TECHNISCHE UNIVERSITÄT
ILMENAU

A QoS Model for Highly Variable Mobile Networks

Dissertation

zur Erlangung des akademischen Grades
Doktor-Ingenieur (Dr.-Ing.)

vorgelegt an der Fakultät für Informatik und Automatisierung
der Technischen Universität Ilmenau

von Herrn Dipl.-Inf. Markus Brückner
geboren am 16.11.1979 in Zeulenroda, Deutschland

Datum der Einreichung	29.05.2017
Datum der Verteidigung	15.11.2017
Gutachter	Univ.-Prof. Dr.-Ing. Andreas Mitschele-Thiel (TU Ilmenau)
	Univ.-Prof. Dr. rer. nat. Jochen Seitz (TU Ilmenau)
	Univ.-Prof. Dr. Paul Müller (University of Kaiserslautern)

urn:nbn:de:gbv:ilm1-2017000519

Thanks

First and foremost I would like to thank my supervisor and reviewers, without whom this work would not get evaluated and therefore ultimately finished. In particular my supervisor Prof. Andreas Mitschele-Thiel ensured, that I would even make it this far. He offered me the room to find my motivation again and start over with a different topic, after my first attempt at a dissertation failed due to prior art quite late in the process. Without his support this journey would have been over years ago – unsuccessfully so.

I am equally indebted to my colleagues at the TU Ilmenau, in the Integrated Communication Systems group and the KASYMOSA project, for providing a stimulating working environment for my research. Prof. Martin Dietzfelbinger for his patience and crucial input regarding the optimization problem embedded in this work. Sometimes, going back to the – not always loved – theoretical foundations of your chosen profession, is the key to the successful conclusion of a project.

To Manuel Osdoba and Philipp Kerling, for bringing in their software development skills, without which the implementation of the ideas presented here would never have been possible, and for asking questions, that shook the assumptions and ideas of my work, until I was finally convinced, that they are sound. Rainer Hildinger, Hartmut Brandt and Manuel Probst, who provided crucial context for the environment, in which the system would have to operate. Without their – sometimes surprisingly patient – explanations and questions, I would not have seen some of the problems and solutions that this work explores.

Writing in a foreign language is always a challenge. I am indebted to Daniel Brady for his invaluable input on the subtleties of the English language. With his commas and „things that sound strange“ he helped to remove many mistakes and ambiguities from the text. Any remaining errors are mine and mine alone.

Last, but not least, I am grateful for the support given to me by friends and family, who encouraged me to pursue this work and, at times, teased me just enough to get the motivation going again. To my son for giving me time to finish the work (although you will probably not understand this until you are quite a bit older) and my wife for supporting me wholeheartedly, even though she feared my motivation would not last for a second attempt. It did, for which I am probably most grateful of all.

Abstract

The goal of a reservation-based QoS system is to guarantee specific transmission parameters to data flows especially in a resource-constrained environment. To do so, it has to balance different requests and find a resource allocation maximizing the overall value to its users.

Traditionally there is an information gap in this optimization process: The network layer operates on individual data flows, whereas higher layers bind multiple flows together to form more complex connections. Without being aware of these relationships introduced on higher layers, the network layer may assign transmission resources to reservations, while not guaranteeing other, additionally required reservations. The result is a sub-optimal allocation, which wastes resources that could ultimately be used to better satisfy users in an already quite strained environment.

This work proposes to make the network layer aware of higher-layer relationships by modeling them as propositional formulas over the resource allocation. Each reservation is represented by a Boolean variable giving its current resource assignment state. By deriving the resulting value of each propositional formula the optimization process can detect – and ultimately eliminate – wasted resources.

The optimization problem is an instance of the *Knapsack* problem with additional Boolean constraints. By transforming it into a Mixed Integer Linear Program existing optimization algorithms like Branch-and-Cut can be used to find optimal solutions in a reasonable time frame. The results of this work indicate that, depending on the scenario, up to 75% of assigned resources may be wasted in a relation-unaware system. Alleviating this problem yields a dramatic increase of the number of admitted application sessions.

Together with proactive resource management, where reservations can be suspended and resumed by the network, this approach is especially suited for highly variable, long delay networks. It is able to autonomously find the optimal resource distribution, without putting additional signaling burden on already limited resources.

Zusammenfassung

Aufgabe eines reservierungsbasierten QoS-Systems ist die Garantie spezifischer Übertragungsparameter für Datenflüsse in Umgebungen mit begrenzten Ressourcen. Zu diesem Zweck muss das System verschiedene Anforderungen gegeneinander abwägen und eine Ressourcenzuteilung maximalen Wertes für die Nutzer finden.

In existierenden Lösungen leidet die Ressourcenzuteilung unter eine Informationslücke: die Netzwerkschicht betrachtet einzelne Datenströme ohne Kenntnis ihrer Beziehungen in höheren Schichten. Durch diese Lücke kann es zur Zuteilung von Ressourcen an nicht nutzbare Reservierungen kommen, die auf andere, nicht aktive Ströme angewiesen sind. Dies resultiert in einer Verschwendung von Ressourcen, welche anderweitig besser zur Erbringung von Diensten genutzt werden könnten.

Die vorliegende Arbeit schlägt ein QoS-System der Netzwerkschicht vor, welches Kenntnis über die existierenden Pfadbeziehungen besitzt. Diese werden als Ausdrücke der Aussagenlogik formuliert, in denen die Zuständer der reservierten Pfade als Boolesche Variablen repräsentiert werden. Durch Auswertung der entsprechenden Ausdrücke kann der Optimierungsprozess verschwendete Ressourcen erkennen und vermeiden.

Das Optimierungsproblem ist eine Instanz des *Rucksackproblems* mit Nebenbedingungen. Durch Umwandlung in ein Mixed Integer Linear Program können durch existierende Algorithmen wie Branch-and-Cut optimale Ressourcenzuteilungen in vertretbarer Zeit berechnet werden. Die vorliegenden Ergebnisse zeigen, dass die Optimierung ohne Kenntnis der Pfadbeziehungen in bestimmten Szenarien bis zu 75% der zugeteilten Ressourcen verschwendet. Eine Lösung dieses Problems erlaubt eine wesentlich effizientere Nutzung begrenzter Ressourcen und damit den Transport einer deutlich größen Zahl an Anwendungssitzungen.

Zusammen mit einem netzseitigen Ressourcenmanagement, welches Reservierungen pausieren und fortsetzen kann, ist dieser Ansatz besonders für stark veränderliche Netze mit langen Übertragungsverzögerungen geeignet. Das Netz findet autonom eine optimale Ressourcenzuteilung, ohne die begrenzten Ressourcen durch zusätzliche Signalisierung zu belasten.

Contents

1. Introduction

When large-scale disaster strikes, one of the most urgent issues after the initial local emergency response is getting communication systems up and running. Coordinating extensive rescue and relief operations is an exercise in logistics, as well as coordination. Ground-based systems like the landlines or mobile networks usually suffer extensive damage, often rendering them unusable. A prime example was hurricane Katrina [IH06] in the southern USA in 2005: destroyed cellphone towers, broken landlines, and, later on, stolen power generators caused a widespread communication break-down which severely hampered the work of rescue forces.

In this situation, independent mobile communication systems, such as satellite networks, can provide services even in the most heavily destroyed regions. With their independent infrastructure, they allow rescue organizations to enter the area, stay connected to their headquarters and coordinate efforts based on a multitude of data otherwise not available.

Communication in a disaster scenario employs a plethora of different systems and protocols: Real-time communication, such as voice or video calls, time-critical emergency messages, and simple data traffic, e.g. maps or mission plans. Event high capacity emergency networks are typically not able to cope with this demand placed on them. Providing a sufficient service via a very limited infrastructure requires an efficient allocation of available resources.

1.1. QoS systems for efficient resource allocation

The fundamental best-effort model of the Internet Protocol is not well suited for efficiently allocating transmission resources in certain scenarios. The protocol stack is mainly designed for robust communication without central control. Mechanisms like the congestion control of TCP aim to provide a fair allocation of resources to all users. However, this fairness can be detrimental to efficiency. Applications which require a certain minimum data rate or maximum latency are better off not communicating at all, instead of wasting precious resources in their futile endeavor.

Enter Quality-of-Service (QoS) systems: even the very early incarnations of the Internet Protocol were able to signal additional information about the preferred treatment of a packet via the Type-of-Service field in the IPv4 header. A simple priority model combined with general information about the preferred packet treatment (e.g. high throughput, low delay or high reliability) provided ways for the network to distinguish different types of traffic and better allocate limited resources.

With the advent of real-time services for the Internet, the need arose for a more rigid system being able to guarantee certain transmission parameters. The resulting Integrated Services architecture provided a way to set up and tear down transmission paths. Network nodes were to set aside the necessary resources, or reject reservations where this was impossible. Similar to telephone networks with their dedicated lines, IntServ provides upfront guarantees, not impacted by the behavior of other network users. For more details on IntServ and its applicability to this work see section 2.2.1.

Where the IntServ architecture is able to provide an efficient allocation of limited resources tailored to the needs of the end user, it suffers from a major drawback: its lack of scalability. The Internet resembles a scale-free network: relatively few highly connected core networks (the Tier 1) carry the majority of traffic, whereas towards the network edge the load lessens considerably. For this reason, IP was designed with the state being held at the end system. At its core, an IP network is nearly stateless: apart from a relatively static set of routing information, the core network does not contain any transient state for individual traffic flows or connections. IntServ violates this assumption by introducing exactly this per-flow state on every router along the way. This leads to a massive amount of state information being stored on the Tier 1 network, quickly overloading the constituent systems.

Combating these scalability issues requires pushing the state back out to the network edge. In Differentiated Services (DiffServ), core routers only provide a fixed set of forwarding behaviors that can be used by edge routers to provide the requested service quality. As it is missing a reservation process, DiffServ does not necessarily provide upfront transmission guarantees. It can, however, be combined with systems that offer such: a more complex reservation-based system towards the end user and a simpler, more efficient behavior mapping at the network core. DiffServ is desribed more in detail in section 2.2.2.

What both architectures provide is a way to express QoS requirements, thus allowing the network to better allocate limited resources and possibly reject traffic that might overload the system.

1.2. Problem statement

Guaranteeing certain transmission properties to flows in the network layer introduces an information gap between the network and higher layers: the QoS model only carries information about individual streams based on the information contained in individual packets. On the transport and, in particular, application layers, more complex protocols prevail. A single TCP stream, for example, already consists of two individual network layer streams carrying data and, crucially, acknowledgements. On the application layer, even more elaborate setups may exist. Protocols like HTTP or FTP introduce dependencies between different TCP connections. Applications such as adaptive video coding might even change the transmission parameters during a session to better cope with changes in the environment.

None of this information is available to the QoS system on the network layer. When allocating resources to reservations, only the the individual flow is considered. In situations with constrained capacity this can lead to mis-allocation: resources may lie dormant because of unmet dependencies on a higher layer. These dormant reservations might even preempt perfectly usable ones, decreasing user satisfaction even further.

Filling this information gap not only allows the network to allocate resources more precisely, but also enables new use cases for the QoS system. Instead of having an all-or-nothing decision with regards to a reservation (and therefore a service) end-systems can, for example, now offer multiple alternatives to the network. By requesting multiple variants of the same service (e.g. multiple output profiles of a video codec), along with information about the relationship between those requests (specifically a mutual exclusion in this case), the end-system provides enough information for the network to gracefully degrade in the case of constrained resources.

This work intends to develop a quality-of-service system that fills the aforementioned information gap. By providing the expressive means to model high-level relationships between paths, the system should improve overall performance in constrained environments and enable use cases like graceful degradation.

1.2.1. Target environment

The research presented here was carried out in the context of mobile satellite communication. However, the results hopefully apply to a broader range of communication networks. The system environment is characterized by:

Long delay Communication in the Internet is mainly characterized by low delay paths. Even data transfer halfway around the world and back generally takes place within round-trip times below 300-400 ms.

In contrast, networks spanning satellites in a geostationary orbit impose a time-of-flight delay of approximately 240 ms in one direction[a]. Together with processing delays, this amounts to a round-trip delay starting at 500 ms, possibly exceeding several seconds.

High packet loss Mobile communication networks in general, and mobile satellite networks in particular, are prone to environmental influences which lead to increased packet loss. Fading, shadowing and interference all increase the chances of destroying crucial information in flight. Appropriate modulation and coding schemes, as well as interleaving are able to protect against certain types of interference. They do, however, have an adverse influence on the available capacity, as well as processing delay.

Low data rate Compared to wired networks, wireless links are generally of low capacity. This is especially true for long-range networks, with low signal-to-noise ratios at the receiver.

Variable link capacity Fading, shadowing and interference are not constant environmental factors for a mobile communication link, but rather change over time (at widely varying timescales). This directly translates into a highly variable available capacity on the link.

QoS-aware MAC layer For special use cases like satellite communication, some kind of resource allocation scheme, fixed or variable, on the shared medium typically exists. Integrating an end-to-end QoS system with this type of intelligent MAC can have two implications: such a MAC layer might benefit from, even need, information about the resource requirements known to the higher layers. On the other hand it can provide information about the capacity on the lower layers to benefit the QoS system.

This target environment provides some unique challenges for the system in order to solve the use cases and requirements described in the following section.

1.2.2. Requirements

The work done in this dissertation is guided by one overarching proposition: relationships between reserved paths in a QoS system are the key to improved network performance.

[a] Geostationary orbits are at a height of approximately 36,000 km. A signal sent in one direction travels 72,000 km (ground-to-ground via satellite) at the speed of light, resulting in the delay mentioned.

Taking them into account enables the system to better allocate resources to applications and satisfy more end-user requirements. By allowing the network a higher autonomy while still retaining the same decision quality, the distributed state of reservations will converge faster in the case of changes. This leads to less transition time, minimizing the performance impact.

This section expands on the requirements behind this proposition by presenting some common example use cases and deriving common system traits from them.

1.2.2.1. Use cases

The research presented here was carried out in the context of the KASYMOSA project (see section 1.3 for details). Its focus on disaster recovery communication therefore inspires the use cases of this work. Only actual QoS-protected communication is considered here. While some level of best-effort traffic can be expected to be present in the network, it has no bearing on the QoS architecture. It is transmitted using any remaining capacity beyond the requirements of the reserved traffic and is the first to be cut when resources degrade.

Data transfer via TCP With TCP [Pos81] being the main transport layer protocol in the Internet, there is no way around it as a use case. From the user's perspective, the protocol offers stream-oriented data transfer with delivery and order guarantees over unreliable networks. From the network's point-of-view, TCP offers automatic adaptation to capacity changes, as well as fair resource sharing without central control.

Delivery and order guarantees are achieved by segmenting TCP traffic into individual packets and acknowledging segments at the receiver. If a segment is not acknowledged, it is retransmitted by the sender. A TCP connection therefore always consists of two flows: the data and the acknowledgement (ACK) flow[b].

For reasons of efficiency over links with a high bandwidth-delay-product, the protocol implements a go-back-N approach: The sender transmits a certain amount of data (as determined by the transmission window) without receiving an acknowledgement. Segments are acknowledged cumulatively, i.e. an acknowledgement of segment X is interpreted as "all segments up to X have been received". The acknowledgement stream is therefore always much smaller than the corresponding data stream, resulting in a highly asymmetric, bidirectional flow.

[b] TCP is capable of transmitting data in both directions of a connection. However, a common use case is the transfer of a resource from a server to a client. This creates an asymmetric flow with data mainly going one direction and acknowledgements traveling the other.

Determining the optimal size of the transmission window is an ongoing research question. It is guided by two conflicting goals:

1. Maximize the throughput of the TCP connection.
2. Prevent overload of the network and the receiver, and share available resources in a fair way.

Preventing the overload of the receiver is achieved by explicitly signaling the *Receive Window* in the TCP header and is not relevant from the network's point of view. Maximizing throughput without overloading the network can be achieved by correctly estimating the bandwidth-delay-product (BDP) of the path (i.e. the maximum amount of data "in flight" before the first acknowledgement can be received). Sending any more data would lead to an overload situation and eventually packet loss, any less would waste resources. The current estimate of the BDP is represented by the *Congestion Window* at the sender of a TCP stream. There is a plethora of control algorithms determining this parameter for all different kinds of use cases. Most of them work according to the same basic principles: ramping up the transmission window quickly (e.g. exponentially) to saturate the available capacity (slow start phase) and relying on transmission errors to detect congestion in the network. Transmission errors are detected by missing ACK packets. Whenever the algorithms detect a congestion, they adapt their transmission thresholds in order to reach a stable BDP estimate.

TCP's congestion control algorithms typically share two common problems:

- Lost packets are considered to be caused by network congestion. Packets that are lost due to other causes (e.g. transmission errors in a wireless network) erroneously may cause the congestion avoidance to kick in. While some extensions of the original TCP proposal address this issue (namely TCP Fast Retransmit and TCP Fast Recovery [APB09]), performance still rests heavily on the stability of the link.
- Estimating the current BDP takes several transmitted packets and received acknowledgements, thus incurring a significant delay. This delay is especially relevant in fast changing environments like mobile communication.

From the point-of-view of the QoS system, TCP has the following properties:

Two mutually dependent paths Due to the nature of TCP's acknowledgement mechanism, each connection consists of two streams in opposite directions. Both flows can only be used in conjunction. If one of the paths cannot be guaranteed, the other is inherently unusable: either no data transfer takes place (then no acknowledgements are needed or sent), or no acknowledgements can be transmitted (then data transfer will stop when the transmission window is exhausted).

Slow reaction to changes Due to the nature of TCP's congestion control algorithms, a connection takes some time to ramp up to maximum transmission speed. For maximum efficiency it therefore relies on a stable link capacity over time.

Flexible transmission speed Depending on the application running on top, TCP in and of itself does not require a specific data rate. The protocol will readily adapt to whatever the network has to offer, as long as it stays stable for an extended period of time (typically several orders of magnitude longer, than the round-trip time of the underlying link).

Susceptibility to packet loss TCP assumes the cause of lost packets to be network congestion and will lower its transmission rate to adapt to the presumed problem. This makes it very susceptible to packets being lost due to transmission errors on a wireless medium. Packet loss has a particularly large impact in the start-up phase of a connection where the packet rate is too low for techniques like TCP Fast Recovery to work properly.

Susceptibility to jitter Without an explicit signaling regarding lost packets, TCP has to rely (amongst others) on timeouts to detect missing ACKs. For efficiency reasons, this timeout needs to be estimated based on the current round-trip-time (RTT) of the connection. Too short of an estimate will trigger needless retransmissions. An overly long estimate, on the other hand, will needlessly delay retransmissions. TCP estimates the correct retransmission timeout based on a weighted average of the RTT, as well as the jitter. A higher jitter will, over time, increase the retransmission timeout to unusable levels, massively slowing down the congestion control feedback loop.

Web-Browsing via HTTP Arguably the most used service in the Internet, the World Wide Web and thereby HTTP [BFF96] in its various incarnations presents its very own set of challenges for a QoS system. Using TCP as the underlying transport protocol it inherits all the requirements and assumptions presented in section 1.2.2.1. and adds yet another layer of complexity. The deprecated version 1.0 is a simple request-response protocol which opens one short-lived TCP connection per downloaded resource (of which there might be many in a single web page containing style sheets, images, script files etc.). This implies overhead for setting up and tearing down the respective connections (and, depending on the QoS system, reservations as well). HTTP/1.1 [Fie+99] already improved on this situation by reusing connections to transfer multiple resources via its Keep-Alive and pipelining mechanisms. Still, modern web browsers open a whole host of connections for each web page loaded in order to parallelize requests and prevent being stuck on slow or stalled resources. With current web applications being distributed via content delivery networks and composed of all kinds of services from different vendors (data providers, script libraries, advertisement networks etc.), these parallel connections may span a large number of target hosts.

The newly standardized HTTP/2 [BPT15] further expands the capabilities of the protocol to include multiplexing support within one TCP connection. While this could, in theory, make it unnecessary in the future to open multiple connections to a single host to benefit from parallel transfer, it is not widely used yet and cannot be relied upon. However, as websites and applications will continue to draw on resources from multiple servers, web browsers will still open many parallel connections to speed up the transfer.

From the QoS system's point of view HTTP has the following properties:

Short-lived connections Many connections in a typical HTTP session are short lived. They are set up, transfer a resource and are torn down. Even with extensions like Keep-Alive, connection lifetimes are in the order of seconds with longer periods of no data transfers taking place.

Variable number of connections Modern browsers open multiple connections to parallelize data transfer even from one communication partner. However, this is only a performance improvement measure. HTTP will work just as well over a single connection or – in the case of multiple data sources for one web application – a set of connections opened sequentially.

1-to-n connection setup When pulling in services from all kinds of vendors, clients need to connect to a large set of different target hosts. This implies diverging transport paths which might at some points only accidentally share the same transmission resources.

Video streaming Real-time applications like video streaming are one of the main use cases for QoS systems. They require high volume, fixed rate transfers without interruptions. As most Internet providers do not offer QoS capabilities to their customers, platforms like YouTube, Netflix and Amazon Instant Video implemented measures to cope with varying network speeds. Their respective players buffer video content (increasing the latency) or switch to different video quality profiles (and thereby change the bandwidth requirements). This switching can occur automatically within an active stream.

The actual bit rate used by a modern video codec depends on many different factors: resolution, complexity of the image content, movement/change rate, quality settings, and encoder profile (constant or variable bit rate settings). Although this makes it hard, if not impossible, to predict the rate used at any given time, codecs are normally able to estimate maximum rates.[c]

Video streaming as a prime example of a QoS application has the following properties:

Constant, high data rate Video streaming, even at relatively low resolutions, uses a high amount of bandwidth at a relatively predictable, constant rate.

Bandwidth flexibility Video codecs are able to trade quality for bandwidth by changing resolution, frame rate, and image or sound quality. These adaptations are typically not gradual, but in distinct steps (e.g. by switching from high definition to standard definition video).

Moderately resilient to jitter In order to provide a smooth display of a video stream, the transmission has to meet certain timing requirements. However, as a streaming service is not interactive, buffering techniques relax the real-time requirements to within the size of the buffer.

Increased resilience to packet loss A video stream is resilient to packet loss to a certain extent. Modern video codecs employ concealment techniques to hide decoding errors from the user. Depending on the application, even if errors become visible, they can be tolerated by the user.

Multicast capability If there is a clear sender-viewer-relationship without any feedback in a video streaming application, transmissions can be multicasted to multiple receivers. From the QoS system's point-of-view, this extends the classic 1:1 relation of a reservation to include 1:n relations. Sender-based adaptation schemes, as mentioned above, that may rely on receiver feedback about the network state are no longer possible. Either adaptation takes place in the network using intelligent network nodes, or the sender offers multiple versions of the same stream to different subscribers (e.g. on different multicast addresses).

[c] For fixed rate codecs this problem is eliminated in total. The codec uses a predefined rate and lowers the image quality where necessary.

Video Conferencing A more demanding QoS use case is a video conference. It inherits all the properties and issues of the Video Streaming scenario and adds strong real-time requirements on top. Video conferencing is an interactive service: two or more people talk to each other in real-time. This requires a very low round-trip delay of less than approximately 400 ms. Beyond that delay, the typical user will have issues keeping a conversation going as people tend to cut each other off and start speaking at the same time. UTMS and LTE acknowledge this fact by limiting the end-to-end delay of their voice traffic classes to 400 ms.

QoS-wise, the Video Conferencing use case has the following properties:

Complex connection hierarchy Depending on the protocol used, a video conference may comprise several interdependent streams: two uni-directional video streams, the appropriate audio streams and, depending on the protocol in question, some kind of control connection. In this setup, not all streams are equally important. If, for example, the control connection breaks, the application might assume a break of the call. Therefore, this connection is a dependency for all other streams. Likewise, users might prefer clear audio over a video signal, as most human communication can be done orally, while a video-only connection is of lower value. This can be viewed as a dependency of the video on its respective audio stream.

High susceptibility to jitter While the Video Streaming scenario can combat jitter by simply buffering for some time in order to smooth out any interruptions in the transmission, video conferencing does not have that option. Each buffer introduces a delay. With video conferencing being a mainly delay-dominated application, additional buffering is necessarily very limited.

High resilience to packet loss Video streaming – depending on the service, e.g. in a digital TV setting – might need to deliver a rather high image quality. This limits the amount of packet loss a user is willing to accept. Video conferencing, on the other hand, allows for a bit more room for error. User will appreciate a clear audio signal and a high-resolution, error free video stream, but communication is still possible even when the visual signal severely deteriorates. A QoS system could use this as room for optimization in a complex environment with several competing services and limited resources.

1.2.2.2. System requirements

With respect to the environment constraints and the use cases, the system needs to support the following set of requirements:

Efficient resource allocation In systems where transmission capacity is at a premium, it needs to be used as efficiently as possible. Efficiency can mean different things on different layers of a transmission system. The physical layer might prefer modulation and coding schemes using as little physical resources as possible for each transmitted symbol. MAC and network layers, on the other hand, might try to minimize protocol overhead (e.g. by employing techniques like header compression).

From the point-of-view of the QoS system, efficiency is in the use of resources to provide a desired transmission behavior (e.g. by assigning transmission resources to specific reservations). Especially when the desired exceeds the available capacity, an efficient QoS system needs to ensure that the overall user satisfaction is maximized. This can imply suspending or removing less important transmissions and requiring end systems to adapt. It specifically implies ensuring, that each assigned transmission resource can be fully used by the intended traffic. No reservation should be assigned resources when any of its relations are not satisfied.

Minimum number of interactions This requirement actually stems from two different network characteristics: first, the reservation process should not have a noticeable impact on the already limited amount of resources. Second, the high transmission delay present in satellite networks presents a challenge to message exchanges involving too many steps. While the setup time of a longer running path might be negligible, reaction times to network events are not. The desired system therefore should minimize the amount of interactions necessary in any part of the protocol.

Adaptation to changing environment In a mobile environment in general, and in mobile satellite communication in particular, we cannot consider the link as a static resource. Variations in the conditions of the physical channel due to changing weather, shadowing effects, or interference, need to be addressed by the QoS system to operate successfully.

Flexibility/Expressiveness General-purpose communication systems call for a general-purpose QoS solution. The QoS layer must not limit the types of relationships expressible to a preconceived set of use cases.

Compatibility The Internet is too large and diverse to expect a complete adoption of a QoS system within a short time frame (if at all). Therefore, the desired system should be able to connect to legacy networks while still providing QoS services on capable parts of the path. This should include cases where one of the end nodes does not implement the QoS system.

Robustness Any transmission necessary to setup and manage communication paths should be robust in the face of message loss or duplication.

1.3. KASYMOSA

The research for this dissertation has been conducted in the context of the KASYMOSA satellite communication system [Wol+13; Brü+16]. The requirements of this project therefore influenced the assumptions and design decisions of this thesis.

KASYMOSA is a mobile satellite communication system for disaster scenarios. The goal of the project was to develop a full disaster communication stack from the application layer down to the antenna. The following environment assumptions were key to the system design:

Highly mobile terminals The KASYMOSA terminals are highly mobile (e.g. placed on cars). This influences the whole communication stack from the need for a tracking antenna to the variable availability of communication resources for the application. As a result of the mobility, the QoS system has to cope with highly variable link capacities due to rapidly changing environment conditions.

Communication infrastructure for rescue and relief operations With the main focus being on disaster scenarios, the system needs to be able to support the demands of rescue and relief operations. Those typically include prioritized communication flows, varying demand, and the need for an infrastructure independent of local ground-based systems, as those are most likely damaged by the disaster. As a main result of this requirement, KASYMOSA is a hub-less communication system without central control over resource allocation.

Geostationary orbit The KASYMOSA system is targeted at satellites in a geostationary orbit. Due to the nature of this orbit, with a height of approximately 36,000 km above the surface of the earth, the system incurs a transmission delay of 240 ms in one direction (time of flight for 72,000 km to the satellite and back down, at the speed of light). Together with processing delays in the stack, this can lead to a round-trip delay in excess of 1000 ms.

In order to offer service guarantees, e.g. for phone calls, emergency messages etc. the system includes a Quality-of-Service component based on the work presented in this thesis. Within the KASYMOSA system, the component interacts with the following parts:

Application The QoS system offers service guarantees to applications. These guarantees are based on demands posed by the application and are fulfilled by correctly configuring the service providers inside the system. Service guarantees in KASYMOSA are guarantees offered by the network service. They are expressed as the standard parameters *data rate*, *transmission delay*, *packet error rate*, and *jitter*. KASYMOSA expects the application to be able to provide its requirements based on those parameters, and aims to guarantee them for as long as the necessary resources are needed and available.

MAC layer Being situated in the network layer, the QoS system interacts with the KASYMOSA MAC layer to fulfill application requirements. The MAC layer is able to distribute communication resources based on the requirements given by the QoS system. To achieve this, the QoS system aggregates the flow-level requirements into DiffServ-like class requirements. The MAC layer in turn reserves the necessary resources and offers an interface to transmit traffic according to the requested transmission parameters.

The goal of the QoS subsystem is the provision of end-to-end service guarantees in a changing environment. Due to the variability present in the communication environment, the QoS implementations on layers 1 to 3 are tightly integrated with each other. The physical layer provides different modulation and coding schemes for different service requirements. The MAC layer implements distributed resource allocation without the need for a central hub. Layer 3 QoS ties everything together. By integrating application requirements signaled by the end-systems into the QoS stack, the network layer provides the input necessary to control the actual delivery of specific service levels over the satellite.

Use-case specific applications providing situational awareness to rescuers complete the KASYMOSA-picture. Fully developed, the system should provide a solution to take to a disaster area, be switched on, and have it automatically provide the necessary service to ensure a successful relief operation.

1.4. Remainder of this work

The remainder of this work is organized as follows: chapter 2 discusses existing QoS solutions that influenced the design in one way or another. This encompasses abstract QoS models like IntServ (section 2.2.1), as well as concrete protocol designs like NSIS (section 2.3.2). The KASYMOSA QoS design is discussed in detail in chapters 3 and 4. The design consists of the QoS model as a framework for expressing requirements and relations, as well as a signaling protocol to transport the relevant information through the network. Chapter 5 presents the evaluation of the system performance against existing solutions. Finally, chapter 6 concludes the work and presents potential directions for future research.

2. State of the art

The following chapter presents the state-of-the-art of Quality-of-Service. As a research area, as well as a practical service, QoS is older than IP, even older than computer networks in general. Wherever resources do not meet demands, similar QoS techniques are employed: whether it is as simple as table reservations in a restaurant or as complex as the capacity planning in water or electricity distribution networks.

In the narrower sense of QoS in IP networks, the research area starts right at the initial specification of the Internet Protocol with a classification model based on the Type-of-Service header field. From there, it has been evolving ever more complex models such as IntServ or DiffServ.

2.1. Classification

For clarity purposes, this work distinguishes between *QoS models* and *QoS protocols*. QoS modeling is concerned with the description of assumptions, guarantees and limitations of an approach. Models are not implementations for specific systems, although they may implicitly or explicitly make assumptions that focus on specific environments. QoS protocols, on the other hand, implement QoS models. They define the communication between different entities in the system, as well as necessary messages and parameter encodings.

Systems cannot always be clearly classified into one or the other of the two categories. QoS signaling is not necessarily orthogonal to the underlying model. Approaches such as RSVP show a clear relation to the underlying IntServ model. By classifying an approach as one or the other, this thesis merely puts an emphasis more on the model or the protocol side of things.

2.2. QoS models

There are two main contenders in the domain of QoS modeling in IP networks: *Integrated Services (IntServ)* [BCS94], a reservation-based, micro-flow oriented model with support for guaranteed services, and *Differentiated Services (DiffServ)* [Nic+98; Bla+98], a traffic-class-oriented model with the ability to scale to large networks. The systems are not necessarily exclusive, but can be used in conjunction to provide end-to-end QoS for users. IntServ, with its clear focus reservations, is used towards the network edge to obtain information about the current user requirements. DiffServ, with its ability to scale, controls the resources in the core network.

2.2.1. Integrated Services

Integrated Services is first and foremost a way of thinking about service guarantees in IP networks. The specification [BCS94] is just as much a standard for designing QoS systems, as it is a collection of ideas. It does include the *Resource Reservation Protocol (RSVP)* (see section 2.3.1) as a reference implementation of parts of the design, but that is not its main focus. The principles and models presented in IntServ have been implemented to varying degrees in other QoS protocols like *Next Steps in Signaling (NSIS)* (see section 2.3.2). RSVP, on the other hand, has been extended to support signaling of more class-like reservations [FAV08], which are not part of the original IntServ specification.

IntServ introduces the notion of a *resource reservation*: an explicit request for specific transmission parameter guarantees, which is either granted or rejected by the *admission control*. It fundamentally shifts the service model of the Internet from a simple, stateless, packet-switched network towards a stateful, circuit-switched approach. While the actual data transport takes place under a packet-switched regime, the necessary resources are set up, managed and torn down like in a circuit-switched network. The system installs state in each network node along the path to enforce the agreed-upon transmission parameters. This is the main issue of the Integrated Services approach: its lack of scalability to large networks due to an explosion of the amount of state in the network core. These issues led to the development of DiffServ (see section 2.2.2).

The IntServ architecture offers four different QoS models: *Best Effort*, *Guaranteed Service*, *Predictive Service*, and *Controlled Load*[Wro97].

Best Effort is the standard mode of operation of the Internet Protocol. Packets are forwarded whenever a network node is able to do so. Congestion in a node leads to packets being dropped. Higher layer protocols are expected to implement mechanisms to cope with varying throughput, delay and packet loss rates.

IntServ goes slightly beyond existing Best Effort mechanisms by proposing different traffic classes as indicators for the relative delay sensitivity of transmitted traffic. Network nodes could use this information, for example, to reorder their transmission queues or adaptively discard messages in overload situations. The authors give no indication as to how an actual implementation of such a service might look or behave.

Guaranteed Service offers the strictest QoS regime. Applications running under this model receive at least the guaranteed service quality (e.g. minimum throughput and maximum delay). If the system cannot guarantee the resources necessary, it rejects the path request.

Predictive Service eases the service requirements to be "fairly reliable" ([BCS94], section 3.1.1). Mainly concerned with a delay bound, applications under the Predictive Service model can expect their delay bounds to be met most of the time, with the occasional infraction occurring. They are expected to calculate their requested delay bounds based on "properly conservative predictions about the behavior of other flows". The standard does not further specify these statements, deferring to the actual application model for details.

Predictive Service is supposed to be less demanding in terms of network resources. Depending on the actual application model, the network can overbook links to a certain degree, allowing more simultaneous requests to succeed.

Controlled Load provides a further "soft" reservation model for IntServ. Not present in the original specification, it was added as a mechanism to provide the service quality of an unloaded network. The authors of [Wro97] specify this as:

- a very high percentage of successfully transmitted packets, with error rate closely resembling the basic error rates of the unloaded medium, and
- an end-to-end delay not much higher than the respective delay of an unloaded network.

The system uses admission control to ensure these parameters even in the case of a highly loaded network. Whereas Predictive Service is just concerned with the delay, Controlled Load additionally requires a certain throughput to be delivered. Network nodes therefore need to be much more conservative, when overbooking their transmission resources.

The IntServ standard contains a collection of ideas not explored in its original implementation. It discusses the idea of rate-adaptive application, able to modify their output data rate based on information they receive from the network. According to the standard, such an application could either be notified implicitly through dropped packets, or explicitly via some kind of control message. This feedback idea is explored further in subsequent approaches like MoSaKa (see section 2.3.3 for further details). It is also part of this work in the form of a suspend/resume mechanism for actively managing paths by the network.

Some rate-adaptive applications like video streams could also be managed by removing less important traffic from a stream first. Modern video codecs output data streams which consist of different parts with varying importance (e.g. an I-Frame in an MPEG video stream is always more important to the overall user experience than a P- or B-Frame). RFC 1633 proposes to mark “expendable” packets in a reserved stream as “preemtable” and let QoS routers drop them first, in case they cannot meet the required service quality. Again, this idea is not further explored in the standard, and is left to other research projects, such as the NOJA multimedia system [Eic08].

IntServ also discusses the idea of a load-based QoS routing. Conventionally, routing is solely based on local information about the structure of the network and the desired destination of the packet. This leads to very stable forwarding decisions where every packet of a stream takes the same path to the target[a].

In a load-based QoS routing system, routers can take the current load situation (possibly along the full path) into account, when making a forwarding decision. Conceivably, a data stream could switch its forwarding path for every packet, theoretically increasing the overall service quality. However, such an approach leads to increased jitter, which hurts the performance of protocols relying on precise timing measurements (e.g. TCP) and is therefore impractical. The authors of RFC 1633 argue that a system like IntServ would be able to take a load-based routing decision at reservation time, reaping most of the benefits, while avoiding the inherent instability. A similar idea is explored in [Vol16] as an approach for a Future Internet.

The original IntServ RFC specifies a first draft of the Resource Reservation Protocol (RSVP). This (incomplete) implementation of the ideas laid out in the Integrated Services architecture is further described in section 2.3.1.

2.2.2. Differentiated Services (DiffServ)

Deploying IntServ on an Internet-level scale quickly highlights a severe issue: forwarding nodes in the network core are part of a huge set of reserved paths. They therefore amass a prohibitively large amount of reservation state and are quickly overloaded. In order to address this problem, the Differentiated Services (DiffServ) architecture [Nic+98; Bla+98] was designed with the explicit goal of avoiding per-flow states in the network

[a] An exception to this are trunked links, where multiple independent paths are combined to form a virtual route, typically to increase capacity. In such a setup, packets of the same stream might be forwarded along different physical paths. Trunking usually occurs on the MAC layer, but can also be implemented higher up (e.g. for multi-homed networks connecting to different ISPs)

core.

DiffServ extends simple priority models as specified by the IP Type of Service (ToS) field [Alm92]. The architecture introduces the notion of a *DiffServ Codepoint* (DSCP), a six bit wide marker encoding the desired forwarding behavior for a packet. The DSCP is stored in the Differentiated Services field, which replaces the older ToS field in the IP header. The specification mandates that QoS-relevant forwarding decisions are to be made based only on the DSCP value, without taking any other parts of the packet into account. Therefore, forwarding nodes in a DiffServ core network do not need complex packet classifiers which would hinder scalability.

Service quality in DiffServ is realized by applying a specific per-hop-behavior (PHB) when forwarding packets belonging to a behavior aggregate. Such an aggregate is marked by a specific DSCP value in the packets' header. The architecture leaves it to the operators to define the semantics of a specific DSCP. Nodes that share a common view on DSCP values form a DS domain. DiffServ does not deal with micro-flows like IntServ. It instead operates on traffic classes, which greatly lowers the amount of state kept in an individual node. The exact specifications for the different traffic classes are not part of the DiffServ specification, but rather subject to Service Level Agreements (SLAs) between network operators. These SLAs are only enforced at the border of a domain by DS boundary nodes. The boundary nodes know the mapping between different DSCP spaces of neighboring domains and have sufficient information about the details of the QoS connected to a specific DSCP value. That way, DS interior nodes, forming the core of a domain, can be implemented much more efficiently. They rely solely on the packet classification, as specified by the boundary nodes, for their forwarding decision. This significantly reduces the computational complexity of the routing and allows the architecture to scale to Internet-level demands.

DiffServ offers a consistent QoS view on the network to end systems. No information about the details of DSCP-mapping inside different domains is needed at the network edge. However, it does not mandate any way to exchange the necessary QoS information at the operator level. The information might be transferred manually (e.g. through contracts between peering networks) or automatically (e.g. by implementing RMD as described in section 2.2.2.1).

The DiffServ model distinguishes between three basic Quality-of-Service classes: Assured Forwarding (AF), Expedited Forwarding (EF) and Best Effort (BE).

Expedited Forwarding [Dav+02] is a building block for a low-latency, low-loss and low-jitter edge-to-edge service within a domain. This rather simple QoS model handles all EF traffic within a single queue. This queue is forwarded with a given minimum rate and protected against other traffic on the same node. Ingress routers limit the amount of EF traffic entering a domain to the minimum forwarding rate provided by the domain, preventing queue buildup – and therefore increased latency – on interior routers. The exact determination of the forwarding rates on individual nodes is not part of the specification. This functionality could be provided via external signaling means like RMD 2.2.2.1.

Assured Forwarding [Hei+99] is more complex than EF. It provides multiple queues for different types of traffic, each with their individual parameters like queue size, priorities, scheduling and excess handling strategies. The DSCP value is structured, containing queue selector and excess treatment information. Ingress routers are therefore able to finely tune the forwarding of different types of traffic, and build edge-to-edge services simply by selecting different DSCP values.

As its name implies, Best Effort traffic is forwarded with whatever resources are available at a node. This includes totally stopping the forwarding, if the other traffic classes fully occupy the current capacity. Best Effort represents the standard forwarding behavior of Internet nodes without any QoS.

DiffServ targets large-scale networks like the Internet. For performance reasons it sacrifices isolation granularity for less state in the core network. For a severely constrained environment like KASYMOSA QoS, this level of scalability is not necessary. However, the lack of explicit reservations in the original DiffServ model prevents the implementation of a resource management as intended in this thesis.

2.2.2.1. Resource Management in DiffServ (RMD)

DiffServ does not mandate any mechanism to set up service level agreements at the domain boundaries. It merely specifies that these have to exist in order to ensure end-to-end QoS for the user. One way to automatically implement and manage those SLAs is Resource Management in DiffServ (RMD) [Wes+02]. RMD introduces the reservation of resources and admission control into the DiffServ architecture.

According to [Wes+02], RMD pursues two main goals: keep DiffServ scalable by limiting complex, per-flow reservation state to a small set of nodes (called “edge nodes”), while still being able to guarantee end-to-end QoS by associating flows with reserved resources. It achieves those goals by still reserving resources for all flows on each relevant node, but aggregating the state information towards interior nodes into traffic class reservations. This allows each flow to be accounted for by a specific reservation (namely the corresponding part of its traffic class on each router), while still keeping the state stored on interior nodes small by minimizing the number of traffic classes.

RMD defines two signaling protocol types: the more complex Per Domain Reservation protocol (PDR), which is responsible for the signaling of detailed reservations on the edges of a domain, and the simpler Per Hop Reservation protocol (PHR) used inside a domain along each hop of a path.

RMD is protocol-agnostic. On the PDR level, protocols such as RSVP or an appropriate model for NSIS can be used. The architecture places a set of requirements on the PDR (e.g. the support of admission control and maintenance of a per-flow state in the edge nodes), but does not define its detailed operation. The specification states that new protocols are to be defined on the PHR level. These are supposed to be in one of two classes: reservation-based PHRs, which install reservation state per PHB on each internal router, and measurement-based admission control PHRs, which rely on the measurement of existing traffic flows through a node for their admission decision. The latter do not install any state on the node. The specification does not indicate how the system distinguishes between different types of traffic (e.g. reserved and unreserved traffic or different priorities) in order to support preemption mechanisms. One of the protocols specified to the PHR plane is Resource Management in Diffserv On DemAnd (RODA) PHR [Wes+03], whose core ideas later migrated into the RMD QoS model of NSIS.

The RMD-QOSM defines RMD in terms of an NSIS QoS model. It specifies the necessary messages and their respective interpretation to actually control DiffServ classes via NSIS signaling. The intended operation is the automation of QoS changes at the domain boundary, depending on current requirements. The NSIS operational model is described in more detail in section 2.3.2.

2.2.3. Component Quality Modeling Language (CQML)

The Component Quality Modeling Language (CQML, [Aag01]) is a system for modeling QoS requirements. In itself, it does not represent a specific model like IntServ or DiffServ, but rather provides the tools to express QoS parameters and their relationships.

Through its tight integration with the Unified Modeling Language (UML, [UML]), CQML is focused on specifying QoS models in all parts of the software design process. Individual components can offer and require certain types of QoS parameters represented by CQML models. While not being concerned with QoS signaling and provision in a network, the approach nevertheless contains modeling ideas worth investigating.

CQML introduces the idea of *Predefined adaptation*, i.e. adaptation of QoS parameters without interaction with the requesting entity. The system assumes, that not all possible values from the value space of a QoS parameter are actually useful in the context of a given use case. If a certain application, for example, supports a high and low bandwidth profile, it is usually not beneficial to allocate any amount of bandwidth between those two. This allocation would not provide enough for the high bandwidth transmission and waste resources for the low bandwidth case. CQML uses the Object Constraint

Language (OCL) to guide the adaptation process. In OCL, boolean expressions define invariants over the parameter value, which always have to evaluate to *true*. Network nodes can use these invariants to guide the resource allocation process.

The system also introduces the notion of *composite QoS parameters*. A parameter composite is formed by introducing dependencies between individual QoS parameters. Again, OCL expressions define the behavior of the parameters under different compositions. CQML introduces three different composition types: sequential (one QoS-capable component using the services of another), parallel-or (a component using one of two other components. The latter are said to be in a parallel-or composition) and parallel-and (a component using the services of two other, but unrelated components, at the same time). For each composition type, different QoS parameters may behave differently. The author gives the example of a composable startup time parameter: for two sequential components, the resulting startup time is the sum of the two constituents. In a parallel-or composition, it is the minimum of the two startup times (whichever dependency is active first provides the service). Finally, in a parallel-and composition, the resultant time is the maximum startup time of all related resources (the service is available when all constituents are active).

CQML is focused on the design of software components and provides a framework for developing a QoS model. It is not an implementation of a working system. However, its notion of QoS invariants modeled as expressions over QoS parameters inspired parts of the research described in this thesis.

2.2.4. HTTP/2 Stream Dependencies

The Hypertext Transfer Protocol Version 2 (HTTP/2 [PR15]) is the next generation transfer protocol for the World Wide Web. Just like earlier incarnations HTTP/2 is a Request-Response-based system for accessing uniquely identified resources on a server. Modern web pages and applications usually comprise dozens, if not hundreds of individual resources like HTML pages, images, stylesheets and program code in the form of JavaScript files. These additional resources are referenced from the content initially accessed by the user and are – depending on their type and the type of their relation to the entry point resource – automatically loaded by a user agent.

HTTP/2 is much more focused on the transport of a high number of resources to a single client, than its predecessors were. To do so efficiently, the protocol supports multiplexing different streams over a single TCP connection. The system offers clients a form of QoS through a stream priority interface to indicate in which order they wish to receive a set of resources. Clients could use this system to prioritize resources that currently block content rendering to the user (e.g. CSS stylesheets), over ones that can be filled in later (e.g. images contained in a web site).

In order to implement resource prioritization, HTTP/2 implements a simple stream dependency system. A stream can depend on another currently active stream, by making the dependency its parent stream. To the multiplexing algorithm, this indicates a preference for the parent, when allocating transmission resources. A stream can be referenced by multiple dependent streams, effectively increasing its priority in the overall stream tree. The protocol also supports a specific kind of dependency called “exclusive”, which removes all existing dependees from a stream, puts the new stream in their stead and makes them children of this new node in the tree. This operation acts as a kind of “insert” into the tree, instead of just adding another child. However, the specification is unclear on the intended use case of this dependency type.

HTTP/2 as an application protocol is not directly applicable to a QoS layer. While its stream dependency system does provide some level of modeling power to supply relation information to a network, it is narrowly focused on the Web and can only serve as an inspiration for a more general QoS system.

2.3. Signaling protocols

Network QoS, as an inherently distributed system, has brought about its share of signaling protocols to synchronize state over different nodes. The two prime contenders are the older Resource Reservation Protocol and Next Steps in Signaling, as standardized approaches for general QoS use cases. Additionally this section takes a look at the MoSaKa signaling protocol, the direct precursor of the work presented in this thesis.

2.3.1. Resource Reservation Protocol (RSVP)

The *Resource Reservation Protocol (RSVP)*, as initially described in [Zha+93] and expanded as a proposed Internet standard in [Bra+97], is the original signalization protocol of the Integrated Services architecture. It provides the necessary protocol operations to signal state changes for flow reservations, that form the basis of IntServ.

One of the central design goals of RSVP was its suitability for signaling large IP multicast distributions. In multicast transmissions, the sender does not necessarily (or even usually) know the receivers. Keeping this in mind, RSVP was designed as a receiver-initiated protocol. While it can efficiently be used as a signaling protocol for unicast streams, receiver-initiated operation especially shines in the multicast case. After a host joins a multicast group, it sends an initiation message along the reverse distribution path of the group towards the source. When the message is processed by a router along the way, which already is connected to the distribution tree and has the appropriate resources reserved, instead of passing it on, this router completes the reservation process by sending the appropriate message downstream. This way, RSVP messages only travel the minimum distance in the network before attaching to a multicast group.

The protocol operation is simplex, i.e. handling a single, unidirectional reservation in one signaling exchange. To describe the characteristics of such a reservation, RSVP uses a *flow descriptor*, a data structure consisting of a *flowspec* and a *filter spec* object. The filter spec contains information necessary to classify traffic as belonging to a specific reservation. The underlying Internet Protocol does not define the context of a flow or a path, but considers the packet as the basic unit of information. In order to build a path on top of IP, each relevant network node needs a way to recognize packets belonging to the same flow, e.g. by source and destination IP addresses, transport layer protocol and source and destination port (where applicable).

The flowspec represents the actual QoS parameters for a specific path. RSVP itself considers the contents of the flowspec as opaque. Their interpretation is subject to the actual reservation model employed by the network.

The protocol was designed as an extensible protocol. One interesting extension, which found its way into NSIS QoS-NSLP (see section 2.3.2.2), is the concept of a *Preemption Priority* and a *Defending Priority* as defined in [Her01]. Instead of serving flows in a "first come, first admitted" manner, or reordering them by a single priority value, this model relies on two values. A flow is admitted, if its preemption priority is higher than the defending priorities of existing flows (existing flows with a lower defending priority are preempted until enough resources are acquired). Once it has be admitted, a flow defends its position with the defending priority. The system can be tuned to be either more stable (by having high defending and low preemption priorities) or more reactive to new reservations (by bringing those values closer together). The single-priority behavior can be modeled by either leaving out this parameter or setting preemption and defending priority to the same value.

RSVP falls short in several respects for the environment envisioned in this work. By design as a simplex protocol, it cannot easily, if at all, bind multiple paths to a more complex reservation structure. As the protocol is always receiver-initiated, a simple reservation setup for a TCP connection already presents a critical problem: each participating system needs to reserve its own receiving flow. Which endpoint would signal the relationship between both paths and thereby "own" the overall structure is unclear. While these problems are far from impossible to solve, no solution exists so far. RSVP also only provides only a rudimentary feedback mechanism for adaptation scenarios. The network can actively tear down reservations in case it can no longer guarantee them. However, it provides no way to enable them again once the capacity is available again, short of the end systems periodically requesting the necessary resources.

2.3.2. Next Steps in Signaling (NSIS)

The quest to establish a flexible system for transmitting signaling information in the Internet led to a whole series of research and specifications that can be (and often is) summarized under the name *Next Steps in Signaling* (NSIS) [Han+05] . NSIS is not so much a protocol, as a framework and collection of ideas to specify protocols for any kind of signaling need arising in a network. The analysis here will focus on the QoS-relevant parts.

To make the implementation of NSIS both simpler and more flexible, the system is divided into two layers: a transport layer implemented by an *NSIS Transport Layer Protocol (NTLP)* and a signaling layer formed by an *NSIS Signaling Layer Protocol (NSLP)*. Each of the layers can be implemented by different protocols for different tasks (in the case of NSLP the different implementations may even run in parallel on the same system). Supplementary specifications to NSIS provide standard solutions for specific layer tasks, which can be replaced, provided that the alternatives keep with the overall model of NSIS.

2.3.2.1. NTLP

Despite its name, the NSIS Transport Layer Protocol is not a ready-to-implement protocol, but rather a meta-specification, that actual NTLP specifications must adhere to. A possible standard implementation of an NTLP is the *General Internet Signaling Transport* protocol (GIST) [SH10; Tse+10], examined later in this section.

NTLP is responsible for forwarding signaling messages between *NSIS Entities* (NEs), that are in a *peer relationship* (i.e. networks nodes communicating directly via an NTLP link). NSIS Entities are network nodes that support the NSIS framework (in the most basic sense: nodes that are able to communicate via a given NTLP). NTLP is only concerned with forwarding messages between these entities. The actual content of the messages is outside the scope of the protocol. It is only concerned with the message type insofar as it needs to make a forwarding decision: a message can either be transparently forwarded to the next hop, or handed to a specific *signaling application* for local processing. NTLPs are especially required to forward unknown message types, for which no local application exists. This is particularly useful for integrating new NSLPs, without deploying them on every intermediate node of a network. It also benefits scenarios where only ingress and egress nodes of a network implement certain signaling functions (e.g. the integration of DiffServ domains into an IntServ path. In that use case, only the network boundaries need access to the full reservation structure).

The decoupling of the transport and signaling tasks shifts the responsibility for maintaining an end-to-end relationship to the NSLP. A specific NTLP link is not concerned with anything beyond two immediate neighbors. According to [Han+05] this approach was chosen to limit the effects of changes to an NTLP to a specific link or network, without affecting the Internet as a whole.

Even though NTLP is not concerned with any issues outside of the link scope, it acknowledges the existence of such issues by offering an optional notification mechanism. A specific NTLP implementation may choose to offer notifications about link changes to any interested signaling application (e.g. an IntServ-based QoS-NSLP, which is interested in detecting route changes to adapt reservations accordingly). The NTLP itself does not handle these notifications in any way, beyond what is necessary for message transport. It is up to the signaling application to trigger any further actions.

NTLP operates in a *path-coupled* mode, meaning that the signaling and data traffic of an NSIS path cross the same NEs. As NTLP does not assume that every node along an IP end-to-end path is an NE: there may be additional IP hops between two NSIS routers. The routes on these hops may even diverge between IP and NSIS traffic. It is the responsibility of the NEs, to configure these additional hops in such a way to as ensure the required reactions to the signaling requests (e.g. by setting up QoS reservations using local means).

One thing NTLP is explicitly *not*, is an actual protocol specification. It is a framework for designing one. An instance – so far the only one – of an NTLP is the General Internet Signaling Transport.

General Internet Signaling Transport protocol (GIST) With NTLP as described in [Han+05] only being an abstract specification of requirements for signaling transport protocols in the NSIS context, a standard implementation is needed for the most prevalent type of network today: the Internet. This role is filled by the General Internet Signaling Transport (GIST) [SH10].

One of the main goals of GIST is the reuse of existing protocols. The Internet protocol world has solutions for nearly every possible communication requirement. Reusing those tried and trusted approaches eases the implementation of GIST and thereby should make adoption faster.

In line with the NSIS communication model, GIST forms connections (*Message Associations* (MA))) between neighboring NSIS entities using an underlying transport protocol. While the specification allows for any kind of protocol, it currently mandates the use of the Transmission Control Protocol (TCP) and the User Datagram Protocol (UDP) for two different kinds of transmission modes:

Datagram mode In *datagram mode*, GIST forwards messages without prior message association setup. Signaling requests are forwarded in UDP packets. This mode severely limits the capabilities of GIST to a "fire-and-forget" forwarding fashion. One of the primary uses of the datagram mode is a special case called *Query mode*. This is used to discover the next NE along the path towards a specific signalization target. In query mode, a packet is sent towards the endpoint for which it is destined, instead of directly addressing the next NSIS node on the path. The packet has the IP Router Alert option header [Kat97; PJ99] set, indicating for every node along the IP path to examine its contents outside the normal forwarding path. The next NSIS node along the path intercepts this packet and may become the corresponding peer of the sending node by establishing an MA. This way GIST discovers unknown forwarding paths towards a target. The approach also allows GIST to cross NSIS-unaware IP routers by exploiting the normal IP forwarding process.

Connection mode This mode forms a message association between two neighboring NEs based on a TCP connection. It provides a whole set of additional features over datagram mode, such as the fragmentation of large messages, reuse of an MA for the transmission of several different signalization processes or security (using the Transport Layer Security protocol [DA99; DR06; DR08], version 1 or above). Connection mode forms long-term relationships between neighboring entities. While initially including more overhead for the setup of the connection (TCP handshake, possibly TLS handshake and other initialization routines), this overhead is spread out over the potentially unlimited number of signaling requests, that can be transported over the same link. The authors of [SH10] state, that "[there] may be any number of MAs between two GIST peers, although the usual case is zero or one", meaning that they expect two nodes to either have no relationship at all, or reuse one connection to handle all their signalization needs.

Connection mode via TCP does have the disadvantage of providing strict in-order delivery for what is basically a message-oriented protocol, where objects may have a specific lifetime. In lossy environments this could lead to urgent messages being blocked by the retransmission of already stale signaling on the same connection. To prevent this kind of problem, RFC 6084 [FDC11] optionally defines the use of the Stream Control Transmission Protocol (SCTP [Ste07]) as the underlying transport layer. SCTP offers multiple independent transmissions in one delivery-guaranteed channel, effectively disabling the strict (and unnecessary) order guarantee of TCP. It also provides capabilities like multihoming, where each endpoint in a connection may be reachable under multiple IP adresses. Together with Datagram Transport Layer Security [RM12] SCTP serves as a secure, reliable transport layer for GIST even in lossy environments.

The different transport modes can be mixed along the same signalization path. RFC 5971 [SH10] gives the example of a network using connection mode in the core, where long-running relationships between NEs are common. The system switches to datagram mode at the network edges, where mobile nodes with frequent path changes may make message association setup prohibitively expensive. The selection of the actual transmission mode is up to the GIST layer. From the point of view of the signaling application, there is no way to request one approach or the other.

How a GIST node makes the forwarding decision depends on its specific implementation. RFC 5971 specifies two *Message Routing Methods (MRM)* (modules that implement the forwarding decision making):

Path-coupled MRM The *path-coupled MRM* is the standard routing method for GIST. It forwards signaling concerned with a specific path along the nodes on this very path. This MRM is therefore most suited for NSLPs that inform or modify entities along the data path (e.g. by creating resource reservations).

Loose-end MRM This MRM “is used to discover GIST nodes with particular properties” along the routing path [SH10]. An example is the discovery of network-address-translation (NAT) nodes, which may require special handling to allow NSIS to work.

The current set of MRMs does not support the signaling of multicast requests. The authors of RFC 5971 claim that the GIST communication model could be easily extended to support such a requirement, provided that the multicast distribution points are NSIS-capable.

NTLPs provide a very flexible signaling transport suitable for a wide range of applications. As they are signaling-agnostic (i.e. they consider the actual signaling messages as opaque), they can be adapted to nearly any kind of environment. GIST (in particular with SCTP as the underlying signaling protocol) fulfills the requirements and environment conditions assumed in this thesis well, and could serve as KASYMOSA QoS’s transport layer. Its only disadvantage is its implementation complexity which stems stemming from the flexibility of the protocol. Section 4.3 goes into more detail regarding the use of GIST for the purposes of this work.

2.3.2.2. QoS-NSLP

The *NSLP for Quality-of-Service Signaling* (QoS-NSLP [MKM10]) provides a framework for signaling QoS-related information via NSIS. The overall design approach is similar to that used in RSVP: state management is done via soft states with periodic refresh. QoS-NSLP deviates from the set of features offered by RSVP by adding sender- and receiver-initiated reservations, bidirectional reservations, and reservations between arbitrary nodes (as opposed to only end-to-end reservations in RSVP). In line with NTLP, QoS-NSLP does not provide support for IP multicast.

In order to keep the QoS signaling process as flexible as possible, the protocol does not mandate a specific QoS model. All necessary information beyond the simple transfer of QoS signaling messages (e.g. admission control or the Resource Management Function) is left to the model. This allows the protocol to be flexible enough to implement anything from IntServ Controlled Load [KFS11] to DiffServ [Bad+10] signaling.

QoS-NLSP specifies four basic message types: RESERVE, QUERY, RESPONSE and NOTIFY. RESERVE and QUERY act as requesting messages, with RESERVE causing changes in the reservation state, whereas QUERY just obtains information. RESPONSE serves as a reaction to a previous request and NOTIFY implements asynchronous notifications. The protocol does not specify the actual content and meaning of the message body or their specific order in an exchange. These topics are left to implementing QoS models that are able to build on the given primitives to provide their service.

In order to identify a specific signaling flow in the network, QoS-NSLP uses the Session ID as given in GIST [SH10]. Building on these identifiers, sessions can be bound together using a BOUND-SESSION-ID protocol object. This allows models to implement unidirectional dependencies between different sessions. [MKM10], section 3.2.8 presents two examples of dependency relations, that can be modeled with bound sessions: the dependency of an end-to-end session on its corresponding aggregate session somewhere in the network (for session aggregation use cases) and the dependency between two sides of a bidirectional session (in which case both unidirectional sessions would include the other in a BOUND-SESSION-ID object). QoS-NSLP allows to further specify the type of session binding by a binding code. The specification includes a list of five atomic binding codes open to extension by the QoS model. The exact interpretation of a session binding is not mandated by QoS-NSLP.

As a flexible framework for building actual QoS signaling protocols, QoS-NSLP is non-opinionated about the actual reservation model. RSVP-style sender-initiated reservation is possible the same way as receiver-initiated or bi-directional reservations. The protocol specification even discusses – albeit in little detail – a stateless operation, which would supposedly operate without any reservation state at all. By specifying the message flow for different reservation models, QoS-NSLP provides a frame into which the actual implementation can just drop its messages and rely on the existing underlying mechanisms for delivery.

State installed in the network is handled a soft-state approach, just as RSVP does: the state information carries a lifetime and expires if not refreshed[b]. While this approach has the disadvantage of causing increased overhead due to periodic refreshes, it provides the huge benefit of a “self-cleaning” network. Especially in the mobile case, where routing paths may change without prior notice, making it hard or even impossible to transmit explicit tear-down messages, a soft-state approach ensures that eventually the left-over reservation state will be removed and the acquired resources freed. QoS-NSLP tries to minimize the overhead caused by periodic refreshes by providing a reduced refresh feature. Instead of transmitting the whole request again for refresh, the request initiator is allowed to transmit only the session ID to indicate an unchanged path.

[b] This seems to somewhat contradict the stateless operation mentioned before. QoS-NSLP is fundamentally a stateful protocol. What a stateless operation could look like and what it would actually be used for is not further specified.

QoS-NSLP QSPEC template While QoS-NSLP does not mandate a specific *Quality-of-Service model (QOSM)*, it acknowledges the fact that there are some recurring elements in each system. RFC 5975 [Ash+10] specifies an extensible message template on which specific QOSMs can be built.

The basic building block specified by the RFC is the eponymous *QSPEC*. It carries all information necessary for a specific QOSM to fulfill its function. According to [Ash+10], "QSPEC parameters provide a common language to be reused in several QOSMs". Each QSPEC can contain up to four distinct *QSPEC objects* to signal different intentions to the QOSM:

QoS Desired This object indicates the QoS parameters desired by the *NSIS QoS Initiator (QNI)* to successfully carry out a transmission.

QoS Available Using this object, the network can indicate which QoS parameter values might be available. This could allow an application to adapt to the currently available resources.

QoS Reserved When carrying out a reservation, the network indicates the resources actually reserved using the QoS Reserved object.

Minimum QoS In order to facilitate a kind of automatic adaptation of the QoS reservation, a QNI may include the Minimum QoS object. The parameters in this object represent the worst conditions under which a reservation can be considered successful. They might (and generally will) be worse, than the QoS Desired, but will allow the application to function. If Minimum QoS is not available, the reservation has to be rejected.

Each QSPEC object consists of a set of *QSPEC parameters* which encode the information necessary to drive the QOSM. RFC 5975 [Ash+10] defines a basic set of parameters useful to most of the envisioned use cases, leaving the option to extend that set where necessary.

Only one parameter is mandatory: the *Traffic Model(TMOD)*. This parameter describes the reservation in terms of a token bucket meter with a token rate and a bucket size, as well as additional parameters like peak rate or maximum packet size. A QNI has to include this parameter in its QSPEC and all QNEs along the path must interpret it.

All other QSPEC parameters are optional. However, the specification does provide a way for a QOSM to mark a parameter as mandatory (the M flag). Using this flag, the initiator can distinguish between parameters that are essential to a reservation, and informational parameters that should not cause a reservation to fail. Two other flags are used to complete the extension mechanism: the N flag (for "not supported") is set by a

QNE if it does not support the interpretation of an optional parameter. This allows the QNI to adapt its behavior, if necessary. The E flag (“error” flag) is used by the QNE to indicate that a parameter is supported but the required demands cannot be met. This allows the reservation process to continue, while still indicating that some restrictions will apply due to unmet optional requirements. These flags provide a standard mechanism to enable extensibility of the protocol, as it allows for QOSMs to include the optional parameters for better performance, while still being compatible with a more basic set of functions of a simpler model.

The additional parameters defined in the specification are divided into three groups: path constraints, such as latency and jitter, traffic handling directives like preemption and defending priorities, and traffic classifiers, e.g. a DiffServ Code Point. QOSMs can draw from this standard set to build their signaling model or extend them by the means described above.

A further extensibility mechanism is the ability to nest QSPECs in one another. A QOSM could define a simpler local QSPEC, just intended for use in one domain (similar to the local interpretation of DSCPs in DiffServ), and carry the initiator QSPEC along with the local signaling as a nested object. In deriving a local QSPEC, a QOSM must not violate the requirements of the initial request, but it is free to transform it into a parameter set more suitable for its needs (e.g. by aggregating it with an already existing local reservation of the same class). Upon leaving the local domain, a QOSM is able to restore the nested object and pass on the signaling as intended by the initiator.

An important aspect of the behavior of a QoS entity is the treatment of excess traffic. If a misbehaving sender exceeds its requested bandwidth, the network has to protect itself and other flows. The QSPEC template specifies an Excess Treatment parameter, indicating how this situation should be handled. Packets can be either dropped, shaped, re-marked as another (signaled) class, or classified as best effort and be transmitted along with the rest of the unreserved traffic. The template also specifies a special excess treatment parameter which effectively disables any policing and shaping by a network node. The network in this case must not in any way impede the traffic flow of a reservation under that scheme. The specification warns implementers to ensure that a node should only accept such a reservation if it can handle any amount of traffic a sender could possibly transmit (e.g. by taking incoming and outgoing link capacities into account).

The specification is unclear on whether multiple requests could be sent within a single signaling transfer. However, it clearly states that under no circumstances can QSPECs for two different link directions travel in the same message. This complicates signaling of path relationships significantly, as it makes it impossible to signal all reservations and their relations in one transaction-like step.

With the stated goal of being a flexible base for the specification of QoS models, the

QoS-NSLP QSPEC template provides a rich set of building blocks from which these can draw. This framework has been used to define various concrete QOSM for specific tasks, such as the QoS-NSLP for IntServ Controlled Load.

NSIS and QoS-NSLP could serve as a basis for KASYMOSA QoS to some extent. By using SCTP as a transport protocol for GIST, TCP's issues in long-delay, high-loss environments can be avoided. However, in order to effectively transmit multiple reservations along with their relations, an extension of the existing QoS-NSLP QSPEC template would be necessary. The implications this has for the rest of the protocol operation are beyond the scope of this work. While KASYMOSA QoS's path model is inspired by GIST's segmented overlay network, it does not use NSIS as its signaling framework, but leaves the adaptation of the protocol to future work.

2.3.3. MoSaKa Signaling

The MoSaKa signaling QoS system [Hei+10; DEB12] is a precursor to the work presented in this dissertation. Its main focus is on supporting efficient QoS handling in a long-delay satellite network by enabling the network to suspend and resume paths when necessary. The approach stems from the observation that link capacity in a mobile satellite communication system is highly volatile, while signaling round-trip times are long.

MoSaKa builds on the idea of network-based adaptation, as presented by preemption models in earlier works, and extends it to include the concept of a temporary suspension of a path. Instead of removing a reservation and requiring the initiator to poll the network for a future capacity change, paths are put into a *suspended* state. In this state, the network still retains control over the path's fate, immediately resuming it when the transmission capacity is sufficient again.

Allowing the network to suspend paths leads to reduced signaling overhead and faster reaction to changes in capacity. Especially the lower overhead is significant, as it is mostly saved in a situation where the network experiences congestion anyway. Having initiators poll for resources at this point only introduces additional traffic, ultimately making matters worse.

Similar to RSVP, Mosaka implements a single-roundtrip signaling process, based on UDP as the transport protocol. An initiator sends out a request towards the peer, which is routed on the intended path. MoSaKa-capable routers along the way intercept the request packets and process them. MoSaKa saves transmission overhead by signaling multiple requests in a single exchange. This allows the system to reserve multiple related paths (e.g. the two directions of a TCP connection) with one signaling exchange.

MoSaka allows both endpoints of a path to initiate the reservation process. While this makes limiting the number of requests for bidirectional reservations possible, it leads to the requirement of symmetric routing. RSVP, as a sender-initiated protocol, discovers the actual downstream path by means of the *PATH* message. The actual request is then transmitted from the receiver to the sender along that path. MoSaKa, on the other hand, transmits the reservation message from the initiator to the peer and back, reserving all requests on the second leg of the exchange. In this model, the transmission direction of the signaling message does not necessarily coincide with the direction of the data flow of a reservation. If such a system is deployed in a network with asymmetric routing, resources may be created on routers which are never crossed by the actual data path. As MoSaka is geared towards satellite systems with a single transmission link over the satellite, this issue does not occur in practice.

MoSaKa fulfills a number of the requirements presented in section 1.2.2.2. Taking an IntServ-like approach with per-flow reservations, the system provides isolation on the path level. With its one-roundtrip signaling process, MoSaKa limits the number of interactions at the initial path setup to the lowest possible level. The system also provides a way to adapt to a changing environment by suspending and resuming paths. This approach significantly lowers the amount of signaling required in case of a temporary resource shortage, as the network is responsible for suspending and resuming paths, and no expensive polling by clients is necessary. However, MoSaKa fails to correctly suspend paths that might be dependent on each other, still requiring the initiator to take action in this regard. MoSaKa also uses periodic refresh messages to manage the lifetime of a path, using additional transmission resources.

The Flexibility/Expressiveness requirement does not fully apply to the system. Its focus is more on the signaling side than on modeling QoS requirements. The model employed in MoSaKa is therefore a simple datarate-only model without any additional complexities. While this could be expanded in the future, no work in that direction has been done so far.

MoSaKa's signaling approach provides compatibility to existing network resources. Similar to the D-Mode of GIST, MoSaKa's signaling messages are always addressed to the remote endpoint of a path, relying on normal IP routing to discover the paths along the way. MoSaKa-capable routers intercept passing messages along the way, taking action as needed. This way, MoSaKa is able to cross legacy networks without interacting with them. MoSaKa does not provide means for the initiator to detect such a legacy system. The protocol assumes that the peer is MoSaKa-capable, as the signaling protocol calls for the messages to be reflected at the remote end. The system is therefore not capable of making partial reservations along a path to a legacy server. As MoSaKa relies on standard IP routing along an end-to-end path, it also does not support asymmetric routing (i.e. different up- and downstream paths) or multicast reservations.

The system employs standard techniques, such as initiator-based retransmissions and a soft-state approach for path lifetimes to cope with lossy networks. Requests are retransmitted and, if necessary de-duplicated, if they fail to be acknowledged. Paths have to be periodically renewed with explicit refresh messages and are garbage-collected, if those messages are absent. This makes the system robust against lost management requests like path deletion, but vulnerable to lost refresh messages.

Overall, MoSaKa, as a specialized protocol for the very environment this work is targeting, already fulfills a large part of the system requirements. This work, as a successor to the MoSaKa system, builds on the experiences of said project and extends it by a relation model. It modifies the state handling of its predecessor, to better integrate multicast reservations and extends its reservation model to accommodate interactions with the relation system.

3. The KASYMOSA QoS Architecture

The focus of this work is on the development of a QoS model suitable for unreliable, long-delay environments. The underlying question is how a flexible and expressive model can improve the overall service quality and help to better utilize the resources available. In line with the requirements described in section 1.2.2.2 the design of the QoS model rests on a set of assumptions:

Reservations are related Most of the reservations in the system do not stand alone, but are related to each other. An example is a reserved TCP connection for a file download: it consists of a high data rate flow path from the server to the client and a low data rate path in the opposite direction, carrying the TCP acknowledgements. Possible relations are discussed in more detail in section 3.3. Modeling these relations should provide the system with a more accurate idea of the best resource allocation in case of overload situations.

Round-trips are expensive Given the envisioned environment this system operates in, a round-trip along a path takes a significant amount of time. The system should therefore provide the network with enough information to make decisions autonomously, without consulting the initiator.

Capacity varies Operating in a highly variable environment, the system cannot be expected to provide stable link capacities under all circumstances. The reservation setup constantly needs to be adapted to currently available resources. However, changes in the environment are transient. Therefore, minimizing the delay to react to changes helps to better match capacity and demand in the system.

These assumptions influence both the service model and the transport system. The following chapter presents the design for a suitable QoS model which provides the expressive capabilities to describe reservations and their relationships. Resource allocation in a constrained environment is an instance of the *Knapsack* optimization problem. Section 3.4 will present the underlying theory, design appropriate value and cost functions, and show an optimization approach based on Mixed Integer Linear Programming to find the best allocation.

3.1. Definitions

The following parts of this chapter use a concise formal description of the model. This description relies on some definitions:

Path A *path* through a network is a sequence of network nodes (vertices) connected by unidirectional data flows (edges), through which packets travel en route from a sender to a receiver. The sequence of vertices includes the sender as the first and the receiver as the last element.

In the context of this work, paths include only nodes that are concerned with the provision of a specific requested service quality. If legacy nodes without QoS support are to be viewed as part of a path (e.g. when looking at the actual routing of packets), they are explicitly referred to.

Initiator The *initiator* of a QoS path is the network entity that sets up the path and therefore owns and controls it. In a path-coupled system, such as the one presented here, the initiator is an endpoint of the underlying network path. While a path might be reserved on behalf of another system (e.g. a node that is not QoS-capable), it passes the initiator nonetheless.

Peer The *peer* of a QoS path is the endpoint opposite of the initiator.

Path segment A path segment is a part of a path between two neighboring nodes. End-to-end QoS is achieved by the start node of each segment applying the appropriate forwarding decisions, shaping the traffic as a result.

Flow A flow is a set of packets to which the same QoS is applied.

Reservation A *reservation* p_i describes the QoS information for specific traffic at a router. Each reservation consists of two parts: a *Parameter* and a *Filter object*

The Filter object designates the traffic, to which a specific reservation applies (similar to the TSPEC object in RSVP). The Parameter object, on the other hand, describes the expected QoS parameters (e.g. data rate or maximum delay) for said reservation.

Each reservation corresponds to a specific segment of a QoS path. Different segments along a path may have different parameter sets, depending on the specifics of the forwarding path. This allows forwarding nodes to adapt reservations as necessary (e.g. to aggregate reservations into larger classes like RMD [BWK04] does).

Reservation set The *reservation set* Q represents the set of reservations currently known to a node, regardless of their state. The reservation set has a corresponding *index set* I containing the indices of all paths in Q. The resource allocation on a router is subject to capacity constraints on individual network interfaces N_i, each with their own corresponding set $Q_{N_i} \subseteq Q$ and $I_{N_i} \subseteq I$.

Reservation state The *reservation state* indicates whether a node currently guarantees the QoS parameters requested in the reservation.

The state of a reservation p_i is expressed by the state variable s_i $(i \in I)$:

$$s_i = \begin{cases} 1 & \text{if } p_i \text{ has the required resources assigned} \\ 0 & \text{otherwise} \end{cases}$$

If the state of a reservation is 0, then no guarantees whatsoever are given as to how (or whether at all) the associated flow is forwarded. This corresponds to the *best effort* service provided by standard IP. Due to the dynamic nature of the resource assignment, the reservation state is transient.

An end-to-end QoS path may rely on the state of more than one reservation along the path to provide the requested service. Due to the distributed nature of decision making in the network, the state of these reservations might differ from one another. It is the task of the QoS system, more specifically, the signaling process, to resolve this inconsistency.

Relation A *relation* is a function $r : \mathbb{B}^k \to \mathbb{B}$, with $\mathbb{B} = \{0, 1\}$, $k \in \mathbb{N}^+$, mapping a k-ary vector of reservation states to a truth value. This models application specific relationships between different flows, such as the two directions of a TCP connection which cannot exist without each other. If a relation evaluates to 1, it is said to be *fulfilled*. Relations are further described in section 3.3.

Each relation has a corresponding *related set* $F_r \subseteq Q$ of children.

Relation set The *relation set* R is the set of relations known to a node.

Resource allocation A *resource allocation* $a_i \in \mathbb{B}^k$ is a k-tuple of reservation states.

$$a_i = (s_{k-1}, \ldots, s_0) \text{ where } s_j \text{ is the state of the reservation } p_j \in Q$$

As such, a_i describes a specific assigment of resources to reservations.

A resource allocation does not necessarily satisfy all relations known to a node. However, barring contradictions in R, an optimal resource allocation should not violate any relations at all.

3.2. Reservations

Reservations – as the name implies – are the core of a reservation-based system. They define the QoS parameters supported and therefore the service-model offered. However, in the context of this work, the exact structure of the reservations actually plays a minor role. The main focus of KASYMOSA QoS is on the modeling of relations between reservations. As such the system is orthogonal to the actual reservation model used (with the exception of parameter interactions as described in section 3.3.3). The reservation model described here is therefore mainly driven by the requirements of KASYMOSA, the project in whose context this research was carried out. Other models with different parameter sets (e.g. NSIS QoS-NSLP's *QoS Desired* and *Minimum QoS* objects) might be possible.

Reservations are inherently local to a specific network node. KASYMOSA QoS provides end-to-end QoS by chaining a set of reservations along a path. The actual handling of a reservation is specific to an individual node and only concerns other network nodes as an event source for state transitions. The following sections therefore always refer to the QoS system of a single node only.

3.2.1. Parameters

At its core, KASYMOSA QoS is a soft-state Token Bucket Metering model [TW10], with support for delay limitation and a priority model. The Parameter object can contain the following information:

Data rate The data rate is the sustained rate with which this reservation forwards traffic through the network. It corresponds to the token rate of the Token Bucket model.

Bucket size The size of the bucket of the Token Bucket Meter. While this is not equal to the maximum burst size, it is indicative of the amount of burst data a reservation is expected to carry.

Excess Treatment If a flow exceeds its reservation, routers can treat the excess traffic by either dropping it immediately or queuing it as *best effort* traffic. Using this field, an application can indicate its preference for one of the two options.

Maximum Delay In the KASYMOSA satellite system, traffic can be forwarded on different physical layer transmission streams. Each stream provides distinct levels of error protection at specific transmission costs in terms of delay and physical layer resources. In order to assist the decision making when mapping reservations onto physical layer streams, applications can indicate the maximum delay their transmission can tolerate.

Priority KASYMOSA's target environment is a disaster area where communication mostly follows a hierarchical structure. Modeling this hierarchy is the goal of the priority model further described in section 3.2.2.

Emergency By setting the emergency flag, applications can indicate that a reservation should always preempt normal communication. Details of the emergency design are presented in section 3.2.2.

Lifetime KASYMOSA QoS is a soft-state protocol, where reservations have to be refreshed to be kept active. As the target environment is a highly mobile satellite system, initiators can come and go at any time without prior notice. In order to keep the network state up-to-date when initiators are not longer able to signal changes, each reservation carries a lifetime after which it has either been refreshed, or expires and is removed by the network.

Traffic belonging to a specific flow is identified by the parameters given in the Filter object. Being an IP-based system, KASYMOSA QoS provides the usual filter parameters, such as source and destination addresses and ports, protocol, and a type-of-service field. Additionally, the system supports reservation merging for use cases like Multicast transmissions. An application can specify one of the following *Collision Policies*:

Reject If the Filter object of a new reservation collides with an existing one (e.g. by referencing the same addresses without further distinguishing features), the new reservation is rejected. This is the standard behavior of the system.

Merge Exact If a new reservation carries exactly the same Filter and Parameter objects as an existing one, the two are merged into a single reservation. This collision policy allows the system to handle multicast applications without reserving additional upstream resources. The point of attachment to the multicast distribution tree for a reservation can simply merge the incoming request with the already existing tree and terminate the reservation process.

3.2.2. Priority model

Designing a priority model can be approached from two different perspectives: either priorities are viewed as simple indications of the user preference and may be weighed against each other in an overall solution, or they follow a more rigid model, where higher priorities always take precedence over lower ones. The former approach offers the system much more room for optimization when calculating a resource allocation, while the latter allows modeling services, such as emergency calls, that always should preempt normal communication.

KASYMOSA QoS's priority model is a hybrid of those two approaches. Using the *Emergency* flag of the Parameter object effectively divides the traffic into two classes: an emergency class, which always preempts the other traffic, and a non-emergency class, which can be preempted where necessary. Within those classes, the system supports a simple preference model with the *Priority* field. Applications can use this value to indicate their preferred solution (e.g. when choosing from multiple alternatives of a reservation). This enables the support for non-preemtable traffic, while still allowing a degree of freedom for reservations with more relaxed requirements.

3.2.3. Reservation state

A reservation in the system can have one of two principal states: *Online*, meaning that the required amount of transmission resources have been allocated at a node and data transport can take place, or *Offline*, i.e. no guaranteed transmission is possible.

Implementing and extending MoSaKa's path suspension mechanism, most offline reservations are actually suspended (i.e. temporarily not served due to limited resources). Whereas its predecessor distinguished between two suspension states (*local* and *remote*) and treated them differently, KASYMOSA QoS introduces a third (*blocked*) to handle relations correctly.

Local This state denotes that a reservation was suspended due to locally insufficient transmission resources. The system will try to re-acquire the necessary resources and serve the reservation again.

Remote Remotely suspended reservations are offline due to a restriction imposed on them from outside the control of a network node. The QoS system neither tries to acquire resources for these reservations, nor considers them as viable candidates for the resource optimization process, until the restriction is lifted.

Blocked KASYMOSA QoS introduces an additional suspended state for reservations blocked by a relation. Assuming, for example, two reservations excluding each other, the system will have to disable one of them, to satisfy all relation requirements (see section 3.3 for details on the relation model). A router will not acquire resources for a blocked reservation (because said resources would be wasted anyway), but will consider it as a potential candidate for fulfilling relations in the resource optimization process.

The full reservation state machine including all transitions is given in appendix A.

3.3. Relations

Relations between reservations are the core idea of this work. The following section introduces a language to model path relations in a flexible way, and uses it to model real-life relations as examples.

3.3.1. Relation Modeling

A relation is a function $r : \mathbb{B}^k \to \mathbb{B}$, expressing an invariant about the states of a subset of the reservation set Q. If a relation evaluates to 0, this indicates that the current state of the subset does not conform to the expectations set by the requester. If any reservation in the subset has resources assigned, they are therefore wasted, as they cannot be used as intended, due to application constraints. The term “application constraint” is deliberately used loosely here. The system should not limit the type of constraint to a predefined set of use cases, but rather support novel and, so far, unforeseen applications.

The relation language has to fulfill two requirements:

1. The language should be capable of expressing arbitrary mappings of allocations to a truth value. Supporting only a predetermined set of mappings would limit the system’s capability to operate in a wide variety of use cases.

2. The language should provide enough expressive power to model the specific use cases found in QoS relations succinctly. Research into possible use cases of the system, as presented in section 3.3.2, showed several recurrent model patterns. The system should be able to express those patterns in a concise manner, without compromising generality.

3.3.1.1. Truth tables

A standard tool for representing an arbitrary mapping from an n-tuple of binary values to a single truth value, is a *truth table* as presented in table 3.1[a]. Simply listing every possible combination of reservation states on the left side and their respective validity

[a] The table presents a shorthand form, where multiple relations share the same left-hand side of the table. Each column on the right-hand side can be viewed as an independent table.

s_{n-1}	...	s_0	r_m	...	r_0
0	...	0	r_m^0	...	r_0^0
$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$	$\vdots$
1	...	1	$r_m^{2^{n-1}}$	...	$r_0^{2^{n-1}}$

Table 3.1.: *A truth table mapping reservation states s_i to validity values r_j^k*

values r_j^k under each given relation r_j on the right, enables the requesting application to model any kind of path relation required.

In such a system, each relation could be represented as a single tuple of bits by interpreting the respective column in the truth table as a binary number. That approach, however, has two drawbacks:

1. Each relation is only concerned with a subset of all the reservation states. However, in order to correctly represent the respective column in the truth table, the relation would need information about every reservation in a node, even the ones it does not constrain in any way. The correct representation would also need to change as the reservation set Q changes from node to node (inserting and removing columns on the left side of the table).

2. Even for a moderate number of reservations, the tuple representing a relation becomes excessively large. If $n = |Q|$ is the number of reservations in a node, then the length of the bit string of each relation becomes 2^n. This is especially an issue for sparsely populated relations only referring to a very small subset of Q.

Both issues can be remedied by representing relations as bit strings relative to their related reservations set F_r. Each bit string effectively represents the right-hand side of a truth table with only the reservations in F_r as inputs. Changes outside the related reservations set will not influence the evaluation of the relation and therefore do not need to be modeled.

3.3.1.2. Boolean expressions

Representing relations as a truth table fulfills the generality requirement, but fails the conciseness criterion. From the modeling point of view, developers might describe paths as being dependent on, or excluding on another, instead of consciously enumerating all possible combinations as required by a truth table. Describing a relation using Boolean expressions is therefore a much better match between the protocol world and its model counterpart. As each function described by a truth table can equally be expressed in a Boolean formula – e.g. by forming it into a Canonical Disjunctive Normal Form (CDNF) – the generality of the approach is not lost.

A Boolean algebra for modeling relations consists of the following elements:

1. The state variables s_i as input variables
2. The result values of relations r_i as output variables
3. The basic operations $\land$ (Boolean AND), $\lor$ (Boolean OR), $\neg$ (Negation)
4. The derived operations $\rightarrow$ (implication), $\leftrightarrow$ (Equivalence), and $\oplus$ (Antivalence, Exclusive OR)

The usual semantics and rules for operations in a Boolean algebra apply and are not further described here.

3.3.1.3. Propositional formulas

While researching the applicability of Boolean expressions to model specific protocol use cases, a limit in their expressiveness continued to show up: the operations of the algebra are well suited to describe relations between any two reservations, but become cumbersome to use in cases where more than a pair of paths is involved. The specific nature of the QoS system often requires modeling not only exact relations (e.g. "p_1 and p_2 depend on each other"), but rather more general propositions about resource allocations (e.g. "p_1 depends on any two of (p_2, p_3 or p_4)", see section 3.3.2 for examples). While Boolean expressions are able to model such cases, the result is akin to enumerating all possible combinations again.

In order to simplify the modeling of some common structures, the Boolean expressions are extended into propositional formulas. These not only contain state variables as inputs, but also more general propositions about the partial resource allocation d_{r_i} linked to a relation r_i.

Propositions can be formed from the following elements:

1. the partial resource allocation variable d_{r_i}
2. an operator $(b_k, \ldots, b_0), b_i \in \mathbb{B}$ for defining a k-tuple of Boolean values. Boolean values are either state variables s_i or propositional formulas as defined in this section.
3. the operator $|v| : \mathbb{B}^k \rightarrow \mathbb{N}$, representing the number of elements of a given tuple v

4. the operators $|v|_1 : \mathbb{B}^k \rightarrow \mathbb{N}$ and $|v|_0 : \mathbb{B}^k \rightarrow \mathbb{N}$ returning the number of 1 or 0 (respectively) in a given tuple v

5. relational operators $=, <, >, \leq, \geq : \mathbb{N} \rightarrow \mathbb{B}$ with their usual semantics over natural numbers

6. natural numbers $n \in \mathbb{N}$

Propositions over integer operands always result in a truth value (i.e. necessarily contain at least one of the relational operators).

This extended definition of a relation modeling language does not compromise the generality of the approach. Propositions do not diminish the expressiveness of Boolean expressions, but rather extend it in some common modeling cases as presented in section 3.3.2. Each Boolean expression is, by definition, also a propositional formula without any of the additional propositions defined above. Transitively, if any relation can be represented using Boolean expressions, it can also be expressed as a propositional formula. Any proposition, as defined above, can equally be represented as a Boolean expression, by simply enumerating all "true"-cases of the proposition and expressing them as a CDNF. As propositions are formed over a finite number of input variables, they can only represent a finite number of possible states and their respective truth values, making an enumeration possible. Therefore, Boolean expressions and propositional formulas, as defined above, are equivalent in their modeling power.

3.3.2. Examples

In order to give clearer insight into the intended use of the relation model, this section presents some common patterns that emerged during the design phase of this work. Each relation is described informally and defined using the formal language presented in section 3.3.1.

Independent A set of reservations is *independent* of each other if its members bear no relationship other than by chance being in the request set at the same point time.

A formal representation – as far as it can be considered useful – of this relation is:

$$r = 0$$

Mutually dependent Reservations that are *mutually dependent* cannot exist without one another. An example would be the two flows of an ARQ-protocol like TCP. If either one breaks, the whole connection stops working.

For a set of reservations $F_r \subseteq Q$ this relation r is modeled as:

$$(|d_r|_1 = |d_r|) \vee (|d_r|_1 = 0)$$

A mutual dependency is fulfilled if either all of the related reservations have resources assigned to them, or none has (by virtue of not wasting resources in this case).

Dependent A *dependent* flow cannot exist without another, but the dependency is not mutual. If, for example, a video streaming source sends an MPEG-4 [MPEG4] video stream with I- and P-frames on different network flows (e.g. marked by an appropriate option header), then the P-frame stream will depend on the I-frame stream, as the former cannot be decoded without the latter due to the inherent dependency relation of the frame types. The I-frame stream on the other hand can very well be used without access to the P-frames. This idea and its impact on QoS is developed more in detail as Scalable Video Coding in Annex G of the MPEG-4/AVC standard [AVC].

Formally, the dependency of a reservation p_i on p_j is modeled as:

$$s_i \rightarrow s_j$$

Exclusive Reservations are *exclusive*, if only one of the set can be active at any given time. This relation can be used to model different alternative parameter sets of the same reservation, e.g. different output profiles of a video source, resulting in different data rates on the link.

The exclusive relation r is formally defined as

$$|d_r|_1 \leq 1$$

At least n The *at least n* relationship indicates that an arbitrary subset of the given reservations can be active to fulfill this request, as long as at least n are enabled. A typical example would be the multiple parallel connections opened by web browsers for performance reasons. While the browser can certainly operate with just one connection, multiple connections increase performance and lower the probability of blocking on slow resources.

The formal definition of this relation r is:

$$|d_r|_1 \geq n$$

3.3.3. Model interactions

Relations in KASYMOSA QoS should be orthogonal to the reservations they constrain. However, depending on the concrete model, there are interactions with specific parameters which have to be taken into consideration when expressing application requirements. The interactions described here are specific to the parameter model presented in section 3.2.1. Others may exist, but are not further explored in this work.

3.3.3.1. Reservation priorities

Dependency relations may lead to a discrepancy between the *stated*[b] and *effective*[c] priorities of reservations. If a relation introduces a dependency $p_i \rightarrow p_j$ (i.e. p_i depends on p_j) and $\varphi_j < \varphi_i$ (i.e. p_j has a lower stated priority than p_i) their effective priorities will be identical.

Assuming an ordering of the reservations with $\varphi_i > \varphi_j$ under a value function for the resource allocation, as presented in section 3.4.3, there are three possible cases regarding the resulting resource allocation:

1. The value of the solution including p_i and p_j is greater than the value including all intermediate reservations p_k with $\varphi_i \geq \varphi_k > \varphi_j$. As this higher value can only be achieved by including p_j as a prerequisite for p_i, the latter "skips" all higher prioritized reservations p_k, effectively taking on the priority of p_i.

2. The solution containing at least one p_k with $\varphi_i \geq \varphi_k > \varphi_j$ is better than the one including p_i and p_j. In this case, p_i is "held back" by its dependency on p_j, effectively lowering its priority to the one of p_j.

3. The value of a solution including all p_k with $\varphi_i \geq \varphi_k > \varphi_j$ and p_j itself is higher than the value of the solution including p_i, p_j and a subset of all reservations in between. As $p_j \rightarrow p_i$, but not the other way around, p_j is even assigned resources before p_i, effectively leading to a priority inversion for the two reservations. In contrast to cases 1 and 2, this case does not apply to reservations in a mutually dependent relationship.

All cases assume a resource limited situation triggering the optimization process in the first place. For cases 1 and 2, p_i and p_j have the same effective priority, while in case 3, p_j takes on a higher priority than its dependent relation. In any case, this situation

[b] The priority as stated in the reservation parameters.

[c] The order in which resources are assigned to the reservation.

could be considered a model error on the side of the relation, as a low priority reservation is a prerequisite for scheduling a high priority one.

3.3.3.2. Lifetime

Similar to the priority parameter, a reservation's lifetime can be influenced by its dependencies. Again, if a path with a lower lifetime is a prerequisite for one with greater longevity, the latter may be effectively limited to the shorter lifetime. The exact behavior depends on the handling of expiring or deleted reservations within a relation. A system could decide to mark said reservations as permanently disabled, causing their dependents to fail early as well. The system could also decide to expire and remove the containing relation along with the reservation. In this case, the lifetime of the dependent reservation would be unaffected, as the dependency relation is removed. Whether this behavior is desirable, depends on the specific use case in question. Application designers again should take into account that a dependency of a path with a longer lifetime on one with a shorter lifetime could be considered a flaw in their relation model.

3.4. Resource allocation

Key to successful operation of a reservation-based QoS system is an efficient resource allocation. Clients reserve resources because their use case is not suited for the best-effort service offered by standard IP networks. Resources, in this respect, are most commonly physical layer transmission resources for which the different reservations compete.

In such a model, each reservation has an associated cost in terms of physical layer resources. Depending on the actual reserved bandwidth, tolerable error rates, delay requirements and the current underlying channel properties, resource demand may vary considerably. Section 3.4.4 describes the relationship between reservation parameters and their associated physical resource requirements in more detail.

Not all reservations in the system are created equal. Some are of higher importance, where others are more expendable. The reservations therefore have different value to the end-users and, in turn, the system. A resource allocation, especially in a constrained environment, should reflect these priorities. Section 3.4.3 discusses multiple alternative value functions and their respective priority model.

Finding the most valuable resource allocation under a given resource constraint is an instance of the *Knapsack problem*. Section 3.4.2 introduces the problem for the purpose of this work and describes its applicability to a resource allocation system.

3.4.1. Requirements

The resource allocation process should achieve two goals:

Maximize user satisfaction Users want to transmit data through the system. They are most satisfied if they have as many high value paths activated as possible (up to the point where all requests are fulfilled).

Waste no resources The system should not assign already constrained resources to reservations that cannot use them due to relation violations. As with any system trying to allocate specific resource blocks from a given overall capacity, there will be some waste due to an imperfect fit. However, this is not considered waste in the sense of deliberately assigning unusable resource blocks. Such "left-over" capacity might be assigned to the transport of best-effort data, which is most likely present in the system anyway.

3.4.2. The Knapsack Problem

A popular analogy of the Knapsack problem[d] is a thief with a backpack burgling a house. The backpack can only carry a limited weight (the resource limit). The thief looks around the house and sees items of different weight and value (their respective resource requirements and values). Obviously the thief wants to maximize the profit of the stolen goods, while still being able to carry them with his knapsack. This specific instance of the Knapsack problem is called the *0-1 knapsack* (because each item can be in the knapsack 0 or 1 times only. No item can be copied.) It is formally expressed as

$$\text{Maximize} \sum_{i=1}^{n} v_i x_i \text{ provided, that } \sum_{i=1}^{n} w_i x_i \leq W$$

Here v_i is the benefit and w_i the associated weight/cost of an item i . The variable $x_i \in \mathbb{N}$ is the number of times the item is included in the knapsack[e]. The value W is the capacity of the knapsack, i.e. the upper resource limit.

Adapting the general Knapsack problem to the resource allocation, as presented here, requires some changes in the setup. Network routers typically have more than two network interfaces. Instead of having a single common resource limit, the router has a constraint for each individual interface. Additionally, reservations can only be

[d] See [KPP04] for a thorough introduction of is problem.

[e] For the 0-1 knapsack $x_i \in \{0, 1\}$. Other variations like the bounded or unbounded knapsack exist, but are of no significance to the resource allocation problem

served by a specific interface $n \in N$ determined by the current routing state. This problem is a variant of the *Multiple Knapsack Problem with Assignment Restrictions*. Dawande et.al. [Daw+00] showed, that this problem is NP-hard.

Adding relations to the problem turns it into its final form:

$$\begin{aligned} \text{Maximize} \sum_{i \in I} v_i s_i \\ \text{provided, that} \sum_{n \in N} w_i s_i \leq W_n \\ \text{and } \forall r \in R : r = 1 \end{aligned} \tag{3.1}$$

where W_n is the maximum gross capacity of the network interface n. The best resource allocation maximizes the overall value, while not exceeding the available resources on any interface *and* fulfilling all relations.

3.4.2.1. Satisfiability of all relations

At first, requiring all relations to be satisfied for a feasible solution seems severely limiting to the general applicability of the system. After all, relations are generic propositional formulas, which, in the general case, are not necessarily satisfiable and whose satisfiability cannot be efficiently checked. This apparent limitation is not relevant for the task at hand for two reasons:

Non-satisfiable relations have no valid use case. A relation which is not satisfiable, i.e. which does not have valid solutions, does not offer additional information to the resource allocation process. The sole purpose of modeling relations is to divide the allocation space into two sets: valid allocations, which do not waste any resources (and therefore should be preferred) and invalid ones, that do. If a relation is not satisfiable, this separation is impossible. All possible resource allocations would be in the same, invalid, class.

The null-allocation is always feasible. The goal of the relation system is to avoid wasting resources by not allocating them to reservations that cannot be used due to their intrinsic dependencies. Under this model, the allocation not assigning any resources at all, is always valid (if not necessarily the best one). The system can therefore assume that for each relation an allocation $a_i = (0, \ldots, 0)$ will produce a valid result. This in turn implies that the satisfiability requirement in equation 3.1

is always fulfilled. Due to the maximization of the target function, the best allocation under the current constraints will most likely enable some reservations[f].

3.4.3. The value function

The goal of the optimization is to maximize the overall quality – expressed as a scalar value – of the resource allocation under the given constraints. This value is the sum of the individual reservation values. As those values are defined to be positive, maximizing the sum necessarily means selecting as many of the most valuable reservations as possible. The individual reservation values should therefore accurately reflect the initiator's preferences regarding the availability of each path.

3.4.3.1. Priority values

The priority φ_i of a path p_i represents the preference of the initiator for specific paths over others. It determines the *intrinsic benefit* v_i of the reservation from the point of view of the value function. Priorities are signaled as integer values in the range $[1, \varphi_{max}]$. Depending on the use case, this range can be mapped to the intrinsic value range $[v_{min}, v_{max}]$ in different ways.

Linear transfer function A simple linear priority model could use the signaled priority value as is.

$$v_i = \varphi_i$$

Under such a model, a reservation p_1 with a priority high φ_1 could be matched value-wise by n low-priority reservations $p_{2...n+1}$ with $\varphi_{2...n+1} = {}^{\varphi_1}/_{n}$ (a *preemption ratio* of ${}^{1}/_{n}$). This gives the end-system the option to finely balance how many low-priority paths preempt one with a higher priority, i.e. at which point a certain reservation is no longer "worth" preempting a whole set of others.

The drawback of this approach is a non-constant preemption ratio over the input range. A reservation with $\varphi = 2$ has a preemption ratio of 2 over one with $\varphi = 1$. However, to get the same preemption ratio over a reservation with $\varphi = 20$, a reservation already needs a priority of $\varphi = 40$. In the limited range $[1, \varphi_{max}]$, this means that against every priority beyond ${}^{\varphi_{max}}/_{k}$ the preemption ratio is less than k. Whether this limitation is an issue, depends on the actual use case and is outside the scope of this work.

[f] Insisting on the validity of the null-allocation may seem like a technicality. However, it does enable the direct transformation into an Integer Linear Program as presented in section 3.4.5.3.

Exponential transfer function Keeping a constant preemption ratio α throughout the input range requires an exponential transfer function:

$$v_i = \alpha^{\varphi_i}$$

Under this model, α reservations of priority φ_i are required to match one reservation of priority $\varphi_j = \varphi_i + 1$. Depending on the selection of the base α, the individual priority classes can be moved further apart or brought closer together.

Strict priority model A special case of the exponential model is a strict priority system, where a higher priority always preempts lower classes, no matter how many of those are on the line. To achieve this the base of the transfer function needs to be equal to the total number of reservations known to the system

$$\alpha = |Q|$$

The transfer function therefore becomes

$$v_i = |Q|^{\varphi_i}$$

Considering the generic case of two input values φ^m and $\varphi^n = \varphi^m + 1$, the transfer function should never cause a path of the latter priority to be preempted by any number of paths of the former. According to the transfer function given, the two input values result in the following benefit:

$$v^m = |Q|^{\varphi^m} \tag{3.2}$$

$$v^n = |Q|^{\varphi^n} = |Q|^{\varphi^m+1} = |Q| \cdot |Q|^{\varphi^m} \tag{3.3}$$

The closest a lower class could come to preempting a higher class under such a regime would be a situation where all but one elements of Q are of lower priority.

In this case the combined value of all low-priority reservations is

$$v^* = \sum_{\substack{k \in I \\ \varphi_k = \varphi^m}} \varphi_k = (|Q| - 1) \cdot |Q|^{\varphi^m} \tag{3.4}$$

From equations 3.3 and 3.4 follows, that $v^n > v^*$, i.e. a set of paths with priority φ^m could never out-compete even a single reservation of priority φ^n.

Using the exponential transfer function does have a significant practical drawback: depending on the base and the range of possible priorities, the resulting benefit values become very large. Regarding the strict priority model, the base is already large to start with (the total number of paths known to a node) and may be raised to quite a significant power. Common computing hardware cannot handle numbers that large efficiently. Care should therefore be taken when selecting this transfer function. If the number of different priorities is low, it may be more efficient to handle each class on its own, running the optimization algorithm multiple times. Specifically the non-preemtable use case is much more efficiently solved by optimizing twice.

3.4.3.2. KASYMOSA QoS's transfer function

The selection of the actual transfer function depends on the intended use case. If the priority indicates a mere preference without any actual guarantees, a linear function might be appropriate. For more strict use cases, an exponential transfer function with a sufficiently high base provides better separation of different priority classes. However, the decision is system-wide: mixing different transfer functions in one network leads to inconsistent interpretations of the reservation priority values and therefore has to be avoided.

As described in section 3.2.2, KASYMOSA QoS employs a simple preference model with respect to its priority values. A linear transfer function, as presented above, is used to calculate the intrinsic values of reservations. Whether or not the selection of the transfer function has a significant influence on the performance of the system (e.g. by limiting the degrees of freedom for the optimization algorithm) is a question beyond the work presented here.

3.4.4. A cost function

Where the value function defines preferences for different solutions in the resource allocation process, the cost function assesses their viability. If the total *required* capacity of a solutions exceeds the *available*, it simply cannot be implemented, no matter how beneficial it would be.

The cost function calculates the amount of transmission resource units (e.g. $^{\mathrm{bit}}/_{\mathrm{s}}$) used for a specific reservation. As the capacity of a link is given in the same units, simply summing up the individual costs of all enabled paths results in the overall cost of a solution[g]. For viable solutions, this value is less or equal to the current capacity of the link in question:

$$w = \sum_{i \in I} s_i w_i$$

[g] This model assumes, that any overhead for transporting multiple transmissions over the same link (e.g. protocol headers to distinguish individual flows), is included in the individual cost of each reservation.

Calculating the individual path cost w_i is slightly more involved. Initiators request the net data rate c_i needed to fulfill their service. Depending on the setup, this may already include the necessary overhead for packetizing the data stream and adding the necessary headers down to the IP layer. Beyond this, initiators cannot make useful assumptions about the actual transmission capacity needed on a specific link. Where wired networks like Ethernet only add a slight, fixed overhead in form of the frame headers to each packet, wireless technologies might have a much more complex relationship between net and gross data rate of a path.

In the KASYMOSA project, the main focus is on a QoS-capable satellite link, an example of a very complex cost function. Depending on the current link quality and the QoS requirements of the path, a reservation might need vastly different transmission resources to guarantee the same net QoS parameters at different points in time. Loss-sensitive traffic like TCP could be transmitted using much more redundancy at the physical layer, whereas audio stream would need to be less protected. The former therefore needs more gross capacity than the latter to transmit the same net rate. The correct cost calculation can only be done by tight cross-layer integration between network, MAC and physical layers.

3.4.4.1. Ethernet

In contrast to the value function, on which all network nodes have to agree in order to achieve a predictable system behavior, the cost function is local to a specific link. KASYMOSA QoS (in the context of the KASYMOSA project) implements two cost functions: one for Ethernet links and one for the satellite link. The simpler Ethernet function adds an estimated overhead of 3% to each reservation:

$$w_i = c_i \cdot 1.03$$

The factor is a compromise between having a more complex QoS model and overestimating resource usage for many of the paths. It is based on the assumption that the payload of an average Ethernet packet will be half the maximum possible payload of 1500 bytes. There will be transmissions with larger payload sizes (e.g. file transfers via TCP), as well as smaller ones (e.g. voice calls, which focus on delay). The overhead of the Ethernet header, including the preamble, on such an average frame is approximately 3%. This model will overestimate the overhead for large frames and underestimate for small frame sizes.

A more accurate alternative to this simple overhead estimation requires a more complex QoS model which includes at least the average packet size, along with the data rate, to allow for a more precise overhead calculation. The system could try to infer the necessary parameters from the type of traffic reserved, e.g. protocol, well-known ports, total data rate or even observe the actual traffic to adapt over time. As the Ethernet part of any satellite system is most likely *not* the bottleneck of a path, estimation errors are not expected to influence the system performance. Better overhead estimation algorithms are therefore left to future research.

3.4.4.2. Satellite

The satellite cost function is slightly more involved than simply adding a fixed percentage of overhead. The lower layers have different possible configurations for different service types, delay and loss requirements, and link conditions. A certain robustness against environmental influences may, for example, be achieved by transmitting a more redundant signal over a shorter time or spreading out a less redundant signal over a longer period. The former is more susceptible to bursts of decoding errors (e.g. signal distortion by passing a tree) and therefore may need more redundant information to still be able to recover. The latter spreads out the transmission over a longer period, diminishing the effect of short interruptions on the overall transmission. It does, however, pay the price of incurring a higher delay, something that might not be desirable for specific types of traffic. Due to the increased redundancy in the first approach, a transmission needs to trade increased resource usage for a lower transmission delay, a fact that is reflected in the cost function.

At each specific point in time in KASYMOSA QoS's operation, the relation between net and gross data rate, i.e. net data rate and required transmission resources can be expressed as the *code rate* $f_c(p) = {}^{k}\!/\!_{n} = {}^{\text{net rate}}\!/\!_{\text{gross rate}}$. This ratio depends on the specific QoS requirements, as well as the current link situation, and is known to the physical layer. The QoS system can request this information from the lower layer and calculate the resource requirements of each reservation p as:

$$w_i = \frac{c_i}{f_c(p_i)}$$

Implementing the cost function on a satellite terminal (or any other system with a variable code rate) is therefore only possible in close cooperation with the physical layer. Information about the current link state and the resulting modulation and coding decisions needs to be relayed to the cost function in order to correctly optimize the resource allocation.

3.4.5. An optimization algorithm

Based on the value and cost functions presented above, the system has to perform two different optimization tasks. First of all, a router has to calculate the optimal resource allocation in case of a capacity restriction. Crucially it also has to calculate the "best case" solution, assuming no limits at all.

3.4.5.1. Best case allocation

In a QoS-system without relations the best case allocation is simple to calculate: it is the sum of all requested paths. However, the introduction of relations changes things. Consider the case of an adaptive video streaming application with different output profiles: such a system might offer a number of different stream qualities (resolutions, compression qualities etc.), depending on the current available data rate. In terms of KASYMOSA QoS's modeling this could be expressed as an Exclusive relation over different reservations with decreasing priority. At any point in time, only one of the reservations would be online, with preference given to the high-quality (and therefore high-priority) ones. The best case allocation under such a regime is no longer simply the sum of all paths in the relation, but the one with the highest priority. More complicated relations might yield even more complex solutions.

3.4.5.2. Resource adaptation

When adapting to a limited transmission capacity, the optimization process effectively has simply one more constraint to follow. Section 3.4.5.3 will show that both the resource limit and the relation information can be modeled as invariants in an Integer Linear Program. Existing algorithms and their implementations efficiently solve the problem sizes presented in this work, making the use of the system feasible.

The resource adaptation process has to incorporate the emergency preemption mechanism presented in section 3.2.1. To do so, the optimization algorithm is run twice: first, with only the paths having the Emergency-flag set, leaving $W_{rem} = W - \sum_{i \in I} s_i w_i$ as the remaining capacity. The second step will only consider non-emergency paths for optimization under the W_{rem} constraint. This approach eliminates the need for a more complex priority transfer function that guarantees isolation within a common value space.

3.4.5.3. Integer Linear Programming

Integer Linear Programming problems are optimization problems where an objective function has to be either maximized or minimized under some given constraints. Both the objective and the constraint functions are linear. In the special case of an Integer Linear Program, the constituent variables are restricted to be integers. More specifically, for ILPs solving the 0-1-Knapsack problem, they are binary.

By transforming the resource allocation problem with its relation constraints into an ILP, a huge body of existing algorithm research and implementation becomes accessible. Existing solvers like COIN-OR CBC [COIN] implement well researched algorithms that solve the optimization problem, albeit being NP-hard, in acceptable time for the problem sizes at hand.

Mapping the resource allocation problem to an ILP is a straightforward process[h]:

Variables Every state variable s_i is also a variable in the optimization problem. When mapping the relation constraints, additional, non-solution helper variables can be created. While these are normal variables from the point of view of the ILP solver, they are not part of the final resource allocation.

Objective function The objective function can be directly expressed as an ILP objective function:

$$\text{Maximize} \sum_{i \in I} v_i \cdot s_i$$

Capacity constraints The capacity constraints can be directly expressed as ILP constraints as well:

$$\text{For every interface } n \in N \text{ ensure, that } \sum_{i \in I_N} w_i \cdot s_i \leq W_n$$

where W_n is the specific gross capacity of the interface n.

Relation constraints The propositions representing the relation constraints are created bottom-up, based on their abstract syntax tree. Integer helper variables are introduced, where necessary.

[h] For clarity reasons this work deviates from the usual matrix and vector notation of ILPs and uses explicit summation instead. The two notations are equivalent, but the one used here is more aligned with the standard expression of a Knapsack problem used so far

Conjunction The conjunction of a set V of variables can be expressed as:

$$0 \leq \sum_{s_i \in V} s_i - |V| \cdot y \leq |V| - 1$$

The binary variable y holds the result of the conjunction.

Disjunction The logical OR of a set V of variables can similarly be expressed as:

$$0 \leq |V| \cdot y - \sum_{s_i \in V} s_i \leq |V| - 1$$

Again, the value of the binary variable y is the result of the disjunction.

Negation The negation of a variable s_i is transformed to an equality constraint:

$$y = 1 - s_i$$

Expressing the counting operators based on the definitions given before, is a straightforward process:

Count-One $|d|_1$ The Count-One operator for a tuple d becomes a simple sum:

$$y = \sum_{s_i \in d} s_i$$

Count-Zero $|d|_0$ The Count-Zero operator on a tuple d is expressed by counting the ones of the negated tuple:

$$y = \sum_{s_i \in d} 1 - s_i$$

The cardinality operator does not have an equivalent as the ILP. While it could be transformed as the sum of Count-One and Count-Zero on the same tuple, the tuple size would actually be a known constant and be used directly, when constructing the ILP representation.

The comparison operators used in the propositional formulas are directly supported by the ILP solver and need no further transformation[i].

[i] Formally, an ILP only contains "≤" constraints. However, since the other operators can easily be expressed in terms of this inequality, most modelers, including the COIN-OR front-end used in this work, automatically transform them where necessary.

Building up the full relation from these individual transformations is merely a matter of following the abstract syntax tree to its root, reusing helper variables where necessary. The top-most constraint representing the full relation r is simply restricted to value greater 0, ensuring that the solution is only valid if the constraint holds.

$$1 \leq r$$

In a similar fashion, the state variable for reservation p_i that is not eligible for optimization (i.e. a remotely suspended reservation), is restricted to at most 0, ensuring that no solution, which requires p_i to be online, is feasible.

$$s_i \leq 0$$

Based on the transformed representation of the optimization problem, an ILP solver is then able to find an optimal assignment to the contained state variables s_i.

See appendix B for a detailed example of a transformation.

4. The KASYMOSA QoS Transport System

The very nature of a network QoS system is to have a distributed architecture. State information is scattered over multiple nodes, each of which can be a source for events pertaining to said state. This necessitates a synchronization protocol along with the corresponding transport system.

KASYMOSA QoS's use case imposes the following requirements on the communication subsystem:

Minimal number of interactions One of the key requirements of the system is a short convergence time over low-bandwidth, long-delay links. As with the QoS model in chapter 3, the basic assumption is that round-trips (and thereby protocol interactions) are expensive and are to be minimized.

Network-originated signaling Operating in a highly volatile environment leads to a significant amount of QoS-related events originating in the network and being signaled to path endpoints.

Support for multicast reservations One of the use cases, stemming from the KASYMOSA project, requires the signaling of multicast reservations. The signaling subsystem needs to support the reservation and management of multicast distribution trees.

Robustness A mobile communication environment is always fraught with the risk of losing packets. The signaling subsystem must provide robustness against packet loss, duplication and corruption. This includes a protection against stalling independent signaling processes on a lost packet in another transmission.

4.1. Protocol primitives

Chapter 3 describes the system's model objects that represent network's state. Accompanying those is a set of protocol actions to change said state. The protocol supports the following primitives:

Reserve Using the *Reserve* action, an initiator requests the creation of a set of QoS paths from the network. The accompanying reservation and relation objects describe the new state as desired by the initiator. Reservation requests are subject to the network's admission control mechanism. After such a request has been granted, the appropriate QoS is guaranteed until further notice. Section 4.1.1 describes the reservation process in more detail.

Suspend/Resume KASYMOSA QoS strives to minimize protocol interactions by suspending and resuming reservations in the same way as first demonstrated in the MoSaKa system [Hei+10], KASYMOSA QoS strives to minimize protocol interactions by being able to suspend and later resume reservations. See section 4.1.2 for details on the mechanism.

Refresh In a soft-state system like KASYMOSA QoS, reservations have to be continually renewed or they expire. This approach ensures an eventually consistent network state without lingering reservations, even in the case of a sudden disappearance of an initiator. The main refresh mechanism of the system is implicit, by observing the usage of the data path, as described in section 4.1.3. To support reservations with only intermittent traffic, the protocol also includes an explicit *Refresh* primitive to be sent by the initiator in case of longer intentional transmission pauses on a path. Both mechanisms work by resetting the life-timer of their respective reservations, preventing them from expiring.

Delete Even though KASYMOSA QoS is a soft-state system, which could just let reservations expire by no longer refreshing them, the protocol still supports an explicit *Delete* action. The soft-state timeout is always a trade-off between the refresh overhead and the convergence time in case of the end-of-life of a reservation. Introducing an explicit delete action allows the initiator to choose longer expiration timeouts, while still being able to tear down a reservation on short notice. Section 4.1.4 presents a detailed discussion of the intricacies of deleting reservations and their relations.

4.1.1. Reservation

The *Reservation* action requests the creation of new state in the network. The action is accompanied by a set of reservations and relations to be added to the respective sets in a node. Upon reception of such a request, a node will try to allocate the necessary resources locally and, if necessary, further downstream. It will then signal the results back to its upstream.

KASYMOSA QoS's concept of relations as well as the ability to suspend and resume paths influences the handling of reservation requests. Whereas the classic admission control process takes an accept/reject decision, KASYMOSA QoS's is slightly more nuanced. If a reservation is currently blocked by a relation, it is suspended, rather than rejected. This indicates to the initiator that the system took note of a reservation, but is not currently serving it. A rejection on the other hand, would signify a final decision on the inability to serve said request. The latter would make it impossible for an initiator, to request reservations in an exclusionary relationship, as only one of those would not be rejected.

In case of a resource shortage, the system handles suspension and rejection depending on the current state of a reservation. Reservations that were already admitted, but cannot be guaranteed any longer are suspended. New reservations that have not been admitted yet and are not currently blocked by a relation (in which case no resources are requested for them anyway) are rejected. The decision on which reservations have resources allocated is subject to the value model presented in section 3.4.3. By implementing suspension for existing paths and rejection for new paths, the system puts a slight focus on stability over fairness. All other things being equal, a new transmission request will not be served until an existing one is finished. A system focused more on fairness (i.e. serving more end users equally, instead of providing the best QoS for admitted transmissions) may take the age of a reservation into account and gradually diminish its value over time, until new reservations win out.

A reservation always describes the next (as in: outgoing) path segment from a node's point of view. Similar to the GIST signaling overlay, KASYMOSA QoS builds QoS paths as a series of segments with possibly different parameters. The overlay formed by this approach and how it enables in-network adaptation of reservations, as well as multicast support, is discussed in more detail in section 4.2.

4.1.2. Suspension/resumption

KASYMOSA QoS takes its Suspend/Resume mechanism to support operation in a highly volatile environment from the MoSaKa system. In order to minimize the convergence time in case of environment changes, as well as the signaling effort required for dealing with transient link degradations, the system actively manages paths which are currently not served. This frees the initiator from periodically trying to obtain the necessary resources through the normal reservation process. It indicates that the system regards the reason for not serving a reservation as temporary, whereas a deletion is a permanent decision. The mechanism has two main effects:

- Actively managing the paths on the network side lowers the reaction time to changes. Whenever the capacity changes, a router can immediately react to this change by either suspending, or, in particular, resuming paths. Without this mechanism, end systems would periodically have to check on the current situation. Depending on the check interval, there would be a significant delay in reacquiring a reservation after a capacity increase.
- By removing the signaling burden from the initiator, the system saves a significant amount of signaling messages. Instead of having the initiator periodically request resources, only to fail because the link degradation still persists, the network can simply send a single message when the issue is resolved. This is especially crucial in a highly volatile environment, as the initiator would have to use a very short signaling period to react reasonably quickly to any changes.

In addition to the advantages presented so far, the Suspend/Resume mechanism plays a crucial role in supporting exclusionary relations between reservations. Assuming, for example, an initiator wants to signal two different transmission profiles for a video source, only one of which is used at any given time. Without the path suspension mechanism, admission control could only reject one of the reservations because it is blocked by the relation. However, rejecting it would make it impossible for the requester, to distinguish between a genuine rejection due to insufficient resources and a reservation being blocked by a relation[a].

4.1.3. Implicit refresh

One of the goals of KASYMOSA QoS is the minimization of protocol interactions to save traffic and speed up convergence. Especially when operating on an overloaded link, the additional signaling transmissions generated by the refresh mechanism can make up a significant portion of the overall traffic. As these messages are necessary for the correct operation of the system – as reservations would otherwise expire – they have to be transported with a higher priority, effectively preempting real traffic on the link. This is particularly an issue, as the signaling traffic is very intermittent and can lead to a situation where the system periodically has to suspend and resume reservations in order to just transport refresh messages.

To avoid this issue, KASYMOSA QoS implements a *implicit refresh* approach. This approach rests on two assumptions:

[a] Obviously the rejection could convey such information using specific status codes for blocked reservations. However, this would be equivalent to the suspension of a reservation: the accompanying state would still be kept in the network to be enabled when necessary/possible.

1. QoS routers can observe the data flow related to a specific reservation. As the routers holding a reservation are the very ones that are providing the QoS guarantees entailed by it, they have to be on the forwarding path for the respective traffic. Otherwise they would not be able to influence the QoS with their forwarding decision.

2. Paths that are used are still needed. If data is still transmitted along a path it can safely be assumed that this traffic is to be subject to the guarantees initially requested.

Soft-state approaches mainly serve a simple purpose: remove state in the network, which cannot be removed by the original requester anymore, due to their inability to send signals to a certain set of nodes. This can either be because of a sudden disconnection of the initiator or a change in the routing in the network, which diverts a path via a different set of routers. In both cases, refreshes from an initiator can also no longer reach the respective routers, which in turn let the reservations expire after some time.

A major optimization parameter for the soft-state approach is the reservation lifetime. A shorter lifetime enables quicker reaction to changes in the network, whereas a longer lifetime lowers the amount of refresh signaling necessary. The implicit refresh approach is able to alleviate the trade-off issue by eliminating explicit refresh messages altogether. Resting on the two assumptions given above, KASYMOSA QoS uses the data flow along a path as an indicator that said path is still needed. It does so by taking each data packet as a refresh event, resetting the life timer of the appropriate reservation. This way, no additional resources are needed to indicate the refresh. In the case of route changes or crashes of the initiator, the data flow stops and the usual lifetime management takes control.

Topology change detection Expiring inaccessible reservations after a topology change is only half the issue for continued QoS support. New nodes that have not been part of the forwarding path before, need to have information about the reservation, in order to handle traffic appropriately. Using periodic refresh messages, routers can detect unknown paths and request the necessary information from the initiator. This mechanism fails with implicit refreshes, as the data traffic is in no way different from a simple, unreserved transmission. A new router simply detects an increase in best-effort traffic and cannot assume anything about this change.

There are two types of topology change relevant to this issue: a change where systems on the old route (or parts thereof) can still reach the initiator (figure 4.1) and a situation where this is no longer possible (figure 4.2). Both cases need to be handled slightly differently by the system.

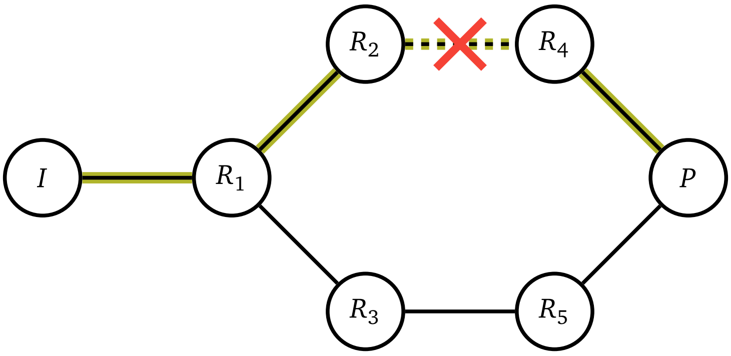

Figure 4.1.: *A topology change leaving parts of the old route connected.*

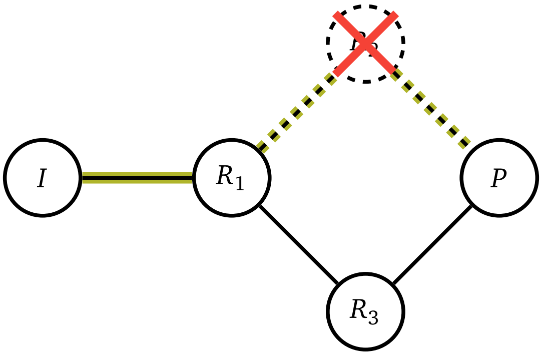

Figure 4.2.: *A topology change leaving the old route completely disconnected.*

The case presented in figure 4.1 can be handled by the normal reservation expiry mechanism, as implemented by KASYMOSA QoS. After the routing switches from R_2 and R_4 to R_3 and R_5, at some point R_2 will expire the path segment it still holds. As it is still able to reach R_1 and therefore, by proxy, the initiator I, the system can detect the route change because of the expiry notification. I or, depending on the actual implementation of the route change handling process, R_1 can then initiate a re-reservation along the new route, based on the knowledge, that data should still be traveling along the reserved path.

Figure 4.2 presents a different issue. In this case, the forwarding node itself has gone down and can no longer expire the reservation as necessary[b]. This case cannot be solved by passively observing signaling messages that would be sent in the normal course of the protocol. Instead, there are a few different options, as to how to tackle this situation:

Periodically send explicit refreshes If an initiator sends periodic refreshes along the path, new routers are able to detect unknown paths and inform the initiator accordingly. As a drawback, this solution reintroduces extra signaling, in the normal operation of the network, leading to the problems discussed above. However, depending on the volatility of the network topology, applications could send much fewer of those signaling messages than necessary for the usual refresh/expiry mechanism.

Explicit route change notification If the route change occurs at a QoS router, the system could explicitly notify the initiator about the change and trigger the re-reservation process. This is the most flexible solution, but requires additional active signaling on the part of the network. This also does not solve cases where the route change occurs at an intermediate router (e.g. when integrating with a legacy network), which does not have a concept of the signaling requirements.

Received QoS verification Applications monitoring the received QoS (e.g. by monitoring parameters like throughput, packet error rates and the like) could be able to detect relevant changes in the transmission quality and trigger a topology change detection. This approach would not detect and try to fix the path interruption until it becomes relevant. As long as a best-effort forwarding in the new intermediate nodes still meets the expected parameters, no path repair would be triggered. While this would limit protocol interactions to situations where they are strictly necessary, it does incur an extended reaction time in the event of a QoS violation. As the end system would have to monitor the received parameters for some time to get statistically valid estimated values and then trigger the process of rebuilding the path, the required QoS will be violated for an extended period. As an added detriment, the signaling necessary for repairing the path will occur in a situation where the network is already congested (hence the violation of the QoS parameters in the first place).

KASYMOSA QoS does not implement any of the presented approaches. As it focuses on mobile satellite communication, where route changes to a different satellite link are

[b] This case is representative for all cases where information about an expiry can not get through to the initiator. This could also be caused by a network split, where the relevant routers are still working, but cannot send signaling messages.

uncommon or even impossible, the path repair mechanism is currently outside the scope of the system.

4.1.4. Deletion

In a soft-state system like KASYMOSA QoS, unused reservations will expire eventually. However, this mechanism is intended as a fail-safe fallback in case of breaking changes in the network. Relying on it for normal path tear-down would imply wasted resources for at least one path lifetime, as the earliest a router could expire a reservation would be, by definition, after that period of inactivity.

Supporting explicit path tear-down solves this issue. When an initiator no longer requires a specific reservation, it sends out a *Delete* request. Routers that receive such a request can free any assigned transmission resources.

Introducing the concept of relations into the QoS model affects the system's ability to delete reservations. Whereas independent reservations can always be deleted without any regard for their environment, things are slightly more complicated if a reservation is part of a relation. Deleting such a reservation influences the state of others in the same relation to some extent. Depending on the exact behavior of the system, these could become impossible to serve. There are four possible approaches to this issue:

Refuse removal of the reservation Whenever a reservation is contained in an existing relation that is not to be removed along with it, the system can refuse the removal of said reservation. The reason for this refusal is indicated to the initiator, so that appropriate action can be taken.

This approach considers the removal of a bound reservation as a modeling error. It therefore errs on the safe side and refuses to carry out such a transaction. While this ensures that no initiator removes any reservation accidentally, it increases the number of protocol interactions in this case.

Remove the reservation from any relations it might be part of A deleted reservation no longer exists for the system. It therefore cannot conceivably be part of any relation. This approach assumes that whatever relation two paths had is no longer intended by the initiator. Deleting a reservation under this regime is essentially the same as in a system without relations: a reservation can always be removed and leaves no traces in the system.

This approach does not seem viable as it allows the creation of invalid relations. Assuming a reservation is used in a binary operator like $\wedge$ or $\vee$: there is no clear concept of how the relation should behave. The reservation could be replaced by a static constant 0 or 1, marking it either as permanently offline (see the respective approach below) or online. Both cases can be problematic. Consider the latter case of marking a reservation as permanently online: Whereas a $\wedge$ relation would behave as if this reservation didn't exist any longer (which could be considered correct on the grounds that it was removed), as $s_i \wedge 1 = s_i$, a $\vee$ relation would effectively become useless. As $s_i \vee 1 = 1$, the state of s_i would no longer have any bearing on the relation at all. Relations like $s_i \oplus 1$ would behave even worse, effectively blocking the related reservations from being enabled ever again.

Transitively remove any relation the reservation is part of If the system considered reservations as the "foundation" for relations, then it could transitively remove any relation that a reservation is part of. However, without removing all other related paths, this approach would lead to an issue: the system state could contain paths which are unusable from the application's point of view, but are no longer constrained by any relation, thereby wasting bandwidth.

Transitively deleting relations *and* related paths might lead to a chain reaction, with the deletion of a single path tearing down a whole reservation structure. It could be argued, that signaling a single deletion constitutes a modeling error anyway, and keeping the network state consistent is therefore of overriding importance. However, the same argument can be made for the refusal to delete bound reservations. The latter approach errs on the side of caution by refusing to execute a potentially dangerous action, while the former focuses on faster cleanup of potentially invalid relations.

Mark the reservation as permanently offline and deleted If an initiator deletes a reservation, it indicates that it doesn't intend to use the corresponding capacity anymore. This indication could be taken as a sign that it has no intention of fulfilling any prerequisites for other reservations, as indicated by the relations once modeled. By marking such a reservation as permanently offline, the system can reflect this in a consistent way in its internal state. Any further resource allocation would have to take this indication into consideration when calculating its value. In line with considering such a signaling as a modeling error, the system should inform the initiator about the fact that the reservation was part of a relation, giving it a chance to rectify the situation.

This approach is something of a compromise between approaches 1 and 3. On the one hand, it immediately enacts the change requested by the initiator by calculating a new solution under the assumption of the new path state. On the other hand, it still protects reasonably against a deletion cascade by only suspending reservations where necessary.

KASYMOSA QoS's implements approach 1. Deleting a reservation from a relation is considered a modeling error and rejected by the system. To avoid having to delete reservations due to life-time expiry (where no originator of any signaling could be blamed for the error), the system unifies the life-times of all reservations in a relation, ensuring, that a relation and their reservations will be deleted together[c].

To allow an initiator to delete a whole relation with its constituent reservations, the system ensures that the multiple deletion requests in one signaling process are executed in an order which prevents errors.

4.1.5. Modeling parameter updates

The protocol primitives do not include an UPDATE request. To simplify modeling and implementation of the request structures, reservations are considered static. However, the relation model can be used to flexibly model updates where necessary. By injecting a new request into the network and correctly relating it to an existing reservation, an initiator can choose between different replacement strategies.

Unconditional replacement If a reservation is to unconditionally replace another, the initiator simply signals the deletion of the old path and the reservation of the new one in the same request. Routers will free the resources currently assigned to the old path and try to acquire them for the new reservation. If this process fails, the user is left without any reservation at all (the old one has been torn down, the new one is rejected). This is exactly what this strategy is trying to achieve: replace the old path even at the risk of losing the reservation altogether.

Conditional replacement A less risky strategy for the initiator is to replace a path with a new reservation only if this can be successfully achieved. Otherwise, the old state will be kept. This can be achieved by reserving the new state within an Exclusive-relation to the existing one. Routers can then try to enable the new reservation. If this fails suspend it and enable the old version. Once the new state is in place, the initiator can send a delete request for the old state and the relation. While this strategy increases the signaling overhead for parameter updates (because it requires at least two round-trips: one for the request and one for the deletion), they are rare enough to be negligible. Their main use case would be the adaptation of a reservation due to current network conditions, something, that can just as easily be signaled via an exclusive relation in the path setup process.

[c] See section 3.3.3.2 on why different lifetimes in a relation should be considered a modeling error.

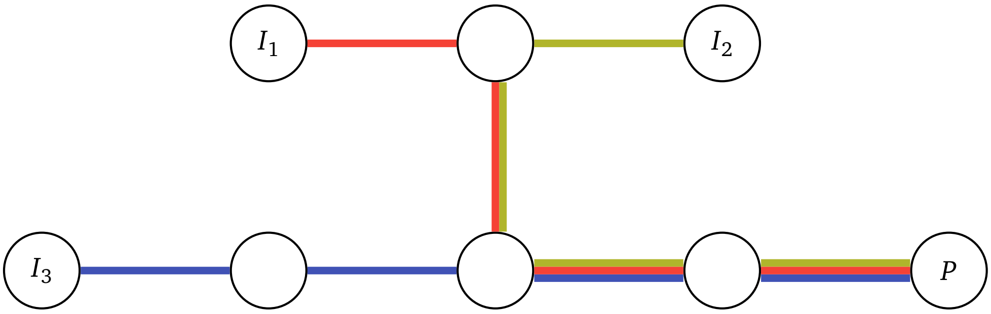

Figure 4.3.: *Unicast reservation with end-to-end paths*

4.2. Path structure

Reserving a QoS path in an IP network necessarily means creating state on multiple independent systems along the way. While we might think of the path as an end-to-end structure, it actually comprises multiple nodes, each dedicating a part of their transmission resources. How those nodes implement the reservation is transparent to the user. Conceptually, there are two approaches to this problem:

1. Paths are always end-to-end. This approach, taken by systems like MoSaKa, considers the end-to-end path as one single entity signaled to each router along the way. Those then provide their service directly to the initiator of the reservation.

2. Paths are hop-by-hop only. Systems like GIST[d] or DiffServ[e] expect the requested service always from their next-hop neighbor without any mandate on how this node actually acquires the necessary resources.

Approach 1, depicted in figure 4.3, is simple to implement in a path-coupled system: the initiator sends the reservation request along the intended path, where each router on the way can intercept it and reserve the necessary resources locally. All path maintenance – with the possible exception of garbage collection for expired paths in a soft-state system – is done by the initiator.

Approach 2, in contrast, is much more involved: figure 4.4 shows a reservation of three independent end-to-end paths in a hop-by-hop system. Each path consists of a multitude

[d] GIST as a transport system does not mandate any reservation scheme at all. The hop-by-hop nature of GIST is embedded in the structure of the transport overlay.

[e] DiffServ does not mandate reservations per se. However, conceptually, the system implements a per-hop QoS behavior that may include reserved capacity for individual traffic classes.

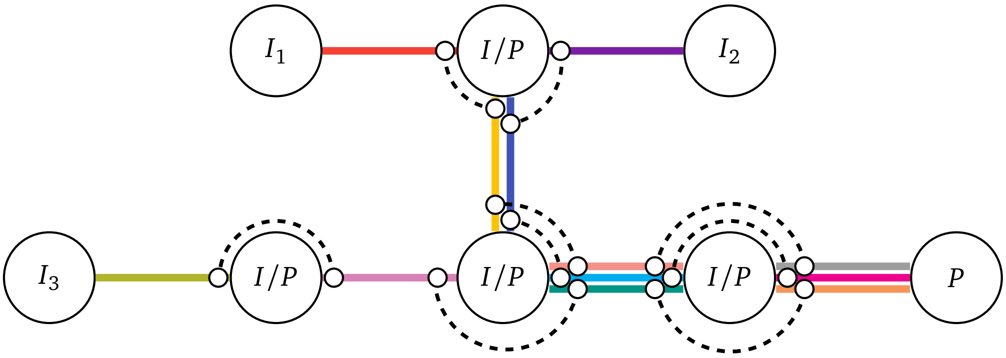

Figure 4.4.: *Unicast reservation with hop-by-hop paths*

of segments between neighboring nodes, linked within each router. Every path segment is an independent reservation which secures the necessary down-stream services. In such a system, each router fulfills the dual role of a peer (i.e. the endpoint for a reservation) and of an initiator (i.e. the originator of the next segment towards the actual path endpoint). While this approach incurs a higher complexity on each router – at the very least it now has to keep two reservations, one incoming and one outgoing, as well as their connection – it does have the advantage of allowing a more flexible adaptation of reservations. Routers can change the actual reservation parameters to account for any additional overhead, and easily merge reservations into aggregates (or even split aggregates for that matter, e.g. to transport them over multiple outgoing links).

4.2.1. Multicast support

The real benefit of a hop-by-hop solution lies in the simple implementation of multicast reservation support. Figure 4.5 shows a possible multicast reservation setup in an end-to-end system. The initiator I_1 was the first to reserve the multicast path all the way towards the source node P. Afterwards, when I_2 and I_3 joined the same multicast group, their paths were terminated at their respective attachment points to the existing distribution tree.

Setting up this kind of reservation in an end-to-end system is simple enough: routers just need to be aware of multicast reservations (which is not an issue, given the fact that multicast uses a specific IP address range) and merge them where applicable. However, deleting paths is a different matter. Assume I_1 wanted to leave the multicast group and tear down its respective reservation. Without special consideration, this would break the distribution for I_2 and I_3 as well, since they rely at least in part on a path not controlled

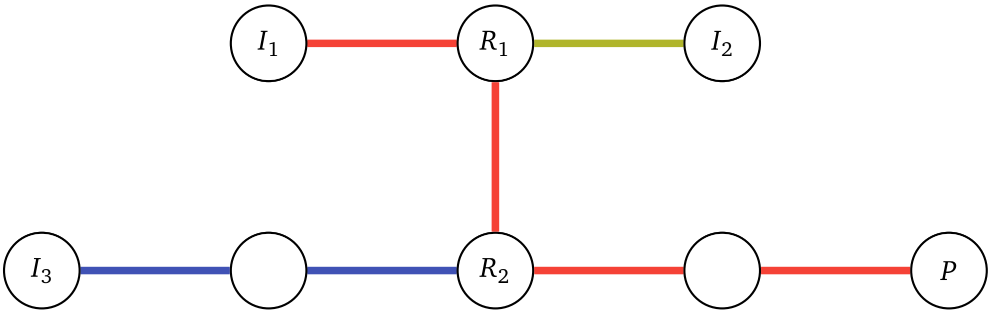

Figure 4.5.: *Multicast reservation with end-to-end and partial paths*

by themselves. Routers would need to detect this situation and transfer ownership of the still required path segments to another initiator. This could either be achieved by taking over the control within the router itself or handing it over to one of the remaining initiators (in this example this would preferably be I_2, as it is the only user of the segment R_1–R_2). In any case, this incurs additional signaling to synchronize the change in the network.

Realizing the same structure in a hop-by-hop fashion simplifies things considerably. Figure 4.6 shows the same setup using this approach. Each path segment is controlled by its originating router. The end-to-end paths are formed by linking individual segments together. For multicast support, multiple segments may be linked to a single one (as demonstrated, for example, by linking the red and blue segments to the yellow one). Reservation setup is just as complex as in the end-to-end approach: routers have to recognize multicast reservations and merge them where appropriate. However, tearing down a reservation becomes much simpler: routers just need to remove the links between the respective path segments, forwarding the deletion request only if the last link for an upstream segment was removed. No additional signaling to transfer ownership is necessary.

4.3. Signaling transport

KASYMOSA QoS's signaling transport subsystem should fulfill the following requirements:

Minimize signaling overhead In line with the overall goal to minimize protocol overhead, the transport subsystem should not introduce more signaling than absolutely

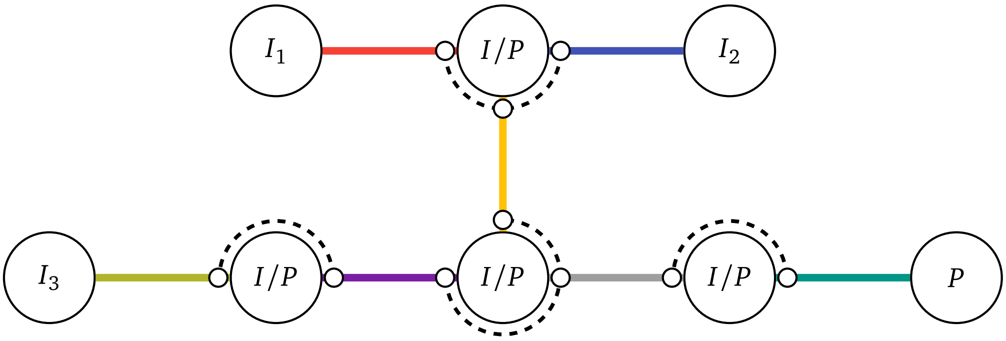

Figure 4.6.: *Multicast reservation with hop-by-hop paths*

necessary. In the ideal case, each transmission of a request requires only two messages: the request and an answer.

Provide robust transport The signaling subsystem rests on the assumption of a robust message transport. The transport system should be able to cope with message losses and duplications. Due to the specific nature of the signaling application (related messages are infrequent and self-contained), order guarantees are unnecessary. The system can reasonably expect to have a signaling transaction finished before starting a new one.

Low transport delay Signaling interactions occur at random points in time and are short in nature. The transport system should be able to forward them with the minimum delay. It should not entangle independent signaling requests: message loss for one request should not incur any queuing delay for unrelated requests that happen to be taking the same forward path.

Integration of legacy networks The signaling subsystem must not rely on specific features of the underlying network. In particular, it needs to be able to cross legacy systems without QoS support. The transport system should, however, be able to detect such systems and indicate their presence to the signaling subsystem.

4.3.1. General Internet Signaling Transport

The General Internet Signaling Transport protocol, as described in section 2.3.2.1, seems like the canonical answer to all signaling needs in a system like KASYMOSA QoS. The protocol offers one-shot, stateless signaling, as well as a permanent signaling overlay between neighboring NSIS entities. The former, achieved by using *Datagram mode*, is intended for short-lived signaling needs, especially when integrating mobile nodes into the network. The more long-term *Connection mode (C mode)* creates an overlay of TCP connections to transport signaling messages with delivery and ordering guarantees. GIST is able to cross legacy networks by using a specific forwarding mode (*Query modeQ mode*, an adaptation of the Datagram mode) whenever no routing state exists. In Q mode, the entity sends messages directly towards the signaling target (e.g. QoS path endpoint) with the router alert option header set. GIST-capable routers along the way intercept those packets and interpret them, setting up routing state as necessary.

Due to the design of TCP as a transmission protocol, signaling messages can only arrive at a neighbor in the order they were transmitted. A loss of a TCP segment of one message will lead to all subsequent messages being held up until the retransmission of said segment is successful. As GIST, in its typical mode of operation, transmits all messages between neighboring nodes over the same TCP connection, a message loss even affects unrelated signaling operations. This issue can partially be alleviated by the use of SCTP as the underlying transport protocol [FDC11]. Instead of a single TCP connection, RFC 6084 specifies the use of an SCTP association in C mode. Leveraging the multi-stream capabilities of SCTP, e.g. by assigning different NSLPs to different streams, GIST is able to isolate signaling flows of those applications against each other. The RFC does not mandate an option to separate individual signaling message exchanges (e.g. individual reservations) within the transport layer. SCTP could support guaranteed delivery without order guarantees (and therefore solve the stalling issue) on the signaling message level. However, this would require changes to the existing specification, to provide the necessary guarantees beyond the NSLP level.

GIST can also operate in Q mode (or D mode, respectively) only. Messages are then sent as individual UDP datagrams, and do not suffer from the stall problem presented above. However, the main advantage of C mode, guaranteed delivery, is lost in this mode of operation. The QoS application will have to ensure retransmissions on its own.

Given these issues, there is no benefit to implement KASYMOSA QoS as an NSIS QoS-NSLP within the context of this research. It would create significant effort without appropriately simplifying the implementation and evaluation of the system. However, when pushing the system from research to production, the adaptation to this standard protocol could be beneficial. Future work should therefore include an analysis of the exact nature of the adaptations required, both on the KASYMOSA QoS and NSIS side.

4.3.2. KASYMOSA QoS transport layer

KASYMOSA QoS implements an own signaling transport tailored to its specific use case. Combining minimal interactions with reliable message transmission, it tries to provide the best solution for the envisioned operation environment.

4.3.2.1. Reliable message forwarding

The transport layer forwards messages without any preceding handshake, to keep signaling delay low. This approach precludes transport protocols like TCP or SCTP, which have a connection setup phase. The protocol of choice for connection-less transmissions is the User Datagram Protocol (UDP), similar to GIST D/Q mode. UDP does not offer any delivery guarantees, leaving any possible reliability algorithm to the application.

In order to implement reliable transmission, while still limiting the amount of messages transmitted, KASYMOSA QoS closely integrates the acknowledgment/retransmission mechanism with the QoS protocol itself. The system operates in a request-response manner, with each QoS request being answered by an appropriate response, be it successful or not. The reliability layer exploits this protocol design and assumes the individual request as the basic unit of retransmission (as opposed to a message as protocols like TCP would do). Requests can be acknowledged within the normal protocol operation by their respective response (e.g. a successful reservation as a reaction to a reservation request) or, where necessary and appropriate, using the special return code PENDING. The latter enables a router to acknowledge the successful reception of a request, while indicating that a higher-layer protocol operation will take additional time (e.g. reserving a specific resource which might take longer to acquire).

Retransmission is triggered by a timeout. The determination of actual timeout value is outside the scope of this work. Routers should choose their retransmission timeouts based on information they may possess about the future path of a request (e.g. longer timeouts if a request has to cross a satellite link), as well as any history information (e.g. any prior retransmission attempts for a specific target and their timing behavior).

While this approach tightly couples the transmission protocol with the QoS layer, it has the benefit of removing additional protocol messages, as would be transmitted by TCP or SCTP. In the standard use case, where messages are successfully transmitted, this system uses exactly the minimum amount of messages possible, one request and one reply, while still maintaining a delivery guarantee in the case of an unreliable link.

The transport layer does not implement a congestion control mechanism, such as exponential back-off. Signaling messages are sufficiently infrequent to render such mechanisms unnecessary. The protocol also does not implement segmentation mechanisms like TCP, tailoring its messages to the path MTU. While relying on IP for segmentation can lead to increased message loss (due to the individual IP packet loss rates adding up, when a signaling message is segmented), signaling messages are typically small enough, to stay below the required minimum MTU for IP anyway. Future research is needed,

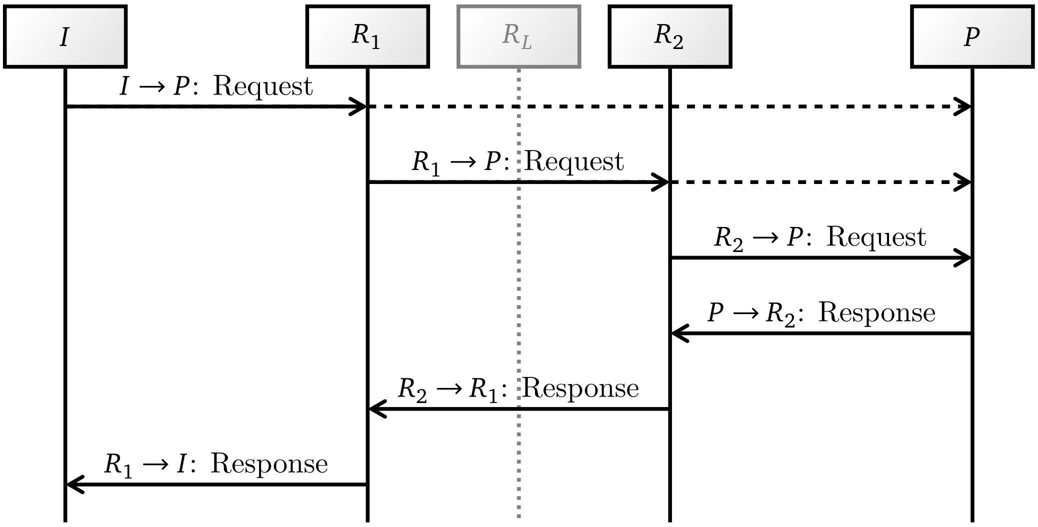
I
R_1
R_L
R_2
P
$I \rightarrow P$: Request
$R_1 \rightarrow P$: Request
$R_2 \rightarrow P$: Request
$P \rightarrow R_2$: Response
$R_2 \rightarrow R_1$: Response
$R_1 \rightarrow I$: Response

Figure 4.7.: *Example of a reservation process in the KASYMOSA QoS transport layer*

to assess, whether these assumptions hold for links with a higher load. In these environments the system will have to handle more and, due to request aggregation, larger signaling messages.

4.3.2.2. Neighbor discovery

KASYMOSA QoS's neighbor discovery algorithm is inspired by the Q-mode of GIST. Request messages are addressed to the final target (i.e. the path's endpoint) and are routed using normal IP routing. QoS-capable routers along the way intercept those messages, interpret them as needed, and send on the appropriate requests towards the target. Figure 4.7 shows an example of a request traversing two routers. While the forward messages are always destined for the peer P, return messages are addressed at the node that initially sent the request.

By addressing forward requests to the target node, the system is able to transparently bypass legacy routers (as depicted in figure 4.7 by the grayed-out router R_L). These systems will simply not intercept the message and forward it via the usual IP routing.

Bypassing an intermediate router could lead to a violation of the QoS guarantees. Figure 4.8 depicts a situation, where an QoS-incapable interior router of a network could lead to such a violation. If the reservations p_1 and p_2 exceed the capacity of the link R_L——R_E, the legacy router R_L will not be able to detect this (as it has no QoS capabilities

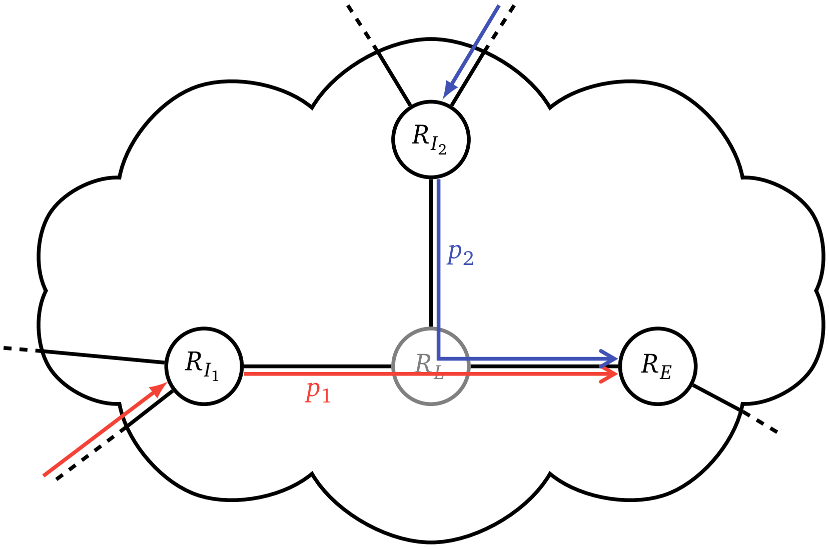

Figure 4.8.: *Possible collision of two reservations on a legacy router*

whatsoever). In this case, packets are forwarded by R_L in a best-effort manner, which will invariably lead to packet loss and queuing delays for both reservations.

Detecting and resolving such an issue is outside the scope of this work. A possible solution could involve routers sending signaling messages with a fixed outgoing TTL/Hop Count field in the IP header. If the next intercepting node receives a hop count, which was decreased by more than one, there is a legacy node on the path which did not interpret the request. Routers can then either reject the request or just provide information to the client about the best-effort path segment.

4.3.2.3. Last Node Behavior

When integrating a new QoS architecture into a legacy network the system can encounter a situation, where the last QoS-capable node of a path is not the intended end-point. However, KASYMOSA QoS relies on the last node of a path to return a response to a request to trigger the overall response process (see figure 4.7 for an example of a message sequence chart. Here, P triggers the response process towards I by responding to the last request sent by R_2). If such a response is not given, the protocol will fail a reservation at some point, effectively preventing the integration of legacy nodes as peers.

NSIS QoS-NSLP [MKM10] discusses this problem in the context of its topology change detection. The specification gives three distinct cases for the last node of a signaling path:

1. The last node is the intended receiver of the signaling request. This is the case presented in figure 4.7. The protocol operates as normal, no special handling is necessary.

2. The last node is configured as a proxy for the actual signaling target. This case is a special case of the "legacy router"-scenario. The last router simply terminates the path and indicates success to the initiator.

   If the intended reservation is for a data flow *towards* the peer *and* the reservation stops just one hop short of the target, the resulting reservation setup is actually indistinguishable from a normal signaling process. In this case the last resource reservation would be at the outgoing interface of the last router towards the peer anyway, something which can easily be achieved.

3. The last node is not explicitly configured as a proxy for the actual signaling target. If a router discovers that there are no more QoS-capable nodes towards a target, it assumes the proxy role for the target. The actual detection of this condition is not part of this work. The transport system could interpret network management messages like ICMP "Port Unreachable" (which would be sent by a standard target system upon reception of a UDP datagram for a port, which is not open) to detect availability and QoS-capability of the intended receiver.

In KASYMOSA QoS's use case, the primary goal is to assign satellite transmission resources to the requested paths. In such a scenario, the receiving satellite terminal could be configured as a proxy for all but the known QoS-capable end systems. While this would effectively terminate each QoS path before even entering the Internet at large, it would most likely offer the required QoS. Satellite resources are very scarce compared to resources in standard infrastructure networks. The bottleneck of a path would therefore most likely be on the satellite link, something which could be effectively handled in this configuration.

4.3.2.4. Asymmetric routes

KASYMOSA QoS, as presented here, cannot operate in an environment with asymmetric routes. Reservations are made hop-by-hop with the responses being transported back via the same link. Figure 4.9 depicts a case in which the routing breaks the signaling process. A reservation p_1 is forwarded to router R_2. At some point in the process (after reserving local resources and potentially creating further signaling messages not shown in the figure) R_2 sends a reply message to R_1. Due to the routing setup of the network this messages would have to travel along the path $R_4 \to R_3 \to R_1 \to R_2$. The QoS subsystem at R_4 intercepts the message and drops it because from its point of view it received a reply to a request it never sent. The message is never forwarded along the dotted parts of the return path.

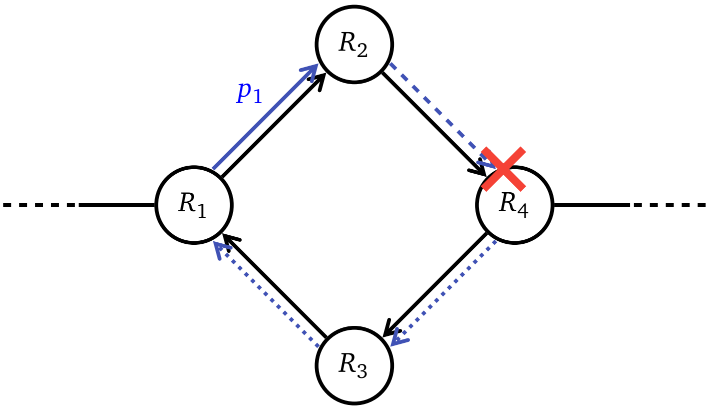

Figure 4.9.: *Broken routing of a reply message*

Even if all routers in this scenario forwarded the reply, until it reached its intended recipient, KASYMOSA QoS would have other issues in such an environment. The system is able to signal reservations for data flows, that travel in the opposite direction of the signaling flow. In an asymmetric routing environment, this would lead to resources being reserved at the wrong routers. Figure 4.10 illustrates the issue: the reservation p_1 originates at the initiator I and travels along the blue path towards the peer P. Along the way individual path segments are reserved according to the algorithms presented in this work. As the reservation is actually intended for a flow, that travels from the peer to the initiator (e.g. the downlink portion of a TCP connection), the actual data flow d_1 is decoupled from the reservation flow. Consequently, routers R_3 and R_4 will have no resources assigned to the reservation, while R_2 will reserve resources for a data flow it never forwards.

This issue could be tackled by introducing a two-phase reservation protocol. Reservations for paths in the direction of the signaling flow would be reserved in the forward phase. Once the signaling process reaches the last node on the path, it enters the reverse phase. Any reservations in the direction from the peer to the initiator are reserved in this phase. Future work will have to look at the exact implications of this approach on the signaling process, the number of signaling messages and possible changes to the

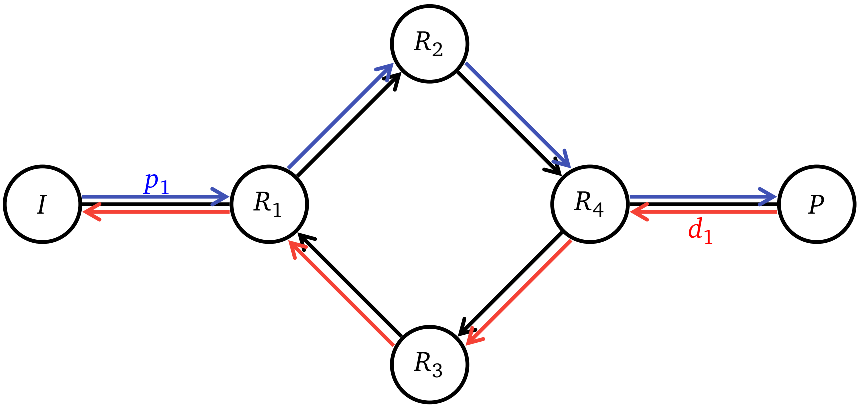

Figure 4.10.: *Wrong path coupling in an asymmetric routing setup*

signaling protocol itself.

One last issue in an asymmetric setup stems from the concept of related paths. When related paths take different routes through the network, additional signaling might be necessary to synchronize their state and correctly calculate the state of their relations. Again, the exact implications on convergence time, wasted resources or required signaling are outside the scope of this work and have to be investigated in the future.

4.3.2.5. Routing cycles

Due to the distributed nature of the routing system in the Internet, configuration errors can lead to cycles in the routing graph. The architects of IP anticipated that possibility and designed a mechanism, to gracefully fail in such a situation, instead of continuously forwarding packets. Each IP packet carries a Time-to-Live (IPv4) or Hop Count (IPv6) field in its header. This field is initialized to a specific value when creating the packet and decremented at each router the packet passes. Once the field reaches zero, the packet is discarded and an appropriate error message is sent to the originating node (e.g. ICMP Time Exceeded for IPv4).

The transport system of KASYMOSA QoS effectively undermines this mechanism. Transport packets are intercepted at each QoS-capable router, their requests are interpreted and new transport packets are created where necessary. The TTL is reset each time, preventing IP from detecting loops in the forwarding graph. As requests are aggregated into forwarding messages without regard to their history (e.g. whether they arrived in a message together), a common Hop Count for a set of requests is impossible in any case. The basic forwarded unit is the request, not the message. Therefore, cycle detection has to take place on this level.

The actual reaction of the system to such a cycle depends on the value of the *Collision Policy* of a request:

Reject A routing cycle will invariably lead to a node receiving a request with the same filter twice. This collision leads to a rejection of the request, preventing it from traveling any further. This is the correct behavior, albeit unintentional due to a collision rather than a cycle detection.

Merge Exact If a "Merge Exact" request returns to a node, it has already been requested from, the system will basically merge it with itself (the system has no means of detecting the duplication, as the request, even though it shares filter and parameters with it's existing copy, will have a new unique identifier). Merging a request with itself leads to a dead-lock in the protocol. The newly arrived request waits for the request it was just merged with to finish. This request in turn waits for its forwarding chain to return, the very chain, that ends with the request just merged. As both requests are therefore transitively waiting for each other, the protocol dead-locks. At some point, this dead-lock will be broken by a timeout, failing the request. This is again the correct result, albeit with a very long delay and potentially superfluous retransmissions of requests.

Actively detecting route cycles can speed up error detection, specifically in the case of request merges. Simply attaching a Hop Count field to each request (as IP does with its packets) will not solve the routing cycle issue. It will not change the behavior in the Reject-case at all. In the Merge-case the behavior will depend on the exact hop count value and the length of the cycle. If the cycle is shorter than the remaining hop count, the system will display the same erroneous behavior as without the counter. However, it will add the issue of selecting the correct maximum hop count. Too low, and requests will not go through at all, too high, and the mechanism is rendered useless for the merge case.

To detect cycles, no matter their length, KASYMOSA QoS could use a distributed marking algorithm. If a router receives an unmarked request, it attaches a random mark value. Routers record all marks of requests passing through them. Should a router detect a filter collision between two requests that have the same mark value, but different identifiers[f], it rejects this request with a ROUTE CYCLE DETECTED error code. Mark values are deleted, when either the request is finally answered or based on a timeout.

[f] The "different identifiers" requirement distinguishes requests that returned via a cycle in the routing graph from duplicates sent by another router. Simple retransmissions of a request will share the same identifier

4.4. Security considerations

As any other protocol installing state in the network and influencing the forwarding of traffic, KASYMOSA QoS has significant security implications. This section discusses the main issues, without being a thorough security analysis.

Denial-of-service via parameters The most basic attack against nodes sharing a path with an attacker is the reservation of a high-priority, high-volume path (or multiple low-volume ones), effectively eating up all available resources. Once a path is admitted and used, especially if it cannot be easily preempted due to its priority, legitimate requests will be rejected due to insufficient resources.

To prevent such an attack, service providers have to specify what constitutes a legitimate reservation and how much resources each partner is allowed to request. These questions go beyond the mere technical implementation of a QoS system, into the realm of Service Level Agreements.

Denial-of-service via filter The filter object controls which traffic to apply the QoS to. By using appropriate filters, malicious systems could effectively mount denial-of-service attack against other systems or networks. If a malicious system shares at least one node on a path to a target with the system under attack, it could conceivably disrupt the traffic flow by simply forging the source/destination addresses in the filter object and requesting a very low QoS. If the attacked system has no colliding filter already in place, the request would go through and disrupt any best-effort transmissions, as well as preventing possibly legitimate future requests from succeeding. The closer the shared node is to the target system, the greater the damage, as more and more parts of the Internet could effectively be “disconnected” from the point of view of the attacked node.

Due to KASYMOSA QoS’s segmented structure making it common for one system to reserve resources on behalf of another, this type of attack can only be prevented at the network edge. By verifying, that the requesting system is actually allowed to act for the source and destination addresses specified in the filter object, routers can stop malicious requests from entering their domain. How the necessary trust relationships are established and what additional information (e.g. request signatures forwarded throughout the path) might be needed is beyond the scope of this analysis.

Path disruption by spurious suspend/delete messages In its current incarnation all routers in the QoS system trust each other to only send signaling for resources they actually have requested or serve. By injecting spurious suspends or deletes, an attacker could disrupt legitimate reservations by other users. KASYMOSA QoS uses random 128-bit UUIDs as reservation identifiers. These IDs are not disclosed beyond the involved systems. An attacker would therefore have to intercept the appropriate messages or guess their UUIDs, making this attack highly unlikely.

Denial-of-Service on the signaling layer By simply sending massive amounts of signaling messages into the system, an attacker can delay or even disrupt the execution of legitimate requests. This attack is nearly impossible to defend against, especially if it occurs from multiple sources (distributed DoS). Its effects can be somewhat alleviated by limiting the amount of signaling messages, an individual node can inject into the system. However, since the handling of reservations is a rather expensive process for the router (possibly triggering resource acquisition and optimization processes), even low-volume attacks from multiple attackers may quickly overload the core network without being detectable at the network edge.

Information disclosure The concrete structure of communication links in a system can be sensitive information in certain use cases. Reservation meta-data could reveal information about communication (and therefore organizational) structures or pattern (when communication takes place) and serve as a basis for more targeted attacks. Malicious systems can effectively "probe" for existing reservations by purposely creating colliding filters and evaluating the network response. Depending on the level of detail (e.g. "Do systems A and B have any connection at all" vs. getting all the individual reservations), even a low-volume probe could reveal the necessary information.

Similar to the denial-of-service via the filter object, this attack can only effectively be defended against at the network edge. By clearly specifying and enforcing which system is able to create what type of filter, edge routers can stop malicious requests from entering the network.

This analysis is by no means exhaustive. It highlights the main areas of interest for a detailed security analysis and tries to give hints to possible countermeasures. Securing the protocol is beyond the scope of this QoS-focused work.

5. Evaluation

This work claims that KASYMOSA QoS is well suited for long-delay links with variable capacity. The evaluation first establishes a set of parameters in section 5.1 which define "well" in the context of this work. Results of the analysis and the conclusions drawn from them, are presented afterwards in sections 5.2 and 5.3.

5.1. Performance indicators

To evaluate the improvements of KASYMOSA QoS over existing reservation-based solutions, the following performance indicators are measured:

Wasted bandwidth The central claim of this work is that a misalignment between protocol-intrinsic flow relations and the underlying reservation structure causes transmission capacity to be wasted on reservations that cannot actually be used by applications. A key indicator for the performance of the system is therefore the amount of wasted bandwidth compared to existing solutions.

Remaining sessions The fundamental goal of a QoS-system is to transmit as much traffic as possible, in compliance with its requested parameters. In cases of broken paths, applications might either stop sending traffic or not receive their desired QoS when transmitting via Best-Effort. In both cases, the performance on related, still active paths may be impacted (e.g. the remaining path in a TCP connection).

This parameter is closely related to the wasted bandwidth (i.e. remaining sessions go down if wasted bandwidth goes up). It is still worth looking at those parameters separately. Even with zero wasted capacity, a system will not always be able to fully serve all QoS traffic as intended. If the available capacity decreases, some reservations will have to be suspended, no matter whether a systems is relation-aware or not. By keeping relations intact, the system can expect to impact fewer application protocol sessions (i.e. relations) than without any regard to path dependencies.

Signaling overhead The need to transmit signaling information puts further strain on potentially already limited resources. A QoS system is therefore better if it requires less signaling overhead, both in terms of messages and total bytes transmitted, to achieve the necessary state changes. As the amount of bytes transmitted is highly dependent on the actual encoding of the signaling protocol, this work focuses on the message overhead alone. Messages tend to be small, so the dominating factor for the signaling overhead is their number instead of their actual size.

Convergence time As a system with distributed state, KASYMOSA QoS requires synchronization effort in the case of changes. Especially in long-delay networks like satellite communication systems synchronization always incurs a noticeable time overhead. A lower convergence time provides applications with faster start-up for their reservations as well as quicker recovery after changes in the environment. This parameter has a large influence on the overall performance in terms of wasted resources, since a quicker synchronization lowers the amount of time resources are wasted due to state inconsistencies.

5.2. Efficient resource allocation

An efficient allocation of the available resources to the reservations present is key to the performance of the system. One of the main assumptions of this work is an improvement of the allocation quality in terms of less wasted bandwidth and more remaining sessions over existing solutions without a relation model.

In a full-fledged system, both parameters are highly dependent on parameters like round-trip time or client behavior[a]. To eliminate these influences, the resource allocation quality is evaluated in isolation.

The simulation assumes a competition of all known paths for the same variable resource (e.g. overall satellite capacity). 100% of the available capacity are assigned to reservations[b]. After limiting the capacity and optimizing the remaining bandwidth allocation, resources will, to a certain extent, be assigned to reservations which cannot be used due to their relations not being fulfilled. A relation-aware system should exhibit significantly less of this wasted capacity and in turn be able to serve more QoS traffic[c]

[a] The wasted bandwidth over time depends on, among other things, the question: when, if at all, does a client detect a broken path setup and how does it react to this state? A client waiting for the missing resource to return will waste more bandwidth, over time, on potentially remaining paths, than one immediately giving up its related reservations. However, it will save signaling overhead, by not sending any change requests in the case of short resource outages.

[b] This is achieved by creating a random set of initial reservations and their respective relations and assuming the sum of their resource requirements as the total capacity.

[c] Barring any relation conflict, an ideal optimization algorithm should never waste any bandwidth at all.

than its relation-unaware counterpart.

To evaluate the applicability of the system for different operation environments, the following parameters are varied in the simulation:

Remaining capacity KASYMOSA QoS is designed for environments with highly variable link capacities. Depending on the severity of the capacity change, the relation-aware system may change its performance in relation to a dependency-unaware implementation. The expected outcome is a widening performance gap with more severe capacity losses for both wasted bandwidth and active session.

Number of reservations A smaller number of reservations will inevitably remove degrees of freedom from the optimization algorithm. Disabling one session out of 10 will free up a greater portion of the overall capacity than disabling 1 out of 100. On the other hand, this will make it harder to allocate as much of the remaining bandwidth as possible, simply because the "offcuts" will be larger. This is especially an issue for a setup with many "mutually dependent"-relations, which can only be enabled or disabled as blocks. The overall performance in terms of active session should therefore be somewhat impacted for relation-aware systems.

Relation complexity Relations bind multiple paths together to be treated as a unit. The larger these units get, the higher the probability of a relation-unaware system making mistakes and rendering sessions unusable for their respective clients.

Relation type KASYMOSA QoS is capable of expressing a wide variety of relations. In order to estimate the influence of the relation type on the overall performance, four different types of relations are evaluated: *Mutually Dependent*, *Dependent*, *Exclusive*, and *AtLeastN*, all as presented in section 3.3.2.

Value variance Section 3.3.3 presents some issues with mixing reservations of different priorities (and therefore ultimately different intrinsic values) in a relation. Varying this parameter gives insight into the influence of these modeling issues, especially with respect to different types of relations.

Table 5.1 summarizes the value space for the parameters described in this section.

In order to assess the improvements of a relation-aware system over existing solutions, the simulations are carried out using three different implementations:

Path-based Optimization (PO) The optimization process is based solely on the value of individual paths. This is akin to how RSVP-based systems handle resource changes.

Parameter	Range
Remaining capacity	10% - 90% initial capacity
Number of reservations	8 - 512 reservations
Relation complexity	2 - 20 reservations
Relation type	*Mutually Dependent*, *Dependent*, *Exclusive*, *AtLeastN* (with N being half of the reservations in the relation)
Value variance	$1 \leq v_{max} \leq 255$ is the upper bound of a uniform distribution in $[1, v_{max}]$ from which reservation values are drawn at random

Table 5.1.: *Simulation parameters for evaluating wasted bandwidth and remaining sessions*

Correlated Time (CT) The "Correlated Time" optimization, as a simple improvement over the purely path-based approach, prefers to keep paths, that were signaled at the same time, together. Without a formal relation concept, this approach implicitly assumes paths from the same signaling process to be dependent on each other. For optimization purposes, paths are first ordered by their intrinsic value (as defined by the transfer function) and, if they have the same value, then correlated by their arrival time (derived from the signaling packet they arrived in). This approach should be effective to model mutual dependencies (e.g. TCP sessions). It should fail to provide adequate allocation quality for other types of relations.

Relation-Aware (RA) The relation-aware system implements the full modeling power as presented in this work. It should be able to totally avoid any wasted bandwidth and provide a benefit over the two reference systems.

5.2.1. Mean wasted capacity in all scenarios

Analyzing the mean of the wasted capacity over all scenarios gives a first glimpse on the benefit, relations are able to provide. Figure 5.1 shows the percentage of the remaining capacity which is wasted for different adaptation scenarios. The figure does not show the Relation-Aware scenarios, as those are not wasting any resources at all. Their respective figures would be horizontal lines at $y = 0$.

The figure clearly shows the possible gains through application of a relation system. On average, relation-unaware systems waste between 50% and 70% of the remaining capacity.

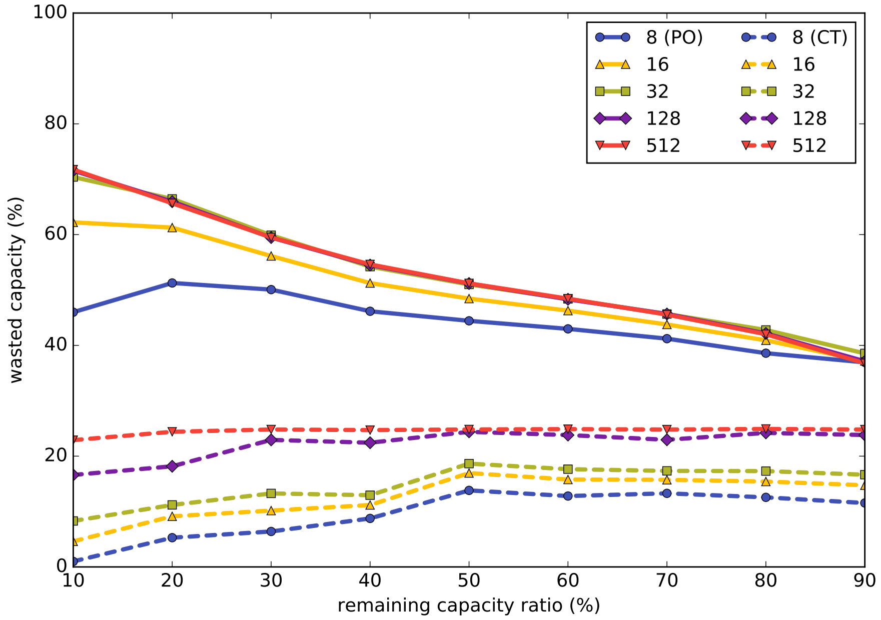

Figure 5.1.: *Wasted capacity as a fraction of the remaining data rate for different problem sizes. (The 128- and 512-paths scenarios for Path-based Optimization nearly coincide, hiding the purple 128-path curve.)*

For all reasonably large scenarios, the system behaves identically. All lines in the graph match closely. An exception is the 8-path scenario. In this case, according to the results, the wasted capacity decreases by 25 percentage points when compared to larger setups. This is not an inherent improvement in performance for very small setups, but rather an artifact of the way wasted capacity is calculated. It is defined as the sum of all paths, which are part of an invalid relation:

$$c_w = \sum_{p \in Q_w} c_p$$

and

$$p \in Q_w \leftrightarrow s_p = 1 \wedge (\exists r \in R : r = 0 \wedge p \in F_r)\}$$

What is missing from this picture is the *unassigned capacity*: "cutoff" capacity that cannot be assigned to any of the remaining paths. When the number of paths is low, each individual path is of significant size compared to the overall capacity. If a path cannot be fit into the solution, it is therefore likely to leave a bigger portion of the remaining capacity unused. Increasing the number of paths lowers the relative size of this gap. Figure 5.2 shows unassigned capacity for two select scenarios: the smallest one with 8 paths and the largest with 512 paths. The solid line again indicates the wasted capacity. The dashed line, on the other hand, shows the sum of the wasted and the unassigned capacities, effectively displaying the portion of the remaining data rate that cannot be used for QoS-protected traffic. The 8-path scenario clearly exhibits a much larger portion of unassigned capacity. For the 512-path scenario, the unassigned capacity is absolutely insignificant, as it is within less than 1/50 of the wasted capacity[d]. Looking at the overall unusable capacity, the two scenarios are much closer together and the perceived advantage of smaller scenarios disappears entirely.

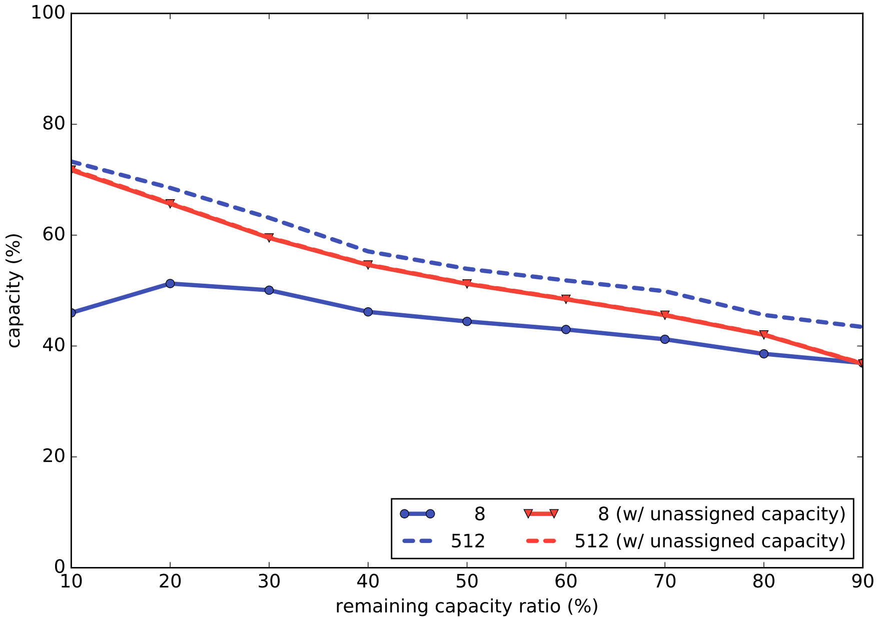

Figure 5.2.: *Wasted and total unused (including unassigned) capacity as a fraction of the remaining data rate for different problem sizes. This graph exemplifies the increasing influence of unassigned capacity due to a packing mismatch for very small scenarios. For 512 paths, the cutoff becomes negligible to the point of hiding the corresponding line in the graph.*

[d] At most, the cutoff is just smaller than the smallest disabled path (otherwise that path would be enabled). In the worst case, of 10% remaining capacity, at least 51 paths will be enabled (typically more, as the system tends to prefer smaller paths with the same priority), making the cutoff smaller than the average of the remaining paths.

After establishing in figure 5.1, that all reasonably large scenarios behave the same, all the following analyses will be based on a 512-paths setup.

5.2.2. Remaining Sessions

Wasting less capacity is just one part of the story. From the point of view of user satisfaction, it is much more interesting to look at how many sessions the system is able to support in a constrained situation. These two parameters are not fully orthogonal, as wasting bandwidth diminishes the capacity for keeping sessions online. They are not fully aligned either, as even a perfect resource allocation algorithm needs to disable paths in an overload situation. Figure 5.3 shows the overall performance of all three optimization algorithms over a whole range of adaptation scenarios. RA clearly outperforms its counterparts, at times keeping as many as twice the number of sessions active. Interesting to note is the ability of RA to keep 40% of the sessions active with only 10% of the capacity remaining. This is due to the algorithm preferring sessions that are below average in terms of their capacity. The 10% remaining capacity are based on the average capacity for a path. In the original distribution, half of the paths are, by definition, taking up less space than this average. As both the required capacity and path value are randomly assigned and therefore uncorrelated, there exists a certain set of high value, low capacity reservations, which fit into the remaining capacity, driving up the number of active sessions.

Figure 5.3 does not include the results of the Exclusive relation. The behavior of this relation is sufficiently different to skew the results unjustly in favor of the relation-aware approach. To understand why that would be we need to take a more detailed look at the results for each individual relation type.

5.2.3. Relation type

Figures 5.4 and 5.5 display a significant difference in performance for different types of relations. While both figures show a clear performance benefit of a relation-aware system, they also reveal the issue of optimizing towards the wrong goal in the case of the CT strategy applied to Exclusive relations.

From the perspective of a relation, Path-based Optimization randomly enables reservations in order to achieve the maximum value for the current capacity limit[e]. It is therefore expected that the Mutual Dependency shows dismal performance: there are

[e] Of course, the selection of paths to enable is not random at all, but purely based on their intrinsic values. However, as these values are randomly distributed across paths in a relation, the selection looks random from the point of view of said relation.

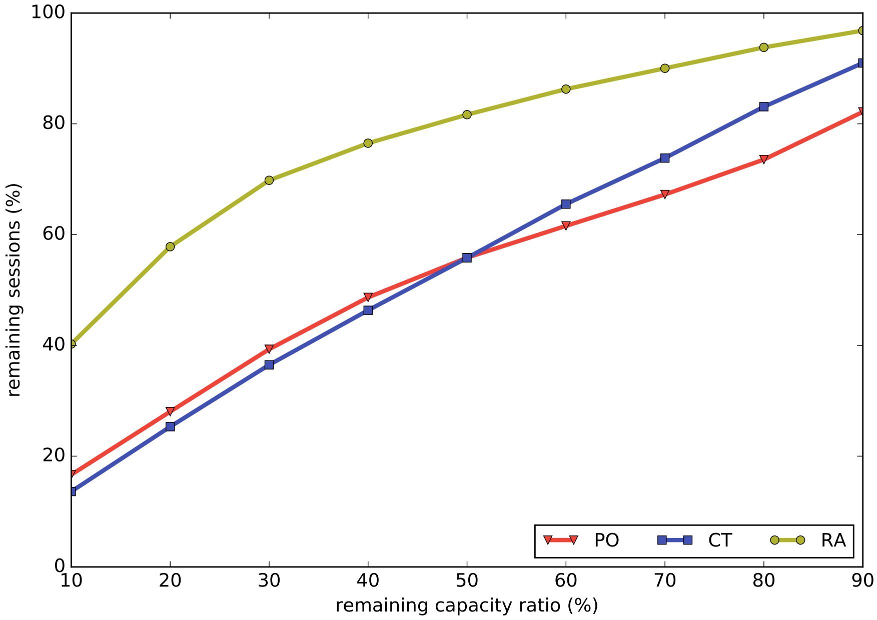

Figure 5.3.: *Percentage of remaining sessions as a function of the remaining capacity. This graph only includes the "Mutual", "AtLeastN" and "Dependent" relation types. The "Exclusive" relation is significantly different. See figure 5.7 for further details.*

only two out of 2^k (with k being the size of the relation) valid cases: all reservations are enabled or none are. The algorithm performs better for one-way dependencies: every case, where at least the required path is online, is a valid solution. This increases the amount of valid solutions to $2^{k-1}+1$ (2^{k-1} solutions with the dependency enabled, plus the null-allocation). Even better still, are the results for the AtLeastN relation. For each relation of size k there are $1+\sum_{i=k/2}^{k}\binom{k}{i}$ valid solutions, a number even bigger than $2^{k-1}+1$. When – from the perspective of the relation – randomly selecting solutions, the optimizer is therefore much more likely to encounter a valid solution, decreasing the resulting overall wasted capacity.

Correlated Time optimization treats every relation (i.e. paths that were reserved at the same time) as a Mutual dependency. It is therefore not surprising that it does not waste any capacity for either AtLeastN, Dependent or Mutual relations. The solution is already the optimal case for the Mutual Dependency. It is also a special case for AtLeastN and Dependent relations: both relations allow the null-solution, as well as the full solution (i.e. all paths enabled). Therefore, an optimization strategy producing only

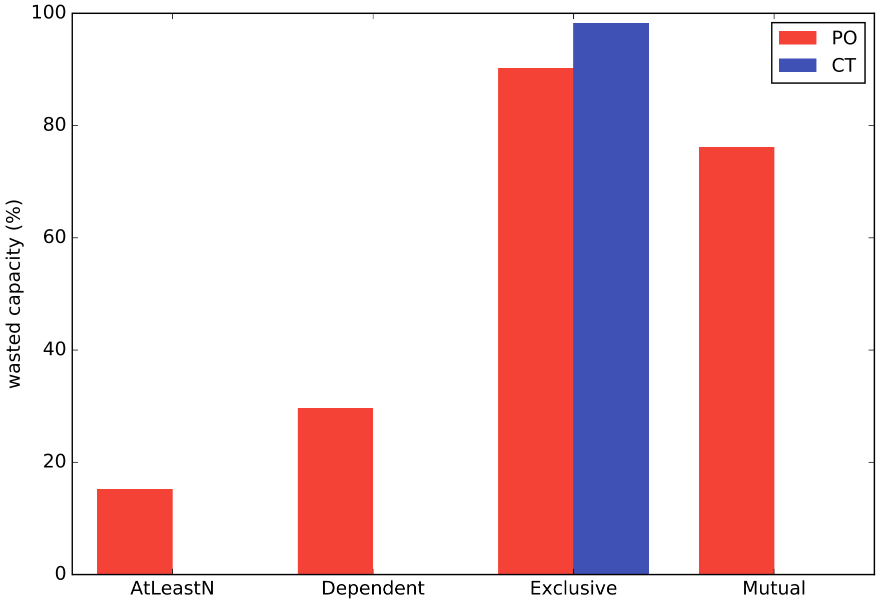

Figure 5.4.: *Overall wasted capacity for different types of relation. The Relation Aware result is not shown, as this strategy does not waste any resources.*

these two types of solutions might be unnecessarily restrictive (which will show in the "remaining sessions" result below), but at least not produce any wasted capacity.

Both strategies show their worst performance for the Exclusive relation. They try to enable as many paths as possible, a behavior which is exactly opposite to the optimization goal for that relation. The analysis here slightly overestimates the amount of wasted capacity: it counts any path in an Exclusive relation with more than one enabled reservation as wasted. The highest valued path could be viewed as not wasted at all because it can be used by the requesting application. The resulting allocation just uses more resources than strictly necessary.

To explore this idea further, we take a look at the remaining sessions for each relation type. The results depicted in figure 5.5 are as expected: relation types that lead to a high amount of wasted capacity result in fewer remaining sessions. There is simply less capacity remaining to distribute to valid relations. In any case, the Relation Aware system is able to retain the highest proportion of sessions after adapting to a diminished link capacity. Time Correlated performs the same for three of the four relation types, and, not incidentally, the same as RA for the Mutual dependency.

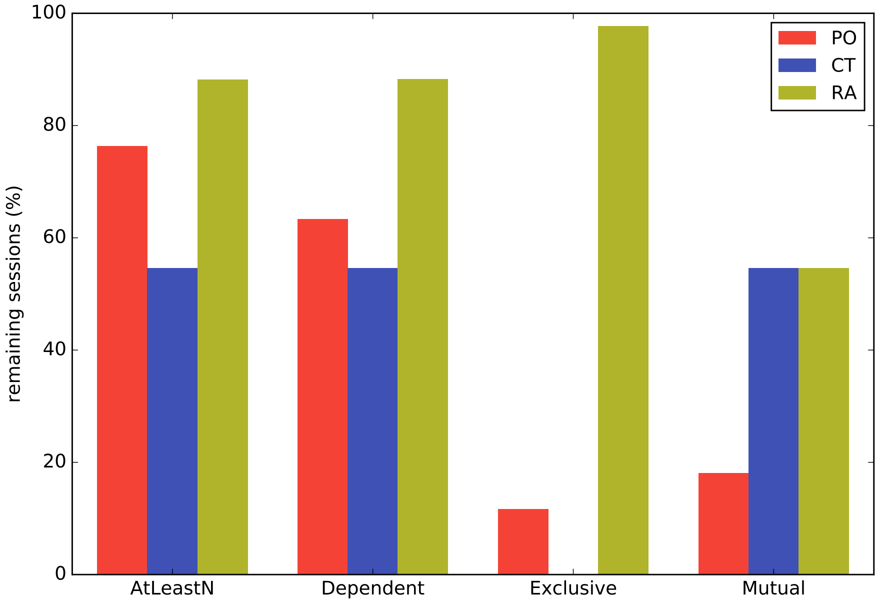

Figure 5.5.: *Overall remaining sessions for different types of relation*

Again, the behavior of the Exclusive relation stands out and warrants further investigation. A deeper investigation of the exact relation behavior for different adaptation scenarios in figure 5.6 shows a similar picture as before: the Relation Aware approach consistently outperforms Path-based Optimization and coincides with the Correlated Time results for Mutually dependent relations. The AtLeastN and Dependent relations allow the system to keep more relations online, even for severe adaptation scenarios, than the Mutual dependency. They simply offer more valid solutions to the optimizer. Instead of having to cut a whole session, the optimizer can gracefully degrade it by disabling single paths. This is especially relevant for the Dependent relation in severely constrained scenarios (below 25% remaining capacity). Where AtLeastN can only be scaled down to the value of N for a session, Dependent can be downgraded to a single path (the prerequisite path on which all others depend). This allows the system to keep many more sessions online. [f].

It is interesting to note the relative performance of PO and RA for AtLeastN and Dependent relations in medium to low adaptation scenarios. Both strategies keep more

[f] This is emergent behavior of the relation model in the optimization process. The optimizer does not take the number of remaining sessions into account at all.

AtLeastN relations online, although in theory the Dependent relation could be scaled down much further. This is due to the Dependent relation removing a degree of freedom by prescribing one specific path (the dependency) to be enabled. If this path is of low priority – the very modeling error described in section 3.3.3 – it may be disabled first by both strategies. PO works purely based on the path relation and is much more likely to disable a session, hence the huge performance loss for that strategy. However, even the Relation-Aware system will encounter the point where keeping a low priority dependency online is less beneficial than disabling it in favor of a high priority dependent path from another relation. RA will take the whole relation with it, freeing up more capacity for other sessions and therefore exhibit lower performance loss. The AtLeastN relation does not specifically mark out one path as important, giving the system more room to shift around capacity. This effect reverses for very severe adaptation scenarios, where the better scalability of the Dependent relation becomes relevant.

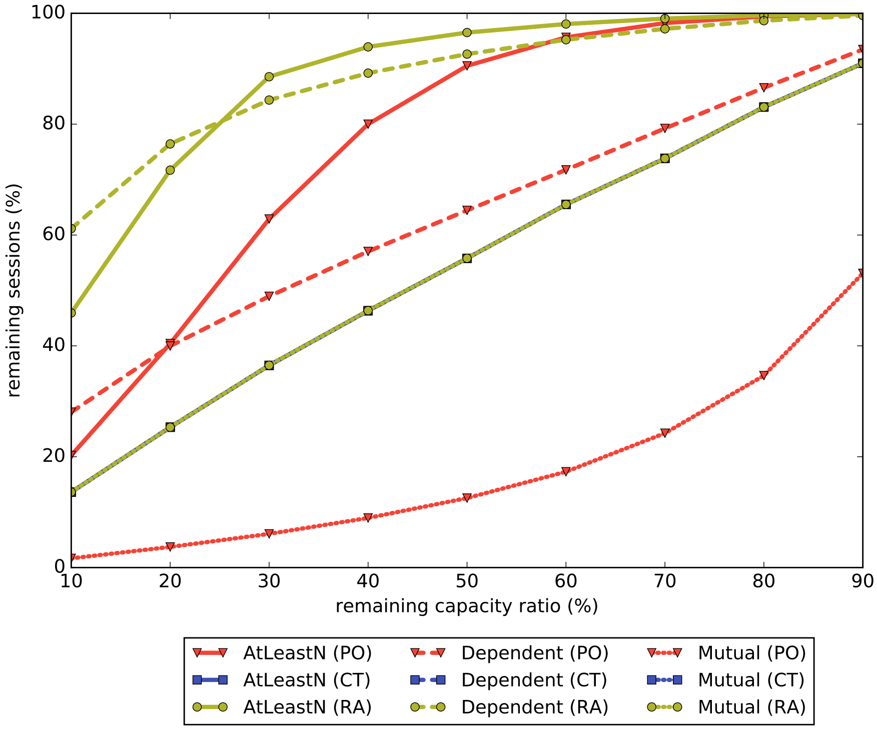

Figure 5.6.: *The optimization performance on different relation types. The Correlated Time (blue) approach coincides with the Relation-Aware optimization for the Mutual dependency and is therefore nearly invisible in the graph.*

For reasons of visual clarity the results for the Exclusive relation have been moved to figure 5.7. The graph shows an exceptionally good performance for the RA system and basically no performance for PO or CT. These results are slightly misleading, however. The simulation calculates the initial resource requirement by summing up all individual paths (as a relation-unaware system would do) and reduces the capacity based on that. As each relation only requires one out of up to 32 paths to be online, the system is able to support 100% of the sessions, even when adapting to quite severely constrained situations. The PO optimization, on the other hand, does not show any decent performance, even at high remaining capacity, precisely because the system was not build for this use case. It enables as many paths as possible, making the Exclusive relation invalid almost by definition (as this relation literally indicates "any one of them, but not more"). The CT optimization strategy performs even worse: because it implicitly assumes a mutual dependency between paths reserved at the same time, it optimizes for exactly the wrong target. Instead of enabling one path per relation, it enables all paths or none. Therefore, it does not produce any valid relations at all.

However, the relation validity under-estimates user satisfaction for the Exclusive relation. If the user did not care about the wasted capacity in the relation, she would still be able to use the best online path from each session and get her data transmitted. This significantly increases the performance figure for the PO system, although it still falls short of the relation-aware approach. CT also performs better in this category. However, its performance is worse than the PO strategy, as it either enables all reservations or none. With decreasing capacity, the number of relations where all paths can be enabled (and therefore the number of sessions containing at least one usable path) decreases as well.

5.2.4. Relation complexity

Looking at the behavior of the system from the relation complexity perspective yields the expected results: with increasing complexity, the wasted capacity increases and the remaining sessions go down. Figure 5.8 shows the wasted capacity again for the Path-based Optimization and Correlated Time approaches over all relation types. The CT result shows a nearly constant wasted capacity contributed exclusively, as it were, by the Exclusive relation.

Digging deeper into this result in figure 5.9 reveals nearly 100% wasted capacity for the Exclusive relation. The amount drops slightly with larger relation sizes. This is due to the algorithm considering the relation to be a Mutual dependency. It therefore either enables all reservations (which is *not* a valid solution for Exclusive) or none (which is valid for the Exclusive relation). With increasing relation complexity, the overall number

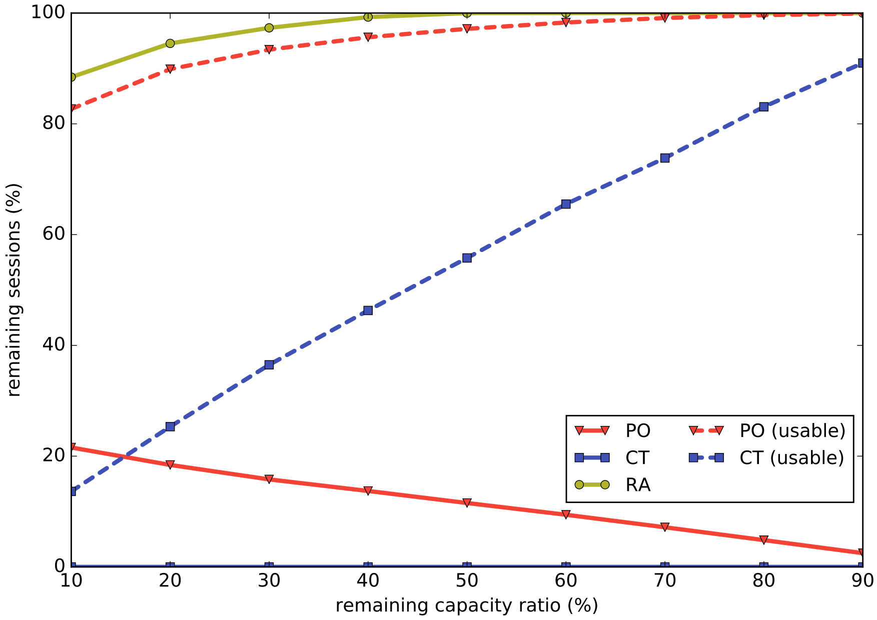

Figure 5.7.: *Performance of the optimization algorithm on the Exclusive relation. This use case is not supported without some kind of relation model and therefore performs exceptionally poorly. Dashed lines show the usability of at least one path in a session ignoring potentially wasted capacity within the same relation.*

of relations in a scenario goes down. Therefore, each relation takes up a larger chunk of the total capacity available. When adapting to a lower capacity, the “offcut” becomes larger: capacity that cannot be assigned at all because none of the remaining sessions fit. The non-wasted capacity in this picture is therefore not used to transport QoS-protected data, but rather stays unused in this specific scenario.

For the Path-based Optimization approach, increasing complexity also increases the amount of wasted capacity. From the point of view of the relation, the algorithm, again, randomly enables paths to fill up the available capacity. It has therefore an increasing probability to produce invalid result with increasing relation size.

Looking at the detailed results in figure 5.9 reveals the Exclusive and Mutual relations to be the dominating factors in the overall result. Both waste nearly 100% of the remaining capacity for larger relation sizes. Both relations have only two valid solutions out of 2^k (with k being the number of paths in the relation). In the nearly random selection process, they therefore both have a $1/2^{k-1}$ probability of producing a valid solution and

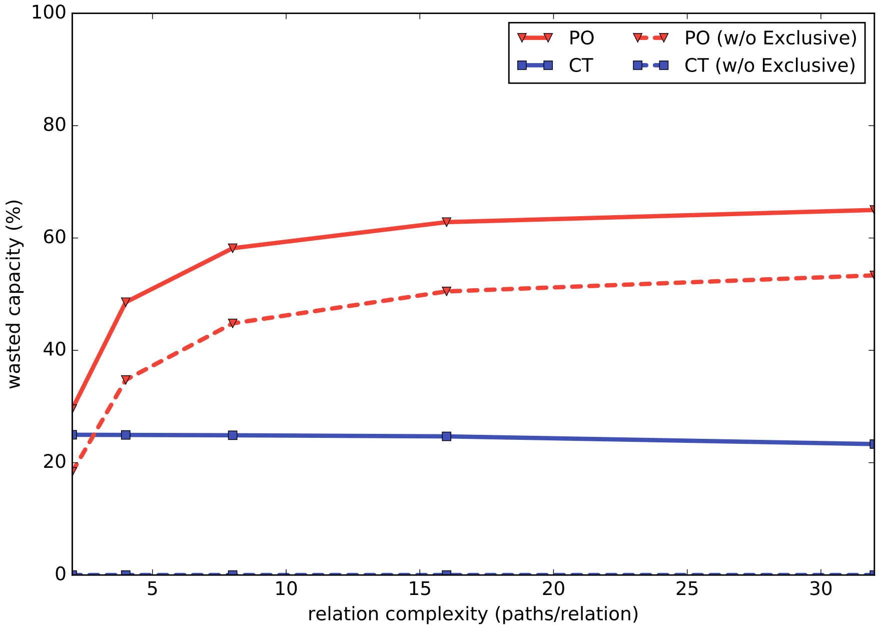

Figure 5.8.: *The wasted capacity as a function of the session complexity.*

not wasting any capacity. For smaller relations, the effect becomes less pronounced, resulting in a decreasing amount of wasted resources.

The proportion of remaining sessions depicted in figure 5.10 again exhibits the expected behavior: Relation-Aware optimization by far outperforms the other systems for every session size. This is due in large part to the inclusion of Exclusive relations. Where they slightly improve on the overall system performance for the Relation-Aware algorithm, they severely lower the result quality for PO and CT systems. Removing this relation type from the analysis narrows the performance gap to 10-25% for the scenarios presented here.

5.2.5. Path value variability

The variability of path values within a relation does not have a significant influence on the performance for any of the presented systems. Figure 5.11 shows a very slight improvement for PO at very low variabilities ($v_{max} \leq 2$). This is due to the AtLeastN, Dependent and Mutual dependencies, which all waste slightly less capacity in these scenarios. As the optimization algorithm for PO will order paths solely based on their

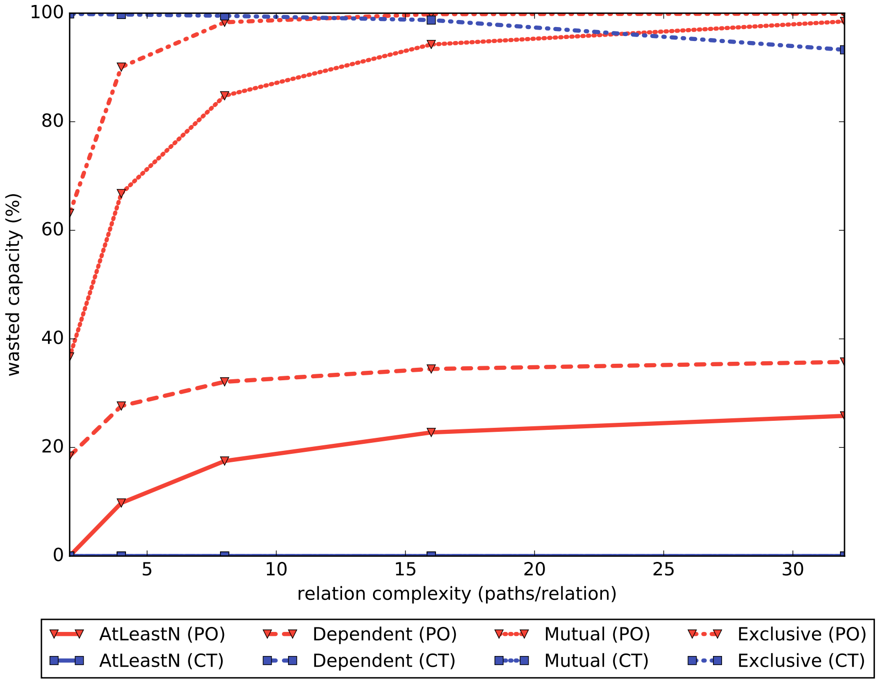

Figure 5.9.: *The wasted capacity as a function of the session complexity.*

intrinsic value, less variability in a relation leads to the constituent paths more likely being "kept together" in the process, leading to less wasted capacity. Other than that, the value variability does not influence the performance of the system.

5.2.6. Conclusion

The results of the adaptation analysis in large parts confirm the expectations from section 5.2. There are significant performance gains in terms of wasted capacity, as well as remaining online sessions to be had, when modeling relations in a QoS system. This performance gap increases with increasing relation complexity. Especially the purely Path-based Optimization is increasingly likely, to accidentally destroy relations (in particular for Exclusive and Mutually dependent relations, which have only a minuscule number of valid solutions compared to their overall solution space for larger relation sizes).

Correlated Time optimization provides acceptable performance for any kind of relation

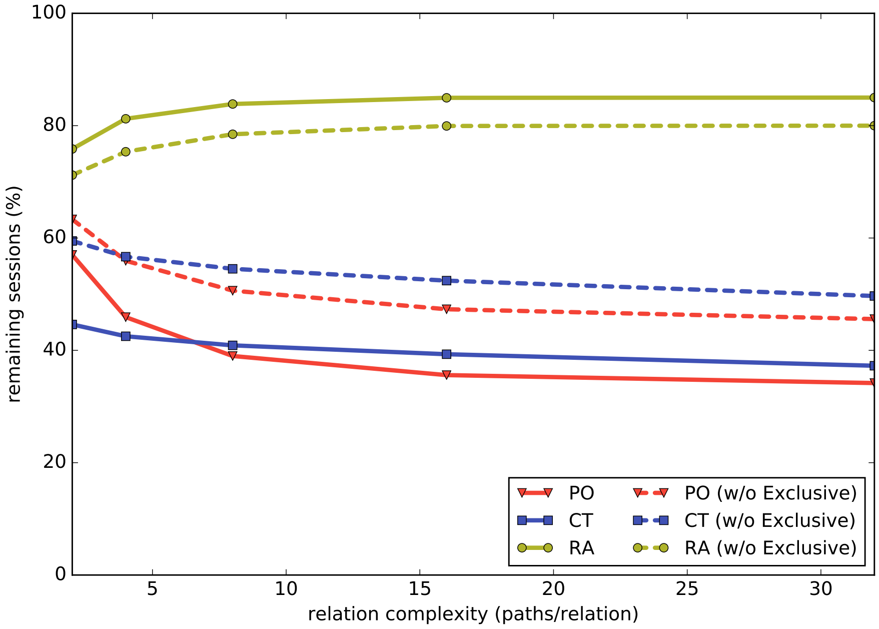

Figure 5.10.: *Remaining sessions as a function of the session complexity.*

which includes the full solution (i.e. all paths enabled) as a valid result. It is still outperformed by a Relation Aware system in terms of remaining sessions because it is not able to take advantage of the room for graceful degradation which a specific application might provide. However, due to the inherent assumption that every relation is a Mutual dependency, the approach fails to provide any adequate performance for any kind of Exclusive relation. The use case of signaling paths that are *not* to be enabled until they are needed, is simply not supported by systems without some kind of relation model.

5.3. Signaling

Changing the way the network reasons about reservations and their relationships might have an influence on the signaling process as well. Whereas existing systems rely on the initiator for a consistent path state, KASYMOSA QoS has more information available and should therefore require less interaction with the end systems. To evaluate what influence the relation model has on the signaling process, this section looks at two

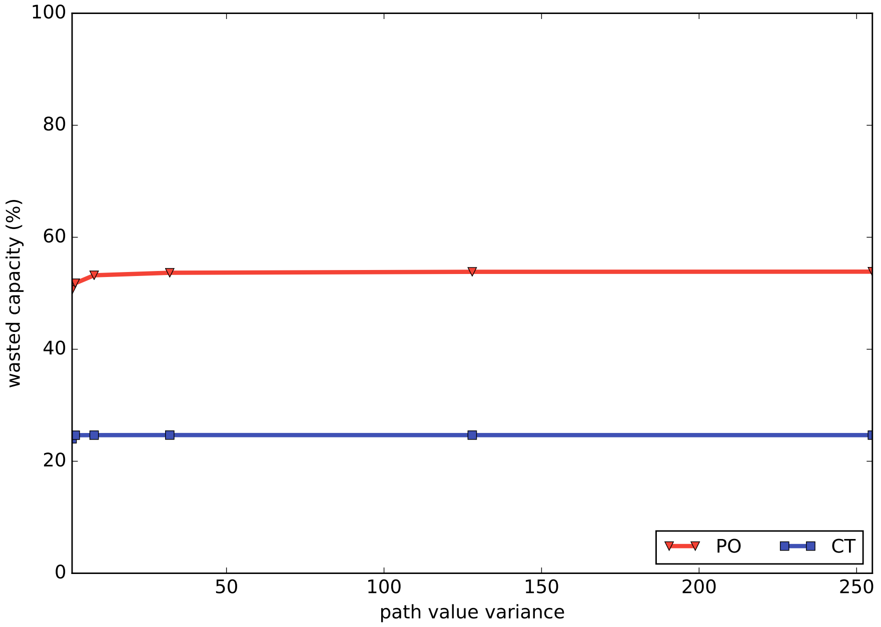

Figure 5.11.: *Remaining sessions as a function of the session complexity.*

performance indicators: *convergence time* and *signaling overhead*.

The convergence time is the time between a change in the reservation state at one system (triggered by a request or a change in the network environment) and the point in time at which all reservations along the path show a consistent state. Managing QoS reservations is just a means to an end. From the user's point of view, signaling and reserving resources is no more than a delay she might be willing to pay for a guaranteed service quality. Similarly, from the point of view of the system, the signaling process is a necessary evil to synchronize state over all systems contributing to the requested service. Minimizing the convergence time should therefore help to provide the best possible service quality with the minimum amount of resources.

Signaling overhead is a necessary evil for both the end user and the system. Distributed state, by necessity, needs information transfer to be consistent. In an in-band signaling system, such as the Internet[g], signaling and user data compete for the same transmission resources. A higher signaling overhead therefore automatically leaves less capacity for reservations.

Both parameters are not fully orthogonal. Due to the fact that each signaling transmission takes time, more signaling overhead leads to a longer convergence time. However, there are additional timing constraints, which may increase the convergence time even beyond the simple time needed to transmit signaling messages and acquire resources.

The signaling performance is influenced by different system parameters:

Transmission delay KASYMOSA QoS is specifically designed with long-delay networks in mind. A primary concern is therefore the influence of the transmission delay on the overall signaling performance.

Packet loss rate Due to environmental factors, mobile systems typically exhibit significantly higher packet loss than wired networks. In order to cope with such losses during the signaling, and eventually reach a consistent system state, a QoS system will need to employ some form of retransmission algorithm. Depending on the retransmission algorithm used, a higher packet loss rate will, to a varying extent, increase both convergence time and signaling overhead.

System type As we are comparing KASYMOSA QoS to existing solutions, we again need baselines to measure against. As we are focusing on the signaling process, these systems are slightly different than before.

> **Modified RSVP** The first system we compare against is a modified RSVP without relations or suspend-resume capabilities. Whereas standard RSVP is only able to reserve a single path with one signaling transfer, the modified version is able to aggregate multiple requests. This is done to evaluate the influence of the suspend-resume-mechanism, as well as the relation model, instead of message aggregation.
>
> **MoSaKa** The predecessor of KASYMOSA QoS introduced the notion of an active, network-based suspend-resume mechanism, allowing the system to lower signaling overhead for short-lived capacity issues. By evaluating this system, we can distinguish the influence of the suspension mechanism from that of the

[g] Depending on your point-of-view a QoS system can be considered to be both an in-band and an out-of-band system. For most existing systems, from layer four on up the signaling is out-of-band as it employs different transmission channels (i.e. protocols) than the actual data. On layers three and below, however, most systems mix data and signaling packets on the same channel and should therefore be considered in-band systems.

relation model. This system should show improvements especially in terms of the convergence time when a degraded capacity is restored. Without active resume, end systems can only rely on periodic reservation attempts. These are necessarily only transmitted with a certain interval to keep the signaling overhead down. This retry period inherently introduces an additional delay to system convergence, which is eliminated through the network-driven resume process.

KASYMOSA QoS Finally, this section evaluates the full relation-aware system. There are gains to be expected in signaling overhead and convergence time, due to better decisions by the network when adapting to changing conditions. Where the other systems rely on the initiator to signal the correct path setup for each relation, KASYMOSA QoS is able to calculate the desired outcome directly in the network, eliminating the need for interaction during the convergence phase.

5.3.1. Network environment

The example network in figure 5.12 serves as an environment for the signaling overhead analysis. The figure shows a representative example of a satellite network with the central satellite link in the middle and a terminal attached to either side. Due to the nature and range of satellite systems, they tend to include only one central link with fast distribution networks on each end.

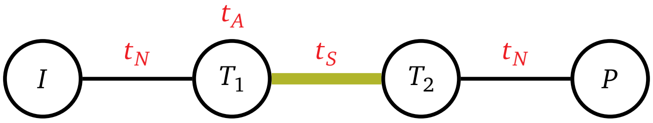

Figure 5.12.: *Representative example network for convergence the time and overhead analysis. The edge represents the satellite link between terminals T_1 and T_2. Additional network nodes between the terminals and the respective path endpoints are omitted. Their transmission and processing delays are summarized as t_N. t_S and t_A represent the transmission and resource acquisition delays of the satellite link, respectively.*

The timing variables given in the figure are defined as follows:

t_S The one-way transmission delay imposed by the satellite link. This includes the processing delays of the physical and MAC layers, as well as the actual time-of-flight of the signal to the satellite and back.

t_A The resource acquisition delay for sending a piece of data via the satellite link. Depending on the underlying system and state of the link, this may range from close to zero to even exceeding $2 \cdot t_A$. If the system uses statically assigned transmission resources or the appropriate transmission resource is already allocated from a former transmission, there will be no significant delay. However, if the system dynamically allocates physical resources and the allocation process involves other partners on the link, the acquisition may take more than one round-trip time. The KASYMOSA MAC link acquisition process opts for the middle ground: distributed resource allocation without a central coordinator. Each station periodically signals its resource requirements on a broadcast channel to every other station. Once all requests are collected, every station performs a deterministic resource allocation algorithm to determine its (and the other's) transmission resources. Due to the system design, this process takes at least $4 \cdot t_S$ to complete.

For the sake of this analysis, it is assumed that the resource acquisition will, on average, take $2 \cdot t_S$, while at the same time acknowledging that this value varies largely, depending on the system and its current state.

t_N The overall transmission and processing delay imposed by the networks between the path endpoints and the satellite terminals. These are either wired wide-area networks or wireless local networks. In any case, they impose much less delay than the satellite link, so $t_N \ll t_S$ by at least two orders of magnitude.

5.3.2. Initial reservation

Whether a system is relation aware or not, the initial reservation has to pass through the whole path at least twice until the initiator can use the requested path, resulting in two signaling messages transmitted along the path. The exchange will take at least:

$$t_R = 2 \cdot t_N + 2 \cdot t_A + 2 \cdot t_S \approx 6t_S$$

This assumes that the acquisition of the resources for the reservation and the transmission of the signaling packets can take place in parallel. If these two events have to be serialized, the delay will increase even further.

In case of a successful acquisition or a total rejection of all necessary resources, relation-aware and -unaware systems are level. However, if not all necessary resources could be acquired, the systems start to diverge. Whereas a relation-aware system will still receive an acceptable service level in one exchange (that includes not receiving any service at all), relation-unaware systems will have to handle different use cases differently:

Mutual If at least one of the mutually dependent paths fails, the initiator needs to tear down others that may have been granted. Assuming that each node immediately releases allocated resources on reception of the delete message, the tear down process adds $\sim 3 \cdot t_S$ to the total convergence time.

Dependent If the granted reservations include the dependency, the initiator can just use the paths as they are. Should the dependency not have been granted, the initiator has to signal another request, deleting a matching path of the dependent set and replacing it with the dependency. This increases the overall convergence time by $\sim 6 \cdot t_S$, as the initiator needs to receive the network response before being able to use the associated paths.

AtLeastN The best strategy for the AtLeastN relation is to request the full resource set, and react to the response where necessary. If the network can guarantee at least the required number of reservations, no further action is necessary. If there are less than N reservations confirmed, the initiator has to tear down the remaining paths. The resulting message exchange increases the overall delay by $\sim 4 \cdot t_S$.

Exclusive The only viable strategy for the initiator in a relation-unaware system is to request the best path of the relation first. In case of a failure, without further information, an adapted binary search strategy could be applied:

1. Remove all paths with higher costs than the last request from the relation (as they will not be guaranteed by the network anyway).
2. From the remaining paths, select the cost-wise median for the next request. If there is no such path, continue with step 4. If the request fails, repeat with step 1. If the request succeeds, continue with step 3.
3. Remove all paths with a lower cost from the relation. Since the resources for a higher-cost path could already be acquired, it would not be beneficial to request them. Continue with step 2.
4. Now the maximum possible size for a path from the relation has been established. In the initial relation, locate the path with the highest value at or below the established cost. If this is different from the one already reserved, delete the reservation and replace it with the high-value path.

For "well-behaved" relations (i.e. a strong correlation between cost and value of the constituent paths) the last step will not yield a reservation and is unnecessary. This process adds at least $\sim \lceil log_2(k) \rceil \cdot 6 \cdot t_S$ to the overall convergence time of a relation of size k.

Other relation structures might need an even more complex reservation pattern. In any case, a relation-aware system will never be worse than a relation-unaware one when reserving a set of resources.

In terms of signaling overhead, each additional message exchange adds two extra messages to the process. The relation-aware system will therefore have the lowest possible number of messages: only two, if no informational messages (e.g. PENDING) are sent.

5.3.3. Resource degradation

Figure 5.13 illustrates the issue of additional adaptation events in a relation-unaware system. In the example, three paths are mutually dependent. The relation-aware system releases the whole set of resources in one go, avoiding repeated adaptation later. The relation-unaware system needs two additional signaling processes, when the link goes down further. As the system has no notion of a relation or wasted resources, it guarantees these paths until there is not enough capacity left.

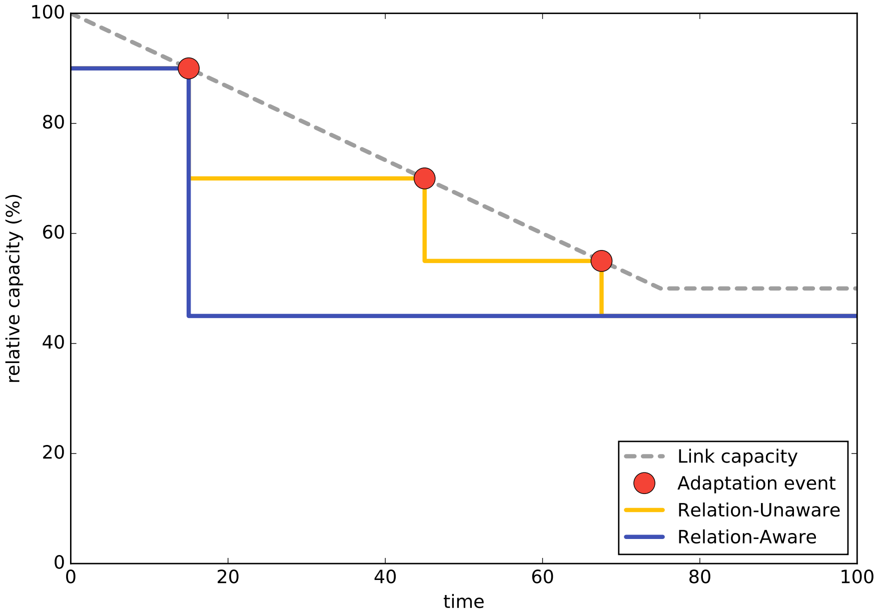

Figure 5.13.: *Adaptation events in a relation-unaware system compared to a relation-aware.*

To avoid these issues, the system could rely on the initiator to correct relation violations, introducing additional signaling overhead and convergence delay.

In the event of a link degradation, the appropriate router in each system signals certain paths as no longer guaranteed. For RSVP this means deleting the path, whereas MoSaKa and KASYMOSA QoS are able to suspend it. Both types of messages only need to be transmitted towards the endpoints to create a unified state throughout the network. The total transmission time is therefore $3 \cdot t_S + t_N \sim 3 \cdot t_S$ (assuming the transmission towards both endpoints runs parallel).

KASYMOSA QoS's adaptation process ends there. However, both relation-unaware approaches need to signal a deletion or suspension for the appropriate related reservations. Depending on exactly where in the network the change occurred, this introduces different additional convergence delays. If the adaptation took place on the near side of the satellite link with respect to the initiator (i.e. for outgoing paths from I to P), the added convergence delay amounts to $\sim 3 \cdot t_S$. Assuming that the update is signaled in both directions at the same time, the initiator would be notified after t_N about the change and start correcting immediately. The correction would follow the initial signaling towards P with a delay of $2 \cdot t_N$. The network would reach a consistent state the latest $3 \cdot t_N + 3 \cdot t_S$ after the causal change event.

If the adaptation takes place for an incoming path from P to I, it originates on the far side of the satellite link (as reservations are always outgoing to the QoS router). It therefore reaches the initiator after $t_N + 3 \cdot t_S$. The initiator's response reaches the peer after $2 \cdot t_N + 3 \cdot t_S$, bringing the total convergence time to $3 \cdot t_N + 6 \cdot t_S \approx 6 \cdot t_S$.

In an RSVP-like system, where paths are permanently deleted, the process ends with a stable resource allocation. Suspend/Resume-systems like MoSaKa, on the other hand, are extremely prone to oscillations. In a naive implementation, the initiator simply signals a path suspension for related paths. This frees up resources on the degraded link, possibly triggering a resumption of the initially suspended path, which in turn triggers a resumption of the related paths by the initiator, restarting the whole adaptation process again.

In order to avoid these oscillations, a Suspend/Resume-system has two options: either the router only signals resource changes and lets the initiator take care of suspending the correct paths, or the initiator ignores relation violations and does not signal at all. The first approach adds an end-to-end signaling transmission to every resource change, bringing the convergence time to $\sim 3 \cdot t_S$ and $\sim 6 \cdot t_S$ for near- and far-side events respectively. The latter approach always limits the convergence time to $\sim 3 \cdot t_S$, like a relation-aware system does, but leaves resources unused, possibly triggering another adaptation process later on, as illustrated above.

The inherent additional delay of a correcting initiator can also lead to inefficient resource allocation in fast-changing environments, as illustrated by figure 5.14. If the degradation of a link is fast enough, that a second adaptation event on a path from another relation becomes necessary, the resulting corrections massively overshoot the adaptation target capacity, leaving a significant amount of resources unused, only to correct it again in a later resource increase step.

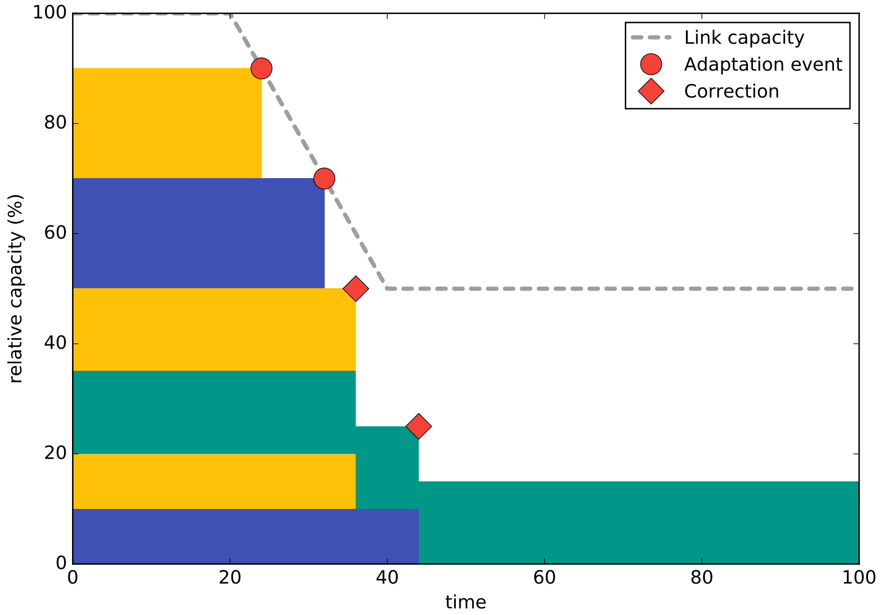

Figure 5.14.: *Correcting initiator overshooting the adaptation target in a fast-degrading environment. The figure illustrates the resource usage at the adapting router.*

One final problem for a correcting initiator is, again, the Exclusive relation. Its intended use case is to provide graceful degradation to a service via different QoS levels (probably representing different user experiences). In case of a link degradation, the correcting initiator needs to restart the allocation process described in section 5.3.2 to find the best alternative reservation, that will provide some service level. This again significantly increases overhead and convergence time.

5.3.4. Resource increase

When a constrained link necessitates the deletion or suspension of reservations, the transmission intent of the end systems usually does not cease to exist. Once capacity is available again, they therefore have an interest in reallocating their resources. However, depending on the system, this requires different amounts of effort.

Suspend/Resume systems, such as MoSaKa and KASYMOSA QoS, can notify clients immediately upon return of their resources. Once a router has sufficient capacity available, it transmits a Resume message towards both path endpoints in a similar fashion to the preceding Suspend. Routers along the way reacquire their respective local resources and forward the resume message, just as they would do in case of the initial reservation. Just as before, where KASYMOSA QoS's allocation is correct right away, MoSaKa's may result in wasted resources.

The RSVP-like system does not have the active feedback mechanism. The initiator can only rely on periodic re-reservation requests until communication can resume. However, this presents the problem of weighing signaling overhead against reaction time. Figure 5.15 illustrates the issue of a mismatching retry interval with regard to the actual availability of resources. The blue event denotes the point in time when enough resources to serve the requests would have been available again. As the RSVP system only periodically attempts the acquisition process, it does not detect the available capacity until the the of the retry interval. The delay between availability and detection is dead time, where the resources stay unused.

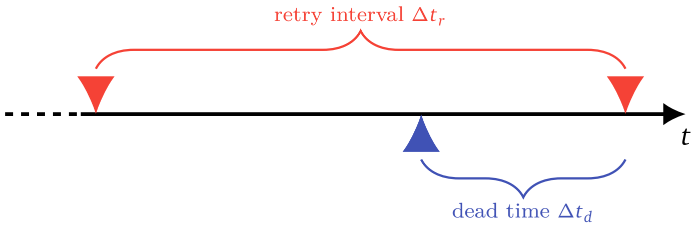

Figure 5.15.: *Dead time due to mismatch between re-reservation attempts and actual capacity increase events*

Assuming a uniform random distribution of the resource availability point over the retry interval, the dead time on average amounts to $\Delta t_d = {}^{\Delta t_r}/_{2}$. The requesting system therefore has to decide on the best interval for the current state of the network. Intervals that are too short cause increased signaling overhead due to unnecessary retries, too long, and the long detection delay decreases performance severely. Similar to the retransmission timer in TCP, a system may employ sophisticated approximation techniques to try and estimate the best interval length. In a quickly varying network, like a vehicle moving in an urban environment, the capacity may return on much shorter time scales than in a slowly evolving one, like a nomadic setup with weather as the only influence. However, such estimation algorithms rely on frequent events to provide the timing measurements necessary to converge on an optimal value. A network environment that is sufficiently stable, to be of any use at all, will typically not have enough adaptation events to provide a useful and stable optimal retry interval to the average user.

5.3.5. Message loss

The analyses above only apply to a loss-free transmission of the signaling messages. In case of a lossy environment, another delay comes into play: the retransmission timeout. Setting the correct timeout is a wide area of research. Solutions exist from simple fixed retransmission intervals to the complex RTO calculations implemented in TCP, which take round-trip time and jitter into account. They all have, by necessity, one thing in common: the retransmission timeout t_R is always longer than the round-trip time of the signaling process, significantly so in jittery environments. Therefore, whenever packet is lost, t_R dominates the overall convergence time.

The actual impact on the convergence time hugely depends on the loss ratio γ and the desired success probability ρ. The number of required transmission attempts n for a given loss rate and success probability can be calculated as

$$n = \left\lceil \frac{\log(1-\rho)}{\log(2\gamma-\gamma^2)} \right\rceil$$

The argument of the logarithm in the denominator is the combined loss probability of two dependent packets. As a successful signaling transmission requires the transmission of a request and a response, the probability for a signaling failure in any step increases accordingly. In a lossy environment, $\rho = 1$ is impossible to achieve. The success rate will asymptotically approach 100% for the number of retries $n \to \infty$.

Figure 5.16 visualizes the relation between the packet loss rate and the required transmission attempts for some example success probabilities. The figure shows, that for a reasonable probability of successfully transmitting a signaling request, the amount of transmission attempts grows quickly with packet loss rate. To achieve a 99% probability of successfully transmitting a request in a lossy environment with 10% packet loss, the system already needs an average of 3 retransmission attempts. Typically, systems will limit the number of retransmission attempts to a certain threshold. The vertical black dashed lines show the tolerable packet loss rate for a maximum of three transmission attempts. In order to achieve 99% probability of success, the initiator can only tolerate up to ∼ 11% loss rate in the network. Accepting an impractically low success probability of 50% would let the system work in very lossy environments with up to ∼ 55% loss rate[h]. Even after the first message loss, the retransmission timeout becomes the dominating factor in determining the overall convergence time. Further message losses only deteriorate the performance further.

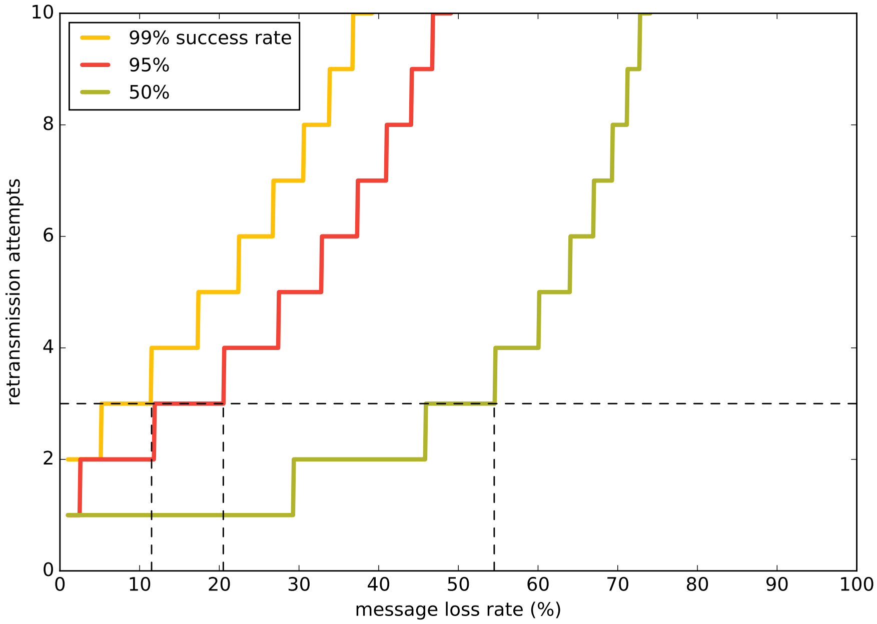

Figure 5.16.: *Average transmission attempts as a function of the packet loss rate.*

This analysis assumes a uniformly distributed message loss with constant probability. In a mobile network, with its link quality largely depending on the current environment, this is not the case. Message losses tend to be correlated in time, increasing the likelihood of the signaling exchange failing overall. End systems can try to circumvent this correlation by using retransmission intervals which are longer than the average message loss burst. This may further increase the overall convergence time of the process.

If a signaling exchange fails to transmit successfully, it may leave inconsistent state behind. Some nodes might have been notified of a change, and allocated or released resources, while others might not. In this case, another timeout becomes relevant: the

[h] “Work” here means: successfully completing a signaling exchange. Whether any practical application protocol could cope with this kind of environment is debatable.

lifetime of a reservation. By necessity, the lifetime of an object is longer than the round-trip time. This gives the data traffic, used as an implicit refresh, time to reach even the furthest node after a path has been granted[i]. Depending on the application, lifetimes will typically be at least an order of magnitude above the RTT of the network. Therefore, for a failed signaling transmission, the lifetime becomes the dominating factor when determining the convergence time.

In conclusion: in a lossy environment, independent of the system used, the retransmission timeout quickly becomes the dominating factor of the convergence time. Depending on the desired success probability, even a relatively low loss probability of 1% already requires a retransmission, making the RTO dominant. The analyses in this section refer to individual signaling exchanges. Every time a relation-unaware system needs additional messages, to reach the desired network configuration, message loss compounds the performance degradation compared to to a relation-aware system.

5.4. Overall network capacity

The influence of the additional signaling overhead and delay times on the overall network capacity is hard to estimate. It depends on parameters like the network overbooking, stability, packet loss, transmission and resource acquisition times, as well as the applications communicating over the network (which in turn influence parameters like relation size and structure, data transmission behavior and reservation parameters).

The determining factor of the network performance is the ratio between stability and convergence time. The more stable a network is, i.e. the less frequent changes in the request setup or capacity occur, the less influence the actual QoS system has. Even a relation-unaware QoS system can achieve low wasted capacity over the long run in a reasonably stable environment. The actual adaptation process may result in wasted capacity at first, but a correcting initiator will eventually solve this issue. Relation-unaware systems may take longer to converge, but with a sufficiently long network change interval, this difference becomes negligible.

However, if the capacity varies more quickly, with intervals approaching the order of magnitude of the signaling round-trip time, differences become more pronounced. Depending on the application and network setup, relation-unaware systems at least double the convergence time for most events. For a given ratio $m = {}^{t_C}/_{t_R}$ of the stable period t_C and the round-trip time t_R, the relative performance of a relation-unaware compared to a

[i] This implicitly assumes the same RTT for signaling and data traffic. Depending on the MAC and PHY layer QoS, this is not necessarily the case. Different buffering strategies or modulation and coding schemes can significantly increase the RTT for certain types of traffic.

relation-aware system can never exceed $\frac{m-1/2}{m-1}$. As figure 5.17 illustrates, the performance of the relation-unaware system drastically declines for shorter stable periods. The curve displayed in the figure is a maximum, assuming otherwise perfect operation. Including message losses, retry mismatches and additional signaling to approximate more complex relation structures, the real performance is likely to be significantly worse.

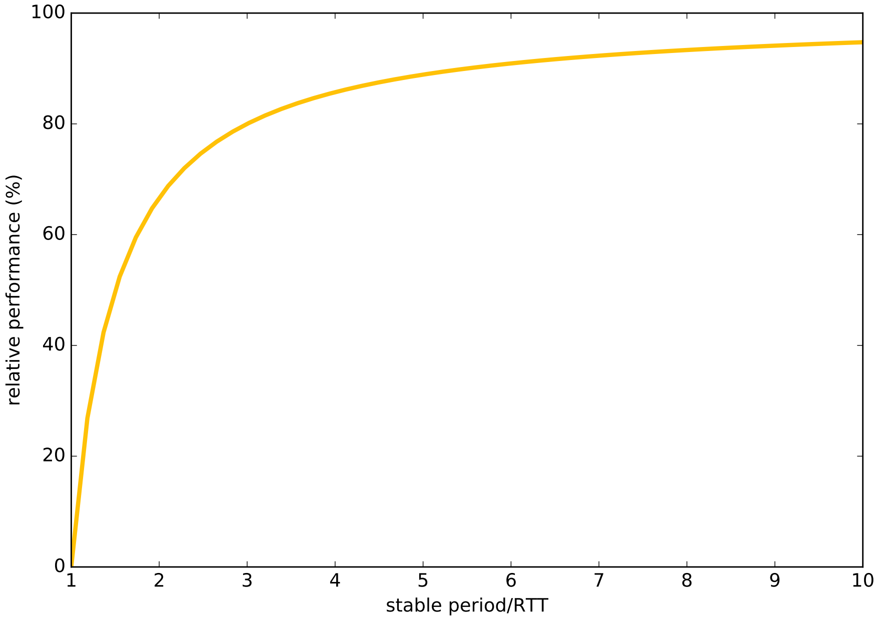

Figure 5.17.: *Relative performance of a relation-unaware system for a given ratio* t_C/t_R.

6. Conclusions and future work

No academic work would be complete without some concluding remarks and an outlook on future work. Section 6.1 gives a summary of the key findings of the work and tries to interpret them according to the goals set in the beginning. The following section 6.2 presents new and open questions identified during the project and provides possible directions for future research.

6.1. Conclusions

Path relations are a crucial factor of the performance of reservation-based QoS systems in mobile satellite networks. In an environment where resources are comparatively scarce and transmission delays long, they provide valuable information to enable networks to quickly adapt to changes. Quickly adapt they must: effects like fading, shadowing and interference change the link capacity rapidly in a mobile environment.

Existing QoS-solutions work on an individual flow level: each reservation is treated independently. However, this opens an information gap between network and application. Transport protocols, such as TCP, or applications like video streaming or SIP form complex path hierarchies with multiple interdependent reservations. Something as simple as a TCP file download already requires two mutually dependent paths: a high-volume downlink, transporting the actual file data, and a comparatively low-volume uplink, carrying acknowledgements. One without the other is useless, wasting resources that could otherwise be better employed in the constrained environment.

This work proposes to model relations as propositional formulas representing invariants over the reservation states in a router. By evaluating such an expression, a router can assess whether the current (or a planned) resource allocation fulfills the application requirements or not. Propositional formulas, with their ability to express any value pattern the relation invariant might have, provide the necessary flexibility to integrate the relation model with any kind of application protocol. Use cases like exclusive reservations for different quality levels of a video stream – impossible to model efficiently in traditional systems – become a matter of simply defining the correct invariant.

Finding the best current resource allocation on a router is an instance of the well known Knapsack problem:

$$\begin{aligned}\text{Maximize } &\sum_{i\in I} v_i s_i\\ \text{provided, that } &\sum_{n\in N} w_i s_i \leq W_n\\ \text{and } &\forall r \in R : r = 1\end{aligned}$$

The relation requirement ensures, that no feasible solution wastes any bandwidth. The very definition of "wasted bandwidth" as allocated resources, that cannot be used, always provides a feasible solution. If no resources are assigned, the solution is, by definition, valid.

This resource allocation problem can be transformed into an Integer Linear Program, giving access to a wide body of algorithm research for efficiently finding solutions. Using an existing Branch-and-Cut solver, practically relevant problem sizes can be tackled by the system.

The result is a QoS system, that does not waste resources and keeps more user sessions online when adapting to degrading link conditions. Traditional systems can waste up to 70% of the remaining capacity on a severely degraded link. This in turn leads to between 20% and 50% fewer remaining user sessions when compared to a relation-aware system after adaptation.

Making better decisions based on relation information enables the network to lower signaling overhead and convergence time as well. Assuming the initiator corrects relation violations in a traditional system, the amount of signaling (and therefore the convergence time) at least doubles. Specific use cases like exclusive relations between paths deteriorate performance even further. As existing systems are simply not designed with these use cases in mind, they need complex approximation algorithms, involving much additional signaling, to achieve the performance of a relation-aware solution.

Did this work fulfill the requirements set in section 1.2.2? The presented system efficiently allocates resources to related paths, avoiding wasted resources which can otherwise comprise up to 70% of the remaining link capacity. By making better decisions based on the additional information, the relation-aware system avoids signaling necessary for correcting allocation errors and is able to support use cases that are practically impossible to efficiently realize in existing solutions. Together with a Suspend/Resume mechanism taken from the MoSaKa system, KASYMOSA QoS is able to adapt to variable environments, as they are prevalent in mobile communications. Propositional formulas as a way to express relation invariants do not limit the end user to specific preconceived use cases, but allow applications to model any kind of static path relationship. Dynamic path relations are not supported, and are left to future research. The system is compatible with existing networks by bypassing non-QoS-capable routers. While this possibly degrades the QoS-performance, it is a necessary capability in an environment where global change of the infrastructure has basically become impossible. Future work should look into the possibility of adapting KASYMOSA QoS to the NSIS protocol framework. By basing the work on the foundations of an accepted standard, adoption may be quicker and more seamless. Last, but not least, the underlying transport system operates robustly in lossy environments without stalling independent signaling streams.

6.2. Future work

Even though KASYMOSA QoS manages to achieve its goals, the system is far from being a perfect solution. As it is to be expected in any kind of research, new questions arose along the way. This section briefly discusses those questions and tries to give indications as to where solutions might be heading.

Efficient relation representation Currently, the system transmits relations through the network by directly encoding their propositional formulas. This representation is not necessarily very efficient for small, densely populated truth tables resulting from use cases like TCP. More efficient solutions could transmit the function index of the truth table (for small, densely populated use cases) or a list of valid table indexes (for larger, more sparsely populated tables). How this influences the representation as an Integer Linear Program remains an open question. As any truth table can be represented into a Boolean expression (a subset of propositional formulas), the transformation to an ILP, as presented in this work, is always possible. Whether the resulting ILP is sufficiently efficient (in particular in the presence of the counting operators $|d|_1$ and $|d|_0$), remains unclear.

Dynamic path relationships The relation model presented here only allows to model static path relations. Relations that could change over time require some kind of temporal model not contained in this work. Future research needs to clarify what kind of use cases benefit from a dynamic relation model and how such a model could flexibly, yet efficiently, be represented.

More complex reservation models The KASYMOSA QoS reservation model is a simple Token Bucket Meter, combined with an equally simple priority system. Section 3.3.3 highlights some interactions between the reservation and relation models for such a restricted case. Future research should answer the question how more complex models with in-built adaptation capabilities like the “QoS Desired” and “Minimum QoS” of NSIS NSLP-QSPEC Template interact with relations and whether a combination is feasible at all.

Adapting QoS to modern service-based infrastructures In the collaboration with partners in the KASYMOSA project, it became apparent that the expressive capabilities of a QoS system on the network layer are too limited to be used in a multi-layered application. Even something as seemingly simple as creating the correct Filter object becomes an issue when using a service architecture where the different components might hold different information about the transfer in question. In such a system, the top-most application layer might know about the rate requirements for its payload data, but not be able to provide information about the actual transport layer: which protocols are used, what connections will be opened etc. Architectures like SOAP leave these details to the service layer underneath the applications, typically not exposing such information at all. How QoS requirements can be integrated into such a system, including the necessary adaptations at the lower layers, is an open question. Existing approaches like CQML could provide the necessary tools to do so, and should be investigated further.

Integration of DiffServ-like systems Where IntServ (and therefore KASYMOSA QoS) handle micro-flows, i.e. individual reservations for specific packet flows, systems like DiffServ provide QoS by aggregating many flows into a Behavior Aggregate. The reasoning behind the aggregation is a scalability problem, when applying IntServ to Internet-sized networks. It remains to be seen how, and whether at all, a relation model could be integrated with DiffServ, and how such a system would cope with relations within a Behavior Aggregate or between different BAs.

Better integration with lower layers The KASYMOSA system already provides a certain degree of cross-layer integration between the physical, MAC and network layers. A distributed resource allocation algorithm on the MAC layer uses aggregated information from the network layer to request the correct amount of physical resources. However, when adapting to a constrained environment, this algorithm does not take detailed information into account, resulting in a certain amount of “left-over” capacity, where no network layer reservation fits. A more efficient system might try to minimize these superfluous resources by transmitting, and taking into account, the resource levels useful to the higher layers.

Minimize/accelerate signaling based on relation information In the current system, the adapting router is responsible for transmitting information about suspended paths to all other nodes. In theory, as all nodes along a path have the same information about relations, signaling the offending path and letting other nodes calculate dependencies on their own, should suffice. Taking this idea a step further could mean relying on globally available information on the satellite link (e.g. from the distributed allocation algorithm on the MAC layer) to speed up signaling. Instead of having the signaling cross the satellite link first, all attached terminals could use the available information to calculate the expected outcome at the adapting node and start signaling right away on both sides of the slow link. Research questions here include how much information needs to be shared between the terminals, and how accuracy, and therefore, possibly required corrections, behave with incomplete information.

A. State Machines

Figure A.1 depicts the full state machine of a KASYMOSA QoS reservation. The picture does not include lifetime or retransmission mechanisms, as these are handled separately.

One thing to note is the “Offline” state with its three sub-states “RemoteSuspend”, “LocalSuspend” and “Blocked”. Whereas the two suspend states stem from the Suspend/Resume mechanism, as already presented in MoSaKa, the Blocked-state is specific to KASYMOSA QoS. Blocked reservations are reservations for which resources could potentially be provided, but enabling them would violate a containing relation. They behave differently from a locally suspended relation in that no resources are acquired from lower layers. However, in contrast to remotely suspended relations, they are viable candidates for the optimization process.

Another noteworthy detail is the missing transition between locally and remotely suspended states. If such a transition were possible, a reservation could deadlock: if two nodes suspended the reservation at the same time due to locally insufficient resources, they would both, afterwards, receive the signaling by the other node, and enter the RemoteSuspend state. As remotely suspended reservations are, for all intents and purposes, non-existent to the resource allocation, none of the nodes would try to reacquire resources for that path, perpetually keeping it suspended.

Figure A.2 depicts the external refresh cycle of a reservation. The state machine controls and reacts to the two timers for Refresh and Deletion of a path. Both timers are restarted by data being transmitted on the path, avoiding the need for explicit refresh messages. A noticeable extension of the state machine is the “idle” state, in which both timers are stopped. This state is entered whenever the corresponding path is suspended for any reason. As suspension usually takes place in an already constrained environment and should stop the initiator from transmitting data along the path. By requiring refreshes for suspended reservations, the system would risk deleting these paths due to their timers running out. To avoid submitting explicit refresh messages for suspended paths and thus using already limited resources, the refresh mechanism is disabled in this environment. This does introduce the issue of paths being perpetually suspended if the node, that initially triggered the suspension, disappears. However, the problem could be solved by topology change detection or external timer mechanisms without introducing additional signaling.

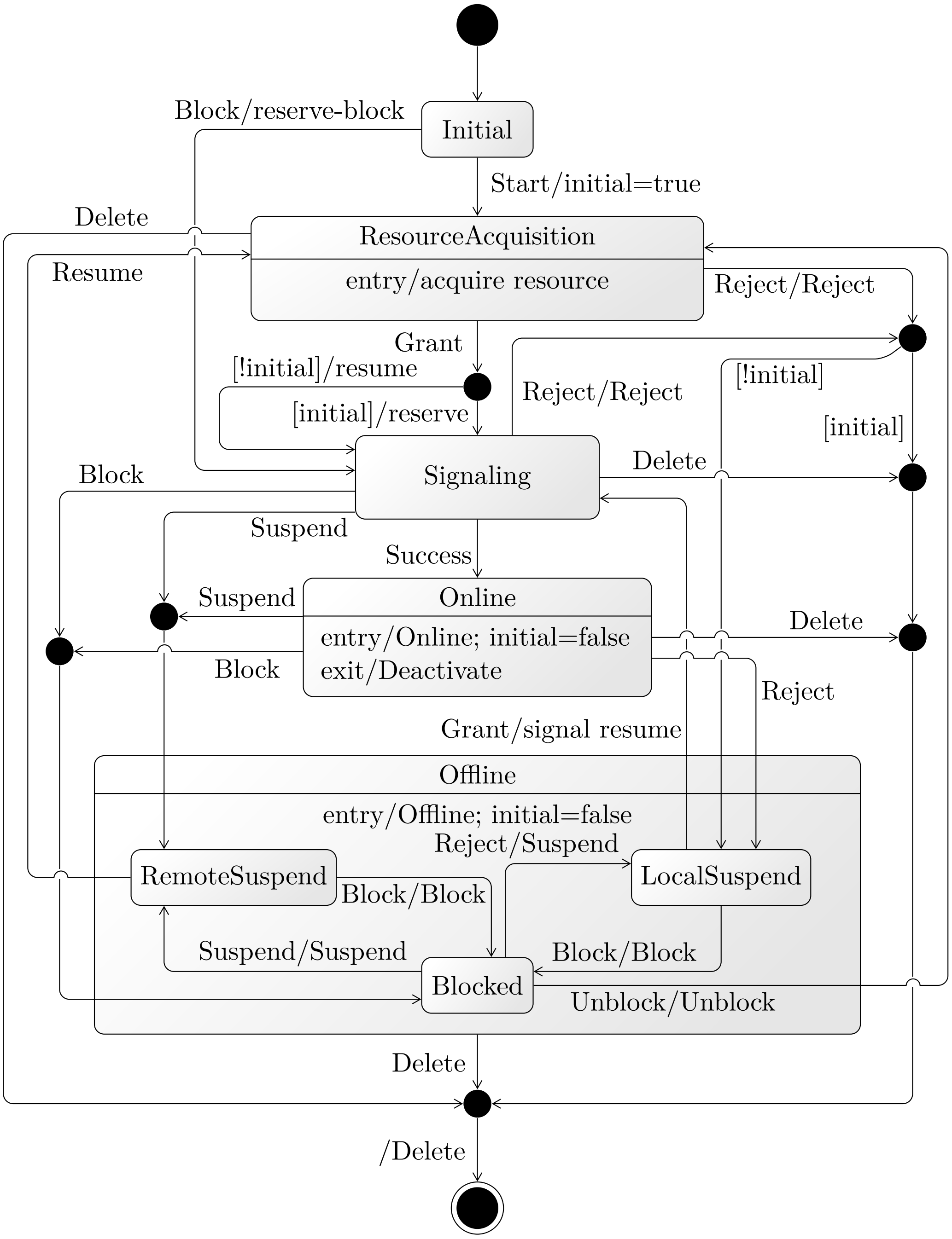

Figure A.1.: *Reservation state machine*

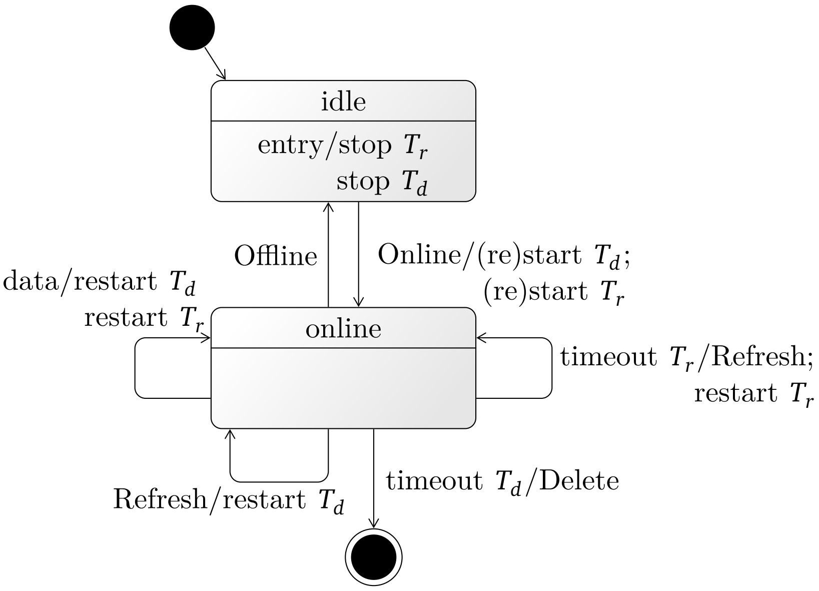

Figure A.2.: *The refresh cycle of an individual reservation*

B. A complex transformation example

For this transformation example, the following reservation and relation setup is assumed:

- Path p_0(rate: 10000 bit/s, prio: 2)
- Path p_1(rate: 5000 bit/s, prio: 2)
- Path p_2(rate: 3000 bit/s, prio: 1)
- Path p_3(rate: 1000 bit/s, prio: 1)

Paths p_0 and p_1 form a mutually dependent pair (e.g. the video and audio connection of a streaming solution), as do p_2 and p_3. The two pairs are mutually exclusive (e.g. they might represent alternative versions of different quality of the same stream).

The resulting relation has the value 1 for $s = (s_3, s_2, s_1, s_0) = (0,0,0,0)$, $(0,0,1,1)$ and $(1,1,0,0)$, resulting in the propositional formula:

$$r = (|s|_0 = 4) \vee \overline{s_3}\,\overline{s_2}s_1s_0 \vee s_3s_2\overline{s_1}\,\overline{s_0}$$

Transforming this setup yields the following ILP:

		Notes
Maximize	$2 \cdot s_0 + 2 \cdot s_1 + 1 \cdot s_3 + 1 \cdot s_3$	
provided, that	$10000 \cdot s_0 + 5000 \cdot s_1 +$ $3000 \cdot s_2 + 1000 \cdot s_3 \le c_{max}$	assumes identity between net rate and physical resources
	$(1-s_3)+(1-s_2)+$ $(1-s_1)+(1-s_0) = y_0$	$\vert s\vert_0$
	$0 \le (1-s_3)+(1-s_2)+s_1+s_0-4 \cdot y_1 \le 3$	$\overline{s_3}\,\overline{s_2}s_1s_0$
	$0 \le s_3+s_2+(1-s_1)+(1-s_0)-4 \cdot y_2 \le 3$	$s_3s_2\overline{s_1}\,\overline{s_0}$
	$0 \le y_0 + y_1 + y_2 - 3 \cdot r \le 2$	the complete r
	$1 \le r$	ensure $r \ne 0$

Using these constraints an ILP solver will find the optimal resource allocation for the current capacity limit c_{max} (the example implicitly assumes only one serving interface, hence only one capacity limit). As the null-solution is included in the ILP as y_0, there will always be a feasible solution, even if no paths can be served at all.

Bibliography

[Aag01] J. Aagedal. “Quality of Service Support in Development of Distributed Systems”. PhD thesis. Department of Informatics, Faculty of Mathematics and Natural Sciences, University of Oslo, 2001.

[Alm92] P. Almquist. *Type of Service in the Internet Protocol Suite.* RFC 1349 (Proposed Standard). Obsoleted by RFC 2474. Internet Engineering Task Force, July 1992. url: http://www.ietf.org/rfc/rfc1349.txt.

[APB09] M. Allman, V. Paxson, and E. Blanton. *TCP Congestion Control.* RFC 5681 (Draft Standard). Internet Engineering Task Force, Sept. 2009. url: http://www.ietf.org/rfc/rfc5681.txt.

[Ash+10] G. Ash, A. Bader, C. Kappler, and D. Oran. *QSPEC Template for the Quality-of-Service NSIS Signaling Layer Protocol (NSLP).* RFC 5975 (Experimental). Internet Engineering Task Force, Oct. 2010. url: http://www.ietf.org/rfc/rfc5975.txt.

[AVC] ITU-T Rec. H.264. *Advanced video coding for generic audiovisual services.* Tech. rep. Identical to ISO/IEC 14496-10:2012. International Telecommunication Union, Jan. 2012. url: http://handle.itu.int/11.1002/1000/11466.

[Bad+10] A. Bader, L. Westberg, G. Karagiannis, C. Kappler, and T. Phelan. *RMD-QOSM: The NSIS Quality-of-Service Model for Resource Management in Diffserv.* RFC 5977 (Experimental). Internet Engineering Task Force, Oct. 2010. url: http://www.ietf.org/rfc/rfc5977.txt.

[BCS94] R. Braden, D. Clark, and S. Shenker. *Integrated Services in the Internet Architecture: an Overview.* RFC 1633 (Informational). Internet Engineering Task Force, June 1994. url: http://www.ietf.org/rfc/rfc1633.txt.

[BFF96] T. Berners-Lee, R. Fielding, and H. Frystyk. *Hypertext Transfer Protocol – HTTP/1.0.* RFC 1945 (Informational). Internet Engineering Task Force, May 1996. url: http://www.ietf.org/rfc/rfc1945.txt.

[Bla+98] S. Blake, D. Black, M. Carlson, E. Davies, Z. Wang, and W. Weiss. *An Architecture for Differentiated Services*. RFC 2475 (Informational). Updated by RFC 3260. Internet Engineering Task Force, Dec. 1998. url: http://www.ietf.org/rfc/rfc2475.txt.

[BPT15] M. Belshe, R. Peon, and M. Thomson. *Hypertext Transfer Protocol Version 2 (HTTP/2)*. RFC 7540 (Proposed Standard). Internet Engineering Task Force, May 2015. url: http://www.ietf.org/rfc/rfc7540.txt.

[Bra+97] R. Braden, L. Zhang, S. Berson, S. Herzog, and S. Jamin. *Resource ReSerVation Protocol (RSVP) – Version 1 Functional Specification*. RFC 2205 (Proposed Standard). Updated by RFCs 2750, 3936, 4495, 5946, 6437, 6780. Internet Engineering Task Force, Sept. 1997. url: http://www.ietf.org/rfc/rfc2205.txt.

[Brü+16] M. Brückner, P. Drieß, M. Osdoba, and A. Mitschele-Thiel. "A Dependency-Aware QoS System for Mobile Satellite Communication". In: *IEEE Wireless Communications and Networking Conference, WCNC 2016*. Doha, Apr. 2016.

[BWK04] A. Bader, L. Westberg, and G. Karagiannis. *RMD (Resource Management in Diffserv) QoS-NSLP model*. Ed. by A. Bader, L. Westberg, and G. Karagiannis. Imported from research group DACS (ID number 304). Feb. 2004. url: http://doc.utwente.nl/66552/.

[COIN] *COIN-OR Branch-and-Cut MIP Solver*. Website. Sept. 2016. url: https://projects.coin-or.org/Cbc (visited on 12/06/2016).

[DA99] T. Dierks and C. Allen. *The TLS Protocol Version 1.0*. RFC 2246 (Proposed Standard). Obsoleted by RFC 4346, updated by RFCs 3546, 5746, 6176, 7465, 7507. Internet Engineering Task Force, Jan. 1999. url: http://www.ietf.org/rfc/rfc2246.txt.

[Dav+02] B. Davie, A. Charny, J.C.R. Bennet, K. Benson, J.Y. Le Boudec, W. Courtney, S. Davari, V. Firoiu, and D. Stiliadis. *An Expedited Forwarding PHB (Per-Hop Behavior)*. RFC 3246 (Proposed Standard). Internet Engineering Task Force, Mar. 2002. url: http://www.ietf.org/rfc/rfc3246.txt.

[Daw+00] M. Dawande, J. Kalagnanam, P. Keskinocak, F.S. Salman, and R. Ravi. "Approximation Algorithms for the Multiple Knapsack Problem with Assignment Restrictions". In: *Journal of Combinatorial Optimization* 4.2 (2000), pp. 171–186. doi: 10.1023/A:1009894503716. url: http://dx.doi.org/10.1023/A:1009894503716.

[DEB12] P. Drieß, F. Evers, and M. Brückner. "The MoSaKa QoS System: Architecture and Evaluation". In: *International Journal On Advances in Telecommunications, vol 5, nr 3&4, 2012* 5.3&4 (Sept. 2012), pp. 216–228. issn: 1942–2601.

[DR06] T. Dierks and E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.1*. RFC 4346 (Proposed Standard). Obsoleted by RFC 5246, updated by RFCs 4366, 4680, 4681, 5746, 6176, 7465, 7507. Internet Engineering Task Force, Apr. 2006. url: http://www.ietf.org/rfc/rfc4346.txt.

[DR08] T. Dierks and E. Rescorla. *The Transport Layer Security (TLS) Protocol Version 1.2*. RFC 5246 (Proposed Standard). Updated by RFCs 5746, 5878, 6176, 7465, 7507, 7568, 7627. Internet Engineering Task Force, Aug. 2008. url: http://www.ietf.org/rfc/rfc5246.txt.

[Eic08] A. Eichhorn. "Content-Aware Multimedia Communications". PhD thesis. July 2008. url: http://uri.gbv.de/document/gvk:ppn:571021468.

[FAV08] A. Farrel, A. Ayyangar, and JP. Vasseur. *Inter-Domain MPLS and GMPLS Traffic Engineering – Resource Reservation Protocol-Traffic Engineering (RSVP-TE) Extensions*. RFC 5151 (Proposed Standard). Internet Engineering Task Force, Feb. 2008. url: http://www.ietf.org/rfc/rfc5151.txt.

[FDC11] X. Fu, C. Dickmann, and J. Crowcroft. *General Internet Signaling Transport (GIST) over Stream Control Transmission Protocol (SCTP) and Datagram Transport Layer Security (DTLS)*. RFC 6084 (Experimental). Internet Engineering Task Force, Jan. 2011. url: http://www.ietf.org/rfc/rfc6084.txt.

[Fie+99] R. Fielding, J. Gettys, J. Mogul, H. Frystyk, L. Masinter, P. Leach, and T. Berners-Lee. *Hypertext Transfer Protocol – HTTP/1.1*. RFC 2616 (Draft Standard). Obsoleted by RFCs 7230, 7231, 7232, 7233, 7234, 7235, updated by RFCs 2817, 5785, 6266, 6585. Internet Engineering Task Force, June 1999. url: http://www.ietf.org/rfc/rfc2616.txt.

[Han+05] R. Hancock, G. Karagiannis, J. Loughney, and S. Van den Bosch. *Next Steps in Signaling (NSIS): Framework*. RFC 4080 (Informational). Internet Engineering Task Force, June 2005. url: http://www.ietf.org/rfc/rfc4080.txt.

[Hei+10] M. Hein et al. "Perspectives for Mobile Satellite Communications in Ka-Band (MoSaKa)". In: *EuCAP'2010: The 4th European Conference on Antennas and Propagation.* Barcelona, Spain, Apr. 2010.

[Hei+99] J. Heinanen, F. Baker, W. Weiss, and J. Wroclawski. *Assured Forwarding PHB Group.* RFC 2597 (Proposed Standard). Updated by RFC 3260. Internet Engineering Task Force, June 1999. url: http://www.ietf.org/rfc/rfc2597.txt.

[Her01] S. Herzog. *Signaled Preemption Priority Policy Element.* RFC 3181 (Proposed Standard). Internet Engineering Task Force, Oct. 2001. url: http://www.ietf.org/rfc/rfc3181.txt.

[IH06] Select Bipartisan Committee to Investigate the Preparation for and Response to Hurricane Katrina. *A Failure of Initiative - Final Report of the Select Bipartisan Committee to Investigate the Preparation for and Response to Hurricane Katrina.* Tech. rep. U.S. House of Representatives, Feb. 2006. url: http://www.gpo.gov/fdsys/pkg/CRPT-109hrpt377/pdf/CRPT-109hrpt377.pdf.

[Kat97] D. Katz. *IP Router Alert Option.* RFC 2113 (Proposed Standard). Updated by RFCs 5350, 6398. Internet Engineering Task Force, Feb. 1997. url: http://www.ietf.org/rfc/rfc2113.txt.

[KFS11] C. Kappler, X. Fu, and B. Schloer. *A QoS Model for Signaling IntServ Controlled-Load Service with NSIS.* Internet Draft. Expired 2012-03-07, no successor, last accessed: 2013-05-23. Internet Engineering Task Force, Sept. 2011. url: http://tools.ietf.org/html/draft-kappler-nsis-qosmodel-controlledload-14.

[KPP04] H. Kellerer, U. Pferschy, and D. Pisinger. *Knapsack Problems.* Springer-Verlag Berlin, 2004. isbn: 3-540-40286-1.

[MKM10] J. Manner, G. Karagiannis, and A. McDonald. *NSIS Signaling Layer Protocol (NSLP) for Quality-of-Service Signaling.* RFC 5974 (Experimental). Internet Engineering Task Force, Oct. 2010. url: http://www.ietf.org/rfc/rfc5974.txt.

[MPEG4] *Overview of the MPEG-4 Standard.* Tech. rep. 2002.

[Nic+98] K. Nichols, S. Blake, F. Baker, and D. Black. *Definition of the Differentiated Services Field (DS Field) in the IPv4 and IPv6 Headers.* RFC 2474 (Proposed Standard). Updated by RFCs 3168, 3260. Internet Engineering Task Force, Dec. 1998. url: http://www.ietf.org/rfc/rfc2474.txt.

[PJ99] C. Partridge and A. Jackson. *IPv6 Router Alert Option*. RFC 2711 (Proposed Standard). Updated by RFC 6398. Internet Engineering Task Force, Oct. 1999. url: http://www.ietf.org/rfc/rfc2711.txt.

[Pos81] J. Postel. *Transmission Control Protocol*. RFC 793 (INTERNET STANDARD). Updated by RFCs 1122, 3168, 6093, 6528. Internet Engineering Task Force, Sept. 1981. url: http://www.ietf.org/rfc/rfc793.txt.

[PR15] R. Peon and H. Ruellan. *HPACK: Header Compression for HTTP/2*. RFC 7541 (Proposed Standard). Internet Engineering Task Force, May 2015. url: http://www.ietf.org/rfc/rfc7541.txt.

[RM12] E. Rescorla and N. Modadugu. *Datagram Transport Layer Security Version 1.2*. RFC 6347 (Proposed Standard). Updated by RFC 7507. Internet Engineering Task Force, Jan. 2012. url: http://www.ietf.org/rfc/rfc6347.txt.

[SH10] H. Schulzrinne and R. Hancock. *GIST: General Internet Signalling Transport*. RFC 5971 (Experimental). Internet Engineering Task Force, Oct. 2010. url: http://www.ietf.org/rfc/rfc5971.txt.

[Ste07] R. Stewart. *Stream Control Transmission Protocol*. RFC 4960 (Proposed Standard). Updated by RFCs 6096, 6335, 7053. Internet Engineering Task Force, Sept. 2007. url: http://www.ietf.org/rfc/rfc4960.txt.

[Tse+10] T. Tsenov, H. Tschofenig, X. Fu, C. Aoun, and E. Davies. *General Internet Signaling Transport (GIST) State Machine*. RFC 5972 (Informational). Internet Engineering Task Force, Oct. 2010. url: http://www.ietf.org/rfc/rfc5972.txt.

[TW10] A. Tanenbaum and D. Wetherall. *Computer Networks*. Prentice Hall PTR, 2010. isbn: 978-0132126953.

[UML] *UML 2.4.1*. OMG Standard. also adopted as: ISO/IEC 19505-1 and 19505-2. Aug. 2011. url: http://www.omg.org/spec/UML/2.4.1/ (visited on 04/01/2014).

[Vol16] T. Volkert. “Hierarchisches Routingmanagement - Autonomes Netzwerkmanagement für ein dynamisches Routing unter Berücksichtigung von Qualitätsanforderungen”. PhD thesis. Mar. 2016. url: http://uri.gbv.de/document/gvk:ppn:848832272.

[Wes+02] L. Westberg, A. Császár, G. Karagiannis, Á. Marquetant, D. Partain, O. Pop, V. Rexhepi, R. Szabá, and A. Takács. "Resource Management in DiffServ (RMD): A Functionality and Performance Behavior Overview". In: *Protocols for High Speed Networks*. Springer. 2002, pp. 17–34. url: http://qosip.tmit.bme.hu/~csaszar/csata-pdf/westberg02rmdperformance.pdf.

[Wes+03] L. Westberg, M. Jacobsson, M. de Kogel, S. Oosthoek, D. Partain, V. Rexhepi, P. Wallentin, and G. Karagiannis. *Resource Management in Diffserv On DemAnd (RODA) PHR*. IETF Internet draft. Internet Engineering Task Force (IETF), Sept. 2003. url: http://www.watersprings.org/pub/id/draft-westberg-rmd-od-phr-04.txt.

[Wol+13] A. Wolf, H. Brandt, B. Hamet, R. Hildinger, S. Lipp, M. Brückner, F. Evers, and P. Drieß. "On Distributed Resource Allocation in Fully Meshed Satellite Networks". In: *International Communications Satellite Systems 2013 Conference*. Florenz, Oct. 2013.

[Wro97] J. Wroclawski. *Specification of the Controlled-Load Network Element Service*. RFC 2211 (Proposed Standard). Internet Engineering Task Force, Sept. 1997. url: http://www.ietf.org/rfc/rfc2211.txt.

[Zha+93] L. Zhang, S. Deering, D. Estrin, S. Shenker, and D. Zappala. "RSVP: A New Resource ReSerVation Protocol". In: *IEEE Network Magazine* 7.5 (Sept. 1993), pp. 116–127.

Index

Raw data and Source code

An electronic version of this thesis, all raw data used in the evaluation chapter and the relevant source code can be accessed at `https://www.geekbetrieb.de/thesis/`.