vs

Masterthesis

Increasing network utilization and utility for time-sensitive applications

Peter von Zameck Glyscinski

Course of Study:	Informatik
Examiner:	Prof. Dr. rer. nat. Dr. h. c. Kurt Rothermel
Supervisor:	Dr. rer. nat. Frank Dürr, Jonathan Falk, M. Sc. David Hellmanns, M.Sc.
Commenced:	June 9, 2020
Completed:	December 9, 2020

Abstract

Nowadays real-time networks are indispensable when it comes to modern industrial applications. Traffic has to be transmitted with hard or soft delay bounds and guaranteed throughput. Handling this traffic and maximizing the throughput in a network can be very challenging depending on the type of traffic. On the one hand, there is planned traffic in a network which is easy to handle because of its predictable nature. On the other hand, there is unplanned traffic, e.g. event-triggered traffic, which is much harder to manage. This event-triggered traffic is needed to allow applications to improve control loop quality or give them the possibility to handle alarm storms, where one event forces many reactions in the network. Thus, techniques are needed to handle unplanned traffic, such as event-triggered traffic. To do so, over provisioning a network is not an option because it would be to expensive and the resources of the network would not be optimally used. This means new options have to be investigated to handle unplanned traffic for time-sensitive applications.

This thesis introduces a new approach where traffic shaping is used to handle traffic of time-sensitive application like event-triggered traffic. The new approach helps to improve the overall utilization in the network while guaranteeing a certain bandwidth for very application. To do so shaping with application knowledge at the network edge is used to optimize the performance, and in-network shaping with limited knowledge about the network situation helps to cope with worst-case situations.

The approach has the benefit that it is almost stateless and does not have the requirement of any per-stream basis knowledge except of a metering algorithm at the edges of a network. In addition to that the shaping at the edges of a network allows a higher send rate at the application side. This gives applications the possibility to improve utilization in the network, while also limiting single applications from greedily spamming the network with to much data.

The improvements and the increasing overhead as well as necessary technologies of this new approach will be introduced in this thesis. The evaluation shows that it is possible to guarantee a certain bandwidth and delay for every stream while increasing the utilization for specific network scenarios and traffic characteristics such as event-triggered traffic. The thesis also provides the theoretical background on how to calculate a worst-case estimation for the delay, drop rate of messages and the throughput for this new approach. These estimations are done using the Network Calculus framework. Furthermore the fairness between streams and limiting factors for this approach are discussed and evaluated.

Contents

List of Figures

List of Tables

1 Introduction

Nowadays real-time networks are indispensable when it comes to modern industrial applications. Traffic has to be transmitted with hard or soft delay bounds and guaranteed throughput. Handling this traffic and maximizing the throughput in a network can be very challenging depending on the type of traffic. On the one hand, there is planned traffic in a network which is easy to handle because of its predictable nature. On the other hand, there is unplanned traffic, e.g. event-triggered traffic, which is much harder to manage. This event-triggered traffic is needed to allow applications to improve control loop quality or give them the possibility to handle alarm storms, where one event forces many reactions in the network. Thus, techniques needed to handle unplanned traffic, such as event-triggered traffic. To do so, over provisioning a network is not an option because it would be to expensive and the resources of the network would not be optimally used. This means new options have to be investigated to handle unplanned traffic for time-sensitive applications.

To face all these challenges and guarantee Quality of Service(QoS), an important element of the traffic engineering toolbox is traffic shaping. Traffic shaping can be used as an application-specific traffic shaping law at the sources of a network. Applications emitting event-triggered traffic can, depending on their application-specific traffic shaping law, decide on their own whether or not to send data to optimize their application-specific objective, e.g. maximize their performance. Using this application-specific traffic shaping law, the bandwidth of the whole network can be distributed equally or with different weights between all of the applications in the network. This prevents the whole system from congestion and results in a reduced message-loss-rate in the network. On the other hand, a drawback will be a lower network utilization in case of event-triggered traffic, because its characteristic is not to send all the time. Since the whole bandwidth of the network is distributed between the applications in the network and the applications are limited by a fixed maximum bandwidth configured in the application-specific traffic shaping law, they are not able to send more traffic if an application is not sending. This means when an application is not sending data, it will block bandwidth that could be used by other applications which results in an underutilized network [AAK00].

In order to overcome these problems one may just increase the bandwidth of the application-specific traffic shaping law and in case of congestion the switches of the network just buffer or drop messages. The problem with this technique is that an application which greedily spams the network with messages might oust the data from other applications in the network. This can be prevented by using an in-network shaper (or Active Queue Management (AQM) mechanism) [AAK00; FJ93; GHS15; WSKS02] which operates with switch-local knowledge only and has the objective of optimizing fairness between the incoming streams in case of temporary overload situations. Usually these in-network shapers can only be used with a feedback mechanism which can rather be active or passive like the congestion window algorithm in TCP. With this feedback mechanism the sender can be forced to reduce its sending rate and packet drops are minimized. However the previously introduced scenario with time-sensitive event-triggered traffic typically uses

connectionless communication which has no passive feedback mechanism. Also AQM-mechanisms have a high complexity and, therefore, they are usually not implemented in switches, which have only limited computing resources.

To overcome the previous shown problems of an in-app shaper or an in-network shaper this thesis introduces a new hybrid approach where the shaping techniques are combined. That is, we have an ingress-shaping with application knowledge at the network edge to optimize the performance, and an in-network shaping with limited knowledge about the network situation to cope with worst-case situations. The new approach to shape event-triggered traffic for time sensitive applications reserves a certain bandwidth for every stream in the network. It also allows temporary increases of the send rate of multiple streams. This temporal increase of the send rate copes with the characteristic of event-triggered traffic and therefore helps to increase the utilization in the network.

In order to guarantee a bandwidth for every stream in the network, packets are marked by using a metering algorithm at the first hop after every application, e.g. the edge ports of a network. With this marking it is possible to categorize the traffic in conform traffic and un-conform traffic, which might lead to congestion in the network. The marked packets are then shaped based on this marking at the egress of every port of the switches. Packets which were sent with an un-conform rate are dropped if congestion at the egress occurs and the switch is not able to buffer packets anymore. For Packets which were sent in a conform way it is guaranteed that they will never be dropped. With this technique it is possible to guarantee a certain bdanwidth for every stream in the network, at which packets will never be dropped.

This approach has the benefit that it is almost stateless and does not have the requirement of any per-stream basis knowledge except of the metering algorithm at the edges of a network. In addition to that an in-app shaper is used to allow a higher send rate at the application side. This gives applications the possibility to increase utilization in the network, while also limiting single applications from greedily spamming the network with to much data.

The improvements and the increasing overhead as well as necessary technologies of this new approach will be discussed in this thesis. The evaluation shows that it is possible to guarantee a certain bandwidth and delay for every stream while increasing the utilization for specific network scenarios and traffic characteristics such as event-triggered traffic. The thesis also provides the theoretical background on how to calculate a worst-case estimation for the delay and the drop rate of messages as well as the throughput for this new approach. These estimations are done using the network calculus framework [LT01]. Furthermore the fairness between streams and limiting factors for this approach are discussed and evaluated.

The remaining part of this thesis is organized as follows. After this introduction, Chapter 2 will give an overview about technologies which are needed to understand the following parts of the work, followed by a short section about related work in Chapter 3. In Chapter 4 the system model that is used in this thesis is explained followed by Chapter 5 with the problem description. Then in Chapter 6 the new approach of this thesis is introduced, as well as design decisions which have to be done when configuring a network with this new approach. Chapter 6 also shows how to do worst-case estimations on a given network using the network calculus framework. After that Chapter 7 provides an evaluation on the findings of Chapter 6 and shows the approaches limits. Finally Chapter 8 concludes the thesis and also gives an outlook on future work.

2 Technical Background

This chapter is divided into one section about the fundamental network technologies, which are used in this thesis, and the methods which are used to simulate and analyze the new approach of this thesis.

2.1 Network-Technologies

In this section the fundamental network technologies, which are used in this thesis, are explained. These are shaping and policing and the token-bucket-shaper as well as color-shaping. Also the forwarding process in switches of the IEEE 802.1Q [18] standard are explained.

2.1.1 Traffic Shaping

The two important techniques from the network engineering toolbox which are used in this thesis are traffic shaping and traffic policing. Traffic shaping is a technique to deal with temporary spikes in networks and distribution of the demand on resources in a network over time. This is achieved by either limiting the rate at which data is sent, e.g. delaying packets, or throttling the bandwidth at which a source is sending data[AGN12]. A predefined traffic profile which defines parameters like burst size, rate and delay variations is used to deal with this temporary spikes. A burst is an accumulation of data which gets sent with a temporary higher rate, the burst rate, through the network. The duration of the burst is defined by the burst size which is the amount of data that can be sent at the burst rate before throttling the sent rate again.

These traffic profiles are also used in traffic policing but instead of limiting the rate or throttling the bandwidth at which data is sent, traffic policing tries to achieve the goals of the traffic profile by either marking or dropping traffic[CSYM02]. Both marking and dropping packets assumes a feedback mechanism, such that the source of the traffic lowers its send rate. Both traffic shaping and traffic policing have the goal to provide a Quality of Service (QoS) in the network, such as lowering congestion in the network or guarantee maximum delay bounds. The main difference of these two techniques is that traffic shaping will try to distribute the load in the network over time to overcome temporal spikes. To this end, the shaper includes a queuing element which can temporarily store traffic and send it when the load on the network gets lower again. Traffic policing does not have this queuing element and thus just drops traffic and anticipates the source of the traffic to lower its send rate or resents its dropped data. Figure 2.1 shows the difference between policing and shaping network traffic. The spikes in the network policing chart are just cut at the limiting bandwidth, e.g. policed and thus just dropped. The spikes in the second diagram are shaped and thus traffic gets delayed and sent after the incoming traffic rate drops under the maximum bandwidth again.

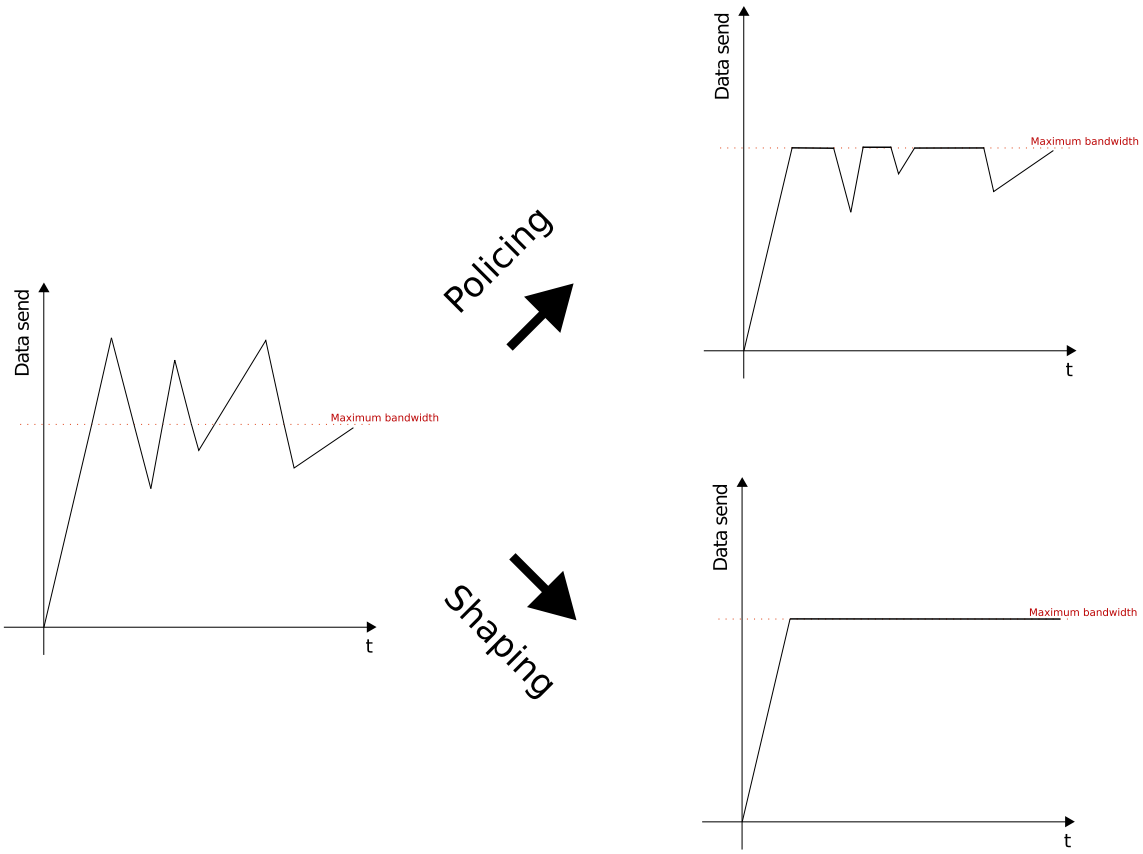

Figure 2.1: Traffic policing vs. traffic shaping.

As most traffic shaping algorithms can also be transferred to a traffic policing algorithm by just decreasing the maximum buffered data size of a shaper to one packet, there is no differentiation between traffic shaping and policing in the following sections and only the term shaper instead of shaper or policer is used.

The two shaping techniques which are used in this thesis are the token-bucket shaper and color shaping. These two techniques are explained in the following sections.

Token-Bucket Shaper

Figure 2.2 shows how the token-bucket shaper is build.

The shaper is working with a token-bucket which can store tokens up to a predefined depth Bu and refills the bucket with new tokens at a defined token rate of $rt\frac{tokens}{s}$. Whenever the shaper wants to send a message from its input queue to its connected successor, the shaper has to consume a specific amount of tokens from its bucket to send the message. The amount of tokens which have to be consumed by the token-bucket shaper are typically one token for every byte the message has. So to forward a message which has a size of, e.g. $512b$, the token-bucket shaper has to consume 512 tokens from its bucket to forward the message to its connected successor. Another configuration

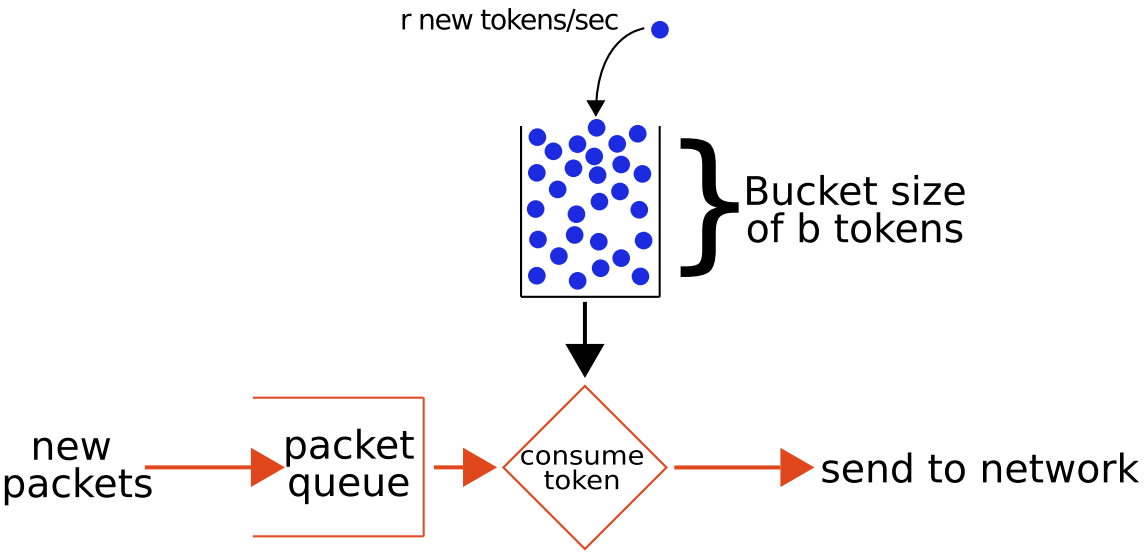

Figure 2.2: Architecture of a token-bucket shaper.

is that one token equals one message, but this is only recommended if all packets, arriving at the shaper, have the same size. Otherwise the data rate at which the token-bucket shaper forwards incoming messages may vary a lot.

If the bucket has not enough tokens left to send a message, it has to wait until enough tokens with rate $r \frac{tokens}{s}$ have been refilled again. If the token rate of the shaper is lower than the rate of arriving messages and the input queue gets full, new messages are dropped.

Color Shaping

A color shaper shapes traffic according to a color marking, which was applied to a traffic stream beforehand. To do so a color shaper consists of two components, a marker which marks incoming traffic and a dropper which forwards or drops messages according to its color marking. These two components can be directly connected with each other but do not have to. For example it is possible to use the marker at the ingress of a switch and the dropper at the egress. A common algorithm which is able to mark traffic according to a traffic profile is the Two Rate Three Color Marker [HG99]. With this algorithm it is possible to maintain fairness between multiple streams while limiting the egress of an shaper to a bandwidth according to its traffic profile. The algorithm marks the arriving packets of a flow on the ingress of a router or switch according to their rate of incoming packets. Packets which are under the so called Committed Information Rate (CIR) are marked green. If a packet passes the CIR it will get marked yellow if it still keeps under the Peak Information Rate (PIR). If a packet also passes the PIR it will be marked red.

In case of a congestion the dropper will decide which packets are dropped based on the color of a packet. Packets which are marked red typically get dropped right after they get marked red. Packets with a yellow marking might be dropped with a higher probability than green packets.

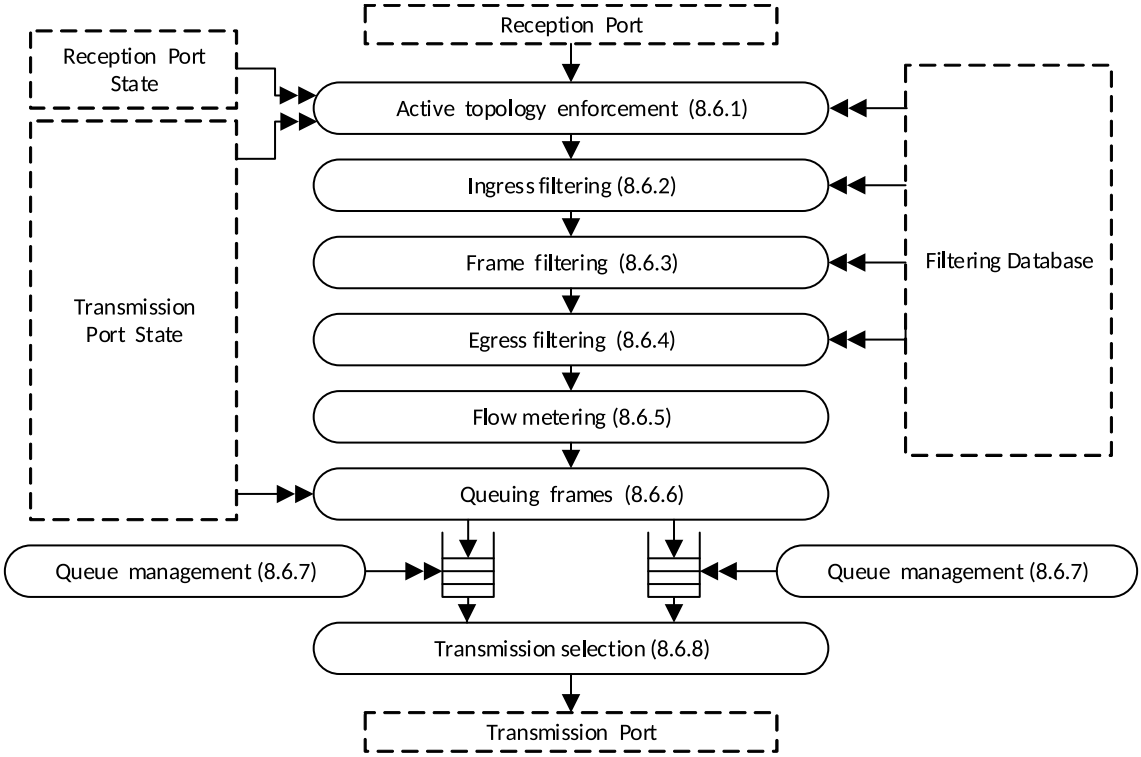

Figure 2.3: Message forwarding process of the IEEE 802.1Q standard[18].

2.1.2 Message Forwarding in Switches

Figure 2.3 shows the message forwarding process for bridge operations in the IEEE 802.1Q standard[18]. The system model in Chapter 4 assumes that this process can be supported by the switches of the model. Packets which arrive through the reception port are forwarded to an active topology enforcement process which whether or not to use the learning and forwarding controls of the switch. Then the packet is forwarded to the ingress filtering process, where packets not conforming to a standard are discarded. This is also the position where the previous introduced Two Rate Three Color Marker can be placed. After that the frame filtering process does filtering decisions like reducing ”the set of potential transmission Ports”[18] of a received packet based on different metrics . The egress filtering, like the ingress filtering, removes non-conform packets. Then in the flow metering process packets may be classified based on their corresponding flow and the destination MAC address, the source MAC address, the VID, its priority or connection identifier[18]. This is also the position where policing or other AQM techniques are applied. After the flow metering a packet gets forwarded to its corresponding egress queue(s) by the queuing frame process. Finally the transmission selection process forwards packets from the transmission queue to its corresponding transmission Port.

2.2 Simulation and Analysis

This thesis uses two techniques to evaluate its new approach. The first technique is the network calculus framework which is used to do worst-case estimations on several performance metrics. A drawback of the network calculus framework is, that it is only capable to give estimations of the worst-case situation of a network. Since the improvements of the new approach are expected to occur in the average, network simulation is used to estimate performance in more realistic scenarios. To do these simulations the OMNeT++ framework is used and will also be introduced in this section.

2.2.1 Network Calculus

Later in section 6.2 a worst-case estimation on networks which are using the new approach is presented. This worst-case estimation is done by using the *Network Calculus* (NC) framework which can be used to analyze communication networks. With its help it is possible to calculate deterministic upper bounds and give guarantees for the traffic and delay in the network for worst-case situations[LT01].

An analyzed network is seen as a system in which traffic is represented by different flows. Arriving traffic of a flow is abstracted as an arrival curve $\alpha(t)$ which gives the maximum cumulative amount of data that can have arrived until time t. The throughput of the components of a system, e.g. a switch or a router, is calculated by using service curves. A service curve $\beta(t)$ gives the minimum cumulative amount of data that was served by the component until time t. Both, arrival curves and service curves are represented as a cumulative function [LT01] R which is defined as follows:

$$R = \{\forall t > 0 : \exists \epsilon > 0 : 0 \leq R(t) \leq R(t+\epsilon), R(t) = 0\ sonst\}. \tag{2.1}$$

For instance the arrival curve of the previously introduced token-bucket shaper can be estimated with the following equation [LT01]:

$$\alpha(t) = Bu + r_{token} * t \tag{2.2}$$

Because the token bucket shaper has the opportunity to forward packets in bursts, the arrival curve of the token bucket shapers output starts with an offset which is as big as the bucket depth Bu. Then the curve rises with token rate rt.

On these cumulative functions upper bounds can be derived by using the so called (min,+)-algebra [LT01] which is a commutative half-ring. To determine the different performance metrics of the network calculus framework the following operations of the (min,+)-algebra are used [LT01]:

1. *Addition*: $\forall t \in X : (f + g)(t) = f(t) + g(t)$
2. *Subtraction*: $\forall t \in X : (f - g)(t) = f(t) - g(t)$
3. *Minimum*: $\forall t \in X : min(f, g)(t) = min(f(t), g(t))$
4. *Maximum*: $\forall t \in X : max(f, g)(t) = max(f(t), g(t))$
5. *Convolution*: $\forall t \in X : (f * g)(t) = \inf_{0 \leq s \leq t}\{f(t-s) + g(s)\}$
6. *Deconvolution*: $\forall t \in X : (f \oslash g)(t) = \sup_{s \geq 0}\{f(t+s) - g(s)\}$

An advantage of the network calculus framework is that it is possible to concatenate multiple systems by using the convolution. This gives a realistic estimation of the whole system and worst-case estimations on a whole system model can be done[LT01].
Furthermore it is possible to use the deconvolution to calculate the virtual delay and the backlog of a system.

Definition 2.2.1
Backlog [Fid10]:
$Backlog \leq \alpha \oslash \beta(0)$

Definition 2.2.2
Virtual Delay [Fid10]:
$Virtual\ Delay \leq \inf[w \geq 0 : \alpha \oslash \beta(-w) \leq 0]$

With the backlog it is possible to determine the maximum buffer which is needed for a system [Fid10]. This is estimated by determining the maximum vertical distance between an arrival curve α and a service curve β. The virtual delay is the maximum horizontal distance between the arrival and the service curve and can be used to determine the maximum delay of data which is sent through a system.
The performance metrics can be used on every communication system. This on the one hand makes it possible to determine the maximum buffer size needed in a switch by using the addition on all arrival curves entering a switch and then calculating the backlog. The delay as well as the output of this cumulative arrival curve can also be calculated. On the other hand a clipper [LT01] makes it possible to clip this cumulative arrival curve to the limits of a fixed buffer size in a switch and then makes it possible to determine the performance metrics on this resulting flow.

A drawback of NC is that it is only capable to do worst-case estimations. This means that in average non-worst-case scenarios the performance of a system might be much better than estimated. Another thing is that NC uses a fluid model for its cumulative functions R which means, that the data of a function can be arbitrarily grained. Compared to a real world scenario where the granularity of data is bounded by the packet size, this will also lead to differences between calculated and measured worst-case estimations in real world.

The Neclas-Lib

To do a worst-case estimation for the new approach of this thesis the neclas-lib [Zam18] is used and extended to evaluate the NC model of this thesis. The python library makes it possible to use the showed NC operations above with little effort and without the need of any extra libraries. It also gives the opportunity to implement new operations, like the clipper, which are needed to do worst-case estimations on networks with the new approach of this thesis.

In this library the arrival and service curves are represented as piece-wise linear functions, which consist of multiple linear pieces that are concatenated to a linear function. These piece-wise linear functions are representing the cumulative functions R which can be periodically extended to infinity.

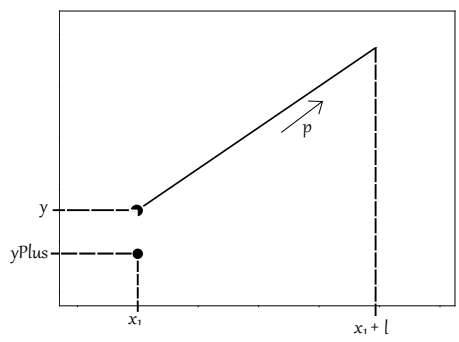

Figure 2.4: A linear piece with its parameters.

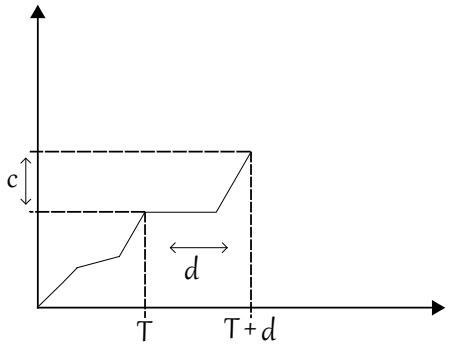

Figure 2.5: A linear function consisting of multiple linear pieces.

Figures 2.4 and 2.5 show how linear pieces and a piece-wise linear function are built. Every piece-wise linear function itself stores a list with non-periodic and a list with periodic linear pieces that are used to extend the function to an arbitrary length. A parameter T marks the beginning of the periodic part from the piece-wise linear function and a parameter c stores the offset on the y-axis of one periodic repetition. Each linear piece has a length l, a slope p and start of its interval x as well as its value at position x which is called y and the value for position $> x$ called y_+.

2.2.2 OMNeT++

OMNeT++[1] is a discrete event simulation framework which can be used to simulate communication networks and other distributed systems[VH08]. Because the previous introduced NC framework only gives estimations and guarantees for worst-case situations, the OMNeT++ simulation framework is used, to get statistical information about non-worst-case situations. In OMNet++ it is possible to simulate random traffic in a network that has certain characteristics. With this simulation the behavior of the new approach of this thesis can be analyzed in a more realistic scenario. With the results of this simulations it is possible to do a statistical evaluation on metrics which can not be calculated using the NC framework. These metrics are the average delay of messages in the network, the throughput, as well as the amount and rate of dropped messages in the network for non-worst-case situations. Furthermore simulating traffic has the advantage that data is not a fluid model anymore, like it is in NC, and thus gives more realistic results.

The core simulation framework of OMNeT++ is written in C++, which makes it fast and its modular simulation software allows to be easily customized, reused, extended and integrated into larger applications [VH08]. It also provides several utilities such as an IDE based on eclipse, a tool to analyze recorded simulations and the possibility to run simulations with an graphical visualization framework, which allows to interact with the simulation during run time.

[1] $https://omnetpp.org/$

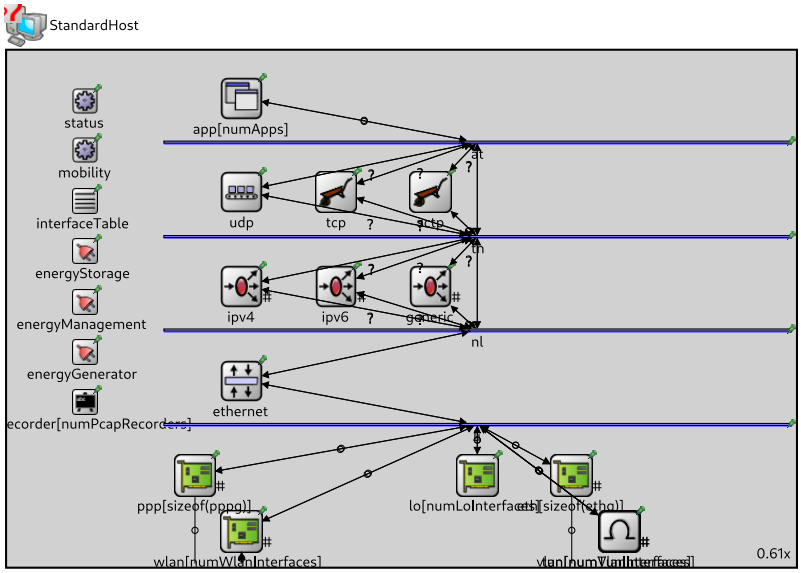

Figure 2.6: Structure of the StandardHost module of the INET project.

The simulation model of OMNeT++ consists of Modules which either are simple modules or more complex compound modules which them self consists either of simple modules or also of compound modules. Thus a network consists of one or several modules which are communicating with each other through messages, that may contain arbitrary data [VH08].

A benefit of the OMNeT++ framework is its huge ecosystem of projects which extend the framework and provide additional modules that can be used in the simulation [FHC+19]. The projects which are used in the evaluation of this thesis are the Network Simulator for Time-sensitive Networking (NeSTiNg)[FHC+19] and its underlying INET Framework[2]. These projects provide several helpful components, such as traffic generators, an EthernetSwitch-component or a simulation model for UDP which is why they are used in this thesis. The important components of the Projects, which are used in the evaluation of this thesis, are introduced in the following sections.

INET StandardHost

Figure 2.6 shows the structure of the StandardHost module of INET. This host was used in the simulations of the evaluations in Chapter 7. The host is structured into multiple layers which fulfill the standard OSI model. On the highest level of the host there are one or multiple *app* modules which are representing the application, presentation and session layer of the OSI model. These apps have to implement the *IApp* interface of INET. An example for an IApp module, which will also be used later in the evaluation, is the *UdpApp* which either generates UDP traffic with a specific characteristic, or receives UDP traffic from other UdpApps. The apps are connected through a *MessageDispatcher* to the transport layer which is represented by the udp, tcp and the sctp modules. These modules again are connected through a MessageDispatcher with the network layer where the ipv4, ipv6 and the generic module handle the addressing routing and traffic control. After that the data link

[2] *https://inet.omnetpp.org/*

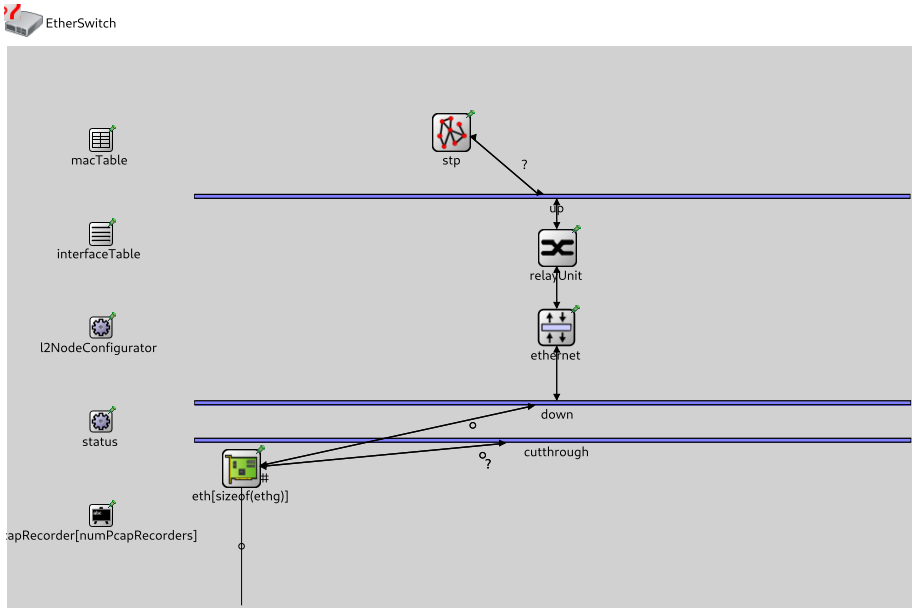

Figure 2.7: Structure of the EtherSwitch module of the INET project.

layer is represented by a *ethernet* module which then gets connected to different traffic interface modules such as the *IEthenetInterface*(eth), the *IWerelessInterface*(wlan), the *IVlanInterface*(vlan), the *IPppInterface*(ppp), the *ILoopbackInterface*(lo) and the *ITunnelInterface*(tun). These interfaces also contain the physical layer where raw bit streams are transmitted or received.

INET EtherSwitch

The next module which is used in this thesis is the *EtherSwitch* module of INET which is shown in Figure 2.7.

The switch has one or multiple *IEthernetInterface*(s) which are connected thorough a two MessageDispatchers to a ethernet module to a relayUnit. This relayUnit handles the mapping between ports and MAC addresses and forwards messages to appropriate ports. It also forwards messages if the upper *ISpanningTree* module is connected, to do route planning in the network.

INET LayeredEthernetInterface

The *IEthernetInterface* of the hosts and in the switches of the network are replaced by the *LayeredEthernetInterface* of INET in the evaluation of Chapter 7. This interface is divided into a macLayer module, which is connected to the upper ethernet module and a phyLayer module which represents the pysical layer of the OSI model. The macLayer module as a default is implemented by the *EthernetMacLayer* module of INET which is shown in Figure 2.8 and also is used and extended for the evaluation in Chapter 7. The layer differentiates between the out-going flow, which is on the left, and the in-going flow which is on the right. Out-going messages are buffered in the queue module which is connected to a server module that forwards the messages in the queue if all underlying modules allow to push a message. Then the streamer module can be used to modify messages which

are going out of the macLayer. Per default this module just passes message to the next module which is the fcsInserter. The fcsInserter inserts the Frame Check Sequence(FCS) for the frame of the message and passes it to the outboundEmitter module. This module just monitors and records out-going traffic and can be configured to only monitor and record traffic that matches with specific characteristics.

At the in-going flow there is an inboundEmitter, which also just monitors the incoming flow and again can be configured to only monitor messages with a specific characteristics. This inboundEmitter is connected to a fcsSchecker module which checks the FCS of incoming messages and then forwards the message to the upper connected module of the EthernetMacLayer.

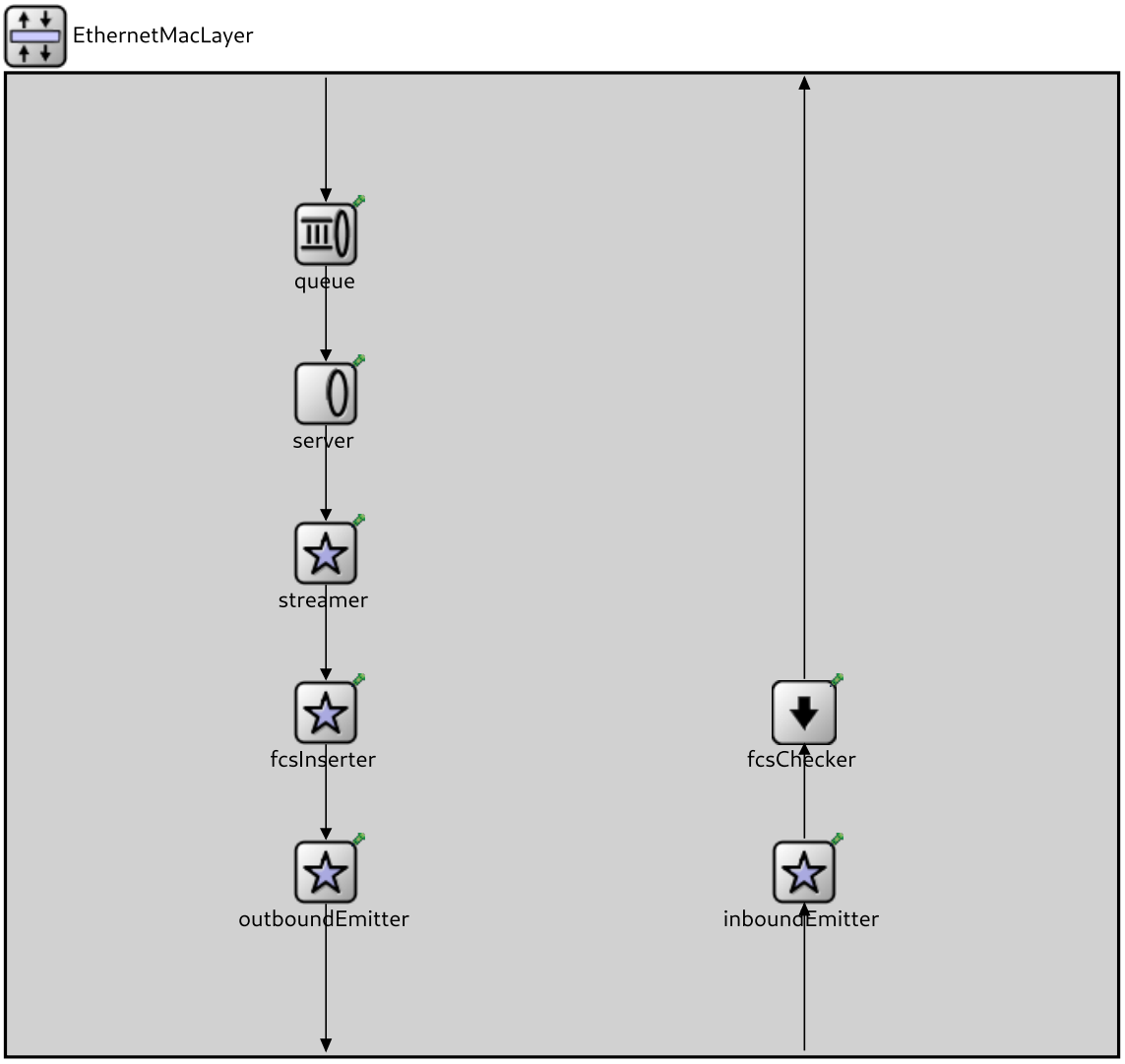

Figure 2.8: Structure of the EthernetMacLayer module of the INET project.

3 Related Work

There are many works which used per-hop forwarding algorithms like the token-bucket or the leaky-bucket algorithm as a traffic shaping mechanism [AAK00][HAB90][SCY+96]. For instance in the work of Alam et. al.[AAK00] the leaky-bucket algorithm was used to shape UDP traffic on the application side. Furthermore in the work of Ghoreishi et. al. [GHS15] it was used as an in-network shaping mechanism. These works showed that the leaky-bucket or the token-bucket algorithm do well in preventing a network from becoming congested. They also provide a simple way to distribute the overall bandwidth of a network fairly and equally between all the sending applications in a network. But as stated before the whole network might be under-utilized with this technique if not all of the nodes are at the peek send-rate of their leaky bucket configuration.

When a token-bucket shaper is used, it is important to find a feasible configuration in order to meet different requirements. Glasmann et. al.[Kli11] published a paper where they estimated token-bucket parameters for videoconferencing systems in corporate networks. In their work they estimated feasible bandwidth requirements to determine an effective bit rate for the videoconferencing systems in a network. They also provide the theoretical background to configure a token-bucket shaper so that an effect bit rate can be reached.

There are also a lot of other works which present or use other network shaping techniques for ATM or AQM like the BLUE algorithm [WSKS02] or the Random Early Detection gateway(RED) [FJ93]. But most of these works assume a feedback mechanism for the senders to reduce their sending rate in case of congestion which can be active or passive. However the system model of Chapter 4 makes it impossible to use a feedback mechanism. In case of a passive feedback mechanism a connection oriented communication like TCP is needed which has too much overhead and a higher delay [AAK00]. In case of an active feedback mechanism the shaper has to inform the sender before congestion occurs, which is not applicable for the restricted computing resources of the switches assumed in the system model.

There are also works which used a color shaper as an in-network shaper, like it is done in the new approach of Chapter 6. One of these approaches is the Rainbow Fair Queueing (RFQ) which was introduced by Cao et al. [ZZZ00]. RFQ uses color markers at the edge routers of a network to classify incoming traffic according to its send rate into different color levels. The core routers of the network have a monitoring algorithm which checks the utilization in the network. Based on the current state in the router a color threshold at which messages should be dropped is set. The dropper in the routers then drop all packets which are over the color level which was set as a threshold. With this technique it is possible to distribute the bandwidth between the different ingress streams of a switch or router equally. Another approach where color shaping was used is the work of Sahu et. al. [SND+00]. In their work a token-bucket shaper was used to classify arriving traffic into two categories at the edges of a network and then gets shaped by the core routers in case of any congestion. They used the Differentiated Services architecture [NC01] to classify the traffic. A

difference to the approach of this thesis is that the work of Sahu et. al. used TCP traffic which already provides a built-in rate control. This is why their approach will not work in the system model of Chapter 4.

A question which has to be answered when using color shaping is how to mark the traffic in a network. To do so, Clark et. al. [CW98] estimated an approach to profile different levels of best-effort service during network congestion. With their approach it is possible to allocate bandwidth to different users during network congestion. Another approach is shown in the work of Kandlurz et. al. [KSS] where adaptive priority marking was used to provide soft bandwidth gaurantees in a differentiated service architecture. Furthermore R. Stankiewicz and A. Jajszczyk provided a work [NC01] where they analyzed the performance of different metering algorithms such as the Single Rate Three Color Marker, the Two Rate Three Color Marker and the Time Sliding Window Three Color Marker.

4 System Model

In this chapter the system model is explained. Since the main focus of this thesis is to increase utilization and utility for time-sensitive applications which emit event-triggered traffic, the Section 4.1 is about the network traffic in the system model. Then Section 4.2 introduces assumptions about the network where this network traffic is sent.

But before introducing the network traffic and the network, the term utilization in a network has to be defined.

Definition 4.0.1
Utilization

Let r_{max} be the maximum data rate which can be send trough a link of a node in the network. Furthermore r_{send} is the rate at which data is send through the link. Then the utilization U is defined as $U = \frac{r_{send}}{r_{max}}$.

The utilization of a network then can be calculated by computing the utilization of all links $l_1, ..., l_n$ in a network U_{li} and then determining their mean $U_{network} = \frac{\sum_{l_1} l_n U_{l_i}}{n}$. A link and thus also a network is underutilized whenever the utilization is < 1.

4.1 Network Traffic

The traffic in the system model is event-triggered time-sensitive traffic. This means that the connectionless User Datagram Protocol (UDP) is used as a transport protocol, which is faster and needs less overhead than the connection oriented Transmission Control Protocol (TCP) [AAK00]. It is also assumed that the source of every traffic, e.g. the application, has no congestion or rate control and will always try to send as much data as possible, to fulfill its targets. Furthermore there is no active feedback mechanism in the network, to throttle the send rate of a source or to notice it about dropped traffic. Traffic which is sent through a network can be categorized into streams where a stream is defined as follows:

Definition 4.1.1
Network stream

A stream S_i is the traffic which is send through a network and has the same destination dest and the same source s_i.

This means that all data which is send from a source s_1 to a sink $sink_1$ belongs to a stream S_1. It is also assumed that the data of a stream is sent in packets of a certain size through the network and that a source only sends data to one sink.

The event-triggered traffic which is generated by the sources of the network will show a specific behavior when monitoring the network utilization. Because the traffic of these senders is event-triggered they will tend to send traffic only if a sudden event occurs. This has the effect that the data of a sender will consist of smaller or bigger bursts instead of a static load. Whenever such an event occurs, a source will try to send as much data about the event as possible to improve its quality of information. An example for such a source would be a camera in an assembly line which scans incoming parts for the next machines in its line. The more detailed the camera scans the part the more information it can give its following machines and will make them more accurate and faster in their work. But more information also means that more data has to be send over the network. This is why an application, which emit this event-triggered traffic, has to be limited in its send rate and bandwidth. If this would not happen, the stream of a single application is able to oust the traffic of all other streams in the network, by forcing congestion which leads to packet drops. This would lead to the problem that most of the appliactions in the network are not able to transmit enough information to fulfill their objective anymore. Congestion in this context is defined as follows:

Definition 4.1.2
Congestion

Let r_{max} be the maximum data rate which can be send trough a link of a node in the network. Furthermore r_{arrive} is the rate at which data, that should be send through the link, arrives at the node. Then congestion appears every time $r_{arrive} > r_{max}$ holds.

This means whenever more data arrives at a node, e.g. a switch (see Section 4.2), than it can forward through its connected links, congestion appears.

To avoid that a stream is able to oust other streams in a network the following has to hold for every stream in the model:

Definition 4.1.3
Guaranteed Bandwidth

For every stream $S_1, ..., S_n$ there is a bandwidth $B_{S_i} > 0Mbps$ at which packets of a stream will never be dropped.

This means that an application s_i which never sends with a bandwidth higher than B_{S_i} has the guarantee that its packets will never be dropped. To achieve this guaranteed bandwidth for every stream in the network, traffic shaping is used. Where and how traffic shaping is used is defined in the next section.

4.2 The Network

The system model is focusing on networks where event-triggered traffic is send through the network. These networks consists of multiple hosts which are communicating with each other through switches. Furthermore a whole network is static which means that there are no link failures and hosts are not joining or leaving the network ones it was set up. We also expect that the route of every stream, going through the network, is known, e.g. there is a global view on the whole network.

4.2.1 The Host Model

Every host in the network can be a source and also a sink. A source has an application which emits event-triggered traffic, that is sent over the network to exactly one sink. A sink only receives traffic from other sources and does not generate event-triggered traffic. It is also assumed that each Host in a network has a token-bucket shaper at its egress with a token rate rt, a bucket size Bu and a connected queue with size Bq_{TB}. Each token in the token bucket equals the weight of one byte in a packet. The token-bucket shaper can directly be accessed by the applications of a host, which brings several benefits. It allows the applications to schedule its messages according to the shaping law, which is more effective than to wait on a feedback of an in-network shaper or even the receiver of the message[HAB90]. With this knowledge it is also possible for each application to effectively use the whole Bandwidth that is given by the token-bucket shaper. If the bucket of the shaper has not enough tokens the application of a host might reduce its send rate to still be able to send its minimum required amount of information. It is also possible to prevent congestion and packet drops inside the network if the traffic profile of the shaper is well chosen.

A drawback of the token-bucket at the host-side is that even a well configured traffic profile may underutilize the network most of the time [AAK00] which will be explained in the problem description in Chapter 5.

4.2.2 The Switch Model

Figure 4.1 shows the data flow of a simple switch of the system model. Packets arriving at the ingress of the switch are forwarded and stored in a queue with size Bq at the corresponding egress of the switch where the packet should be sent. If there is not enough space left in the queue to store another packet e.g. in case of congestion, this packet gets dropped by the queuing frame process of the switch.

It is now assumed that the switches in the system model can be extended to fulfill the forwarding process for bridge operations of the IEEE 802.1Q standard[18]. The switches in the network especially have the possibility to do color shaping with the given operations of the forwarding process. This means every switch has the possibility to do color shaping by marking a packet at each ingress port and filter packets before putting them into one of the egress queues, based on the queue fill stand and the content of a packet. To do the marking of a packet it is also assumed that each packet has a dedicated field where the color of that packet can be set and also can be kept over multiple hops.

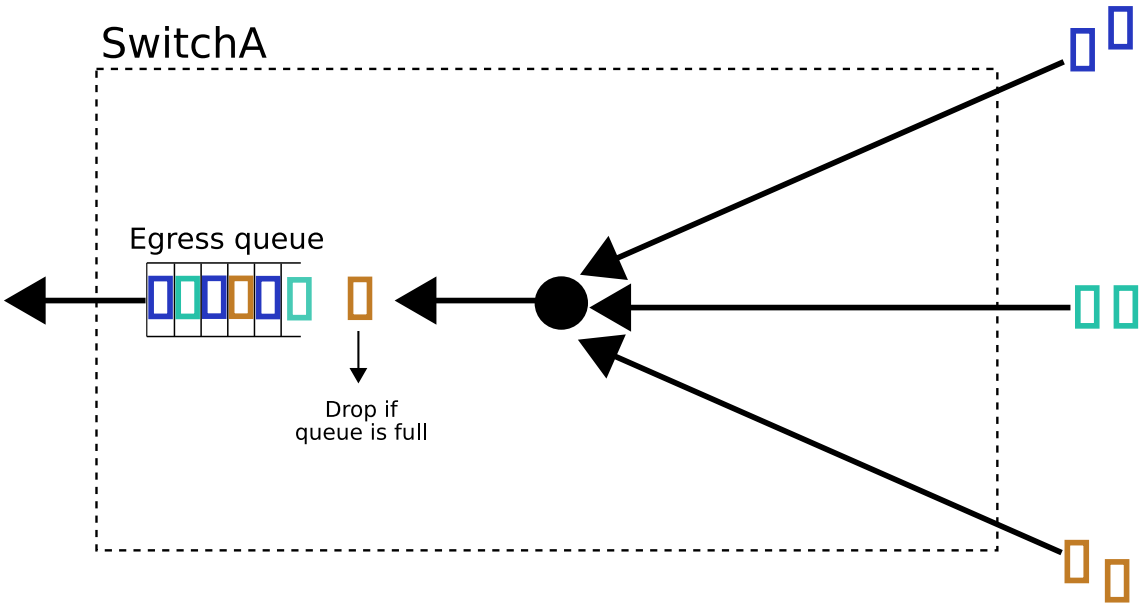

Figure 4.1: A simple switch with three ingress flows and one egress flow with queue. If the egress queue is full, new arriving packets are dropped.

5 Problem Description

In this chapter the problem with networks with traffic of time-sensitive applications is formally defined.

Figure 5.1 shows a simple network with three streams going through one switch. The streams of the network are shaped by the token-bucket shaper of the hosts with configuration parameters rt and Bu. The switch of the network is build like shown in figure 4.1 and the bandwidth of each link in the network is $1Gbps$.

The first problem which now appears is the following:

Definition 5.0.1
Let N be a network with n hosts and m switches with a fixed egress queue size of Bq, where the topology is known. Also the route of each stream $S_1, ..., S_n$ in the network is known and a guaranteed bandwidth $B_{S_1}, ..., B_{S_n}$ in bps for each stream is given. Furthermore it is assumed that if all streams in the network are sending with their guaranteed bandwidth B_{S_i} no congestion in the network occurs and also no more data could be sent without forcing congestion.

Get a network configuration such that definition4.1.3 is fulfilled.

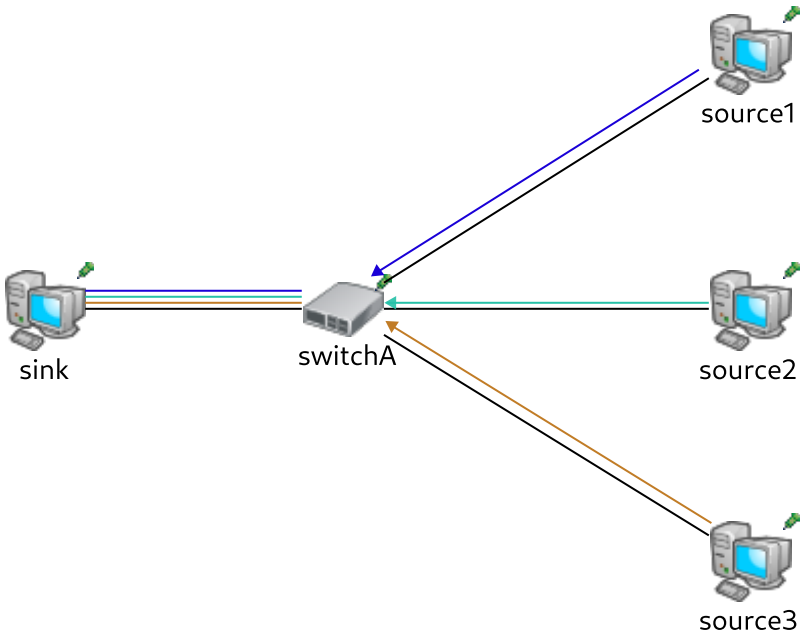

Figure 5.1: A simple network scenario with three sources one sink and one switch.

A naive solution to solve the problem of definition 5.0.1 is to configure the token-bucket shapers in the hosts of the network such that definition 4.1.3 is fulfilled. To do so the token rate of the token-bucket of host i is set to $\frac{B_{S_i}}{1Bps}$ tokens and the token bucket size is set to one. Then all hosts in the network are only able to send data through the network with a bandwidth which is $\leq B_{S_i}$. This also means, because of definition 5.0.1, that with this configuration no congestion will appear in the network and thus the bandwidth for every stream can be guaranteed. For the given network of Figure 5.1 and a given bandwidth of $0.333Mbps$ for each stream S_1, S_2 and S_3 this would mean that the token rate of every token-bucket shaper has to be set to $0.333 10^6 Tps$.

With this naive solution a new problem appears.

Corollary 5.0.1 (Stream characteristic)
Based on the characteristic of the streams in the network, the network will be underutilized.

Underutilized in this context means, that it would be possible to send more traffic through the network without forcing congestion, e.g. there is still bandwidth left in the network that can be used effectively. This corollary holds, because the network will only not be underutilized if all the streams are sending at their maximum rate all the time. If only a single stream pauses its traffic, which will be very common if one of the streams has the characteristic of event-triggered traffic, the bandwidth of that stream will be unused. Therefore the bandwidth could be used by the other streams of the network and thus the network is underutilized. Especially streams with an event-triggered characteristic where events are rare but generate lots of traffic when they occur are problematic. For streams like these a lot of bandwidth has to be reserved, e.g. if there are hard delay bounds which have to be met or a host is not able to buffer the data at the queue of the token-bucket shaper. When now configuring the token-bucket shapers in a pessimistic way like in the naive solution the underutilization will even be higher than with streams which send with many small bursts. Because of that the problem description of definition 5.0.1 is extended:

Definition 5.0.2
Problem Description

Let N be a network with n hosts and m switches with a fixed egress queue size of Bq, where the topology is known. Also the route of each stream $S_1, ..., S_n$ in the network is known and a guaranteed bandwidth $B_{S_1}, ..., B_{S_n}$ in bps for each stream is given. Furthermore it is assumed that if all streams in the network are sending with their guaranteed bandwidth B_{S_i} no congestion in the network occurs and also no more data could be sent without forcing congestion.

Get a network configuration such that definition4.1.3 fulfilled while allowing a high utilization in the network.

This problem will be solved by the thesis. Which is to find a solution with which utilization in a network with event-triggered traffic can be increased, while still providing a guaranteed drop free bandwidth for every stream in the network.

6 Combining In-App with In-Network Shaping

To overcome the problem described in Chapter 5 this thesis improves network utilization and utility by combining an application side and an in-network shaping algorithm, which is shown in Chapter 7. To do so, this chapter will explain the new approach in Section 6.1. After that Section 6.2 will show how to get a worst-case estimation on different metrics for networks with this new approach, by using the NC framework. Finally Section 6.4 will show some general rules when building a network with the new approach.

6.1 Combining In-App and In-Network Shaping

To increase network utilization, which is part of the problem definition 5.0.2, the streams in the network must be able to compensate a reduced send rate of one stream. This is only possible if the token-bucket shapers of the introduced configuration in Chapter 5 are increased. This can be achieved by increasing the token rate and the bucket size of the shapers in the network.

The problem when increasing the token-bucket shapers is that in a worst-case scenario packet drops, as a result of congestion, may occur in the network. This is why the following corollary holds:

Corollary 6.1.1 (Loss of guarantee with higher utilization)
Let N be a network with n hosts and m switches with a fixed egress queue size of Bq, where the topology is known. Also the route of each stream $S_1, ..., S_n$ in the network is known and a guaranteed bandwidth $B_{S_1}, ..., B_{S_n}$ in bps for each stream is given. When now increasing the token-bucket shapers in the network there is a point, that a mechanism inside the network is required to still guarantee a bandwidth for every stream in the network, such that definition 4.1.3 is fulfilled.

This can be proofed by considering the worst-case where all token-bucket shapers send at the same time with their full rate and no in-network mechanism is used which guarantees a bandwidth to every stream. If now increasing the token-bucket shapers token rate and bucket size there will be a point, at which congestion and as a result of that packet drops will occur. Since there is no differentiation between the streams on the egress queues of every switch, it is not possible to guarantee a bandwidth without any drops for the streams in the network anymore. This also means that when increasing the token-bucket shapers even more, there will also be a point where even streams which did not force congestion with their send rate, may be affected from packet drops. E.g. if the three streams in Figure 5.1 share a bandwidth of $1Gbps$ and their guaranteed bandwidth is at $0.33Gbps$, packets of one stream only should be dropped if the stream sends with a rate which is above the $0.33Gbps$. Thus every stream should have a guaranteed bandwidth of $0.33Gbps$ at which no packet drops should occur. In this new scenario one or more streams are allowed to send with a higher rate as their guaranteed bandwidth, because its token-bucket shapers are increased. This may lead to a point where stream S_A forces congestion in the network while stream S_B is sending with its

guaranteed bandwidth. At this point the switch in the network has no knowledge about who forced the congestion in the network. Thus it will just drop arriving packets whether they are from stream S_A or from stream S_B. This will eventually lead to drops of packets from stream S_B and hence quits the guaranteed bandwidth of the stream. Thus a mechanism is needed inside the network, which guarantees a certain bandwidth to every stream in the network again. In this thesis color shaping is used as a mechanism to guarantee a certain bandwidth for every source in the network again.

6.1.1 In-Network Color Shaping

To overcome the previous described problem the approach of this thesis is to combine the token-bucket shapers of the network with additional in-network-shapers. These in-network-shapers will guarantee a bandwidth for every stream in the network, and thus a send rate at which packets will never be dropped. This is done by using color shaping in the switches of the network. The marking process of the color shaper gets positioned on the ingress ports of every switch and the shaper is placed at the egress ports of every switch. In the following the color marking as well as the shaping process in the switch gets explained.

To do the color marking a weaker form of the Two Rate Three Color Marker, that was introduced in Chapter 2, is used. Instead of using two rates with three different colors, this thesis only uses one committed information rate (cir) and in addition a committed burst size (cbs) to mark traffic either red or green. The metering algorithm, which is also called a token-bucket meter[SJ10], has a bucket with tokens and size cbs that is filled with rate cir. Whenever a packet arrives at the marker it gets marked green if there are as many tokens left in the bucket as the packet consumes. If the bucket does not hold enough packets it will be marked red.

Figure 6.1 shows a color shaping switch which uses this token-bucket meter in combination with a shaper.

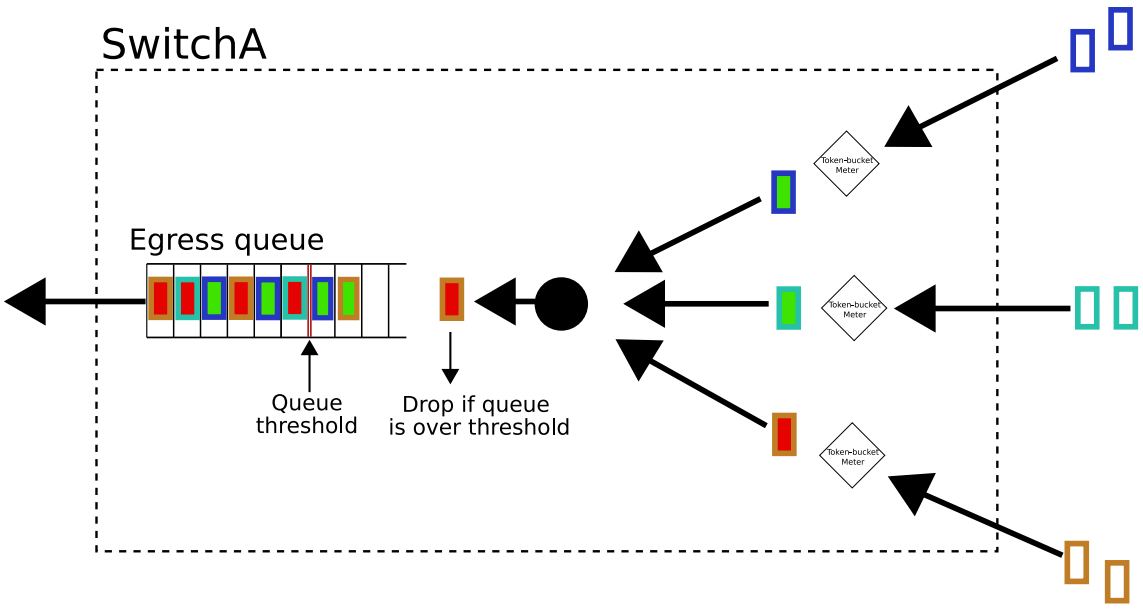

Figure 6.1: Model of the two color shaping switch.

The switch marks incoming packets either green, if they were sent with their assigned and allowed rate, or red if they are exceeding their allowed rate at the ingress port. After that the packets are forwarded to their egress port where the shaper either puts the packets in a fifo-queue or drops them. Packets are dropped if they are red and the queue is filled to a certain threshold Thr in bytes. This threshold need to be configured so that there will always be enough space left for green packets, which we send with the allowed rate. From the queue the packets are send through the egress port with the link rate of the switch. For each port in the switch the following has to hold.

Definition 6.1.1
For a given color shaping switch with m ports for every port i has to hold $\sum_{j \neq i} cir_j \leq r_{link_i}$

This means that for every port the sum of the token-bucket meters cir of all other ports is smaller than or equal to the link rate. Otherwise it would be possible to forward faster green packets to a hop than can be sent over the ports link and thus it would not be possible to guarantee no drops for green packets.

Now it is assumed that a network only consists of n hosts and exactly one switch, that connects all hosts with each other.

The threshold of the egress queues can be calculated by the following lemma:

Lemma 6.1.1
Let N be a network with n hosts and one switch with a fixed egress queue size of Bq and where the route of each stream $S_1, ..., S_n$ *in the network is known. Given an egress port of the switch where streams* $S_1, ..., S_m$ *go through and where stream* S_1 *has the smallest* cbs*, the switch has at least to reserve* $\sum_{i=2}^{m} cbs_{S_i} + 1$ *space in its queue to guarantee no drops of green marked packets.*

To proof this lemma we have to look at the worst case-scenario where one stream fills the queue of the egress port with link rate to its threshold Thr. After the stream has filled the queue the worst-case would be, if all other streams now send at the same time with link rate, while the first stream keeps sending data. In this moment the packets of the other streams get colored green until all cbs tokens in their token-bucket meter are consumed and further packets get marked red. If now all green marked packets arrive at the same time at the egress queue there has to be enough space reserved, such that no green packet gets dropped. In addition, it is also possible that in the moment where all green marked packets of the other streams arrive at the queue, the token-bucket meter of the first source did mark a packet green again which also arrives at the queue. If now the first stream who filled the queue to the threshold was the stream S_1 with the smallest cbs, $\sum_{i=2}^{n} cbs_{S_i} + 1$ space in the egress queue has to be reserved. No more space in the queue has to be reserved, because of the limitation of definition 6.1.1.

From lemma 6.1.1 the following lemma follows.

Corollary 6.1.2
A higher $commitedburstsize(cbs)$ *at the ingress ports of the switches reduces the maximum threshold that can be set in the queue of the egress ports of the switches.*

This corollary follows directly from lemma 6.1.1 because with a higher cbs there has to be more space reserved in every switch queue along the path of the stream. Thus there is less space in the queues left which can be filled with red packets.

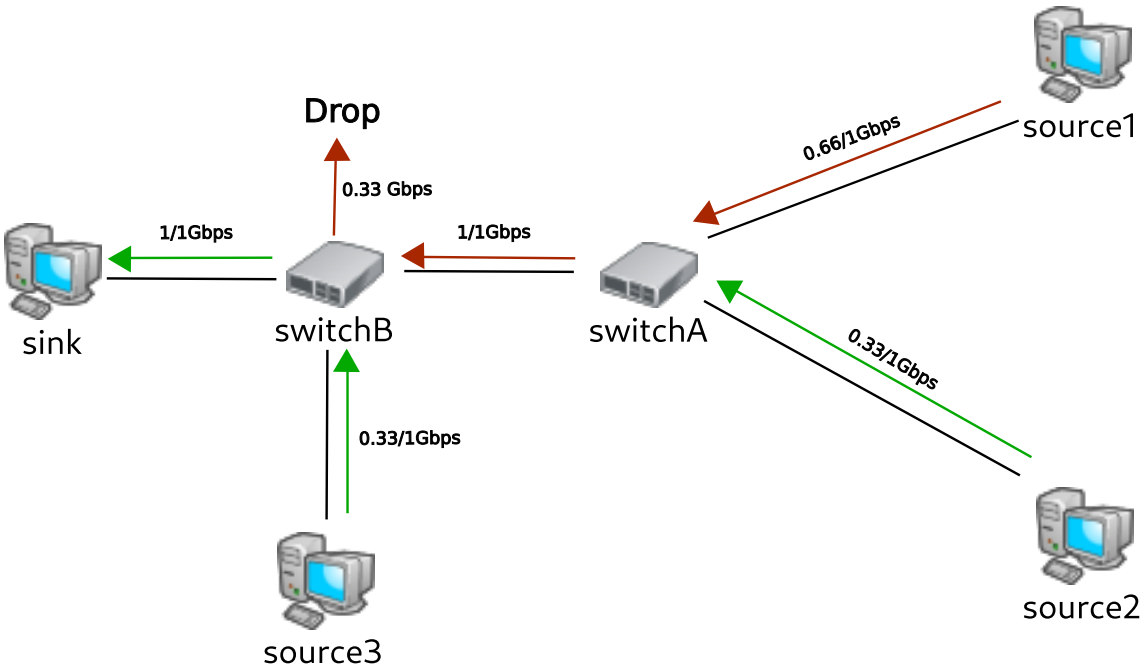

Figure 6.2: Scenario with 3 streams where stream of source1 forces congestion in the network.

This also means that increasing the cbs of the ingress ports of the switches may lead to a higher drop rate if there are streams which are not fully using their maximum cbs. An effective values for the cbs in practice is evaluated in chapter 7.

When using this introduced color shaping method with one switch the following holds.

Theorem 6.1.1
Using the color shaping switch in a network with only one switch and n hosts will guarantee a certain bandwidth for every stream in the network such that definition 4.1.3 is fulfilled.

This follows from lemma 6.1.1 and the previous definitions because the thresholds Thr in the egress queues of the switch can be chosen such that green marked packets will never be dropped. Thus every stream i has a guaranteed bandwidth of cir_i.

With this approach it is possible to increase the token-bucket shapers of the hosts in the network while still guaranteeing a certain bandwidth of every stream in the network, which solves the problem of definition 5.0.2.

A problem with this technique is that it only works for networks with one switch. If the number of hops is increased it could be the case that multiple streams are going through on ingress. This makes it impossible to guarantee a bandwidth to every stream and thus definition 4.1.3 is not fulfilled anymore. This is because the token-bucket meter is not able to differentiate between multiple streams and one stream can consume the whole cir and cbs of the token-bucket meter. Figure 6.2 shows a scenario where packets of streams S_A might be dropped even if it not violates its allowed bandwidth. When the data of stream $S_{source1}$ and $S_{source2}$ get into $switchA$, there will be no drops on $switchA$ because the cumulative data rate of stream $S_{source1}$ and $S_{source2}$ is right at the link rate of $switchA$ which is $1Gps$. But when this cumulative stream from S_A and S_B will arrive at

$switchB$, packets of this cumulative stream will be marked red and then dropped when arriving at the egress port of $switchB$. At this point it can not be guaranteed that the packets marked red are always from stream S_B since $switchB$ does not differ between stream S_A or S_B. A solution for this problem could be to do a per stream differentiation at every ingress of the switches. But this technique would be very expensive, since the state of every stream has to be stored by the token-bucket meter in the switches. Another solution is to mark packets only at the first hop after the in-app-shaper, e.g. the first edge inside a network. This will guarantee a certain bandwidth for every stream over the whole network, at which packets will never be dropped. This method is explained in the next section.

6.1.2 Color Marking at the Edges

To fulfill definition 4.1.3 for networks with more than one hop between a source and a sink, the color marking of the token-bucket meter has to be applied only at the edges of a network. This means that the token-bucket meter is only used at ingress ports in the switches, if these ports are connected to a host. With this technique the token-bucket meters only mark the packets of one stream again and the cir and cbs of every token-bucket meter will only be used by one stream. Since it was previously mentioned that the color marking of one packet is kept over multiple hops an egress port will still always receive colored packets only. What changes with this new technique is the threshold thr which has to be set on every egress queue. This is because now it is possible that a switch receives green packets of a stream with a rate higher than the cir of that stream. This can be formulated by the next lemma.

Lemma 6.1.2
Let $S_1, ..., S_m$ be streams which are going from edge ingress ports through one egress port, where the threshold is Thr. Let $cir_1, ..., cir_m$ be the cir of the token-bucket meters of $S_1, ..., S_m$ and $cbs_1, ..., cbs_m$ their cbs. It is also assumed that $cir_1 \leq cir_2 \leq ... \leq cir_m$. Then it is possible for the egress port to send green packets with link rate r_{link} in bps for a duration db of

$$db = \frac{\frac{Thr*\frac{1}{s}}{\sum_{i=1}^{m} cir_i} + \sum_{i=2}^{m} \frac{cbs_i}{r_{link}}}{r_{link} - \sum_{i=1}^{m} cir_i}$$

This lemma holds because it is possible that stream S_1 fills the egress queue to its threshold Thr with red packets by sending at link rate and then the other streams fill the queue to its maximum with their cbs. If then all streams continue to send only with their cir and hence only green packets will be pushed to the egress queue. This means that when all red packets are forwarded there are $\frac{Thr*s}{\sum_{i=1}^{m} cir_i} + \sum_{i=2}^{m} \frac{cbs_i}{r_{link}}$ green packets in the queue. If now the streams continue to send at their maximum cir the queue will forward only green packets at link rate until the queue is empty. This will be at

$$db * r_{link} - \frac{Thr * \frac{1}{s}}{\sum_{i=1}^{m} cir_i} + \sum_{i=2}^{m} \frac{cbs_i}{r_{link}} - t * \sum_{i=1}^{m} cir_i = 0 \tag{6.1}$$

This equation can then be written as

$$db * (r_{link} - \sum_{i=1}^{m} cir_i) = \frac{Thr * \frac{1}{s}}{\sum_{i=1}^{m} cir_i} + \sum_{i=2}^{m} \frac{cbs_i}{r_{link}} \tag{6.2}$$

which is

$$db = \frac{\frac{Thr * \frac{1}{s}}{\sum_{i=1}^{m} cir_i} + \sum_{i=2}^{m} \frac{cbs_i}{r_{link}}}{r_{link} - \sum_{i=1}^{m} cir_i} \tag{6.3}$$

For this duration the next hop has to make sure it is able to buffer incoming green traffic of other ingress ports. This increases the queue space which has to be reserved for green packets, if an egress queue has traffic which comes from non-edge ingress ports. But before calculating this required queue space the maximum duration of *db* for a series of hops has to be calculated.

Lemma 6.1.3
Let $S_1, ..., S_m$ be streams which come from one ingress queue and are going to one egress queue of a switch of interest. Let $Thr_1, ..., Thr_q$ be the thresholds of the egress queues which are along the path of the stream $S_i \in S_1, ..., S_m$ with the longest distance in hops to the ingress queue of that switch. Furthermore $cir_1, ..., cir_m$ are the cir of the token-bucket meters of $S_1, ..., S_m$ and $cbs_1, ..., cbs_m$ their cbs. It is also assumed that every egress queue that is passed by stream S_i has at least one other stream which sends to the egress queue of the switch of interest. Then the longest duration db at which green packets can be send at link rate r_{link} to the egress queue of the switch of interest are $db = \frac{\frac{\sum_{j=1}^{q} Thr_j * \frac{1}{s}}{\sum_{j=1}^{m} cir_j} + \sum_{j \neq i}^{m} \frac{cbs_j}{r_{link}}}{r_{link} - \sum_{j=1}^{m} cir_j}$.

This lemma holds because if we look at the path of stream S_i then ever switch in its path will have to buffer packets like in lemma 6.1.2 until the burst of green packets of its previous switch is over. This means that in addition to the calculations of lemma 6.1.2 the duration at which packets will be buffered is extended by the thresholds $Thr_1, ..., Thr_q$. If a switch has only stream S_i going its egress queue and no other stream of $S_1, ..., S_m$ then the threshold of that switch do not has to be taken into account for the calculation. This is because this switch will not buffer any new green packets and thus will not extend the send duration of green packets at link rate.

With this duration it is possible to determine the required space for green packets in an egress queue of switch.

Lemma 6.1.4 (Queue space for green packets)
Let $S_1, ..., S_m$ be streams which come from multiple ingress queues and are going to one egress queue of a switch. Let db be the longest duration at which traffic with green packets can come from one ingress queue at link rate r_{link} and from streams $S_1, ..., S_i$. Furthermore let $cir_1, ..., cir_m$ are the cir of the token-bucket meters of $S_1, ..., S_m$ and $cbs_1, ..., cbs_m$ their cbs. Then in the egress queue of the switch at least $db * \sum_{j=i+1}^{m} \frac{cir_j}{s} + \sum_{j=i+1}^{m} cbs_j$ space has to be reserved to guarantee that no green packets will be dropped.

This lemma holds, because for the duration at which the ingress port sends green packets for *sd* seconds, green packets of the other streams have to be buffered.

The amount of data which could comes from all other streams is the data which can be sent at their cir plus their cbs which is $sd * \sum_{j=i+1}^{m} \frac{cir_j}{s} \sum_{j=i+1}^{m} cbs_j$. Lemma 6.1.3 showed that it is possible to determine the duration sd for every ingress port and hence it is also possible to determine the longest duration at which green packets could be sent.

With this new queue threshold the following holds.

Theorem 6.1.2 (Color marking at the edges of the network)
Using a token-bucket meter only at first ingress ports inside a network in combination with a shaper at every egress port of the switches in the network makes it possible to guarantee a certain bandwidth for every stream in the network such that definition 4.1.3 is fulfilled.

This holds because with lemma 6.1.4 it is possible to set the threshold in every egress queue such that no green packets will be dropped. This guarantees a bandwidth to every stream which is the cir of its token-bucket meter.

Also with this technique it is possible to get a configuration such that the problem of definition 5.0.2 is solved. To do so, the given guaranteed bandwidth of every stream is set as a cir of that streams token-bucket meter. Then the token rate and the bucket size of the token-bucket shapers of in the hosts can be increased. This will increase the utilization in the network while maintaining the guaranteed bandwidth for every stream.

What has to be mentioned when using the color marking algorithm is, that red packets don't have any guarantee of fairness. This means whenever red packets are dropped, there is no guarantee that the drops are split equally between the different streams. It is possible that red packets are only dropped from one stream, while all other streams will never be punished for a greedy send rate. This is why the in-app shapers are still needed, to limit streams in their maximum send rate and duration of burst rate, to prevent single streams from ousting the traffic of other streams.

6.1.3 Increasing Overhead and Challenges

When using this combination of in-app shaping and in-network shaping, an implementation overhead which comes with this approach has to be considered. In addition, the following things alos need to be considered when using the color shaping technique.

Requirements to do Color Shaping

The fundamental requirement to use the color shaping technique across a network is the possibility to mark packets. To realize this one of the color marking techniques provided in Chapter 3 can be used.

Another thing that has to be mentioned here is the need of enough queue space in each switch. If there are many streams in the network sharing one link, a lot of space has to be reserved in the egress queue to get the space for lemma 6.1.4. Even for networks with a high bandwidth this can get problematic since today's switches hardware limitations. Even today's switches using chips like

the BCM56980 by Broadcom [1] which are made for high speed networking with a bandwidth of up to $400Gbps$ only have a buffer space of $64MB$. This buffer space has to be divided by all ports of a switch and would only be $1.28Ms$ of buffering at full link speed when using only one channel. So the higher the link speed of a network gets, the lower will be the maximum cbs that can be set for every stream. Also the capacity for red packets which can be stored in the queues will get lower.

Increasing Overhead of the Implementation

As it was stated in Chapter 4 it is assumed that the hosts of a network already have a token-bucket shaper. Also the switches in the network are able to fulfill the forwarding process for bridge operations of the IEEE 802.1Q standard[18]. Continuing on from this scenario the additional overhead of the new approach is the implementation of the token-bucket meter and the shaper at the egress queues to realize the color shaper. Compared to other shaping algorithms such as RED [FJ93], a credit-based shaper [MSL19] or the Rainbow Fair Queuing [ZZZ00], the overhead of the color-shaper is not very high. There is no per stream state needed except of the state of the token-bucket meters at the edge ports of the network. The shaping process of the egress ports can be done without any per-stream state and only by comparing the fill stand of the egress queue and the color marking of a packet.

6.2 Theoretical Worst-Case Estimation

In this section the focus is on how to get a worst-case estimation for a network with the new approach of this thesis, such as the maximum delay and the maximum dropped amount of data for a specific time interval. This is shown by using the network calculus (NC) framework which was introduced in chapter 2. To do the worst-case estimations with NC the first subsection will explain how the output of the token-bucket shapers at the hosts in the network can be modeled as arrival curves. Then section 6.2.2 will explain how these arrival curves of the token-bucket shapers are changed by the Shapers in the switches of the network and how to calculate the output curve, the dropped amount of data and the maximum delay of a switch. Then the last section will discuss how to use this knowledge to calculate the worst-case estimation on a whole network.

6.2.1 Arrival Curves

To get an arrival curve which represents an upper bound for a token-bucket shaper, the already introduced equation 2.2 of Chapter 2 can be used. However in real world scenarios this arrival curve can be tightened up by also considering the maximum link rate r_{link}, at which the token bucket shaper can send packets through the network. To do this first the amount of time it takes to empty the token-bucket at link rate r_{link} has to be estimated. The bucket is empty if the number of tokens send at link rate $r_{link} * t$ minus the maximum number of tokens in the bucket Bu and the produced number of tokens $r_{token} * t$ is zero.

[1] $https://docs.broadcom.com/doc/56980-DS$

(6.4) $t * r_{link} - Bu - t * r_{token} = 0$

This equation can then be written as

(6.5) $t * (r_{link} - r_{token}) = Bu$

which is

(6.6) $$t = \frac{Bu}{r_{link} - r_{token}}$$

Until this point the arrival curve will only rise with the link rate which then is $t * r_{link}$. After the bucket is empty the token bucket shaper is only able to send more data with the token rate r_{token}. This means that the cumulative value of the arrival curve is the amount of data that was sent until the point the bucket was empty $\frac{B}{r_{link}-r_{token}} * r_{link}$ plus the amount of data that was sent after the bucket was empty $(t - \frac{B}{r_{link}-r_{token}}) * r_{token}$. With this estimations the new arrival curve is:

(6.7) $$\alpha(t) = \begin{cases} t * r_{link} & t < \frac{Bu}{r_{link}-r_{token}} \\ \frac{Bu}{r_{link}-r_{token}} * r_{link} + (t - \frac{Bu}{r_{link}-r_{token}}) * r_{token} & t >= \frac{Bu}{r_{link}-r_{token}} \end{cases}$$

In figure 6.3 and 6.4 the difference between the original and the optimized arrival curve can be seen.

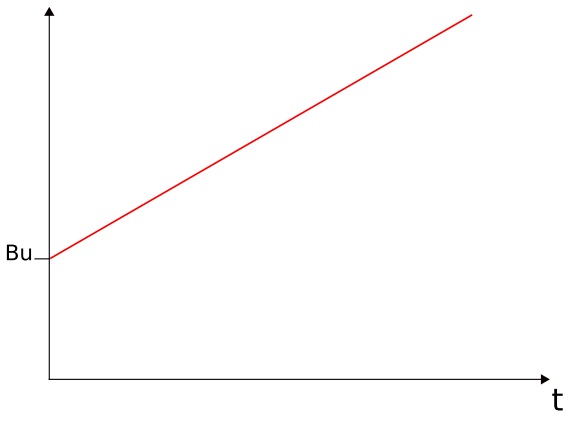

Figure 6.3: Literature

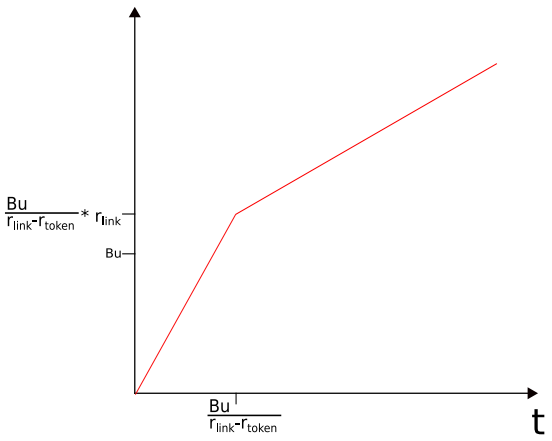

Figure 6.4: Improved curve

Figure 6.5: Comparison between the arrival curve of a token-bucket shaper from the literature and the improved estimation of equation 6.7.

6.2.2 Switch Computations

To calculate the output of a color shaping switch, in this section is explained how to model the shaping process inside the switch. This section also explains how to calculate the maximum delay which can be forced by a switch. To simulate the shaping process of a stream, especially the dropping of data, a clipper[LT01] is used which will divide a stream into data which is forwarded to the queue and data which is dropped. This clipper models the shaper at the egress queue of a switch. To explain how this clipping process works and how to calculate the output of a color shaping switch the following scenario is estimated. We have multiple streams $S_1, ..., S_n$ and a stream of interest S_{shape} which enter a color shaping switch through multiple ingress ports and leafing the switch through one egress port. The egress port of the switch has a buffer size of Bq_{switch} and a service curve β_{switch}. The following sections now explain how to calculate the dropped amount of data of this streams through a clipper and how to calculate the output of the egress port of the color shaping switch.

Shaping a Stream of Interest

To calculate the dropped amount of data for a stream of Interest and also the other streams, the cumulative amount of data which goes through the egress port of the switch has to be considered. This means that for all streams $S_1, ..., S_n$ and the stream of interest S_{shape} the arrival curves $\alpha_1, ..., \alpha_n, \alpha_{int}$ have to be summed up to a cumulative arrival curve α_{cum}

$$\alpha_{cum}(t) = \alpha_{int}(t) + \sum_{i=0}^{n} \alpha_i(t) \tag{6.8}$$

Now a clipper estimates the maximum amount of dropped data at the egress port until time t by calculating:

$$D_{drop}(t) = \sup_{i \leq t} \{\alpha_{cum}(i) - \beta_{switch}(i) - Bq_{switch}\} \tag{6.9}$$

The equation determines the maximum amount of data that was dropped for all times i in the interval of $[0, t]$. The maximum amount of dropped data for a time i is the amount of arrived data $\alpha_{cum}(i)$ subtracted by the already forwarded data $\beta_{switch}(i)$ and the amount of data that can be buffered Bq_{switch}. The biggest difference between the cumulative amount of arrived data subtracted by the amount of already sent data and the queue size, then is the maximum amount which is dropped.

Now to get a worst-case estimation on the dropped amount of data for the stream of interest we have so assume, that as much data of the cumulative dropped data as possible is dropped at the given stream. Which is the minimum of the arrival curve of the stream of interest α_{int} and the dropped amount of data of the cumulative stream D_{drop}.

$$D_{int}(t) = min(D_{drop}(t), \alpha_{int}(t)) \tag{6.10}$$

The minimum is needed because there are two situations which have to be differed. The first is that the amount of dropped data at time t is more than the arrived data of the stream of interest and thus only as much data as has already arrived by the stream of Interest $\alpha_{int}(t)$ can be dropped. The second is that the amount of data arrived by $\alpha_{int}(t)$ is more than the amount of dropped data and thus only these amount is dropped $D_{drop}(t)$.

If there is a guaranteed bandwidth for the stream of interest which is at $g(t)$ this guaranteed bandwidth has to be subtracted from α_{int} before calculating the dropped amount of data. This is because the data which is guaranteed by $g(t)$ is data which can not be dropped and thus has to be removed from the arrival curve of the stream of interest before calculating the dropped amount of data. The resulting equation then is

$$D_{int}(t) = min(D_{drop}(t), \alpha_{int}(t) - g(t)) \tag{6.11}$$

The shaped arrival curves then can be calculated by subtracting D_{drop} from the cumulative arrival curve α_{cum}

$$\alpha_{cumShaped}(t) = \alpha_{cum}(t) - D_{drop}(t) \tag{6.12}$$

The same holds for the stream of interest where the equation is

$$\alpha_{intShaped}(t) = \alpha_{int}(t) - D_{int}(t) \tag{6.13}$$

Not all of this calculations can be done with the neclas library so new functions had to be implemented. Algorithms 6.1 - 6.3 show how the implementation is calculating these shaped arrival curves. The strategy of the main algorithm which is Algorithm 6.1 is to first subtract as much data from the service curve of the switch as possible and assign this data to a cumulative variable f'_{oFlows} for arrival curves $f_1, ..., f_n$ and a variable for the stream of interest f'_{shape}. This is done by first subtracting the guaranteed bandwidth of every stream from both, the service curve and the arrival curves $f_1, ..., f_n, f_{shape}$ and store these bandwidth in f'_{oFlows} and f'_{shape}. Then for the remaining data of the arrival curves of $f_1, ..., f_n, f_{shape}$ are subtracted from the service curve until no more service can β_{switch}. To have a worst-case estimation the data of the arrival curves $f_1, ..., f_n$ is subtracted first from β_{switch} before subtracting the data of f_{shape}. The subtraction is done by determining the maximum data that can be subtracted from β_{switch} with the *CalcClipperMinimum* algorithm (Algorithm 6.3) and then subtracting this value from the service and the arrival curve. Also the value is added to either f'_{oFlows} or f'_{shape}. The *CalcClipperMinimum* algorithm calculates the minimum between a given arrival curve and a given service curve with the limitation, that the resulting curve never has a slope higher than the given arrival curve. Other wise it would be possible that the clipped curve has a higher slope than the original curve which in real world could not happen.

If not enough service is available from the service curve or the possible amount of data from each arrival curve was subtracted from the service curve, the remaining data which can be buffered is determined with algorithm 6.2. These determined amount of data which is returned as $f_{flowBuffer}, f_{otherFlowsBuffer}$ then gets added to f'_{shape}, f'_{oFlows} and then both the shaped cumulative arrival curve and the shaped arrival curve of the stream of interest are returned. In algorithm 6.2 the maximum amount of the remaining arrival curves that can be put into the buffer of the switches egress queue is determined. This is done by illustrating the buffer as a linear function β_{buffer} which always has the value of the buffers size. Then first the maximum amount of data from the remaining cumulative data of $f_1, ..., f_n$ is subtracted from the buffer curve β_{buffer} and stored in $f_{otherFlowsBuffer}$. Then the amount of data from f_{shape} which can be subtracted from the resulting buffer curve is determined and stored in $f_{flowBuffer}$. After that $f_{flowBuffer}, f_{otherFlowsBuffer}$ are returned.

Algorithm 6.1 Shaping of a given set of arrival curves and a arrival curve of interest.

Require: $f_1, ..., f_n, f_{shape} \in R$

Ensure: $f'_{oFlows}(t)+f'_{shape}(t) \leq \beta_{switch}(t)+B_{switch}, \forall t \in R \forall f \in \{f'_{oFlows}, f'_{shape}\} : f(t) >= min(g(t), f(t))$ mit Parametern $g, \beta_{switch}, B_{switch}$

procedure SHAPE($f_1, ..., f_n, f_{shape}$)

$\forall f \in \{f'_1, ..., f'_n, f'_{shape}\} : f \leftarrow min(f, g)$

$\forall f, f' \in \{(f_1, f'_1), ..., (f_n, f'_n), (f_{shape}, f'_{shape})\} : f \leftarrow f - f'$

$\forall f \in \{f'_1, ..., f'_n, f'_{shape}\} : \beta_{switch} \leftarrow sub(\beta_{switch}, f)$

for all $f, f' \in \{(f_1, f'_1), ..., (f_n, f'_n), (f_{schape}, f'_{shape})\}$ **do**

if $\neg$**CapacityLeft**(β_{switch}) **then**

$f_{oFlows} \leftarrow sum(f_1, ..., f_n)$

$f'_{oFlows} \leftarrow sum(f'_1, ..., f'_n)$

$f_{flowBuffer}, f_{oFlowsBuffer} \leftarrow$**CalcBufferClipping**($B_{switch}, f_{shape}, f_{oFlows}$)

$f'_{shape} \leftarrow f'_{shape} + f_{flowBuffer}$

$f'_{oFlows} \leftarrow f'_{oFlows} + f_{oFlowsBuffer}$

return f'_{shape}, f'_{oFlows}

end if

$f_{min} \leftarrow$**CalcClipperMinimum**(f, β_{switch})

$f' \leftarrow f_{min} + f'$

$f \leftarrow f - f_{min}$

$\beta_{switch} \leftarrow \beta_{switch} - f_{min}$

end for

$f_{oFlows} \leftarrow sum(f_1, ..., f_n)$

$f'_{oFlows} \leftarrow sum(f'_1, ..., f'_n)$

$f_{flowBuffer}, f_{oFlowsBuffer} \leftarrow$**CalcBufferClipping**($B_{switch}, f_{shape}, f_{oFlows}$)

$f'_{shape} \leftarrow f'_{shape} + f_{flowBuffer}$

$f'_{oFlows} \leftarrow f'_{oFlows} + f_{oFlowsBuffer}$

return f'_{shape}, f'_{oFlows}

end procedure

Algorithm 6.2 Calculation of the flow which can be put into the given buffer.

Require: $f_{shape}, f_{otherFlows} \in R$

Ensure: $\forall t \in R : f_{flowBuffer}(t) + f_{otherFlowsBuffer}(t) \leq B_{switch}$ mit Parameter B_{switch}

procedure CALCBUFFERCLIPPING($B_{switch}, f_{shape}, f_{otherFlows}$)

$\beta_{buffer} \leftarrow$ Linear function which has always the value B_{switch}

$f_{cumFlow} \leftarrow f_{shape} + f_{otherFlows}$

$f_{buffer} \leftarrow minimum(\beta_{buffer}, f_{cumFlow})$

$f_{otherFlowsBuffer} \leftarrow minimum(f_{buffer}, f_{otherFlows})$

$f_{buffer} \leftarrow f_{buffer} - f_{otherFlowsBuffer}$

$f_{flowBuffer} \leftarrow minimum(f_{buffer}, f_{shape})$

return $f_{flowBuffer}, f_{otherFlowsBuffer}$

end procedure

Algorithm 6.3 Calculate the amount of data that can be removed from the given capacity curve.

Require: f, β_{switch}
Ensure: $f' \leq \beta_{switch}, \forall x \in R : f'(x) <= f(x)$ with parameters T, d, c
procedure CALCCLIPPERMINIMUM(f, β_{switch})
if $\frac{c_1}{d_1} < \frac{c_2}{d_2}$ **then**
$d \leftarrow d_1$;
$m_1 \leftarrow \sup_{T_1 \leq t < T_1 + d_1} (f(t) - \frac{c_1}{d_1} t)$;
$m_2 \leftarrow \inf_{T_2 \leq t < T_2 + d_2} (\beta_{switch}(t) - \frac{c_2}{d_2} t)$;
$T_x \leftarrow \frac{m1 - m2}{\frac{c_2}{d_2} - \frac{c_1}{d_1}}$
$T \leftarrow max(T_x, T_1, T_2)$;
else if $\frac{c_1}{d_1} > \frac{c_2}{d_2}$ **then**
$d \leftarrow d_2$
$m_1 \leftarrow \sup_{T_2 \leq t < T_2 + d_2} (\beta_{switch}(t) - \frac{c_2}{d_2} t)$;
$m_2 \leftarrow \inf_{T_1 \leq t < T_1 + d_1} (f(t) - \frac{c_1}{d_1} t)$;
$T_x leftarrow \frac{m1 - m2}{\frac{c_1}{d_1} - \frac{c_2}{d_2}}$
$T \leftarrow max(T_x, T_1, T_2)$;
else
$d \leftarrow kgV(d_1, d_2)$;
$T \leftarrow max(T_1, T_2)$;
end if
Extend(f, T, d);
Extend(β_{switch}, T, d);
Merge linear pieces of f and β_{switch}

Calculate minimum over all linear pieces of f and β_{switch} with restriction that the slope of resulting linear pieces are never greater than the slope of f and the resulting linear function is rising monotonous and save it to f'
return f'
end procedure

Switch Output and Maximum Delay

To calculate the output arrival curve for the stream of interest S_{int} the deconvolution of the shaped arrival curve $\alpha_{intShape}$ and the service curve β_{shape} that is applied to $\alpha_{intShape}$ has to be calculated. To get the applied service curve β_{shape} the cumulative shaped arrival curve of the other streams which is $\alpha_{otherShape} = \alpha_{cumShape} - \alpha_{intShape}$ has to be subtracted from the service curve of the switch β_{switch}[BF08]. This means for the shaped arrival curve $\alpha_{intShape}$ and the shaped arrival curve of all other streams $\alpha_{otherShape}$ the service curve that is applied to $\beta_{intShape}$ is

$$\beta_{shape}(t) = \beta_{switch}(t) - \alpha_{otherShape}(t) \tag{6.14}$$

After this the output arrival curve is calculated by the deconvolution of the shaped arrival and the applied service curve

$$\alpha_{out}(t) = (\alpha_{intShape} \oslash \beta_{shape})(x) \tag{6.15}$$

The maximum delay is calculated by determining the maximum horizontal distance between the shaped arrival curve and the service curve[Fid10]

$$\inf[w \geq 0 : \alpha_{otherShape} \oslash \beta_{switch}(-w) \leq 0] \tag{6.16}$$

respectively

$$\inf[w \geq 0 : \alpha_{intShape} \oslash \beta_{switch}(-w) \leq 0] \tag{6.17}$$

for the stream of interest.

6.3 Calculate Worst-Case Metrics

The previous sections showed how to get an arrival curve for all token-bucket shapers in a network and how the output curve, the dropped amount of data and the maximum delay of a shaping switch can be calculated. Now this section will discuss how it is possible to calculate worst-case estimations for a stream of interest in a whole network. This could be done by calculating the arrival curves of all token-bucket shapers in the network and then calculating the output of the stream of interest for every egress queue it has to pass. This would also mean that for every egress both the stream of interest and the output of all other streams passing this switch have to be calculated. Also the output of all other switches which goes through the same egress of a switch as the stream of interest have to be calculated. If this is done for every switch which the stream of interest passes, it is possible to determine the minimum throughput of the stream of interest in the whole network. It is also possible to determine the dropped amount of data of that stream by calculating the difference of the throughput of the stream of interest and its arrival curve from the token-bucket shaper. Also the delay can be calculated by determining the delay of each switch the stream of interest went through and then summing these delays up to the overall delay.

These calculations would be very expensive since every output and maximum delay of the switches has to be calculated explicitly. This is why future work could also focus on how to get a more efficient estimation on a whole network using the color shaping switches and a token-bucket shaper at the hosts.

6.4 Network Planing

After introducing the new approach and explaining how to calculate worst-case estimations, this section will show some general rules when designing a network with the new approach. First some general rules on how to choose the parameters of the token-bucket shapers and especially how to prevent them from a to big configuration. Then formulas and rules for the maximum delay bounds of red and green packets are introduced.

6.4.1 Token Bucket Parameters

So far the effect of using in-network-shapers with in-app-shapers in a network and the corresponding possibility to increase network utilization while guaranteeing and maintaining a certain send rate for every stream in a network have been discussed. In this subsection the focus is on how the parameters of the in-app-shaper can be increased, to get a higher utilization in the network. The first question that is answered is which streams get a profit from an increased token-bucket shapers token rate. If the stream is not using the extra bandwidth that is given by a higher token rate then it makes no sense to increase its send and token rate, because this only increases the possibility of congestion in the network. For that reason the following two theorems are applied.

Corollary 6.4.1
There is a point where a higher token rate for the token-bucket shaper will not increase the utilization any more.

Corollary 6.4.2
A higher token bucket size for the token-bucket shaper potentially leads to a higher drop rate in the network.

For Corollary 6.4.1 there are two cases, which have to be observed. The first case is when the token rate gets higher than the actual send rate of its source. If the source sends packets slower than the token-bucket shaper generates tokens, a higher token rate will have no effect on the output of the token-bucket shaper. The second case is when the overall congestion in the network is already to high and a higher token rate would only lead to more packet drops. If the network is already at its limit and more packets will only lead to more drops, the utilization in the network will not increase anymore. This is why the token rate of a token-bucket shaper should not be set higher than the average send rate of its source.

Corollary 6.4.2 applies because the point where the most congestion will happen in a network, is when the bursts of several token-bucket shapers are overlapping. This means that multiple token-bucket shapers are sending with their burst and these bursts are arriving at an egress queue of a switch at the same time. When increasing the duration of a burst by increasing the bucket size of the token bucket shaper, the probability that several bursts are overlapping gets higher. Thus the probability of a higher drop rate also increases. This will decrease QoS and will also not increase the utilization in the network. This means that the bucket size of a token bucket shaper should be chosen as small as possible to reduce the probability of overlapping bursts of multiple streams. If an application needs a high send rate it is better to increase the token rate of a token bucket than the bucket size, to have a better and more precise worst-case estimation on a network configuration.

The last corollary deals with the weight at which a stream can send into the network.

Corollary 6.4.3
A higher burst and send rate of the token-bucket shaper of one stream will lead to a higher throughput of that stream in the network.

The idea of this corollary is that if a stream is sending more packets than the other streams, thus it is present in the network with a higher weight, it may also have a higher throughput in the network. With this it is possible to give a stream, which profits the most of a higher throughput, a ”bigger”token-bucket shaper with a higher token rate and also a higher bucket size. How many streams in the network can get such a ”bigger”token-bucket shaper is evaluated in Chapter 7.

6.5 Delay Bounds

When talking about the delay bounds of the a network with the new approach the following corollary gives a upper bound for green marked packets.

Corollary 6.5.1
Let S_i be a stream which passes switches with queue sizes $Bq_1, ..., Bq_m$ and the link rate of the network r_{link}. Then the maximum delay d_maxg of a green packet of stream S_i is bounded by $d_maxg \leq \frac{\sum_{j=1}^{m} Bq_j}{r_{link}}$.

This holds because the maximum delay a green packet can have in a switch, is if it is put at the last position of the fifo-queue. This means if the queue has a space of Bq_j that in a worst-case, where the packet fills the queue completely, the packet will take $\frac{Bq_j}{r_{link}}$ to get send to the next hop. If this happens at all switches that the packet passes its maximum delay will be $d_maxg \leq \frac{\sum_{j=1}^{m} Bq_j}{r_{link}}$.

For red packets the maximum delay, if they reach their destination, is bounded by:

Corollary 6.5.2
Let S_i be a stream which passes switches with queue thresholds $Thr_1, ..., Thr_m$ and the link rate of the network r_{link}. Then the maximum delay d_maxr of a red packet of stream S_i, if it reaches its destination, is bounded by $d_maxr \leq \frac{\sum_{j=1}^{m} Thr_j}{r_{link}}$.

This holds because red packets will only be put in the queue of a egress port if the threshold is not exceeded. Therefore the maximum delay a red packet can have is when it is put into the queue such that the threshold Thr_j of the queue is reached. Then it will take $\frac{Thr_j}{r_{link}}$ until the packet is sent to the next hop. If this happens at all switches that the packet passes its maximum delay will be $d_maxr \leq \frac{\sum_{j=1}^{m} Thr_j}{r_{link}}$. This also means that red packets have a lower worst case delay than green packet, but also don't have the guarantee to arrive at their destination because then can be dropped.

7 Evaluation

In this chapter the findings of Chapter 6 are evaluated. Which are the effect of token-bucket shapers with a higher token rate and bucket size on the utilization, the effect when also using color shaping on the guaranteed bandwidth and the findings of Section 6.4. To do this new modules for OMNeT++ had to be implemented to support the color shaping of this thesis as well as the token-bucket shaper and different traffic characteristics. How these modules are implemented is described in Section 7.1. After that in Section 7.2 the measured properties of the simulations are explained and how these are measured in the simulated networks. Finally Section 7.3 will show the results of the evaluations which were done using the OMNeT++ and the NC Framework.

7.1 OMNeT++

This section explains which extensions to the NeSTiNg and INET modules had to be done for the OMNeT++ simulations. These are the color shaping, the token-bucket shaping and the event-triggered traffic in the simulations. To simulate the new approach of Chapter 6 in the different network scenarios own modules are defined on top of the NeSTiNg and INET modules. To simulate different traffic characteristics new *ActiveSource* modules are defined which then can be used in the *StandardHost* module of the INET framework. Furthermore the StandardHost had to be extended to use the token-bucket shaper. These extensions are explained in Section 7.1.1. The color shaping is done by a new *ShapingSwitch* module which is build on top of the INET *EtherSwitch* module. How the *ShapingSwitch* module does the color shaping is explained in Section 7.1.2.

7.1.1 EventTriggeredSource

To simulate an event-triggered host the *StandardHost* module of the INET framework has to be extended such that it contains a token-bucket shaper and that it is able to generate event-triggered traffic. For the event-triggered traffic the *UdpApp* module of th INET framework had to be changed. There the *source* module is replaced by one of the new source modules implemented for this thesis. Another extension to the standard the *StandardHost* is that its *EthernetInterface* is replaced by a *LayeredEthernetInterface*. To do the token-bucket shaping a new module the *TokenServer* is used as the server module of the *macLayer* in the *LayeredEthernetInterface* of the hosts. The new modules are explained in the following sections.

EventTriggeredSources

There are three new sources which are implemented and used in the EventTriggeredHosts UdpApp, to simulate different traffic characteristics. These characteristics are the following:

- Static traffic load
- Periodic bursts
- None periodic bursts

The static load is to simulate sources in a network which produce a static predictable load. In the later simulations it is important that traffic like this is not affected by the event-triggered traffic, even if the event-triggered traffic forces a high congestion in the network. The other two characteristics are used to simulate the event-triggered traffic in a network. With the periodic bursts it is possible to simulate scenarios where multiple sources in a network are triggered by the same event and start to send bursts at the same time. Examples for this are alarm storms. The characteristic with non periodic bursts is needed to simulate random event-triggered traffic where a random event forces a source to send much data as one bursts.

All these new sources are build on top of the *PacketSourceBase* module from INET. This module already provides the infrastructure to generate INET packets with a specified packetSize, a packetnameFormat, a packetRepresentation and the packetData.

ActivePoissonSource

The first source is the ActivePoissonSource which generates a packet stream which has a poisson distribution. This source generates traffic with a static load. In addition to the PacketSourceBase module the ActivePoissonSource has the parameter *meanNumPacketsPerSecond*. This parameter specifies the mean number of packets that are generated per second. The implementation uses this parameters value to generate random values with a poisson distribution. This is done by using the build-in poisson-method of OMNeT++. The generated value is then used to determine the time which will be between the current and the next packet, e.g. the inter-arrival time. This time is estimated by calculating $wait\_time = \frac{1s}{poissonValue}$. Because of the poisson distribution the ActivePoissonSource generates a stream where the inter-arrival time between two packets is very equal but not always the same. This leads to a stream with a static characteristic, whit a small variance in the sources send behavior. The characteristic when observing there send rate in $Mbps$ of the ActivePoissonSource can be seen in Figure 7.1. There we can see that the ActivePoissonSource generates a very static load that produces only small changes in its send rate and stays at $\approx 3.3 Mbps$ which is its configured mean send rate.

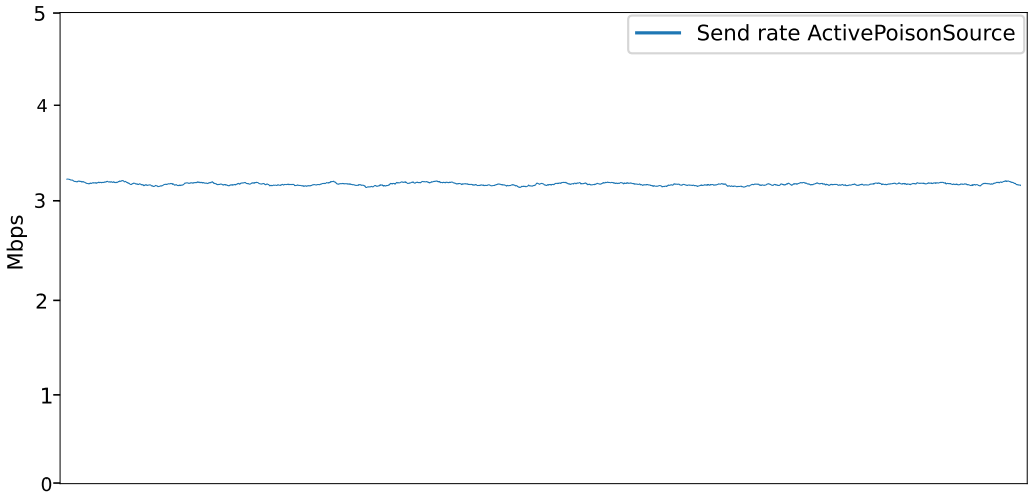

Figure 7.1: Send characteristic of an ActivePoissonSource.

ActiveEventTriggeredSource

The second source is the ActiveEventTriggeredSource(ActiveETS) which generates a packet stream that produces bursts of different sizes. This source is used to generate a stream with periodic bursts up to a predefined size. This is done by using a Cauchy distribution. In addition to the PacketSourceBase module the ActiveETS has a parameters which limits the maximum number of packets per second *maxNumPacketsPerSecond*. Until this limit is reached, packets are produced by using the build-in cauchy-method of OMNeT++ with the same time estimation like in the ActivePoissonSource. The two parameters a and b of the cauchy-method are set to $3 * maxNumPacketsPerSecond$ and $maxNumPacketsPerSecond$. With these values the distribution will produce a high burst at the beginning of every interval until the maxNumPacketsPerSecond are reached. Every interval has a length of one second. With this source it is possible to simulate streams which periodically sends bursts of different sizes through the network which are synchronized with all other ActiveETS in the network. With this it is possible to simulate alarm storms, where many sources generate bursts at the same time. This happens because bursts are always produced in intervals with the same duration of one second and all sources starting to send at the same time in the network. The characteristic when observing there send rate in $Mbps$ of the ActiveETS can be seen in Figure 7.2. Periodic bursts at the beginning of the intervals can be observed where the send rate fast rises and then falls after the burst ends.

ActiveBinomialCauchySource

The last source that is used in this thesis is the ActiveBinomialCauchySource(ActiveBCS). This source is used to simulate non periodic event-triggered traffic by combining the ActiveETS with a binomial distribution. In addition to the ActiveETS the ActiveBCS has the parameter *intervalLength* which specifies the length of the send interval of one burst. The source then starts to send data in intervals which have the specified length. Before every interval the source uses the build-in uniform-method of OMNeT++ to determine whether or not to send data in that interval. If the source decides to send data in an interval, it starts to send packets with the same mechanism as the

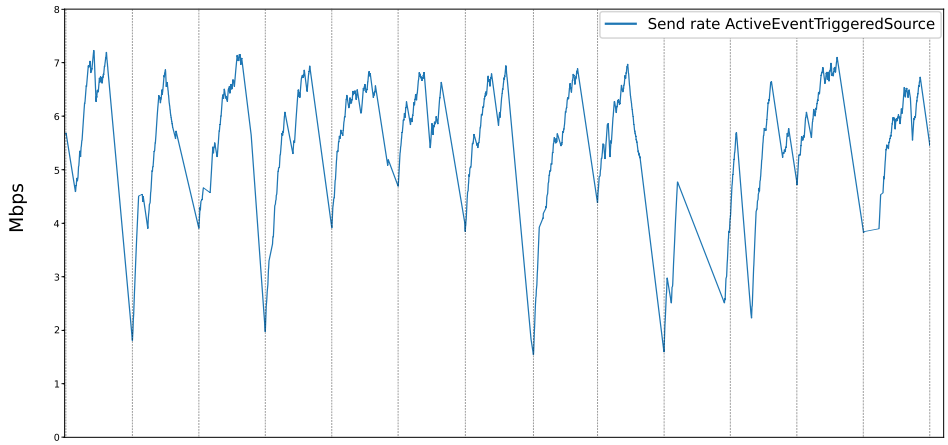

Figure 7.2: Send characteristic of an ActiveEventTriggeredSource. The intervals are marked as dotted lines

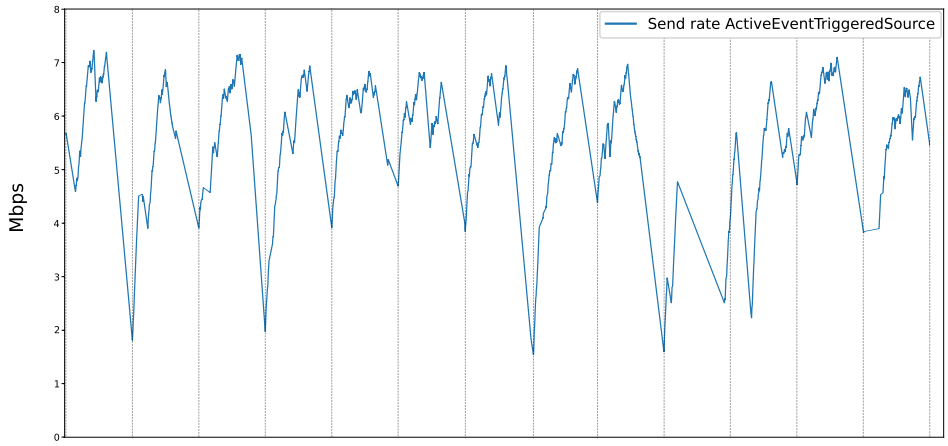

Figure 7.3: Send characteristic of an ActiveBinomialCauchySource. The intervals are marked as dotted lines

ActiveETS until its limit of $\frac{maxNumPacketsPerSecond}{intervalLength}$ in that interval is reached. The characteristic when observing there send rate in $Mbps$ of the ActiveBCS can be seen in Figure 7.3. Compared to the send behavior of the ActiveETS bursts don't have this perfect periodic behavior, since there are intervals where the source does not send any data. This can also be seen when comparing the lower send rate in the Figures 7.2 and 7.3. While the send rate of the ActiveETS only has 6 moments where it drops under $2Mbps$ the send rate of the ActiveBCS does it 14 times. Also the minimum send rate of the ActiveBCS is lower where it two times almost drops down to $0Mbps$.

In addition to these three new implemented sources some simulations are also using the *ActivePacketSource*(ActivePS) to simulate a maximum static load. The source uses a *prdocutionInterval* parameter to determine the time until the next packet should be send. When setting this time as high as the maximum number of packets which can be sent at link rate of the host, it produces a maximum static load.

TokenServer

To get a token-bucket shaper into the Host the server module in the MacLayer of the LayeredEthernetInterface is replaced by a new implemented TokenServer. This TokenServer represents the token-bucket shaper. The module is based on the PacketServerBase-module of INET. In addition the TokenServer gets six parameters which are *tokenConsumptionPerPacket*, *tokenConsumptionPerByte*, *tokensPerSecond*, *initialNumTokens* and *maxNumTokens*. The *tokenConsumptionPerPacket*- and *tokenConsumptionPerByte*-parameters determine how many tokens should be consumed by the token-bucket shaper per packet or byte. *TokensPerSecond* defines the token rate of the shaper and *initialNumTokens* and *maxNumTokens* the initial number of tokens in the bucket and the bucket size. The server internally stores the amount of tokens available in a variable and removes these tokens before a packet gets forwarded. The server tries to forward a packet every time a certain event occurred. These events are that a new packet arrived at the queue which is connected to the server or that a packet can be pushed to the underlying module of the server. These two events follow the push and pull based approach, which is used in the LayeredEthernetInterface. In addition to that there is a Message event which is triggered by a message timer. This message event is used to schedule a timer when there are enough tokens to forward a packet again. Whenever one of these events occur the server tries to forward a new packet to the underlying connected module. To do so the server checks if the connected queue contains a packet and if there are enough tokens available to send the packet. If there are not enough tokens the server schedules a token timer message to the moment where enough tokens available again. New tokens are added to the internal variable whenever a new event occurs. This is done by storing the time where the last event occurred(*lastUpdateTime*). Then the new amount of tokens are calculated by $newTokens = (currentTime - lastUpdateTime) * tokensPerSecond$ where $currentTime$ is the current simulation time where the event occurred. Tokens are not added periodically, like it would happen in real world, because this would not suit the event based system of OMNeT++ where events only should occur if needed and to be more efficient.

7.1.2 ShapingSwitch

The ShapingSwitch-module is based on the INET EtherSwitch-module and uses the MacRelayUnit-component of NeSTiNg as relayUnit and the LayeredEthernetInterface-module with a modified MacLayer as EthernetInterface. The mac layer has two main modifications to the original one. The first is the *TwoColourMarker* module which classifies incoming traffic at the ingress either in green traffic or red traffic. Figure 7.4 shows how this marker, which is put between the fcsChecker of the mac layer and its upperLayerOut, is build. The marker consists of a Classifier, two color marker and a join node. The two color marker is represented by the *TokenBucketMeter* module of the INET project to mark incoming traffic. This TokenBucketMeter marks packets either red or green based on a given committed information rate(cir) and a committed burst size(cbs). Because the TokenBucketMeter can only handle TCP and UDP packets, a *ContentBasedClassifier* of INET is used to send only

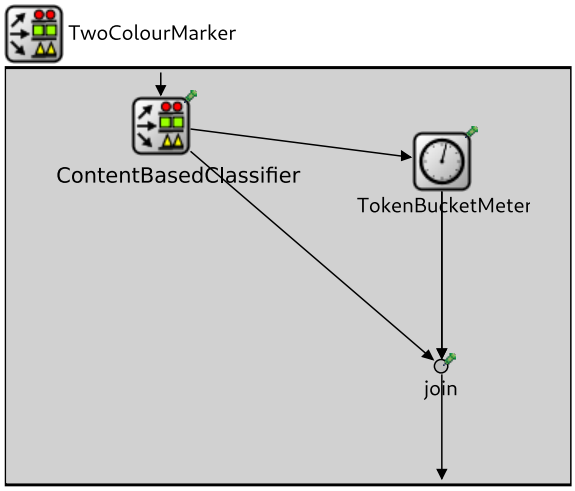

Figure 7.4: Structure of the TwoColourMarker module in OMNeT++ which is used to mark incoming traffic either red or green.

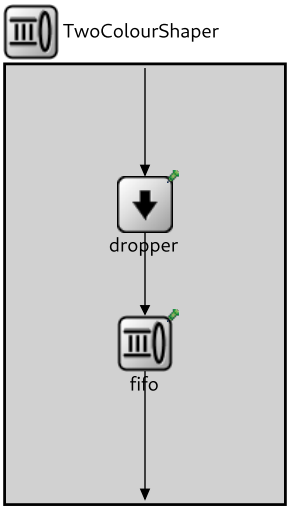

Figure 7.5: Structure of the TwoColourShaper module in OMNeT++ which is used to shape outgoing traffic based on its color marking.

UDP packets to the TokenBucketMeter. Incoming is either send to the TokenBucketMeter if it is UDP traffic and then forwarded to the join node or directly forwarded to the join node. With the ContentBasedClassifier later it is also possible to circumvent the TokenBucketMeter if a port is not an edge port in the network.

The other modification in the mac layer is the *TwoColouShaper* module at the egress side which replaces the default queue in the mac layer. The structure of the two color TwoColouShaper can be seen in Figure 7.5. The shaper consists of a queue which has a dropper in front of it that drops incoming traffic if it is marked red and the queue is filled to a certain threshold. To do so the dropper get the two parameters *packetThreshold* and *queuePacketCapacity*. Whenever a packet arrives at the shaper the dropper checks the current amount of packets in the queue. If the queue contains less packets than the packetThreshold the dropper forwards the packet to the queue. If the queue is

filled to the packetThreshold the dropper checks whether the packet is marked green or not. A green packet is still forwarded until there is no space left in the queue. Packets which are not marked green will be dropped at the moment where the packetThreshold is reached. Furthermore it is possible to set the shaper in an color aware mode with the bool parameter *dropperIncolorAwareMode*. If the shaper is set to the color aware mode the dropper will always forward packets to the queue no matter which color marking a packet has.

7.2 Measured Properties

To evaluate the performance of a shaping configuration the following network properties are measured, the network utilization, message loss, observed delay bounds and whether the guaranteed send rate for every stream were met or not. This section will explain how these metrics are defined and how they are measured during the OMNeT++ simulations.

7.2.1 Network Throughput

With the network throughput it is possible to evaluate the effectiveness of the shapers configuration in a network scenario. This is done by considering the average network throughput during a test scenario and combining this value with the number of dropped messages. The higher the throughput while keeping the number of dropped messages low, the better is the effectiveness of the shapers configuration and the bandwidth of the network is used optimally.

The network throughput is defined as follows.

Definition 7.2.1 (Network throughput)
The throughput of a network $Th_{network}$ is the amount of data A_{ri} which arrives at the sinks of a network in bits per second.

$Th_{network} = \sum_{ri=0}^{n} A_{ri} \frac{bit}{s}$

In this thesis either the average throughput of a whole simulation was measured or the throughput over time. To calculate the the average throughput the sent data of the simulation $A_{overall}$ was summed up and then divided by the duration of the simulation in seconds $D_{simulation}$, which then gives the average throughput $Th_{average} = \frac{A_{overall}}{D_{simulation}}$. To determine the throughput over time the exponential moving average(EMA)[GCR01] for irregular time series was used. The EMA smooths the calculated throughput between two data points, in this case two packets which arrived at a sink, by considering the last measured value and the time between the data points. The for a value v_{i+1}, an smoothing factor α, and the previous EMA value EMA_{prev} and v_i, where time delta between v_i and $v_i + 1$ is t_{delta}, the EMA can be calculated by

$$EMA(v_i{+}1) = \left(\frac{t_{delta}}{\alpha}^{-1} * EMA_{prev}\right) + \left(\frac{1 - \frac{t_{delta}}{\alpha}^{-1}}{\frac{t_{delta}}{\alpha}^{-1}} - \frac{t_{delta}}{\alpha}^{-1}\right) * v_i + \left(1 - \frac{1 - \frac{t_{delta}}{\alpha}^{-1}}{\frac{t_{delta}}{\alpha}^{-1}}\right) * v_{i+1} \tag{7.1}$$

With this technique it is possible to observe the throughput in the network over time.

To measure the amount of data which arrived at the sinks in the network, the amount of arrived packets at every sink in a network is recorded at the UdpApp of the sinks during simulation. Then the amount of data can be calculated by multiplying the amount of packets by the size of each packet.

7.2.2 Message Loss

The second metric that will be used to evaluate a simulation is the amount of dropped messages in a network. Together with the throughput of the network this will show how effective a shaping configuration is. It is also possible to determine the congestion in the network, because whenever a packet gets dropped in the network it will be because of congestion. It is also possible to determine the drop rate of packets in the network or of a stream which is defined as follows.

Definition 7.2.2 (Drop rate)
The drop rate d of a system is defined as $d = \frac{P_{dropped}}{P_{pushed}+P_{dropped}}$*, where* P_{pushed} *is the amount of packets which where pushed in a system during a predefined time interval and* $P_{dropped}$ *is the amount of dropped packets in this interval.*

As a time interval in the evaluations of Section 7.3 the duration of the simulation was used. As a system the whole network was used where the amount of dropped packets were recorded at the dropper of the egress queues of the switches in the network and the amount of forwarded packets in the token-bucket shapers.

7.2.3 Observed Delay Bounds

A very important network property that has to be observed is the delay of received messages. The message delay was measured by measuring the time between the egress of the TokenServer e.g. when the token server forwards a packet from the queue to the lower part of the Ethernet layer of the hosts, and the arrival at the ingress of the UdpApp of the receiving sink.

7.2.4 Guarantee

To evaluate that the guaranteed bandwidth of every stream in the network doesn't get violated the color marking of the edge switches is used. In every test scenario and test run the ingress port of every edge switch marks packets which were sent at the allowed and guaranteed rate of the stream green. Those color markings are captured in a vector along with the *treeid* of the packet. Later when packets are dropped in the network it is possible to check whether the packet was marked green at the edge of the network or red. If a green packet is dropped the guaranteed bandwidth of that stream was violated Even if the color marking is not used or changes during the simulation, it will be possible to recognize the drop of green packets in the network by comparing the treeid of dropped packets with the treeid of green marked packets at the edges of the network.

7.3 Evaluating Different Traffic Scenarios

This section shows the evaluation of different network scenarios to show the different definitions, theorems, lemmas and corollaries of Chapter 6. To do so, first a simple one-hop scenario was used where the effect of the color shaper on a per-hop basis can be evaluated and is also capable to compare simulations with the worst-case NC estimation. Then bigger network scenarios with more than one hop were evaluated, to compare the color shaper on a per-hop basis and the color shaper using the token-bucket meter at the edges of a network to mark packets. After that the delay behavior and previously mentioned delay bounds of Chapter 6 are observed and evaluated, in different network typologies. Finally the effect of giving on or more token-bucket shapers a higher weight and the effect of a fixed switch buffer for a higher network bandwidth are observed.

But before analyzing the results of the different network scenarios some general information have to be discussed.

A first remark that has to be mentioned is that either the mean number or the maximum number of packets of the PacketSources were always set as high as the token rate of the corresponding token-bucket shaper. Otherwise the following would happen.

Lemma 7.3.1
The token rate of a token-bucket shaper has to be as high as the mean send rate of a stream. If not the token bucket will have the same characteristic as a leaky bucket.

If the token rate is to low, the bucket of the token bucket will always be empty and thus will produce no bursts. With this behavior the token-bucket shaper will act like a leaky-bucket shaper. This will remove the burst characteristics of the ActiveEventTriggeredSource or the ActiveBinomial-CauchySource and would lead to wrong estimations for event-triggered traffic.

Another remark is that all the simulations below were made with multiple repetitions. This is because the traffic of the three new PacketSources is generated randomly and thus different seeds have to be used to get a statistical mean value for the analyzes. To check whether the amount of repetitions of one configuration is feasible to get a statistical value, the confidence interval [Le 10] between the results of two different PacketSource configurations were checked. Generally every configuration was run with 10 repetitions where the run number was chosen as the seed for random numbers. Then for these repetitions a confidence interval with confidence level $\gamma = 95\%$ was checked and repetitions were increased if confidence could not be reached.

7.3.1 One-Hop Scenarios

The first simulations were made in a network with one hop which is shown in figure 7.6. Three sources are connected to one sink through $switchA$. Also the points where the throughput and the dropped packets are measured is shown in the figure. Th_1 - Th_o marks the points where the received packets are recorded with which the throughput is calculated. D_1 - D_o are the points where dropped packets are recorded which happens at the dropper of the egress queue. Finally S_1 - S_2 marks the points where the packets are recorded that were sent by each source of the network. This network topology was used to analyze the effect on the utilization in the network for an increased token-bucket shaper configuration for different traffic characteristics. Then the effect of using the

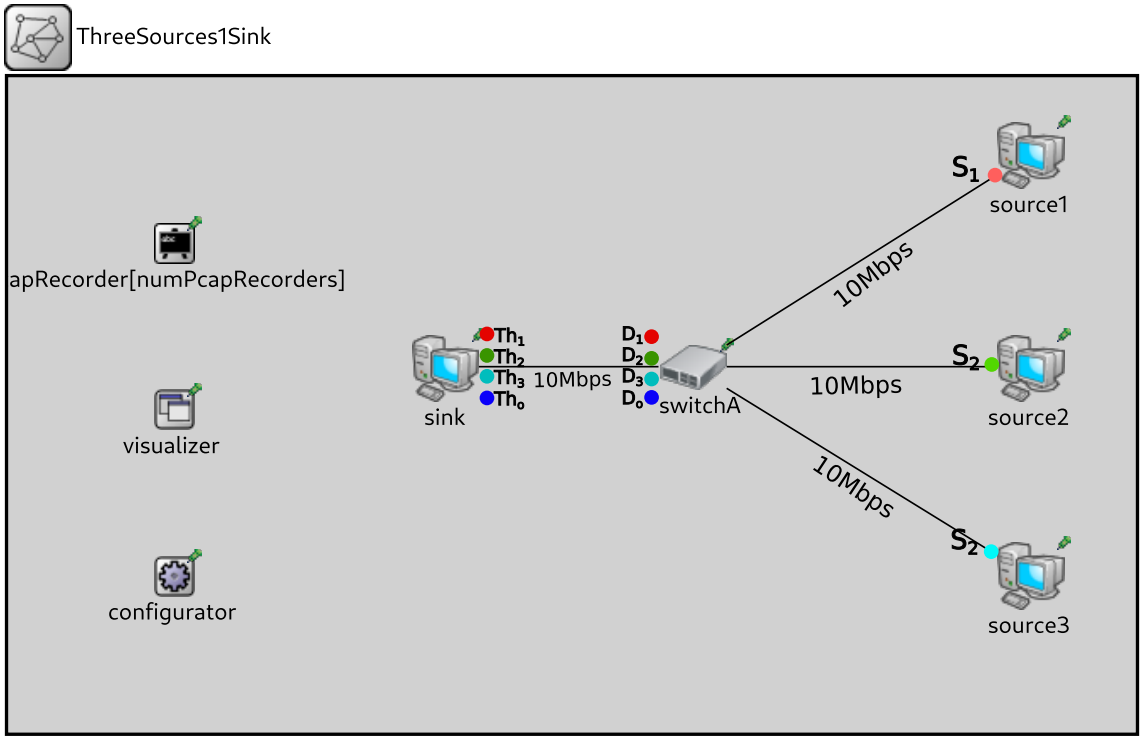

Figure 7.6: Topology of the simulations with a one hop scenario. Three sources are connected with one sink through one switch. Measuring points are marked with a circle.

color shaping switch was evaluated, as well as the right committed burst size for different traffic characteristics. Also the buffer space which is needed for green packets in the egress queue of the shaping switch was evaluated. Finally the results of the simulations were compared to the NC worst-case estimations.

All one-hop configurations, and also all further simulations if not other stated, were simulated with a packet size of $613B$ and with a link rate of $10Mbps$. Because of the overhead of modules in the Ethernet layer and the inter frame gap of 12 byte between packets, the switches were only able to forward packets with a rate of $9.622Mbps$. This is why this rate is used for further estimations when calculating the cir and the cbs. The guaranteed bandwidth according to definition 5.0.2 for every stream in the network was defined as $3.2Mbps$. Because every packet in the simulation has the same size the token consumption of the token-bucket shapers in the simulations were all set to one token per packet. In all simulations the token buckets were full at the beginning of each simulation.

The cbs of the two color marker was set to 1% of the guaranteed link rate of its stream, which will also be used in all further configurations if not other stated. This value was chosen because of the results from the tests in section 7.3.1. For the tests the buffer size was set to a fixed capacity of 1000 packets which means that the egress of a switch can buffer packets up to a half second at the link rate of $10Mbps$. Also this configuration of the switch buffer capacity was used in all further configurations if not other stated.

Parameter	Values				
Packet source	ActivePS	ActivePS s1 ActiveETS s3	ActivePS s1 ActiveBCS s3	ActiveETS	ActiveBCS
Token rate	$635Tps$	$1000Tps$	$1333Tps$	$1666Tps$	$2000Tps$
Bucket size	333 tokens		666 tokens		1000 tokens

Table 7.1: Alternating configuration parameters for source1 and source3 in simulations of the one hop topology.

Increasing the Utilization

The first simulations in the one hop network were made to show the influence of the token-bucket shaper parameters on the utilization in the network. This was measured by increasing the token rate and the bucket size of the token-bucket shapers in the network. The simulations were also made to show that without color shaping in the network, the guaranteed bandwidth for every stream gets lost with a higher token rate or bucket size.

In the simulations source source1 and source3 had alternating parameters for their packet source and their token-bucket shaper which can be seen in Tabular 7.1. All combinations of these parameter values were simulated in multiples runs with different seeds.

The parameters of the token rate and the token bucket were chosen such that the first configuration, with a token rate of $653Tps$ and a bucket size of 333 tokens, does not force any packet drops and hence will fulfill the guaranteed bandwidth for every stream. Then the token rate and the bucket size was increased to see which effect this has on the throughput and the drop rate in the network. Furthermore different packet sources in the UdpApp were used to see the effect of different traffic characteristics on the throughput and the drop rate. The rate of produced packets of the ActivePS in these simulations were chosen such that the sources will always produce a a full load on the token-bucket shaper to simulate a worst case scenario. This was achieved by setting the *productioninterval* to $0.0004s$ which means that packets are produced with a rate of $\frac{1}{0.0004s} * 613B = 1532500Bps = 12.26Mbps$. The maximum number of packets per second for the ActiveETS and the ActiveBCS were always set to the token rate of the token-bucket shaper such that lemma 7.3.1 was fulfilled. In addition to these variable parameters the packet source of source 2 was set to an ActivePoissonSource which produced packets with a mean of $653Pps$ to generate a static load. Also the token rate of source2 was set to $653Pps$ and its bucket size was set to 333 tokens. The droppers in *switchA* were set in the *colorAwareMode* such that they will ignore every color marking. Thus packets were only be dropped if the queue of the egress port was full.

The first results of the simulations can be seen in Figure 7.7. There the drop rate and the corresponding throughput of the simulations where an ActivePS was used is shown. The simulations were made with a bucket size of 333 tokens and an increasing token rate. For the tests with an ActivePS for source1 and source3 the configuration with a token rate of 653 tokens per second did not produce any packet drops during the simulation. Also the the throughput is at a maximum of $9.622Mbps$. When increasing the token rate the drop rate rises while the throughput stays at the same level. This already was expected since source1 and source3 are sending at a maximum rate all the time. This also explains why also increasing the bucket size only increases the drop rate but not the throughput

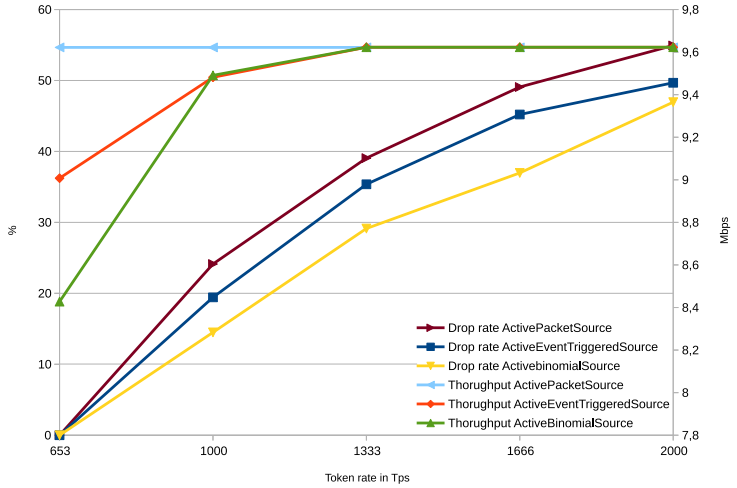

Figure 7.7: The drop rate and the throughput of the simulations with different bucket sizes plotted for different token rates and different PacketSources for source3. The bucket size is = 333 tokens

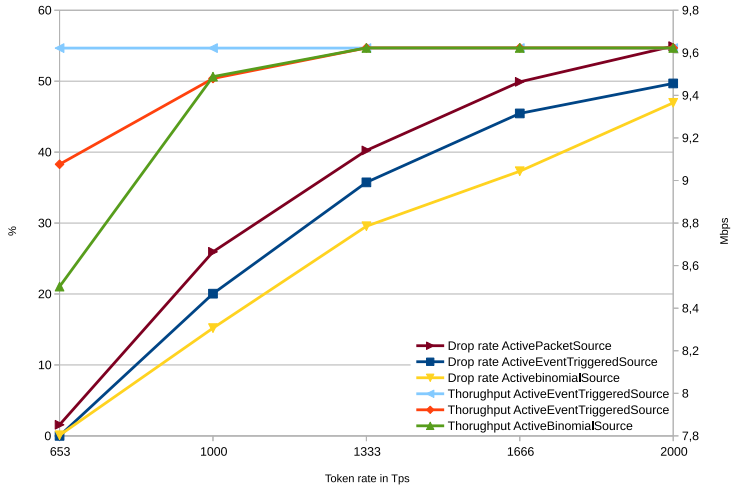

Figure 7.8: The drop rate and the throughput of the simulations with different bucket sizes plotted for different token rates and different PacketSources for source3. The bucket size is = 666 tokens

anymore. This behavior also showed of when increasing the bucket size to 666 tokens (Figure 7.8) or to 1000 tokens (Figure 7.9). Figure 7.10 furthermore shows, that there are packets dropped which are marked green and thus should have the guarantee to not be dropped. Figure 7.10 also shows that the number of dropped packets is very different between *source1* to *source3*. It also shows that *source2* has packet drops even it was sending not higher than its guaranteed rate of $3.2Mbps$. This all shows that for a higher token rate source1 - source3 have no guaranteed bandwidth anymore.

The figures also show that a higher token rate or bucket size does not bring any improvements for networks which are sending with a static rate like in this first configurations.

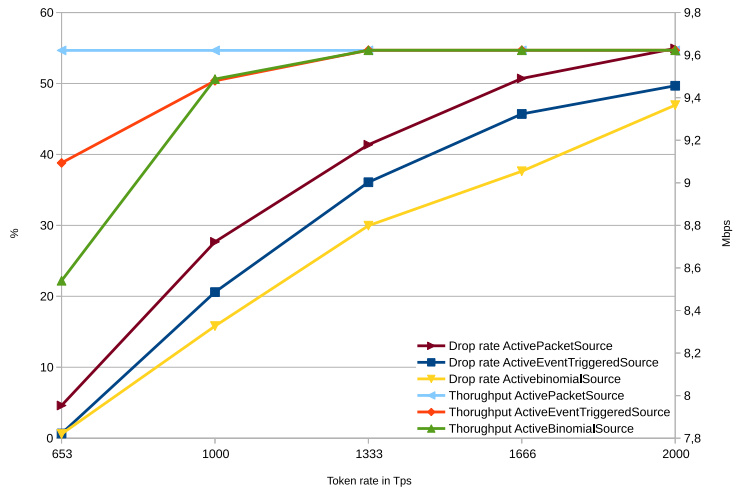

Figure 7.9: The drop rate and the throughput of the simulations with different bucket sizes plotted for different token rates and different PacketSources for source3. The bucket size is = 1000 tokens

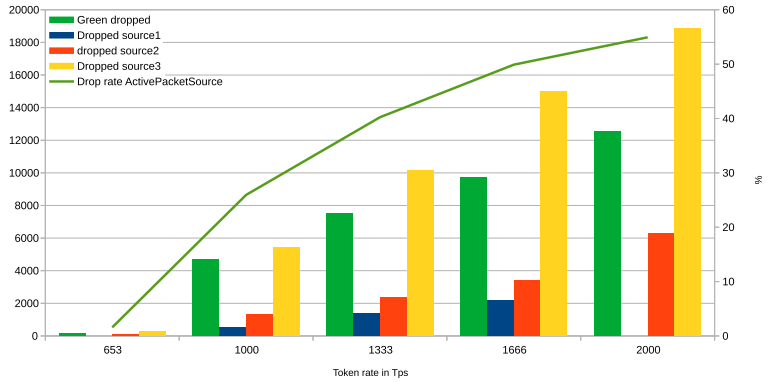

Figure 7.10: The number of drops for every source in the network for different token rates and a fixed bucket size of 666 tokens. Simulation was run with a ActivePacketSource at source1 and source3.

When now replacing the ActivePS of source3 with the ActiveETS the throughput changes as well as the the drop rate. This can also be seen in Figures 7.7 - 7.9 There the curves for the ActiveETS shows that the ActiveETS leads to a lower throughput for a low number of tokens per second. Especially for the first token rate of $653Tps$ the throughput drops from $9.622Mbps$ to $9Mbps$. This can be explained by comparing the throughput over time between a simulation with the AcitvePS and the ActiveETS. Figures 7.11 and 7.12 are comparing the throughput over time between a simulation with two ActivePSs and a simulation with one ActivePS and one ActiveBCS. The bucket had a size of 333 tokens and the token rate was at $653Tps$. Also the cumulative sent data for source1, source2 and source3 is plotted. In figure 7.11 the throughput rises in the beginning to its maximum value of $9,622Mbps$ and all three sources are sending with a constant rate. In figure 7.12 the throughput drops several times when the number of packets which are produced by the ActiveBCS of *source3* and thus the send rate of *source3* drops. Because the token-bucket shaper was configured with a low

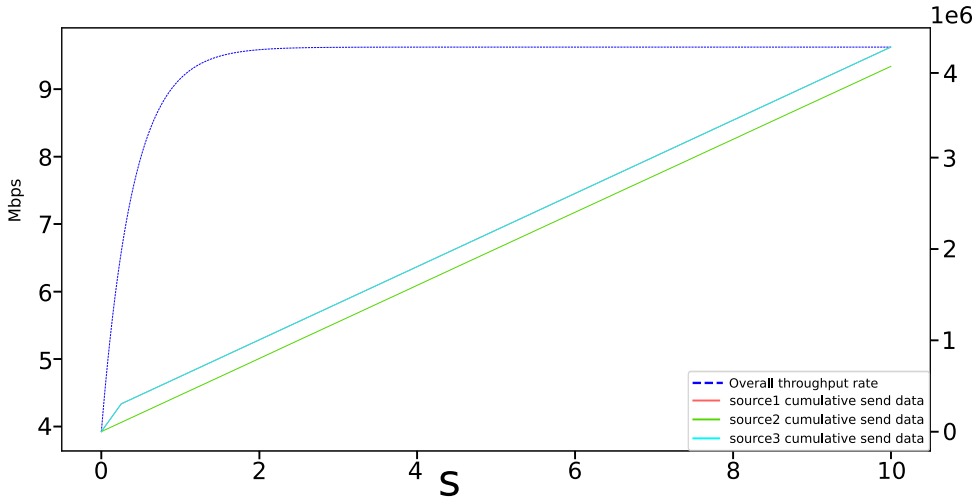

Figure 7.11: Throughput over time for simulation with a token size of 333 tokens a token rate of 653 tokens per second and the *TokenPacketSource*.

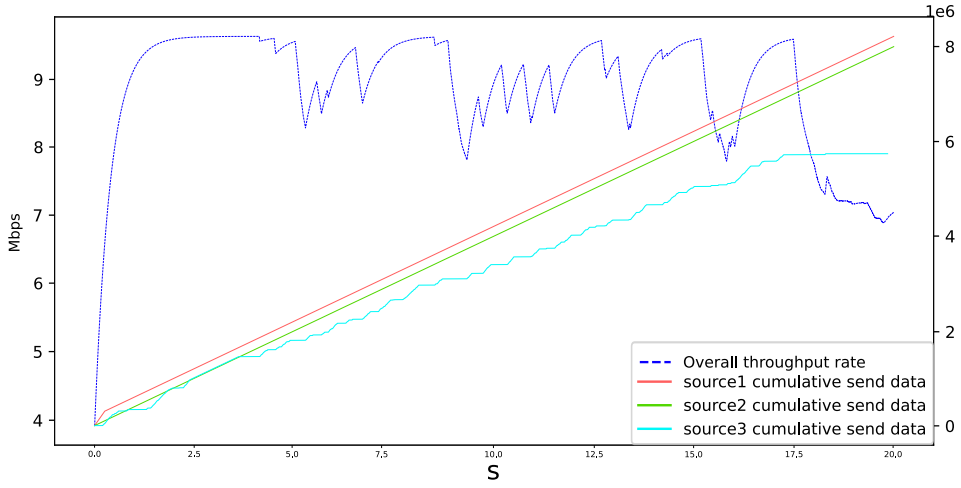

Figure 7.12: Throughput over time for simulation with a token size of 333 tokens a token rate of 653 tokens per second and the *ActiveBinomialCauchySource*.

token rate, source1 is not able to compensate these drops with a higher send rate and thus the overall throughput gets lower in these moments. Drops like these occur in the most of the simulation runs which has the result, that the mean throughput of all simulations is lower than in the simulations with the ActivePS. The same effect happens when replacing the ActiveETS with the ActiveBBCS.

When now increasing the token rate of the token-bucket shapers, source1 is able to compensate the drops and the send rate of source3 and hence the throughput rises. This is why the throughput is at $9.622 Mbps$ again for a token rate of $1333 Tps$ for both the ActiveETS and the ActiveBCS. Figure 7.13 shows the throughput together with the cumulative send data over time for one simulation. In this simulation the ActiveBCS has been used. The token-bucket size was set to 333 tokens at each

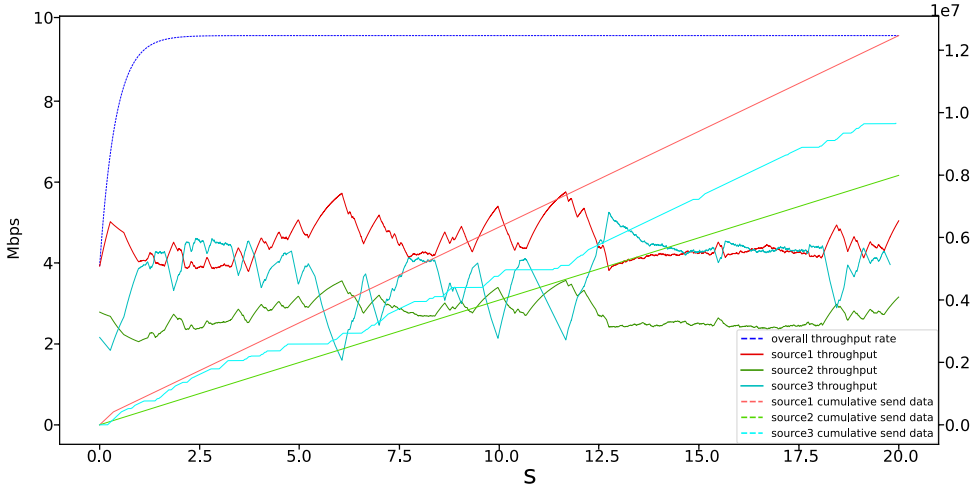

Figure 7.13: Throughput over time for simulation with a token size of 333 tokens a token rate of 1000 tokens per second and the *ActiveBinomialCauchySource*.

host and the token rate was set to $1000Tps$. In addition to the overall throughput in this visualization also the throughput of each stream at the sink is plotted. Whenever source3 stops to send data and its line with the cumulative sent data pauses rising, also the throughput of source3 starts to fall. This effect can be seen e.g. at $\approx 4.8s$ where source3 stops sending data for $\approx 1s$ and the throughput of source3 starts to fall after a certain delay. This delay is in this plot because at the moment where source3 stops sending there are still packets in the queue of the switch which are forwarded to the sink. After the packets in the queue are sent the throughput starts to fall at the sink. When now observing the overall throughput at the sink, when the throughput of source3 starts to fall, it still keeps it rate at $9.621Mbps$. This is because in the moment where the throughput of source3 falls, the throughput of source1 and source2 increases and thus compensates the falling throughput of source3.

Thus an increased token rate and also a increased bucket size leads to a higher throughput in the network, if the streams of the network are not sending all of the time.

The same effects can be seen when also replacing the ActivePS of source1 with an AcitveETS or an ActiveBBC. This can be seen in Figures 7.14 - 7.16. There again the throughput compared with the drop rate for different token rates and bucket sizes are shown. Compared to the previous simulations with an ActivePS the throughput is again lower and never reaches its maximum of $9.622Mbps$. This is because now there is no stream in the network which is always able to compensate a dropping throughput of another stream, like the ActivePS did in the simulations before.

In brief, based on the source characteristic the throughput in a network gets lower and a bigger token-bucket shaper helps increase the throughput and thus also the utilization in the network. What also happens if the token-bucket shapers in a network get bigger, is that the amount of dropped packets increases as well as the number of green packets. These green packets should have the

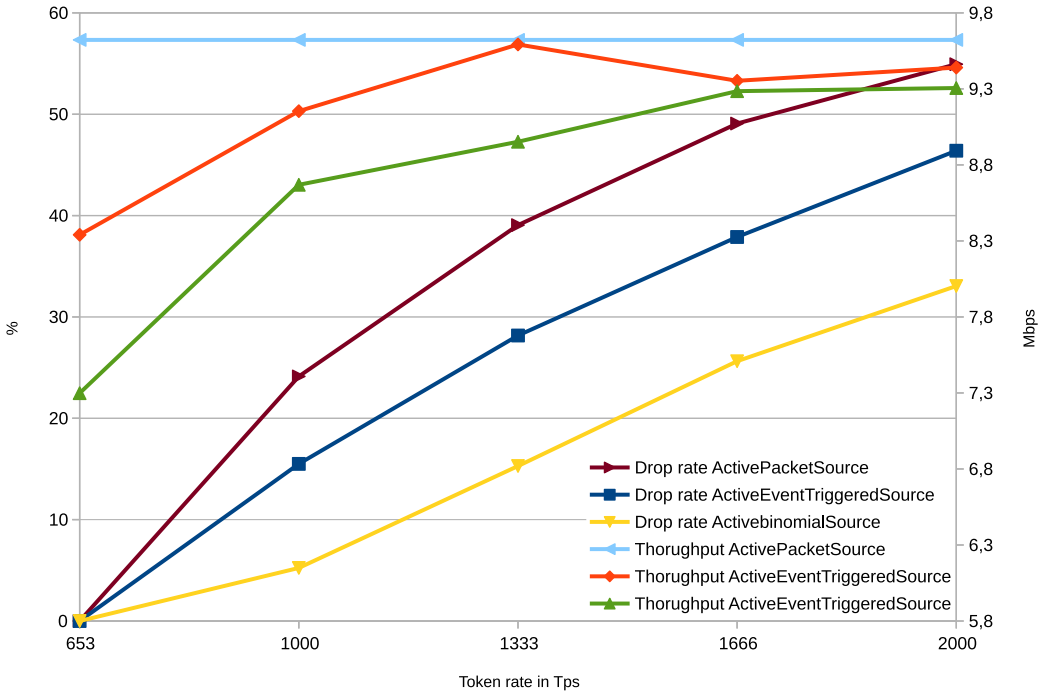

Figure 7.14: The drop rate and the throughput of the simulations with different bucket sizes plotted for different token rates and different PacketSources on source1 and source3. The bucket size is = 333 tokens

guarantee to not be dropped to fulfill the guaranteed bandwidth for every stream in the network. Thus with an increased token-bucket configuration and without using the color shaper it is not possible to fulfill the requirement of definition 4.1.3 anymore.

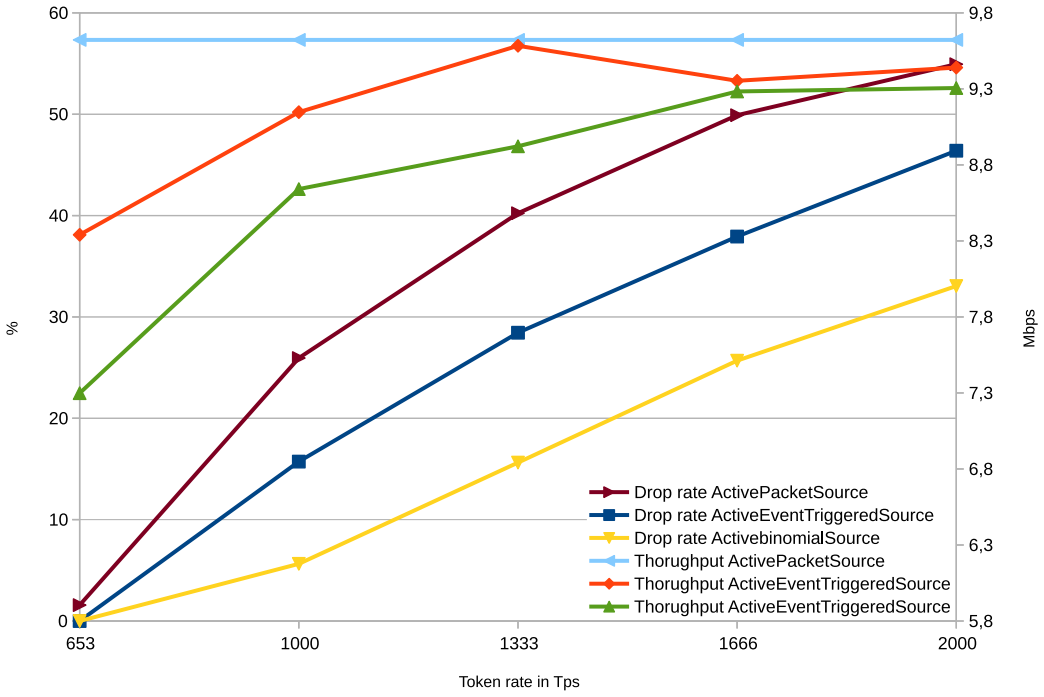

Figure 7.15: The drop rate and the throughput of the simulations with different bucket sizes plotted for different token rates and different PacketSources on source1 and source3. The bucket size is = 666 tokens

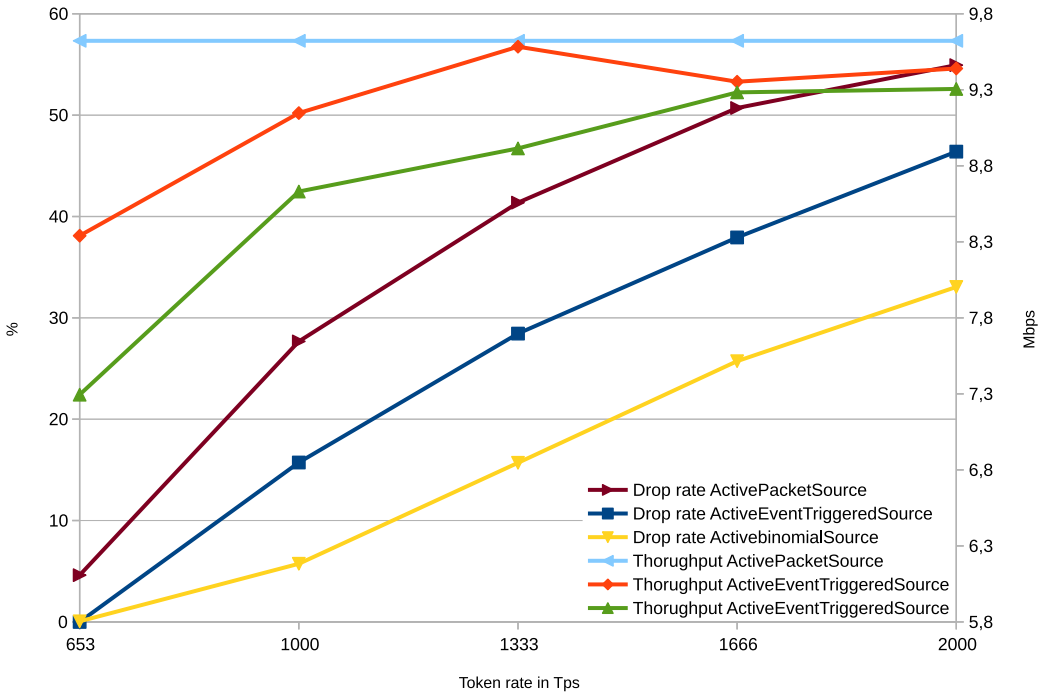

Figure 7.16: The drop rate and the throughput of the simulations with different bucket sizes plotted for different token rates and different PacketSources on source1 and source3. The bucket size is = 1000 tokens

Parameter	Values					
Packet source	ActivePS	ActivePoissonSource		ActiveETS	ActiveBCS	
Cbs	$613B$	$2000B$	$4000B$	$20000B$	$40000B$	$100000B$
Threshold	$1000 - 2 * \frac{cbs}{613B} + 1$			$1000 - \frac{cbs}{613B} + 1$		

Table 7.2: Alternating configuration parameters cbs simulations of the one hop topology.

Evaluate the Committed Burst Size

Before analyzing the effect of using a color shaper in the one-hop scenario now, some simulations with a alternating committed burst size(cbs) were done. These had to be done to find the right cbs for all simulations with the color shaping. To analyze the influence of an increasing *cbs*, a configuration with two sources sending at a higher rate than their guaranteed bandwidth ($Mbps$) and one source sending at its guaranteed bandwidth($Mbps$), was used. A good cbs then is achieved if the source, which in average sends with its guaranteed bandwidth, gets zero or only a small amount of drops. Also the different source characteristics were tested to check if there is a difference between those sources when using a different *cbs*. To do so, the simulations were ran with either the ActivePS, the ActivePoissonSource, the ActiveETS or the ActiveBCS at all sources in the network. As a *cbs* one packet and 0.5%, 1%, 5%, 10% and 25% of the guaranteed bandwidth were used. The different parameters with its values can be seen in Tabular 7.2.

To proof 6.1.1 also the threshold of the switch was alternated between the maximum value before lemma 6.1.1 was violated and one that was higher than lemma 6.1.1 allows. Also the effect of corollary 6.1.2 was checked with this maxed out switch threshold.

The results of tests with a threshold that did not fulfill lemma 6.1.1 can be seen in Tabular 7.3. The tabular shows the amount of green dropped packets during for the different cbs values and packet sources. What can be seen is that especially during the simulations with an ActiveETS and an ActiveBCS there are many green packets dropped. All these green packets were sent with the guaranteed bandwidth of their streams and should not have been dropped to fulfill definition 4.1.3.

Cbs	$613B$	$2000B$	$4000B$	$20000B$	$40000B$	$100000B$
Green packet drops ActivePS	5173	172	0	0	0	0
Green packet drops ActivePoissonSource	711.9	0.5	0	0	0	0
Green packet drops ActiveETS	277.8	38	27.9	32.4	58.9	145.2
Green packet drops ActiveBCS	76.4	12.8	11.2	7.8	14.8	28

Table 7.3: Amount of green dropped packets for different packet sources during the cbs simulations with a to small queue threshold.

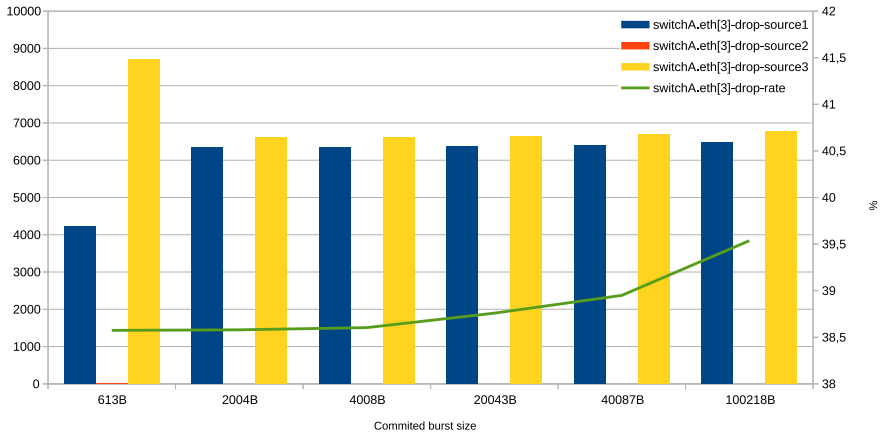

Figure 7.17: Overall drop rate and packet drops of source1 - source3 for the cbs simulations with an ActivePS.

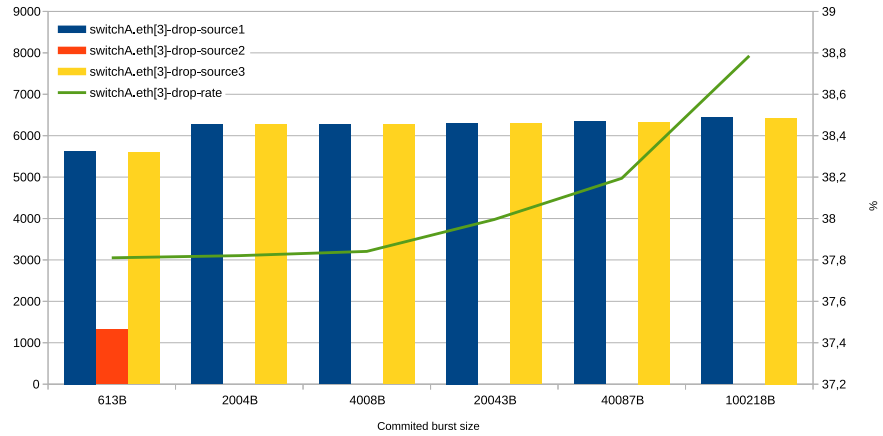

Figure 7.18: Overall drop rate and packet drops of source1 - source3 for the cbs simulations with an ActivePoissonSource.

In Figure 7.17 the results of the simulations with an ActivePS and the maximum possible threshold that fulfills lemma 6.1.1 are plotted. The Figure shows the overall drop rate as well as the number of drops for every source in the network for different cbs-values. What shows off in the figure is that a source which always sends with a very static rate like the ActivePS has no benefit with a higher cbs. More over the effect of corollary 6.1.2 can be observed since the drop rate rises with a higher cbs, because there is space in the egress queue reserved which is not used by source1 and source3. This reserved space gets bigger with a higher cbs and hence the threshold at which red packets are dropped gets lower. This results in a higher drop rate.

Figures 7.18 - 7.20 show the results for the simulations with the ActivePoissonSource, the ActiveETS and the ActiveBCS and the maximum possible threshold that fulfills lemma 6.1.1. Here again the effect of lemma 6.1.2 can be observed by looking at the rising drop rate for a higher cbs. When analyzing the simulations with the ActiveETS and the ActiveBCS the drop rate is lower, because in these simulations sources1 and sources3 are sending in bursts. This has the effect that more of

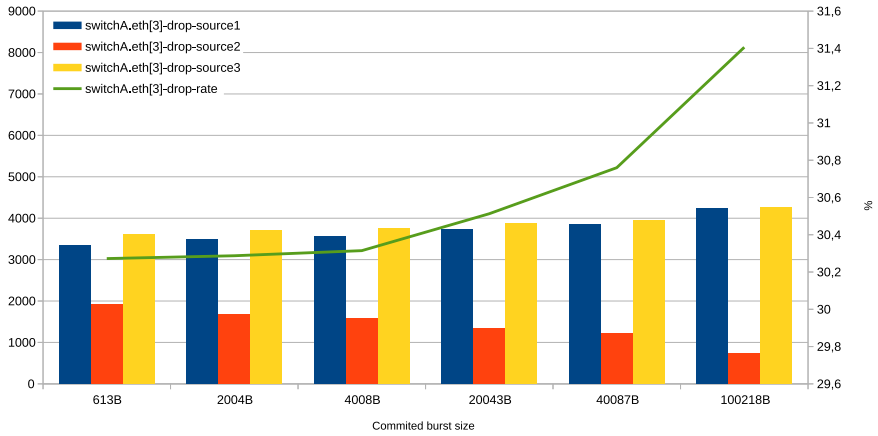

Figure 7.19: Overall drop rate and packet drops of source1 - source3 for the cbs simulations with an ActiveETS.

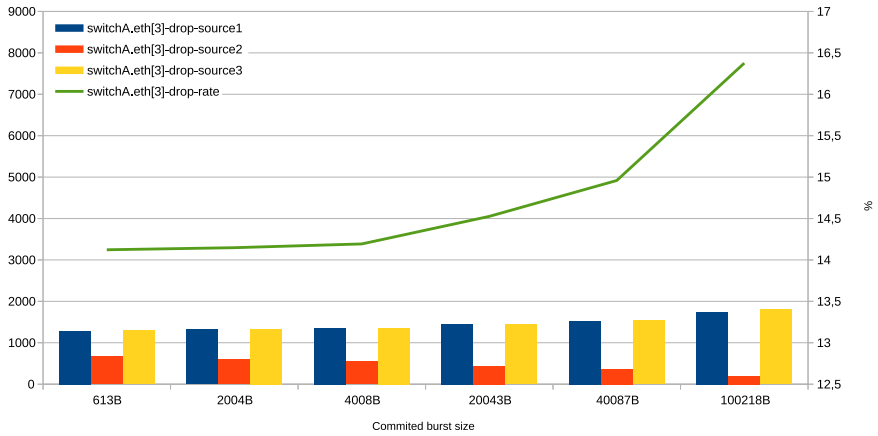

Figure 7.20: Overall drop rate and packet drops of source1 - source3 for the cbs simulations with an ActiveBCS.

their packets get marked green with the tokens of the *cbs* of the marker, since between two bursts the marker is able to refill its bucket again. With this higher amount of green packets the reserved space in the egress queue is more frequently used and hence less queue space is wasted.

When analyzing the effect of a bigger *cbs* on source2 it first shows off, that for an ActiveETS or an ActiveBCS even at a *cbs* of 25% of the guaranteed bandwidth packets are dropped. To guarantee no drops for these sources a *cbs* of 100% of the guaranteed bandwidth would have to be set, which would be not sufficient for the new approach. What also can be observed, is that the amount of dropped packets from source2 is going down with an almost linear factor for a rising *cbs*. On the other hand, the overall drop rate starts to rise for a *cbs* of 1% of the guaranteed bandwidth with all of the used packet sources. Furthermore Figure 7.18 shows that the ActivePoissonSource only needs a *cbs* of 0.5% of the guaranteed bandwidth, such that packets are not marked red anymore.

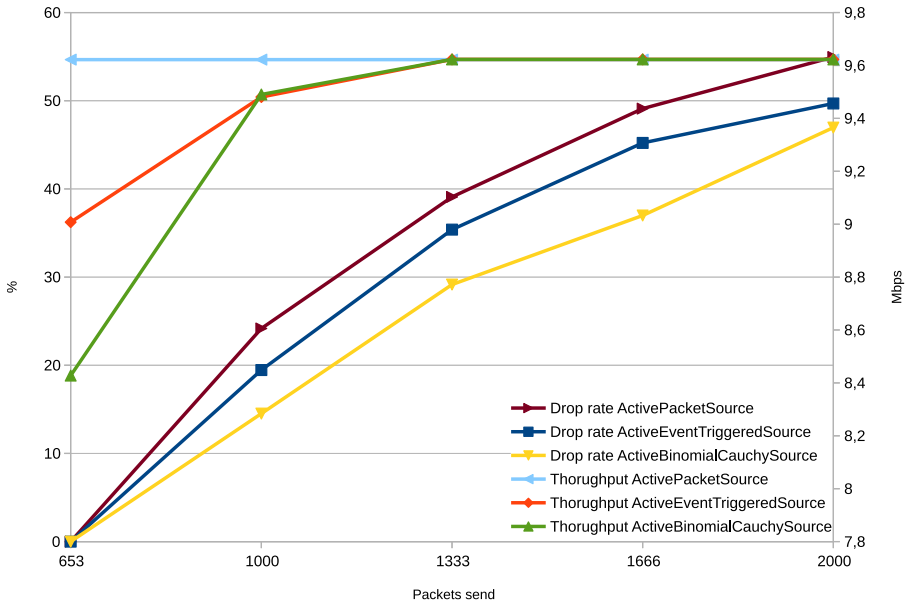

Figure 7.21: The drop rate and the throughput of the simulations plotted for different token rates and different PacketSources for source3. The bucket size was set to 333 tokens and color shaping was used in these simulations

Because of this a cbs of 1% will be used in all further simulations of this evaluation. It is also recommended to use this cbs if there is no requirement that a certain burst of an application has to be marked green.

Apply Colour Shaping

After estimating a suitable commited burst size for the token-bucket meter in the ingress ports of the switch this section analyzes the effect of using the color shaper in the switch. To do so, the previous configurations of section 7.3.1 were used and the color shaper in the egress ports of *switchA* was activated. Especially all possible combinations of Tabular 7.1 were simulated again.

Figures 7.21 - 7.23 again show how the overall drop rate and throughput changes for different token rates and bucket sizes for the combination of an ActivePS on source1 and different packet sources on source3. When comparing these figures with Figures 7.7 - 7.9 it can be observed that when using the color shaper an effect on the overall throughput at the sink or the drop rate in the network is not apparent. The overall drop rate only increased by 0.03% which comes from the unused queue space that has to be reserved for green packets. The same effect holds for the simulations where either the ActiveETS or the ActiveBCS were used on both source1 and source3.

A huge effect of the color shaper can be seen in Figure 7.24. Compared to the simulations without a color shaper with the same traffic there are no green packets dropped anymore. Also when comparing Figure 7.21 with Figure 7.24 it shows, that source1 is not able to oust the traffic of source2 and source3 anymore. There are no packets dropped from source2 and the amount of dropped packets

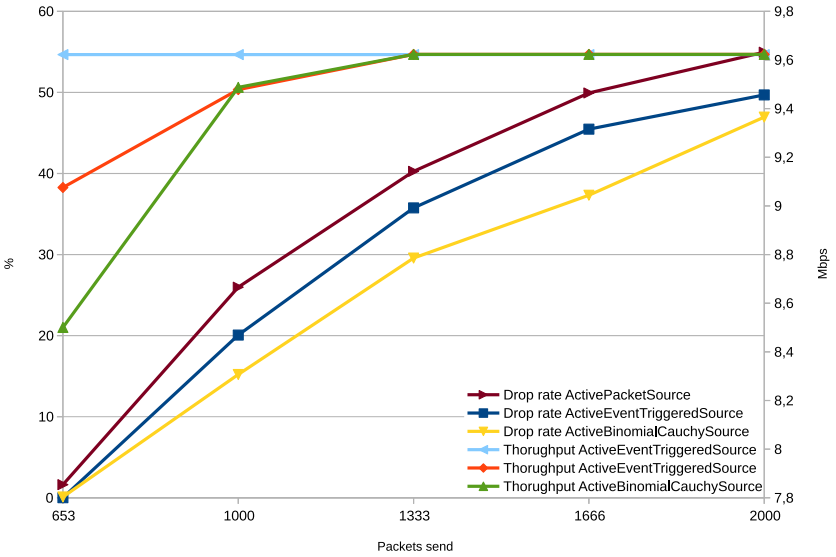

Figure 7.22: The drop rate and the throughput of the simulations plotted for different token rates and different PacketSources for source3. The bucket size was set to 666 tokens and color shaping was used in these simulations

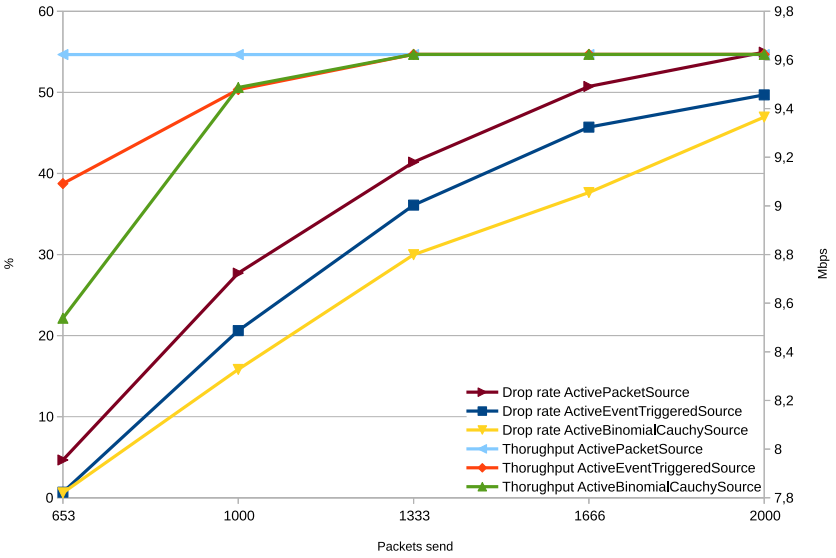

Figure 7.23: The drop rate and the throughput of the simulations plotted for different token rates and different PacketSources for source3. The bucket size was set to 1000 tokens and color shaping was used in these simulations

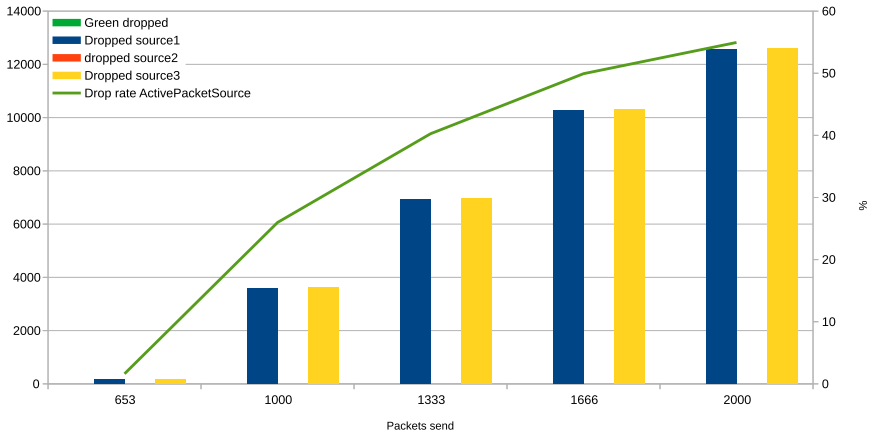

Figure 7.24: The number of drops for every source in the network for different token rates and a fixed bucket size of 666 tokens. Simulation was run with a ActivePacketSource at source1 and source3 and color shaping was used.

of source1 and source3 now is almost equal. This means that in this network scenario the color shaper not only guarantees a certain bandwidth for every stream, it also provides a more balanced and thus also fairer drop behavior.

The guarantee of bandwidth can also be seen when looking at the sources throughput in Figure 7.25. There the same simulation as in Figure 7.13 is shown but this time with the color shaper activated. In the first simulation the throughput of source2 stays most of the time under its guaranteed bandwidth of $3.2Mbps$ and thus also packets of source2 are dropped. In Figure 7.25 where the color shaper was used the bandwidth of source2 is most of the time oscillating around its guaranteed bandwidth of $3.2Mbps$. Because of that, packets of source2 are not dropped and also source1 is not able to oust the traffic of sourc2 anymore.

In brief, when using the color shaper for a one hop scenario like this, a bandwidth for every stream can be guaranteed while almost not increasing the overall drop rate in the network. The color shaper also leads to a fairer drop behavior between the different streams in the network.

Comparing the Simulation with NC

After running the simulations for the one-hop scenario, in this section the results of the simulation are compared with the worst-case estimations of Section 6.2. Figure 7.26 and 7.27 show a plot where curves with the worst-case estimations of NC are compared with the results of a simulation.

Both the simulation and the NC estimations were done for a token rate of $1000Tps$ and a bucket size of 333 tokens. In the simulation source1 and source3 used the ActivePS to produce a maximum load for the token-bucket shaper. Source2 used the ActivePoissonSource to produce packets with an average of $653Pps$.

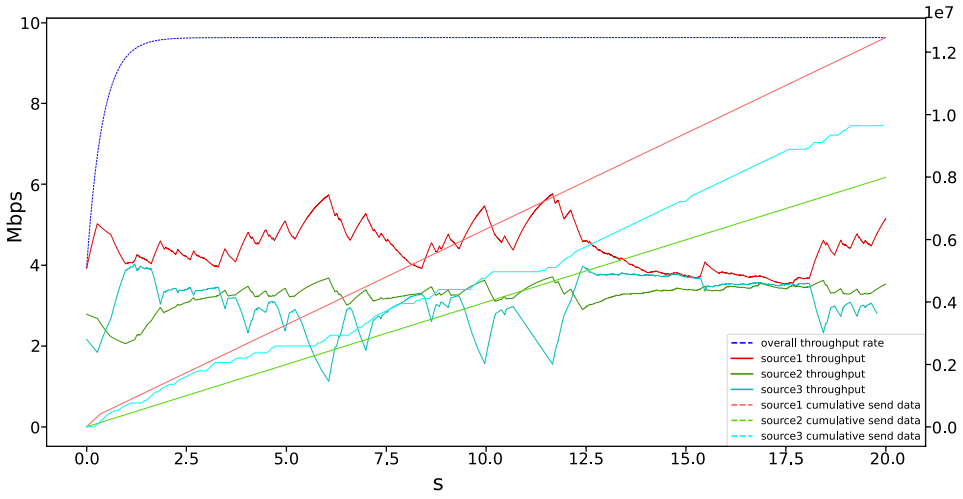

Figure 7.25: Throughput over time for simulation with a token size of 333 tokens a token rate of 1000 tokens per second and the *ActiveBinomialCauchySource*. Color shaping was used in this simulation.

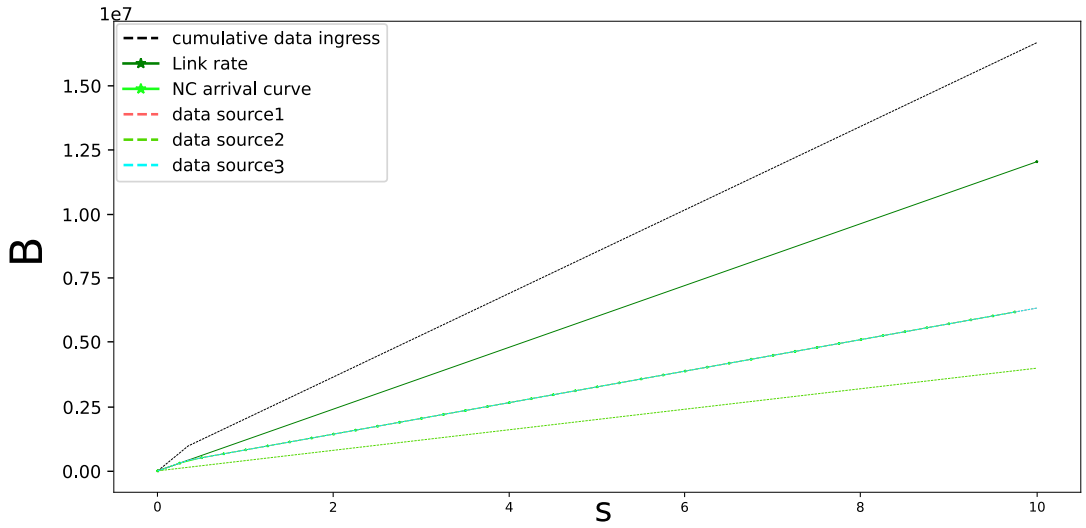

Figure 7.26: Comparison of a NC worst-case estimation vs the results of a simulation with two ActivePSs and one ActivePoissonSource at the ingress of a switch. The token rate is at $1000 Tps$ and the bucket size is 333 tokens.

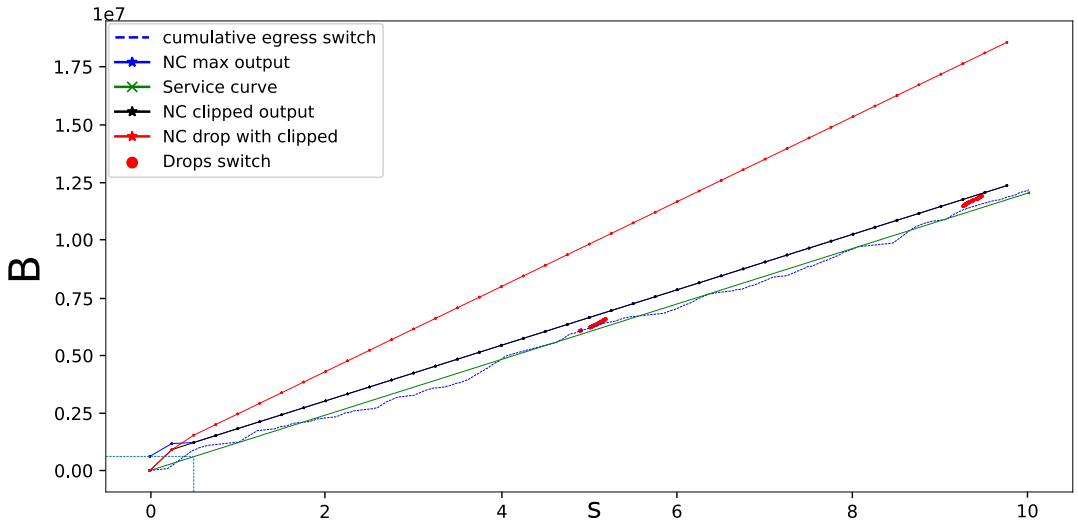

Figure 7.27: Comparison of a NC worst-case estimation vs the results of a simulation with two ActivePSs and one ActivePoissonSource at the egress of a switch. The token rate is at $1000Tps$ and the bucket size is 333 tokens.

In Figure 7.26 we can see that the arrival curve for the NC estimation is the same as the input of source1 and source3. This was expected since source1 and source3 sent their data with the maximum rate which is allowed by the token-bucket shaper. Source2 is under the maximum input of the arrival curve since the rate at which the ActivePoissonSource produces packets is lower than the token rate the token-bucket shaper of source2. It can also be observed, that the cumulative amount of data which comes to the ingress of the switch is noticeably over the link rate of the egress of the switch.

In Figure 7.27 the nc estimations of the egress as well as the real recorded values of the simulation are visualized. What can be observed in the figure is that the cumulative flow going through the egress in the simulation starts increasing lower than the NC estimation. But at $\approx 0.5s$ the curves meets the NC clipped curve and also the NC output curve and rises with the same slope. This means that the NC worst case estimation set an feasible upper bound for the scenario. The curves are not the same from second zero because source2 does not send with the rate which was estimated from the NC arrival curve. This is why the simulation takes longer to fill the queue of the switch and why the curves of the simulation and the NC estimation meet at $\approx 0.5s$. What we also can see is that the amount of dropped packets are lower in the simulation than estimated from NC. This is also because source2 does not send with its maximum rate and hence no packets of source2 are dropped.

The worst-case estimation of the maximum delay in the NC estimation was the maximum horizontal distance between the output curve and the service curve. This is reached at $y = 613000$ where the distance is $0.51s$. The worst case delay of the simulation was $0.506s$, which means that there is only a difference of $0.51s - 0.506s = 0.004s$ between the simulation result and the worst-case estimation in this simulation. Therefore the NC worst-case estimation was very accurate in the delay and also

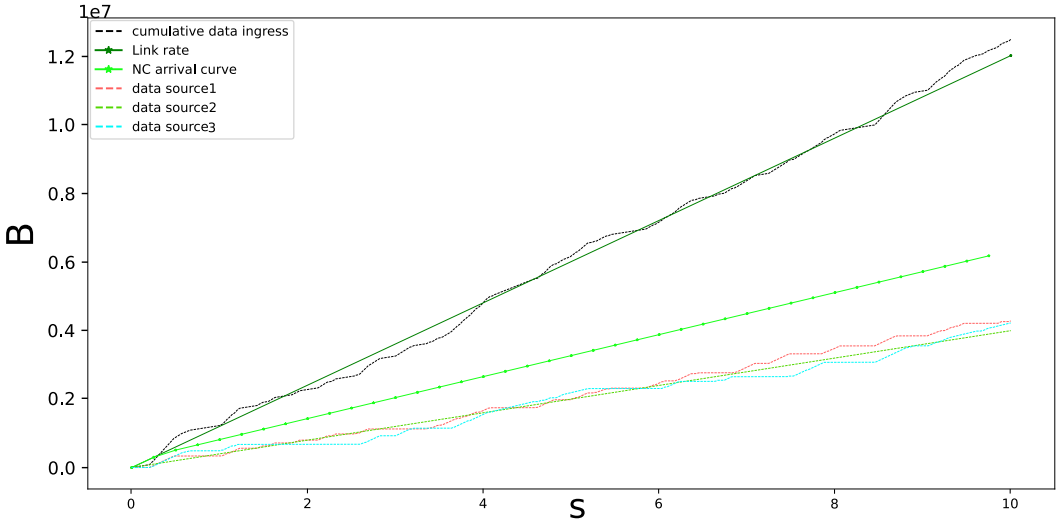

Figure 7.28: Comparison of a NC worst-case estimation vs the results of a simulation with two ActiveBCSs and one ActivePoissonSource at the ingress of a switch. The token rate is at $1000Tps$ and the bucket size is 333 tokens.

provided good upper bounds for the output of the switch. The only estimation which was to high in this simulation were the amount of dropped packets where the NC estimation was higher with a factor of 33%.

However the simulation showed in Figure 7.26 and 7.27 were made with two ActivePS sending with a maximum rate. This means that the simulation was near to a worst case scenario and did not show the average case which is expected for the new approach of this thesis. For this reason Figure 7.28 and 7.29 show a simulation where the ActiveBCS was used instead of the ActivePS. So we have the same configuration but with other packet sources in source1 and source3.

When now comparing the ingress at the switch with the arrival curve in Figure 7.28 it can be observed that all three sources are clearly under the arrival curve. This leads to a huge difference between the NC estimation and the recorded values in the simulation which can be seen in Figure 7.29. First we can notice that the amount of dropped packets is much lower than the NC estimation. The NC estimation expects that $638133B$ are dropped each second which would be $6381330B$ over the whole simulation. But in this simulation in total only 551 packets were dropped which are $551 * 613B = 337763B$ and is only $\frac{337763}{6381330} * 100 = 5.29\%$ of the NC estimation. Also the NC estimated output of the switch is much higher since the cumulative egress of the simulation never reaches the NC maximum output curve. Only the maximum delay of the NC estimation with $0.51s$ was near at the measured maximum delay in the simulation which was again $0.506s$.

In conclusion, the NC worst-case estimation can be used to determine upper bounds on the amount of dropped data and the output of a switch for networks which are using the new approach of this thesis. But these upper bounds are not feasible to do an average estimation for event-triggered

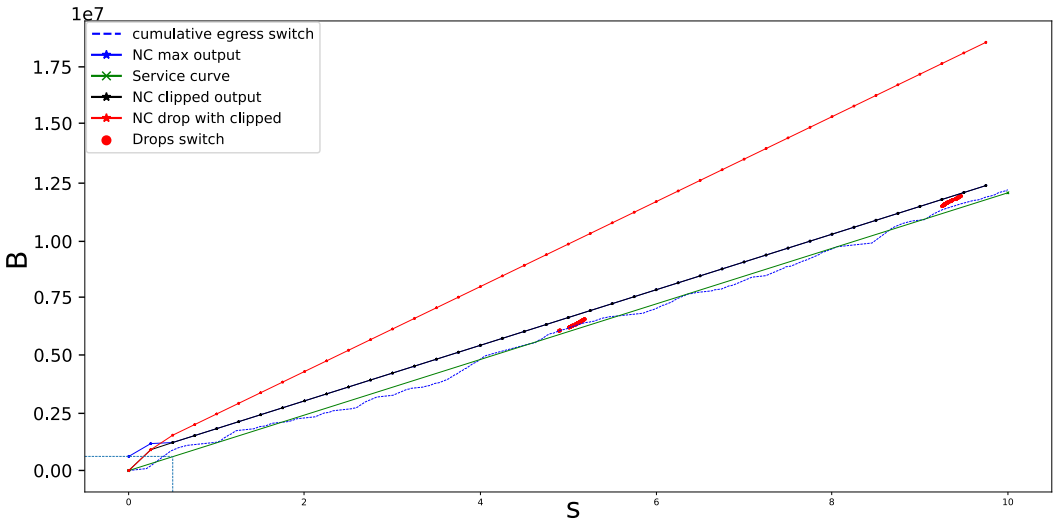

Figure 7.29: Comparison of a NC worst-case estimation vs the results of a simulation with two ActiveBCSs and one ActivePoissonSource at the egress of a switch. The token rate is at $1000 Tps$ and the bucket size is 333 tokens.

traffic where it is expected that worst-case situations are rare. On the other hand NC can be used to get worst case estimation for delay bounds which even for event-triggered traffic, at least in our simulations, are very accurate.

7.3.2 Device to Controller Scenarios

Section 7.3.1 showed that with color shaping it is possible to guarantee a certain bandwidth to every stream in a network and the throughput in can be increased with an increased token-bucket shaper configuration. This section evaluates theorem 6.1.2 by increasing the number of hops in the network and comparing color marking on a per-hop basis and edge basis. To do so, a typical device to controller scenario was chosen. Figure 7.30 shows the network with its topology and measuring points. The network consists of three sources which forward their traffic to one sink over two hops. To measure the overall drops in the network drops have to be recorded and analyzed on every switch in the network. This is why at switchA the two points D_1 and D_2 are and on switchB there are measuring points for all sources in the network $D_1 - D_3$. The overall drops in the network then were calculated by summing up the values of all these measuring points.

The fist simulations were done like in Section 7.3.1, where first no color shaping was used. Hence Tabular 7.1 again was used for the variable parameters. For source2 again a ActivePoissonSource was used and the token-bucket shaper of source2 had a token rate of $653 Tps$ and a bucket size of 333 tokens. Then simulations with the same configurations like in Section 7.3.1 were run.

Figures 7.31 - 7.33 show the drop rate and the throughput for different token rates and token bucket sizes of the simulations without color shaping. This time only the results of the simulations where an ActivePS were used are shown.

Compared to the simulation of the previous section where only one hop was between the sources and the sink, in this simulations the drop rate is in average ≈ 5% lower. This can be explained by the additional hop which means, that there is one more switch in the system which can buffer data in its egress queue. With more buffer the bursts of the stream can better be handled by a system, before it has to drop packets. This is also why the simulations where two ActivePacketSources were used, and hence no bursts are produced, the drop rate does go down but not as much as at the other simulations.

When comparing the number of green dropped packets and the fairness between the sources there is no difference between the two simulations as we can see in Figure 7.35

After that the simulations were ran with color marking on a per-hop basis. This color marking was used on every ingress port of the switches and the threshold of the color shapers were chosen such that lemma 6.1.1 was fulfilled. When now comparing the results of these simulations with the results of Section 7.3.1 then the behavior of dropped packets is different. While in Figure 7.24 the fairness between source1 and source3 is given and no green packets are dropped, the results of the device to controller scenario in Figure 7.35 are different.

When using more than one hop between a source and a sink the combination of color shaping and color marking on a per-hop basis increases the fairness between different sources, but green packets are still dropped. This can be seen in Figure 7.35 where the amount of green dropped packets did go down compared to the simulations without color shaping, but there are still green packets dropped.

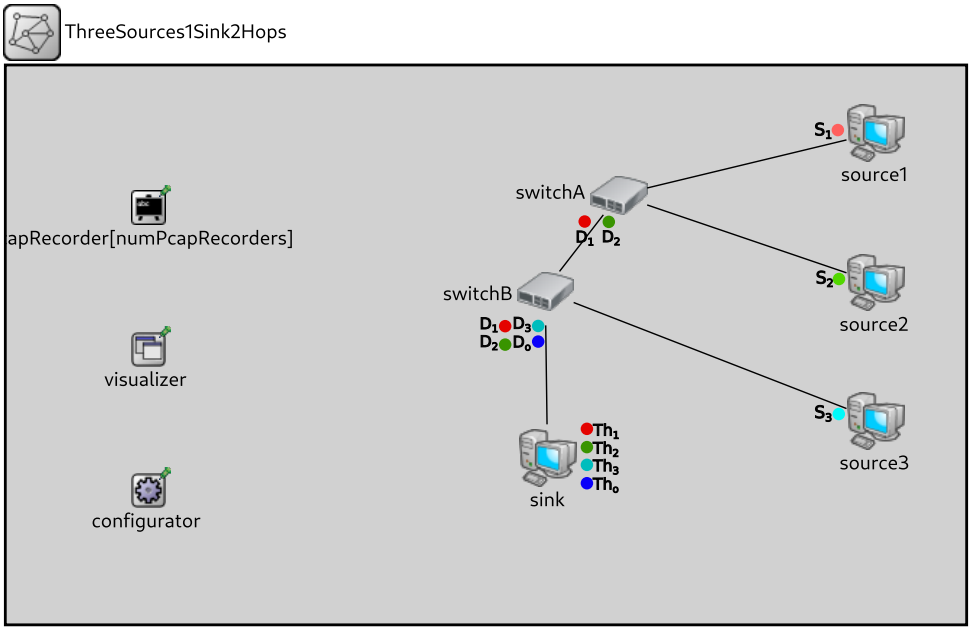

Figure 7.30: Topology of the simulations with a typical device to controller scenario. Two sources are connected to one switch which together with another source is connected to one sink through another switch.

Especially there are still packets dropped from source2 which did never violated its guaranteed bandwidth of $3.2Mbps$. The number of drops even increased for the simulations with a token rate smaller than $2000Pps$. A similar effect can also be seen when comparing the results with and without color shaping where not only ActivePSs were used. In all simulations where only one ActivePS was used, the color shaper on per-hop basis helped to prevent source3 from being ousted by the data of source1. In the simulations where source1 and source3 had either the ActiveETS or the ActiveBCS the color shaper on per-hop basis also helped to get a fairer drop behavior between source1 and source3. But in all simulations packets of source2 were dropped and therefore also green packets were dropped and the guaranteed bandwidth was violated.

This is why the color shaping with color marking at the edges has to be used. After deactivating the color marking on ingress ports which are not at the edge of a network the guaranteed bandwidth was fulfilled again which is shown in Figure 7.36. With color marking only at the edges of a network no more green packets are dropped. Also the fairness between source1 and source3 was increased since the difference in packet drops between them is only at a maximum of 384.6 packets which was 2674.7 packets before with the color marking on every hop. What we also can see is that the drop rate improved by almost 2% for the token rate of $2000Tps$. This is because of the color marking on edge basis the threshold in switchB had to be reduced to 493 packets to fulfill lemma 6.1.4. This behavior can also be observed in the simulations where either the token bucket size was increased or the other sources were used.

Increase the Number of Hops

The same effects can be observed when increasing the number of hops by one. The resulting network is shown in Figure 7.37. Because the number of sources increased by one the *cir* was reduced

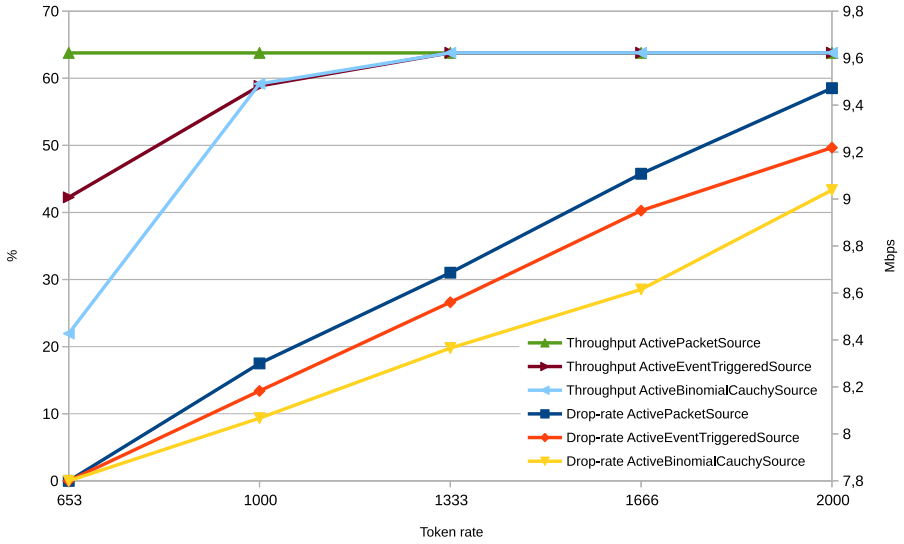

Figure 7.31: The drop rate and the throughput of the simulations with different bucket sizes plotted for different token rates and different PacketSources for source3. The bucket size is = 1000 tokens

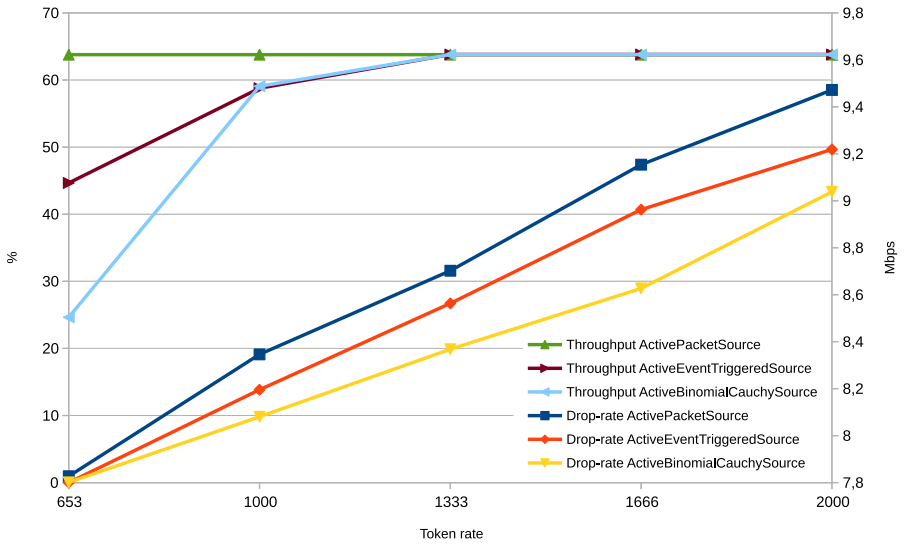

Figure 7.32: The drop rate and the throughput of the simulations with different bucket sizes plotted for different token rates and different PacketSources for source3. The bucket size is = 1000 tokens

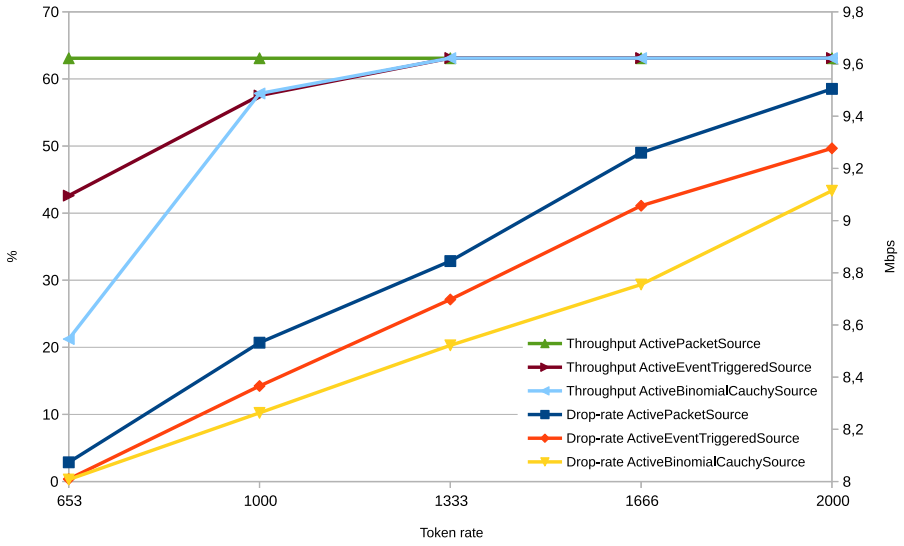

Figure 7.33: The drop rate and the throughput of the simulations with different bucket sizes plotted for different token rates and different PacketSources for source3. The bucket size is = 1000 tokens

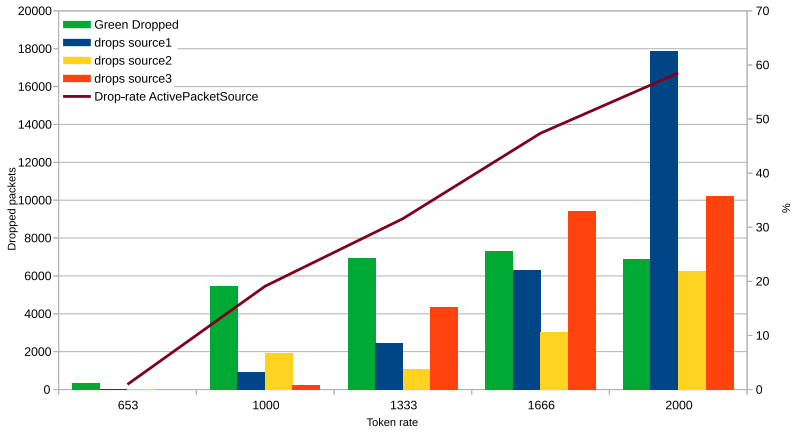

Figure 7.34: The number of drops for every source in the network for different token rates and a fixed bucket size of 666 tokens. Simulation was run with a ActivePacketSource at source1 and source3.

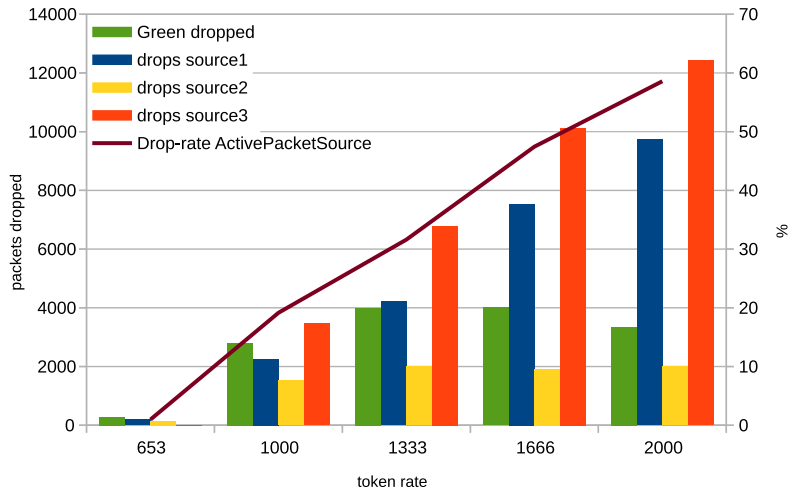

Figure 7.35: The number of drops for every source in the network for different token rates and a fixed bucket size of 666 tokens. Simulation was run with a ActivePacketSource at source1 and source2. Color shaping is used in this simulations.

for this simulations to 2.4$Mbps$ per source. Also the cbs for the tests without color shaping were reduced to 400289B. For the simulations with color shaping on hob-basis the cbs of all edge ports was set to 3000B and the in network ports of switchA to 6000B and switchB to 9000B. Then again simulations were run with the parameters and its values shown in Tabular 7.4.

In these simulations again no green packets were dropped when using color shaping with color marking at the edges of the network. Also the drop rate increased again when using color shaping which can be seen in Figures 7.38 and 7.39. There the drop rate is compared between simulations with and without color shaping for an increasing token rate. The token bucket size for these simulations was at 750 tokens.

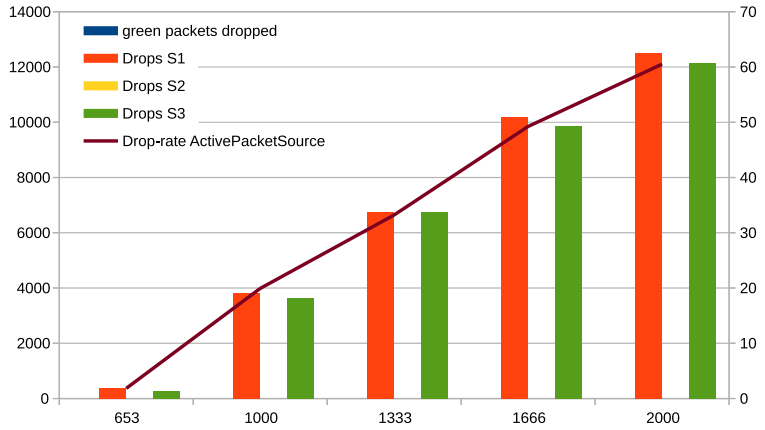

Figure 7.36: The number of drops for every source in the network for different token rates and a fixed bucket size of 666 tokens. Simulation was run with a ActivePacketSource at source1 and source2. Color marking is used at the edges of the network.

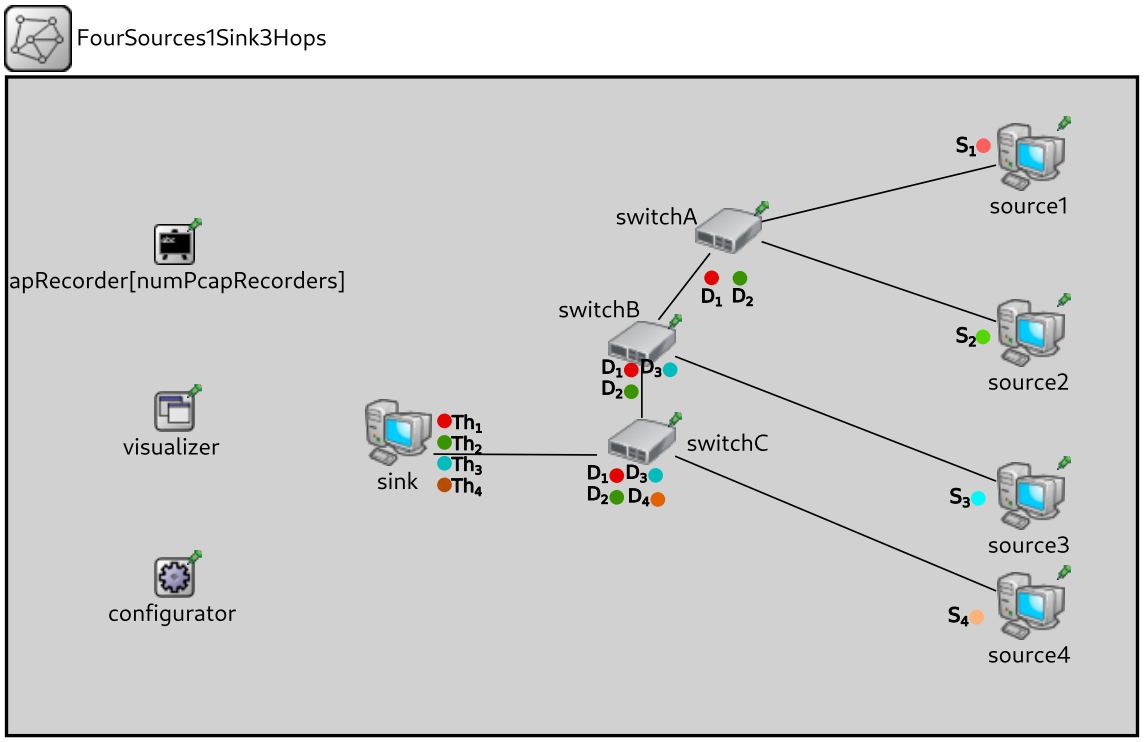

Figure 7.37: Topology of the simulations with a bigger device to controller scenario.

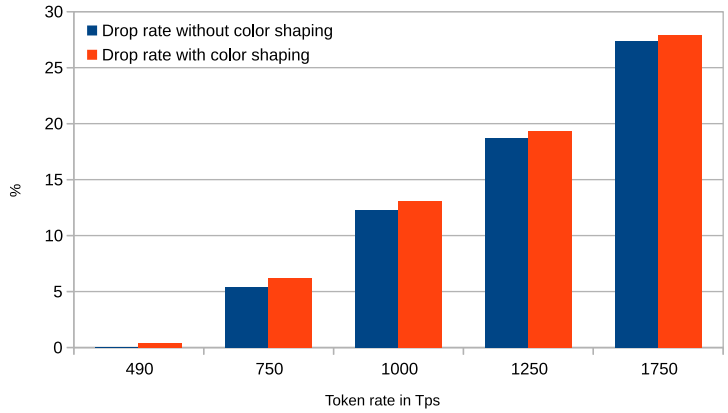

Figure 7.38: The drop rate compared between simulations with and without color shaping for a bucket size of 750 tokens and an increasing token rate. The simulations were ran in the device to controller scenario with three hops. There was an ActivePS on source1 and an ActiveBCS on source3 and source4.

For every token rate it can be observed that the drop rate is $\approx 1-2\%$ higher than without color shaping. This is because to fulfill lemma 6.1.4 the threshold of the color shapers in switchC had to be decreased to 439 packets which leads to a higher drop rate for red marked packets. This is also why the drop rate in Figure 7.38 for the color shaping is higher than in Figure 7.39. In Figure 7.38 source1 used a ActivePS to generate a maximum load for the token-bucket shaper of source1 and source3 and source4 used an ActiveBCS. Thus source1 sends many packets which are marked red by its token-bucket meter and could get dropped by the color shapers. In Figure 7.39 source1, source3 and source4 used an ActiveETS and thus are not producing a maximum load for their token-bucket shapers. This leads to less red packets which have to be forwarded to the sink. Because of the less red packets also less packets can be dropped by the color shapers in the switches which leads to a lower drop rate.

As a result we can say that color shaping on edge-basis improves the fairness between multiple sources when used in a bigger network. It also provides a certain bandwidth for every stream in the network by only increasing the drop rate for $\approx 1-2\%$ compared to a scenario where no color shaping is used.

Parameter	Values				
Packet source	ActivePS	ActivePS s1 ActiveETS s3 and s4	ActivePS s1 ActiveBCS s3 and s4	ActiveETS	ActiveBCS
Token rate	$490Tps$	$750Tps$	$1000Tps$	$1250Tps$	$1750Tps$
Bucket size	250 tokens		500 tokens		750 tokens

Table 7.4: Alternating configuration parameters for source1, source3 and source4 in simulations of the device to controller scenario with three hops.

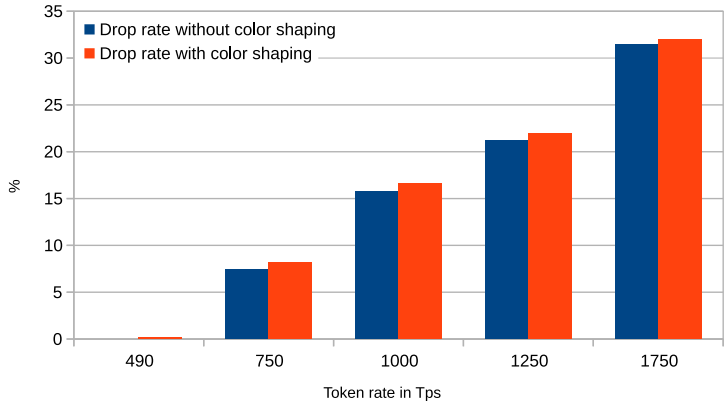

Figure 7.39: The drop rate compared between simulations with and without color shaping for a bucket size of 750 tokens and an increasing token rate. The simulations were ran in the device to controller scenario with three hops. There was an ActiveETS on source1, source3 and source4.

Parameter	Values				
Packet source	ActivePS	ActivePS s1 ActiveETS s3 and s4	ActivePS s1 ActiveBCS s3 and s4	ActiveETS	ActiveBCS
Token rate	$490Tps$	$750Tps$	$1000Tps$	$1250Tps$	$1750Tps$
Bucket size	250 tokens		500 tokens		750 tokens

Table 7.5: Alternating configuration parameters for source1, source3 and source4 in simulations in the network scenario of Figure 7.40.

7.3.3 Bottleneck Scenarios

The previous section showed that the new hybrid approach with a token-bucket shaper and a color shaper in the network with edge based color marking, helps to guarantee a certain bandwidth while allowing a higher utilization in the network. Since the previous network scenarios only showed streams which are going to one sink, in this section a new kind of scenario is evaluated. Figure 7.40 shows the new scenario where two sinks are sharing one hop. This means that the traffic which passes from switchC to switchD will be split at switchD and we have a bottleneck at switchC. In this scenario simulations were run to show again how the utilization can be improved while guaranteeing a certain bandwidth to every stream in the network. To do so, simulations were ran with and without color shaping with the variable parameters shown in Tabular 7.5. The simulations were ran with alternating packets sources, token rates and bucket sizes for source1, source3 and source4. Source2 used a ActivePoissonSource to generate its traffic and it token bucket shaper had a token rate of $490Tps$ and a token bucket size of 250 tokens. Furthermore the cir of every token-bucket meter was set to $2.4Mbps$ and the cbs to $3000B$. For the simulations with color shaping the threshold of switchA and switchB was set to 994 packets and the threshold of switchC and switchD to 736 packets, which fulfills lemma 6.1.4.

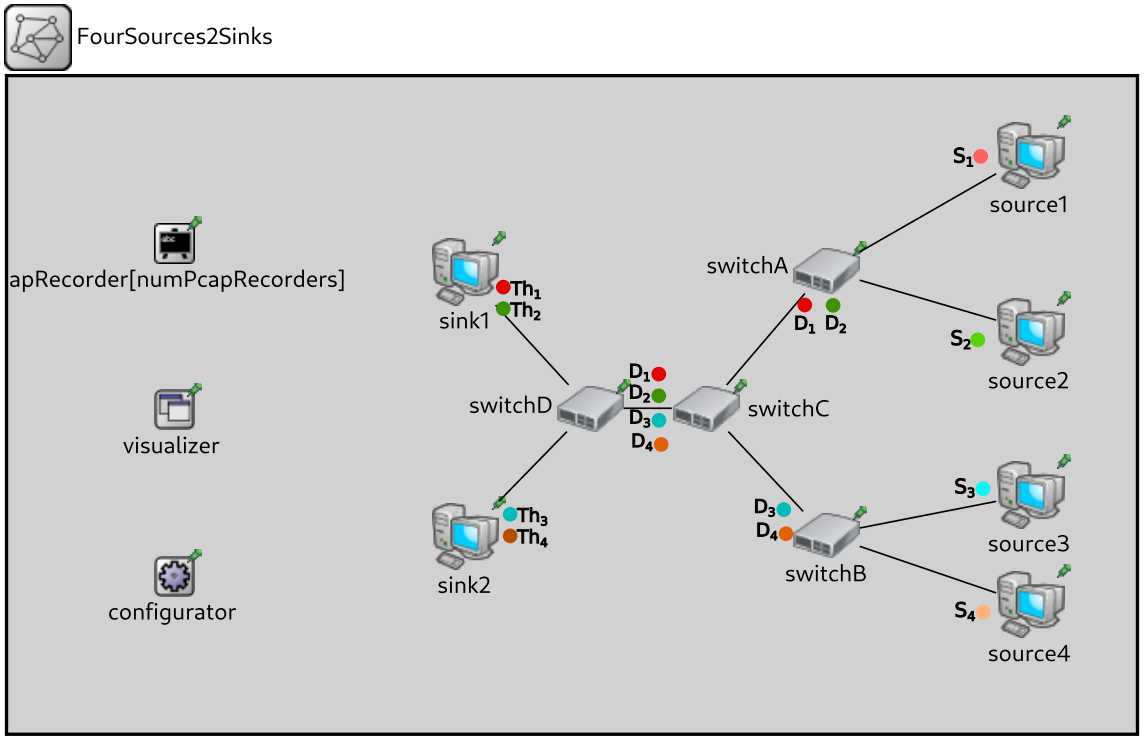

Figure 7.40: Topology of the simulations with two sinks

Like in the simulations before the throughput and also the drop rate increased with a higher token rate and a higher bucket size in the simulations. Again there was a point where it became inefficient to increase the token rate of the token-bucket shapers, because the drop rate did increase much more than the throughput. Also the drop rate increased when using color shaping for up to 2%.

What can be much better evaluated in this simulations is the distribution of the throughput between the different sources and also the sinks in the network. This is shown in Figure 7.41 where the throughput at sink2, sink2 and the lowest throughput of source1 - source4 is compared for different packet sources. The token rate for those simulations was at $1000Tps$ and the bucket size at 500 tokens. Color shaping was not used in these simulations. There we can see that without color shaping the throughput at sink1 and sink2 is very unbalanced for the ActivePS, the ActiveETS, the ActiveBCS and the cobination of ActivePS and ActiveETS. Also the lowest throughput of source1 - source4 is much under the guaranteed bandwidth of $2.4Mbps$ at the simulations with the ActivePS, the ActivePs with the ActiveETS and the ActiveETS. This is because without color shaping there is always one source which is able to oust the traffic of the other sources and thus many packets of the other sources are dropped. This also again leads to the drop of green marked traffic.

When now using color shaping with the token-bucket meter at the edges of the network, the distribution of the throughput changes which we can see in Figure 7.42. There are the results for the simulations with the same configurations as in Figure 7.41 but this time with color shaping activated. What we can see is that especially for the simulations with the ActivePS the throughput at sink1 and sink2 is much more balanced. This also holds for the simulations with the AcitveETS and the ActiveBCS. This is because the color shaper is able to better balance the traffic between the different source due to the color marking. But still for the simulations with a combination of

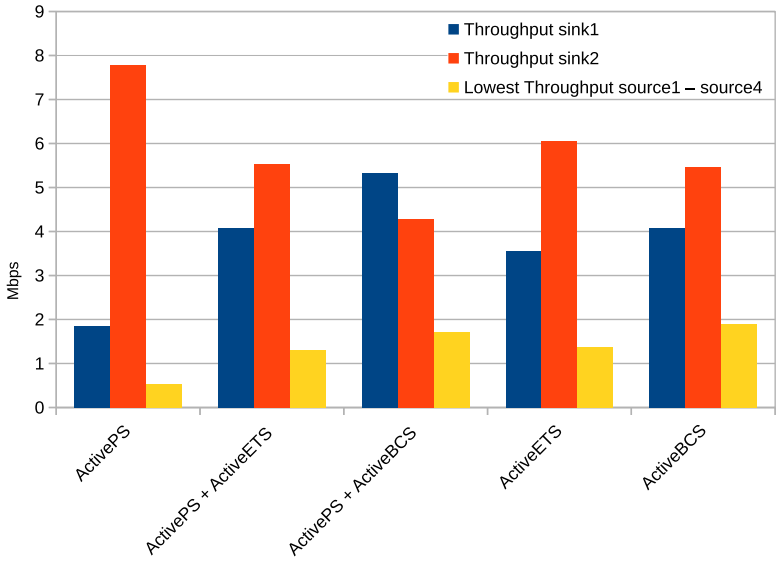

Figure 7.41: Comparison between the throughput of sink1, sink2 and the lowest throughput of source1 - source4 for different packet sources. Simulations were ran in the network of Figure 7.40 with a token rate of 1000 tokens, a bucket size of 500 tokens and without color shaping.

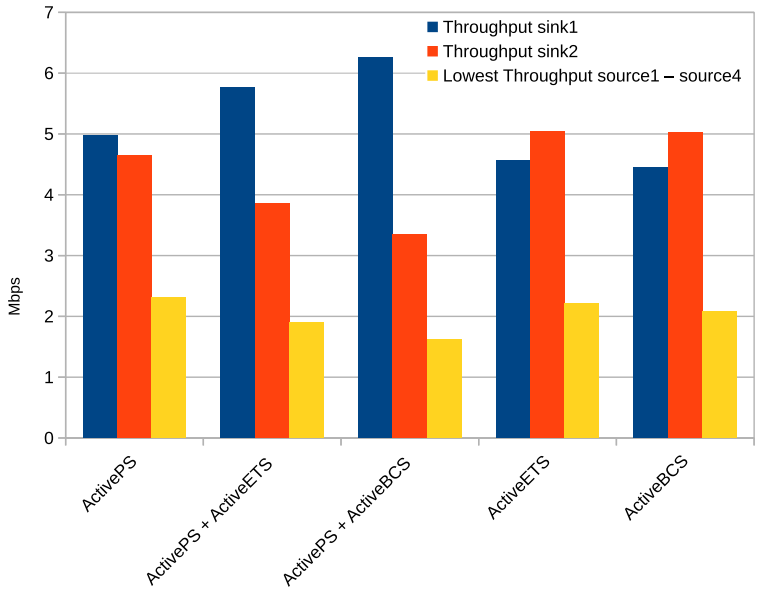

Figure 7.42: Comparison between the throughput of sink1, sink2 and the lowest throughput of source1 - source4 for different packet sources. Simulations were ran in the network of Figure 7.40 with a token rate of 1000 tokens, a bucket size of 500 tokens and with color shaping.

the ActivePS and the ActiveETS or the ActiveBCS the throughput which was measured at sink1 and sink2 is very unbalanced. This shows that the color shaper does not guarantee any fairness between the streams in the network beyond the guaranteed bandwidth. What we can also see is that the lowest throughput of source1 - source4 also increased for every packet source. This is because for every stream a bandwidth of $2.4Mbps$ was guaranteed during the simulations.

7.3.4 Observing Delay Bounds

After observing the effect of the color shaping switch on the throughput, the drop-rate, the fairness and the guaranteed bandwidth for each stream in a network, this section will focus on the observed delay bounds. Therefore the defined guarantees of Chapter 6 are validated and which average delays are observed. To check corollary 6.5.1 and 6.5.2 the device to controller scenario with four hops from Figure 7.37 of the previous simulations was used. Tabular 7.6 shows which parameters were changed between the different configurations of the simulations. Three different packet sources were used for all the sources in the network. Furthermore the threshold of the color shapers was improved from 100 packets to up to 750 packets. In all configurations the token rate of the token-bucket shapers was set to $750Tps$ and the bucket size to 250 tokens. The cir for every token-bucket meter at the edges of the network was set to $2.4Mbps$ and the cbs to $3000B$. Furthermore color marking was used only at the edges of the network.

Figures 7.43 and 7.44 show the maximum and the average delays which were recorded for the different color shaping thresholds and packet sources.

What can be observed is that for an increasing threshold the maximum as well as the average delay increase. This is because with a higher threshold there are more red packets, which can be put into the queues of the switches and thus the overall amount of packets in the queues increases. This has the effect, that the delay increases. Especially for the simulations where only ActivePSs were used to generate traffic this effect is even bigger than in the other simulations, because of the maximum load of the ActivePSs the queues in the switches are filled to their threshold most of the time. This is also why the difference between the maximum and the average delay of the simulations with an ActvicePS are smaller than in the other simulations. While in the simulations with the ActiveETSs and the ActiveBCs the queues of the switches are only filled in burst situations, the queues in the simulations with the ActivePS are filled most of the time due to the static load of the ActivePS. This has the effect that the delay of the simulations with the ActiveETSs and the ActiveBCs varies a lot while the delay of the simulations with the ActivePS does not. But still in general it can be said, that the average delay for green packets of all simulations was lower than the maximum delay. Also the maximum delay for green packets in all the simulations were much lower than the maximum delay which is estimated by corollary 6.5.1 which should be $d_maxg \leq \frac{625000B*3}{10Mbps} = 1.5s$.

Parameter	Values			
Packet source	ActivePS	ActiveETS	ActiveBCS	
Color shaper Thr	100 packets	250 packets	500 packets	750 packets

Table 7.6: Alternating configuration parameters for the simulations where delay bounds are observed.

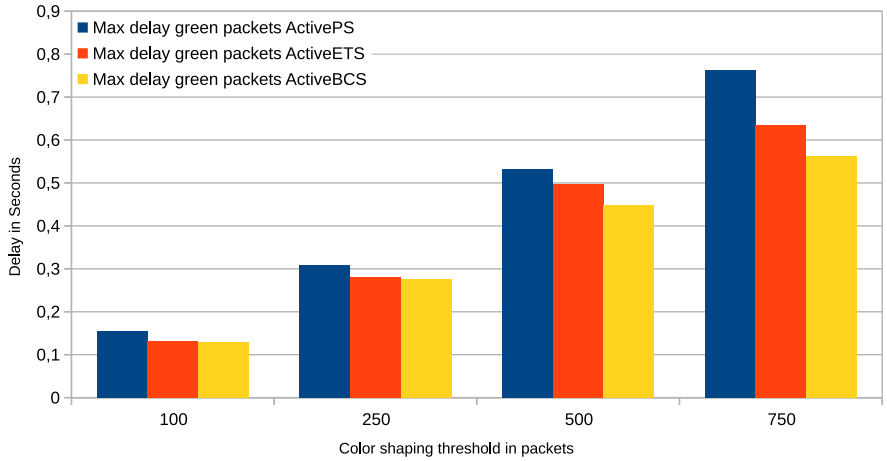

Figure 7.43: Maximum delay of green packets for an increasing color shaping threshold in the switches and different packet sources.

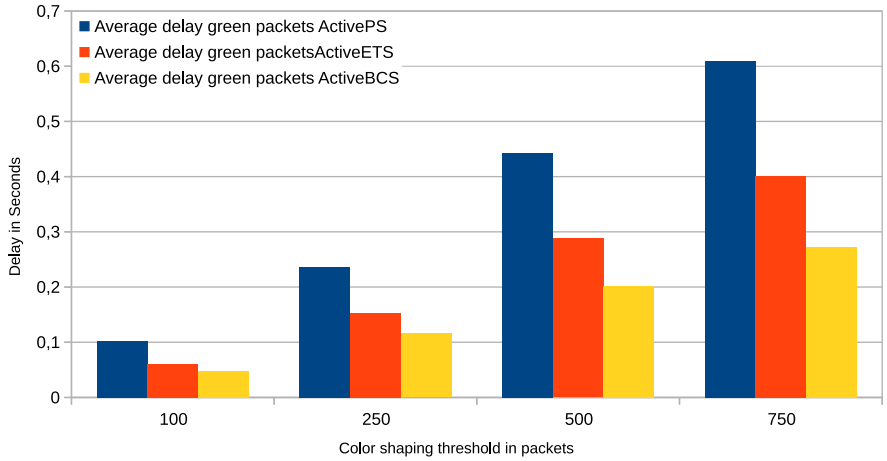

Figure 7.44: Average delay of green packets for an increasing color shaping threshold in the switches and different packet sources.

Similar observations can be done when looking at the maximum and average delay of the red packets which are shown in Figures 7.45 and 7.46. Again the maximum and average delay is shown for an increasing threshold and for different packet sources. Like in the previous results, the average and the maximum delay increases for a higher threshold which again can be explained by the increasing amount of packets, which can be stored in the queue. Also there is a difference between the simulations with an ActivePS and the ActiveETS or the ActiveBCS. This, like before, is because of the different source characteristics where the ActivePS generates a static load which fills the egress queues of the switches all the time, while the other sources fill the queues in bursts. A difference between the red and the green packets can be observed for the average delay. There the delay of the red packets from the ActiveETS is higher than the delay of the ActivePS, which is the opposite to the delay for green packets. This can be explained with the periodic burst characteristic

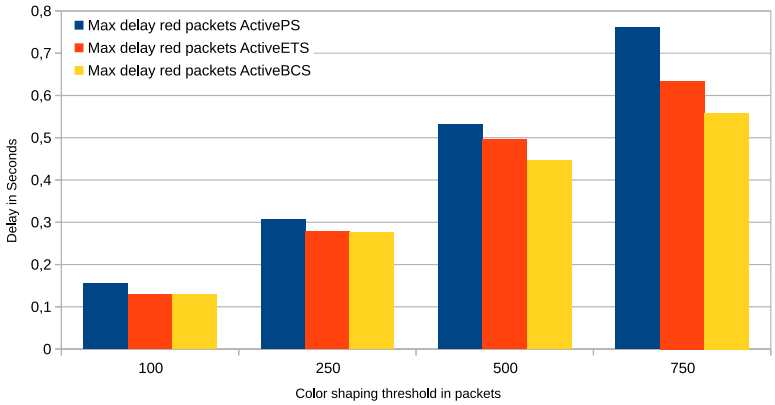

Figure 7.45: Maximum delay of red packets for an increasing color shaping threshold in the switches and different packet sources.

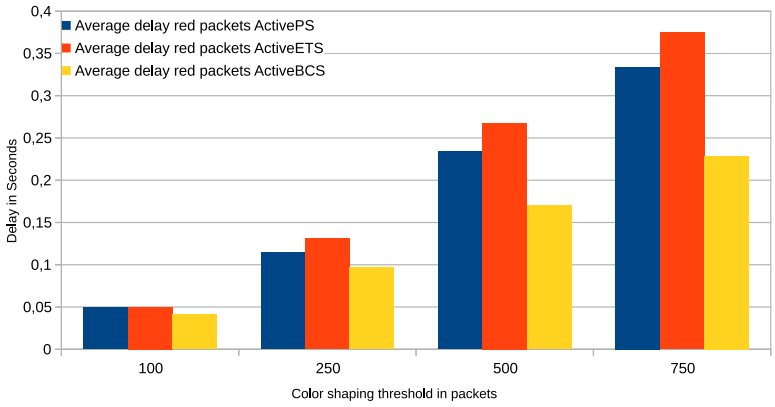

Figure 7.46: Average delay of red packets for an increasing color shaping threshold in the switches and different packet sources.

of the ActiveETS and the topology of the network scenario. Since the ActiveETS sends in periodic bursts, every burst can use an almost full token bucket which means that the ActiveETS can start its bursts with a higher send rate. These bursts will always lead to a situation where switchA has to buffer data which was sent by source1 and source2. These situations will only appear once for the ActivePS in the beginning of the simulation. This is because the ActivePS will always send with the full rate which is allowed by its token-bucket shaper and thus will produce only one burst at the beginning and then send with the token rate of the shaper. This token rate will not be high enough to force congestion at switchA and thus packets will not be buffered at switchA. This extra buffering at switchA leads to a higher average delay for red packets.

What also can be observed for the maximum delay of the red packets is that the worst-case estimation of corollary 6.5.2 this time is much more accurate. For a threshold of 100 packets the maximum delay for red packets, according to corollary 6.5.2, would be $d_maxr \leq \frac{500000b*3}{10Mbps} = 0.15s$, which is

the maximum delay that was recorded for the simulations with an ActivePS. Also the estimation for a threshold of 250 packets, which is $d_maxr \leq \frac{1.25Mb*3}{10Mbps} = 0.375s$, was very accurate for the maximum recorded value, which was at $0.308s$ for the simulations with the ActivePS.

In Conclusion, the worst-case estimations of corollary 6.5.2 based on the network topology and scenario are very accurate. But the evaluation also showed that the average delay for both, green and red packets, can be much lower than the maximum delay for network scenarios like in this evaluation.

7.3.5 Effect of More Weight

In this section corollary 6.4.3 is evaluated. This was done by taking the network of Figure 7.40 and increasing the token-bucket shapers of each source in the network one after another.

The bucket size of every token-bucket shaper was set to 250 tokens in these simulations. Furthermore the cir of every token-bucket meter was at $2.4Mbps$ and the cbs at $3000B$. The threshold of each color shaper was set so that lemma 6.1.4 was fulfilled and the color shapers of switchA and switchB had a threshold of 994 packets. Furthermore Tabular 7.7 shows which parameters were changed for the different simulation configurations. Two different packet sources were used which are the ActiveETS and the ActiveBCS. Then the simulation was done with all possible configurations where two sources have a higher token rate, e.g. weight, then three sources and as a last simulation all sources had a higher weight.

The results of simulations are shown in Figure 7.47 and 7.48. There the mean throughput of sources which had a higher weight are compared with the mean throughput of the other sources in the network, for a whole simulation. Also the amount of sources with a higher weight are illustrated. For both the ActiveETS and the ActiveBCS we can see that sources with a higher weight also get a higher throughput in the network. What we also can see is that when increasing the throughput of more than one source in the network, the throughput of the sources with a higher weight decreases. This is because the sources with more weight have to share the same bandwidth of the network and thus if more sources are using this bandwidth the throughput of the sources decreases.

Parameter	Values	
Packet source	ActiveETS	ActiveBCS
Token rate s1	$490Tps$	$1000Tps$
Token rate s2	$490Tps$	$1000Tps$
Token rate s3	$490Tps$	$1000Tps$
Token rate s4	$490Tps$	$1000Tps$

Table 7.7: Alternating configuration parameters for the simulations with different weight for one or multiple sources.

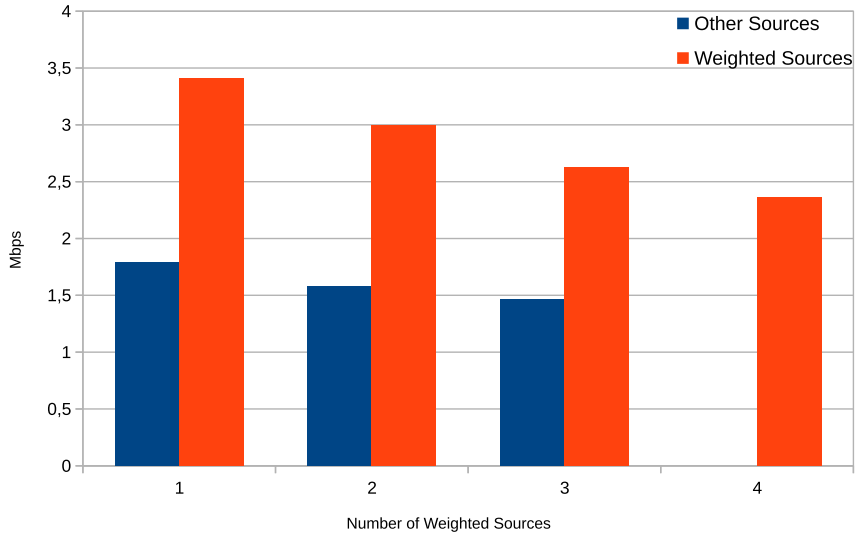

Figure 7.47: The throughput for weighted and not weighted sources with a different number of weighted sources in the network. The simulations were done by using either the ActiveETS in the network of Figure 7.40.

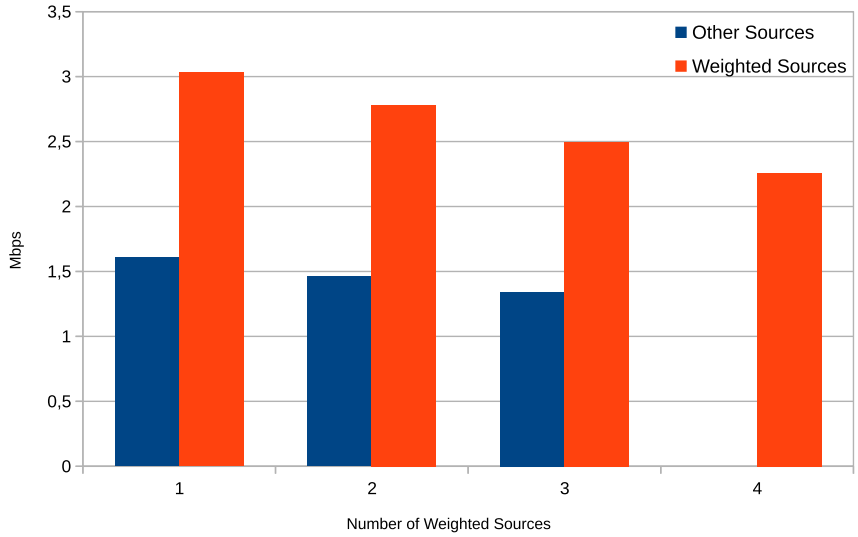

Figure 7.48: The throughput for weighted and not weighted sources with a different number of weighted sources in the network. The simulations were done by using either the ActiveBCS in the network of Figure 7.40.

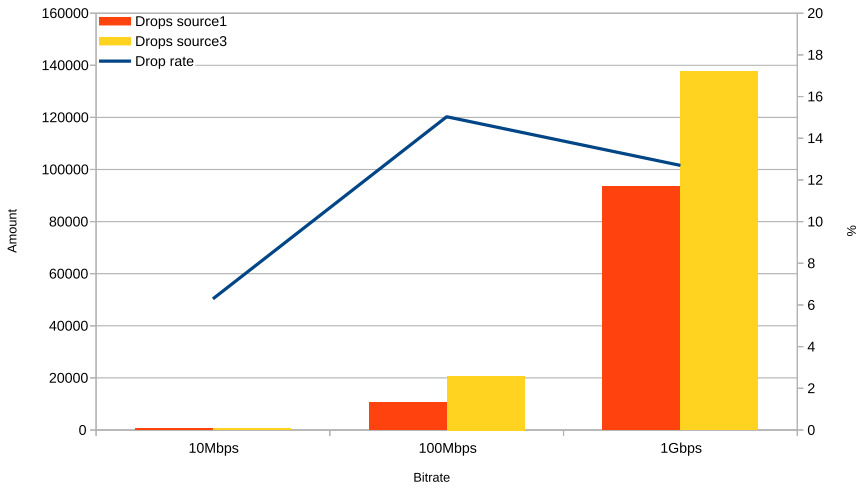

Figure 7.49: Simulation results for an increasing link rate and a static buffer size in the switches.

7.3.6 Effect of Small Buffer Size

In the last simulations the effect of an increasing link rate in the network for a static buffer size was evaluated. Like already mentioned in Chapter 6 the color shapers in the switches only can work as a shaper if they have enough buffer space. This means if the buffer space compared to the link rate gets smaller, the color shaper more and more gets to a traffic policer. Which effect this has on the traffic of a network, was evaluated by increasing the link rate in the two hop scenario of Figure 7.30 while keeping the same buffer space in the queues of the switches. As a base scenario the link rate was set to $10Mbps$ and the queue size of the queues in the switches was set to 1000 packets. Furthermore the token-bucket shaper of source1 and source3 was configured with a token rate of $1000Pps$ and a bucket size of 333 packets, where packets of an ActiveBCS were forwarded. The token-bucket shaper of source2 was configured with a token rate of $653Pps$, a bucket size of 333 packets and an ActivePoissonSource was used to generate packets at source2. The cir of the token-bucket meters was at $3.2Mbps$ and the cbs at $4000B$. Furthermore the threshold of the color shapers in switchA was set to 992 packets and the threshold of the color shapers in switchB to 493 packets to fulfill lemma 6.1.4.

Now simulations were run whit an increasing link rate from $10Mbps$ over $100Mbps$ to up to $1Gbps$. Together with the link rate also all corresponding parameters such as the token-bucket parameters, the cir and the cbs was increased with the same factor as the link rate. Furthermore the threshold of the color shapers were adjusted such that lemma 6.1.4 was still fulfilled in the new configurations. Here also the first effect of an increasing link rate was noticed, even before the first simulations were ran. Because the cbs also increases with the factor of the link rate, the color shaper are not able to buffer this amount of data anymore for the link rate of $1Gbps$ where the cbs would have been $400000B$. At $400000B$ the queue of switchB would have to buffer $2 * 400000B/613B = 1305$ packets only to fulfill the cbs requirements of lemma 6.1.4 which is more than the actual queue size of 1000 packets. This was why the cbs in the simulations with a link rate of $1Gbps$ had to be reduced to $2000000B$ to be able to fulfill lemma 6.1.4 again.

Figure 7.49 shows the results of the simulations. There the overall drop rate in the network for

a whole simulation as well as the overall drops of source1 and source2 are shown. What can be observed is that with a higher link rate the overall drop rate rises from 6.29% for $10Mbps$ to 15% and 12.69% for the higher link rates. This is because with a higher link rate the color shapers in the switches are not able to buffer as much data as before anymore in case of any congestion. While the color shapers in the switches were theoretically able to buffer packets for up to 0.5 second for a link rate of $10Mbps$ the duration goes down to only 0.005 seconds for a link rate of $1Gbps$. This gives the shaper a characteristic which is more like a policer because it is no able to distribute the load over time anymore.

Another effect is that packets of source1 and source3 are not dropped with the same weight anymore, when the link rate increases. For a link rate of $10Mbps$ the amount of dropped packets from source1 and source3 was at 620 and 628, which is only a difference of $\approx 1.3\%$. For a link rate of $1Gbps$ the amount of dropped packets from source1 and source3 was at 93325 and 137450, which was a difference of $\approx 32.1\%$.

In conclusion we can say that if the buffer size of the color shapers in a network are to small, the shaping process is not working anymore. Also the fairness between multiple streams gets worse for a smaller buffer size of the color shapers. This especially is a problem for networks with a high link rate, since the buffer space of a switch can not be increased as easily as the link rate that is supported by the switch. This is because it is much more expensive to increase fast operating memory, which is needed for a high link rate, than the rate at which packets can be propagated.

8 Conclusion and Future Work

This thesis introduced a new approach where traffic shaping is used to handle traffic of time-sensitive applications like event-triggered traffic. The evaluation showed that the new approach improves the overall utilization in a network while guaranteeing a certain bandwidth for very application. To do so shaping with application knowledge at the network edge was used to optimize the performance, e.g. the utilization. This was done by using a token-bucket shaper which limits the traffic of one stream by setting its token rate and bucket size. Furthermore in-network shaping with limited knowledge about the network situation was used to cope with worst-case situations. As a in-network shaping mechanism color shaping was used which marks incoming traffic at every edge port of a network and than shapes the traffic based on its color marking at the egress of every switch.

The evaluation showed that when using the new approach, a certain bandwidth for every stream in a network can be guaranteed and while it is possible to increase the utilization in the network. Also the drop rate and the throughput of a network does not change in a negative manner when using the new approach. In addition the evaluation showed that it is possible to give streams in a network different weights beyond their guaranteed bandwidth. The thesis also provided the tooling to determine the worst-case behavior of a network with the new approach which are the maximum delay, the maximum amount of dropped data and the throughput of a stream of Interest in the network. These estimations were done by using Network Calculus.

For future work the very simple color shaping algorithm could be exchanged by a more complex AQM mechanism. This more complex mechanism then might be able to guarantee a better fairness for the dropped amount of traffic between the streams in a network. It could also be possible to use multiple queues at the egress ports of every switch in the network, to priories green marked traffic. Furthermore the networks, which were observed in this thesis, were all static and thus the approach could be extended to deal with dynamic network scenarios. Finally also the worst-case estimations can be improved such that the calculation of a whole network is done with less overhead. For now the output, delay bounds and the dropped amount of data has to be calculated separately for very switch and stream in the network. This may be improved by finding rules which allow to concatenate switches.

Bibliography

[18] "IEEE Standard for Local and Metropolitan Area Network–Bridges and Bridged Networks". In: *IEEE Std 802.1Q-2018 (Revision of IEEE Std 802.1Q-2014)* (2018), pp. 1–1993. DOI: 10.1109/IEEESTD.2018.8403927 (cit. on pp. 15, 18, 29, 40).

[AAK00] M. Alam, M. Atiquzzaman, M. Karim. "Traffic shaping for MPEG video transmission over the next generation internet". In: *Computer Communications* 23.14 (2000), pp. 1336–1348. ISSN: 0140-3664. DOI: https://doi.org/10.1016/S0140-3664(00)00180-8. URL: http://www.sciencedirect.com/science/article/pii/S0140366400001808 (cit. on pp. 13, 25, 27, 29).

[AGN12] A. M. AlAdwani, A. Gawanmeh, S. Nicolas. "A Demand Side Management Traffic Shaping and Scheduling Algorithm". In: (2012), pp. 205–210 (cit. on p. 15).

[BF08] M. Boyer, C. Fraboul. "Tightening end to end delay upper bound for AFDX network calculus with rate latency FIFO servers using network calculus". In: *2008 IEEE International Workshop on Factory Communication Systems*. 2008, pp. 11–20. DOI: 10.1109/WFCS.2008.4638728 (cit. on p. 45).

[CSYM02] Chunhua Hu, H. Saidi, P. Yan, P. S. Min. "A protocol independent policer and shaper design using virtual scheduling algorithm". In: 1 (2002), 791–795 vol.1 (cit. on p. 15).

[CW98] D. D. Clark, Wenjia Fang. "Explicit allocation of best-effort packet delivery service". In: *IEEE/ACM Transactions on Networking* 6.4 (1998), pp. 362–373. DOI: 10.1109/90.720870 (cit. on p. 26).

[FHC+19] J. Falk, D. Hellmanns, B. Carabelli, N. Nayak, F. Dürr, S. Kehrer, K. Rothermel. "NeSTiNg: Simulating IEEE Time-sensitive Networking (TSN) in OMNeT++". In: *Proceedings of the 2019 International Conference on Networked Systems (NetSys)*. Garching b. München, Germany, Mar. 2019 (cit. on p. 22).

[Fid10] M. Fidler. "Survey of deterministic and stochastic service curve models in the network calculus". In: *IEEE Communications Surveys Tutorials* 12.1 (First 2010), pp. 59–86. ISSN: 1553-877X (cit. on pp. 20, 46).

[FJ93] S. Floyd, V. Jacobson. "Random early detection gateways for congestion avoidance". In: *IEEE/ACM Transactions on Networking* 1.4 (1993), pp. 397–413 (cit. on pp. 13, 25, 40).

[GCR01] J. Glasmann, M. Czermin, A. Riedl. "Estimation of Token Bucket parameters for videoconferencing systems in Corporate Networks". In: (Oct. 2001). DOI: https://doi.org/10.1007/s00591-010-0080-8 (cit. on p. 55).

[GHS15] S. E. Ghoreishi, A. Hamid Aghvami, H. Saki. "Active queue management for congestion avoidance in multimedia streaming". In: *2015 European Conference on Networks and Communications (EuCNC)*. 2015, pp. 487–491 (cit. on pp. 13, 25).

[HAB90] D. A. Hughes, G. Anido, H. S. Bradlow. “Performance of the leaky bucket policing mechanism for small bucket depths”. In: *Electronics Letters* 26.16 (1990), pp. 1305–1307 (cit. on pp. 25, 29).

[HG99] D. J. Heinanen, D. R. Guerin. *A Two Rate Three Color Marker*. RFC 2698. Sept. 1999. DOI: 10.17487/RFC2698. URL: https://rfc-editor.org/rfc/rfc2698.txt (cit. on p. 17).

[Kli11] F. Klinker. “Exponential moving average versus moving exponential average”. In: (Apr. 2011) (cit. on p. 25).

[KSS] W.-c. F. D. D. Kandlurz, D. Sahaz, K. G. Shiny. “Adaptive Packet Marking for Maintaining End-to-End Throughput in a Differentiated Services Internet”. In: () (cit. on p. 26).

[Le 10] J.-Y. Le Boudec. *Performance Evaluation of Computer and Communication Systems*. EPFL Press, Lausanne, Switzerland, 2010 (cit. on p. 57).

[LT01] J.-Y. Le Boudec, P. Thiran. *Network Calculus: A Theory of Deterministic Queuing Systems for the Internet*. Berlin, Heidelberg: Springer-Verlag, 2001. ISBN: 3-540-42184-X (cit. on pp. 14, 19, 20, 42).

[MSL19] E. Mohammadpour, E. Stai, J. Le Boudec. “Improved Credit Bounds for the Credit-Based Shaper in Time-Sensitive Networking”. In: *IEEE Networking Letters* 1.3 (2019), pp. 136–139. DOI: 10.1109/LNET.2019.2925176 (cit. on p. 40).

[NC01] K. Nichols, B. Carpenter. *Definition of Differentiated Services Per Domain Behaviors and Rules for their Specification”, RFC 3086*. 2001 (cit. on pp. 25, 26).

[SCY+96] Y. S. Sun, Chin-Fu Ku, Yu-Chun Pan, Chia-Hui Wang, Jan-Ming Ho. “Performance analysis of application-level traffic shaping in a real-time multimedia conferencing system on Ethernets”. In: (1996), pp. 433–442 (cit. on p. 25).

[SJ10] R. Stankiewicz, A. Jajszczyk. “Performance modeling of DiffServ meter/markers”. In: *International Journal of Communication Systems* 23.12 (2010), pp. 1554–1580. DOI: https://doi.org/10.1002/dac.1126. eprint: https://onlinelibrary.wiley.com/doi/pdf/10.1002/dac.1126. URL: https://onlinelibrary.wiley.com/doi/abs/10.1002/dac.1126 (cit. on p. 34).

[SND+00] S. Sahu, P. Nain, C. Diot, V. Firoiu, D. Towsley. “On Achievable Service Differentiation with Token Bucket Marking for TCP”. In: *SIGMETRICS Perform. Eval. Rev.* 28.1 (June 2000), pp. 23–33. ISSN: 0163-5999. DOI: 10.1145/345063.339342. URL: https://doi.org/10.1145/345063.339342 (cit. on p. 25).

[VH08] A. Varga, R. Hornig. “An Overview of the OMNeT++ Simulation Environment”. In: *Proceedings of the 1st International Conference on Simulation Tools and Techniques for Communications, Networks and Systems and Workshops*. Simutools ’08. Marseille, France: ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), 2008. ISBN: 9789639799202 (cit. on pp. 21, 22).

[WSKS02] Wu-chang Feng, K. G. Shin, D. D. Kandlur, D. Saha. “The BLUE active queue management algorithms”. In: *IEEE/ACM Transactions on Networking* 10.4 (2002), pp. 513–528 (cit. on pp. 13, 25).

[Zam18] P. von Zameck Glyscinski. “Stückweise lineare Funktionen für Network Calculus”. In: (2018). DOI: http://dx.doi.org/10.18419/opus-9981 (cit. on p. 20).

[ZZZ00] Zhiruo Cao, Zheng Wang, E. Zegura. “Rainbow fair queueing: fair bandwidth sharing without per-flow state”. In: *Proceedings IEEE INFOCOM 2000. Conference on Computer Communications. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies (Cat. No.00CH37064)*. Vol. 2. 2000, 922–931 vol.2. DOI: 10.1109/INFCOM.2000.832267 (cit. on pp. 25, 40).

All links were last followed on December 9, 2020.

Declaration

I hereby declare that the work presented in this thesis is entirely my own and that I did not use any other sources and references than the listed ones. I have marked all direct or indirect statements from other sources contained therein as quotations. Neither this work nor significant parts of it were part of another examination procedure. I have not published this work in whole or in part before. The electronic copy is consistent with all submitted copies.

---

place, date, signature