Master’s Thesis

Optimization of an Application Layer Hybrid Error Correction Scheme under Strict Delay Constraint

submitted by
Robert Gogolok
November 30, 2008

Supervisor
Prof. Dr.-Ing. Thorsten Herfet

Advisor
M.Sc. Manuel Gorius

Reviewers
Prof. Dr.-Ing. Thorsten Herfet
Prof. Dr. Raimund Seidel
Master-Thesis for
Mr. Robert Gogolok
Matr.-Nr.: xxxxxxxx

Optimization of an Application Layer Hybrid Error Correction Scheme under strict Delay Constraint

Along with the increasing importance of the Internet Protocol (IP) in home entertainment and distribution of digital media, requests arise for adequate Quality of Service (QoS) on the underlying packet-oriented network. Typical network scenarios incorporate multicast delivery as well as wireless transmission. Both lead to highly variable network conditions.

Recent researches show that especially Application Layer Hybrid Error Coding (AL-HEC) is a very efficient approach for improving the reliability of (wireless) multicast IP-networks. Whereas the Automatic Repeat Request (ARQ) cares for a fast adaptation to varying network conditions, Forward Error Correction (FEC) contributes the scalability to large groups of end devices.

Applications such as real-time media delivery or online gaming formulate strict but application specific constraints concerning the transmission delay and the target error rate. These constraints limit the space of possible coding parameter sets. In order to model the performance of various HEC schemes statistically, recently, a mathematical framework was developed at the Telecommunications Lab. The framework provides a convenient basis to formulate the optimization problem of finding the parameter set leading to the minimal amount of redundancy information on the network. In particular, the tasks to be solved are the following:

• Introducing the functionality of the most recent AL-HEC approach developed at the Telecommunications Lab and describing its parameter derivation according to the available statistical framework.
• Characterizing the properties of the optimization problems coming along with the AL-HEC scheme and describing their search spaces.
• Proposing and analyzing possible methods for finding the solutions to the optimization problems within the allowed parameter space with respect to the minimal redundancy information on the network.
• Implementing and evaluating the methods within the available software framework.

Environment:
The AL-HEC functionality is already implemented as a modification to the c++ open source RTP framework libcrtp1. The library is platform independent but a Linux-optimized solution of the implementation is preferred. A wireless home network environment prepared for the experimental transmission of DVB-IPI compliant video streams is also supplied.

Tutor: Manuel Gorius                      Supervisor: Prof. Dr.-Ing. Thorsten Herfet

1 http://www.gnu.org/software/ccrtp
Statement

Hereby I confirm that this thesis is my own work and that I have documented all sources used.

Saarbruecken, 2008-11-30

...................
Robert Gogolok

Declaration of Consent

Herewith I agree that my thesis will be made available through the library of the Computer Science Department.

Saarbruecken, 2008-11-30

...................
Robert Gogolok
Acknowledgments

I thank my supervisor Prof. Dr.-Ing. Thorsten Herfet for his insightful conversation in August and the opportunity to finish my master thesis at the Telecommunications Lab. I thank my advisor M.Sc. Manuel Gorius for his time, patience and understanding. Many thanks are due to the entire Telecommunications Lab staff of Saarland University for their support.
Special thanks go to my parents and my sister. You gave me your unconditional support through all this long process.
5 Methods of Resolution  
5.1 Full Search  
5.1.1 Algorithm  
5.1.2 Complexity  
5.2 Greedy  
5.2.1 First Transmission  
5.2.2 Retransmission  
5.2.3 Algorithm  
5.2.4 Complexity  
5.3 Improved Greedy  
5.3.1 First Transmission  
5.3.2 Retransmission  
5.3.3 Algorithm  

6 Analysis  
6.1 Performance of Algorithms  
6.1.1 General  
6.1.2 Global Optimum  
6.2 Optimization Results  
6.2.1 Adaptivity  
6.2.2 Physical Limit  
6.3 Tables  

7 Software Implementation  
7.1 Overview  
7.2 Features  
7.2.1 Testing  
7.2.2 Modularity  
7.2.3 Recursion  
7.2.4 Caching  
7.2.5 C Extension  
7.3 Issues  
7.3.1 Interpreted Language  
7.3.2 Parallel Computing  

8 Conclusions and Outlook  
8.1 Summary  
8.2 Future Work  

Bibliography  
A Analysis Data
## List of Figures

2.1 Erasure channel .................................................. 16
2.2 Erasure channel with residual error ............................ 17
2.3 Gilbert Elliot Model ............................................. 18
2.4 Forward error correction example scenario .................... 21

3.1 One sender and multiple receivers in a multiast scenario. ... 23
3.2 Generalized architecture sending packets with the first transmission ... 24
3.3 Generalized architecture using a retransmission round .......... 24
3.4 Parameter $t_{sw}$ .................................................. 26
3.5 Parameter $t_{rw}$ .................................................. 26
3.6 $N_{cc}[1]$ example scenario with one sender and receiver ........ 27
3.7 $N_{rt}[1]$ example scenario with one sender and receiver ........ 28
3.8 End-to-end delay limit ........................................... 28
3.9 Used and available time slots for an example scenario .......... 29

6.1 Number of iterations for search vs. greedy search for an example scenario ... 52
6.2 Number of iterations for greedy search vs. actual number for an example scenario ........................................... 52
6.3 Actual number of iteration steps needed by the greedy search vs. improved greedy search for an example scenario ......................... 53
6.4 Example scenario where the improved greedy algorithm returns a global optimum .................................................. 53
6.5 Example scenario showing the adaptivity of the generalized architecture ....... 55
6.6 Example scenario showing the adaptivity of the generalized architecture versus redundancy information .......................... 56
6.7 Example scenario showing the adaptivity of the generalized architecture with reduced data rate ........................................... 57

7.1 Overview of the equations implemented in the Ruby implementation and the parts implemented in a C extension .................. 62
## List of Tables

<table>
<thead>
<tr>
<th>Table</th>
<th>Description</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>3.1</td>
<td>Parameters representing current network conditions</td>
<td>25</td>
</tr>
<tr>
<td>3.2</td>
<td>Parameters of the Architecture</td>
<td>27</td>
</tr>
<tr>
<td>4.1</td>
<td>Gilbert Elliot parameters</td>
<td>31</td>
</tr>
<tr>
<td>6.1</td>
<td>Extract of optimal parameters for an example scenario when varying $D_{\text{target}}$</td>
<td>58</td>
</tr>
<tr>
<td>7.1</td>
<td>Equations and their corresponding methods in the implementation</td>
<td>60</td>
</tr>
<tr>
<td>A.1</td>
<td>Example scenario used to apply different methods of resolution</td>
<td>69</td>
</tr>
<tr>
<td>A.2</td>
<td>Iteration steps taken by the different algorithms for an example scenario</td>
<td>70</td>
</tr>
<tr>
<td>A.3</td>
<td>Example scenario where the improved greedy algorithm returns a global optimum</td>
<td>70</td>
</tr>
<tr>
<td>A.4</td>
<td>Example scenario to show adaptivity of the generalized architecture</td>
<td>70</td>
</tr>
<tr>
<td>A.5</td>
<td>Example scenario to show adaptivity of the generalized architecture with lower data rate</td>
<td>71</td>
</tr>
</tbody>
</table>
Chapter 1

Introduction

1.1 Motivation

In recent years there has been incredible growth in interconnectedness of consumer devices. Many new electronic devices are being equipped with technologies to participate in wired and wireless networks. In the home or visited network these devices are capable of providing multimedia services such as voice, data and video directly to the consumer. For instance, Wi-Fi enabled smart phones allow to watch live television broadcasts, participate in video conferences etc..

The distribution of multimedia services is thereby dominated by packet-switched networks in which packets are transferred between nodes over data links. Packet-oriented communication does not guarantee a fixed bandwidth and exclusive use of nodes in contrast to circuit switching. Since one link might be shared with many other nodes packets are queued or buffered on each network node to avoid congestion, resulting in variable delay. Supplementary packets sent over data links are exposed to transmission errors.

Media-based services require often multicast. With the multicast technique a group of nodes can be supplied with information. The advantage of multicast is that the bandwidth usage does not multiply with the group size. Duplication of packets happens only where links to the destinations split.

Because of highly variable network conditions adequate transmission of real-time media is required. Generally one aims to reach very low error rates. The nature of real-time asks additionally for avoidance of delays. For instance, Digital Video Broadcasting (DVB) services over internet protocol (IP) based networks emerge. They require a very low packet loss ratio (PLR), e.g. $10^{-6}$, under strict delay constraint, e.g. 100 milliseconds.

To maintain data integrity error detection and correction is needed. The two basic techniques to design error correcting systems are automatic repeat-request, which retransmits packets not received at the destination, and forward error correction, which transmits redundant packets with the beginning of the transmission. A combination of both, the hybrid automatic repeat-request technique, might be used where automatic repeat-request and forward error correction alone can’t reach a certain target loss ratio or delay constraint. Furthermore it is sometimes not possible or desirable to enforce error detection and correction on the network layer or below. For instance, it could be not possible to replace existing standards on the lower layers or it could be desirable to get a platform independent solution that works regardless of the underlying layers. Then the only resolution is to deploy error detection and correction on the application layer.

An approach developed at the Telecommunications Lab of Saarland University introduces a generalized architecture of erasure error correction. Common error correction techniques like automatic repeat-request, forward error correction and hybrid automatic repeat-request
were integrated to adapt to varying network conditions. A mathematical framework has been devised to express the generalized architecture. Based on the mathematical framework and a network scenario an optimization problem of finding the parameter set that accomplishes a predetermined delay constraint and target error rate is given. Additionally the optimal parameter set should be found with respect to minimal redundancy information on the network. Because network conditions change at a moment’s notice, appropriate response times for finding a solution are desirable.

1.2 Goals

The goal of this thesis can be divided into three sub goals. The first goal consists of introducing the functionality of the most recent generalized architecture developed by the Telecommunications Lab at the Saarland University. The second goal is to present a mathematical framework modeling the functionality of the most recent generalized architecture and to formulate the optimization problem inherited with the mathematical framework. To use the generalized architecture on practical systems in the real world, solutions to the optimization problem must be found. This thesis should propose and analyze possible methods for finding the solutions to the optimization problem, which is the third and last goal.

1.3 Organization

To reach these goals this thesis is structured in the following way. In chapter 2, the loss of packets on the application layer is being analyzed. Erasure channels as models of communication channels and therefore data transmission are being presented. The Gilbert Elliot Model as a more realistic model for packet loss is shown. At the end of chapter 2, automatic repeat-request, forward error correction and a hybrid automatic repeat-request are introduced as error correcting methods. Chapter 3 presents the last known approach developed at the Telecommunications Lab of Saarland University for erasure error correction at the time of this writing. The chapter includes information about the functionality of the architecture, the parameters and their limits inherited with the architecture. Afterwards, a mathematical framework for the generalized architecture is being presented in chapter 4. At the end of chapter 4, the optimization problem of finding the optimal parameter set for the generalized architecture is shown. Chapter 5 proposes and analyzes possible methods for finding the solutions to the optimization problem introduced in chapter 4. The following chapter 6 analyzes some aspects of the mathematical framework and of the methods of resolution. Some remarks concerning the implementation of the mathematical framework and the methods of resolution is presented in chapter 7.
Chapter 2

Errors, Modeling and Correction

Multimedia services on the application layer are distributed as consecutive chunks of data packets. The transport layer cares about the delivery of packets from one sender to one or more receivers. The transport of multimedia services over packet-switched networks is impaired by packet losses.

2.1 Packet Loss

Packet loss in packet-switched networks has several reasons. Some examples for packet losses are mentioned in the following. From the source to the destination a packet passes many network nodes. During high network loads packets cannot always travel immediately from one node to the next node till they reach their destination. When the transmission of a packet is being deferred, it must be buffered on the current node. Packets get lost if more packets appear at a node than the node can handle.

Compliance with rules and standards can cause packet loss. Sometimes nodes do not have the required routing information. They do not know on which link to forward the packet to reach the destination. The packet gets discarded. Another example for packet loss is the time to live (TTL) limitation in IP networks. Packets travel with an initial positive TTL value and every node that forwards the packet decreases the TTL value by one. When the TTL value reaches zero the packet gets discarded.

The transmission medium is another important source of packet loss. For instance, wireless transmission has to cope with interferences due to devices operating in the same frequency range and obstacles. Obstacles interrupt or reflect the path of signals and lead to higher error rates.

2.1.1 Erasure Channel

We have discussed so far that packets travel over unreliable paths or channels and in our case get erased as a whole in the consequence. Claude E. Shannon published a paper\[15\] on the limits of reliable transmission of data over unreliable channels. He formalized the concept of information. Furthermore, he designed an upper limit for the amount of information that can be transmitted over an unreliable channel.

A channel consists of an input alphabet, an output alphabet and a probability \( p(i, o) \) that assigns each input element \( i \) and output element \( o \) a transition probability. The transition probability is the probability of receiving the element \( o \) when the element \( i \) was sent over the channel.
Capacity

The so called \( C \) of a channel is a number which allows for a rate \( R \) arbitrarily close to the capacity to reliably transmit data over the channel. For rates above the capacity the transmission is unreliable. Shannon proved also the existence of codes that allow the probability of error at the receiver to be made arbitrarily small. There are no hints on the downside how to design such codes.

Code Rate and Redundancy

When applying error correction codes, often the length of a codeword is \( n \) and the useful information per codeword is \( k \) symbols. The code rate is defined as:

\[
    r = \frac{k}{n}
\]  

(2.1)

The code rate states what portion of the total amount of information is useful, so non redundant.

The codeword consists of \( n - k \) redundant symbols. The redundancy information (RI) is the amount of redundant information in opposite to the useful information symbols and is given by:

\[
    RI = \frac{n - k}{k}
\]  

(2.2)

Channel Model

The erasure channel is an example of a communication channel. Figure 2.1 shows an erasure channel with the input alphabet 0 and 1. The output alphabet consists of 0, 1 and an additional element \( e \) representing an erasure. Since the input alphabet is binary the erasure channel is also known as the binary erasure channel (BEC). In this erasure channel an input symbol is deleted with probability \( p_e \) and correctly received with probability \( 1 - p_e \).

![Figure 2.1: Erasure channel](image)

The channel capacity can be calculated by:

\[
    C_{ERC} = 1 - p_e
\]  

(2.3)

The redundancy information is:

\[
    RI_{ERC, min} = \frac{p_e}{1 - p_e}
\]  

(2.4)
2.1. Packet Loss

Channel Model Under Fault Tolerance

The previous channel gives a reference redundancy information value that is needed to correct the data transmission down to an arbitrarily small residual error rate. But we deal with multimedia services that allow errors to happen to a certain target loss rate.

In the following we calculate the channel capacity and required redundancy information in case of residual error, as presented in [18]. We cascade two virtual erasure channels, as depicted in figure 2.2.

\[
1 - p_e[1] \quad 1 - p_e[2]
\]

\[
\begin{array}{c}
p_e[1] \\
1 - p_e[1] \\
1 - p_e[2]
\end{array}
\quad
\begin{array}{c}
p_e[2] \\
1 - p_e[2] \\
1 - p_e[2]
\end{array}
\]

**Figure 2.2:** Erasure channel with residual error

The right erasure channel provides the residual error:

\[ p_{res} = p_e[2] \] (2.5)

The error rate of the original channel is the combined error rate of the cascaded channels. For the overall packet loss we have:

\[
p_e = 1 - (1 - p_e[1]) \cdot (1 - p_e[2]) = 1 - (1 - p_e[1]) \cdot (1 - p_{res}) \] (2.6)

Solving equation 2.6 for \( p_e[1] \) results in:

\[
p_e[1] = \begin{cases} 
\frac{p_e - p_{res}}{1 - p_{res}} & p_e \geq p_{res} \\
0 & \text{otherwise}
\end{cases} \] (2.7)

The channel capacity under fault tolerance is given by:

\[
C_{RES} = 1 - p_e[1] = \frac{1 - p_e}{1 - p_{res}} \] (2.8)

The redundancy information under fault tolerance can be calculated by:

\[
R_{IRES} = \frac{p_e[1]}{1 - p_e[1]} = \frac{p_e - p_{res}}{1 - p_e} \] (2.9)
Compared to the channel capacity and required redundancy information in the previous section of the erasure channel without residual error, the channel capacity has increased and the required redundancy information has decreased:

\[ C_{RES} = \frac{1 - p_e}{1 - p_{res}} \geq 1 - p_e = C_{ERC} \] (2.10)

\[ R_{I,RES} = \frac{p_e - p_{res}}{1 - p_e} \leq \frac{p_e}{1 - p_e} = R_{I,ERC,min} \] (2.11)

Since we don’t need to try that hard to correct errors anymore, less additional information is needed.

### 2.1.2 Gilbert Elliot Model

The erasure channels in the previous section show theoretical limits but represent not exactly the reality. Errors happen not uniformly distributed. In many situations errors happen as error bursts. Whether an error occurs or not depends additionally on the history of the channel. Therefore the channel shouldn’t be memoryless as the previous ones. The following model emphasizes the sequence of packets.

The Gilbert Elliot (GE) model, presented by Gilbert [10], characterizes the error sequences generated by data transmission channels. The state of the network gets represented by a state diagram with a two-state Markov chain [19] as depicted in figure 2.3.

![Figure 2.3: The Gilbert Elliot model used to characterize the error sequences generated by data transmission channels.](image)

The bad state \((B)\) represents an error, e.g. a packet loss. In the bad state errors occur with (high) probability \(P_B\). On the other hand the good state \((G)\) represents a nearly error-free transmission. Errors occur in the good state with (low) probability \(P_G\). The model is called simplified GE model when the loss probability for state \(G\) is zero and for state \(B\) it is one.

The model is tagged with transition probabilities. If in state \(G\), the probability to stay in the good state \(G\) is \(P_{G|G}\). The complementary event \(1 - P_{G|G}\) is the probability to go to state \(B\) when in state \(G\). On the other hand, if in state \(B\), the probability to stay in the bad state is \(P_{B|B}\). The complementary event \(1 - P_{B|B}\) is the probability to go to state \(G\) when in state \(B\). The steady state probabilities of being in state \(G\) or \(B\) are given by:

\[ \pi_G = \frac{1 - P_{B|B}}{2 - P_{G|G} - P_{B|B}} \] (2.12)

\[ \pi_B = \frac{1 - P_{G|G}}{2 - P_{G|G} - P_{B|B}} \] (2.13)

The Gilbert Elliot model is a markov chain of order 1. This means that the next transition to be done depends on the current state and is not dependent on the transitions done so far or the transitions to be made in the future.
Normally packet loss does not happen statistically independent. The occurrence of error-free or inaccurate periods and the average length of those periods suggest a correlation between consecutive packet transmissions. This behavior is also expressed in the GE model. To stay in one of the GE states for more than one transition can be interpreted as an error-free period, when in state $G$, or as a burst of errors, when in state $B$.

The single-step transition matrix for the two-state markov model is:

$$
P = \begin{pmatrix}
P_{G|G} & 1 - P_{G|G} \\
1 - P_{B|B} & P_{B|B}
\end{pmatrix}
$$

(2.14)

The $k$-step transition matrix is then:

$$
p^k = \left( \frac{1 - P_{B|B} + (1 - P_{G|G})(P_{G|G} + P_{B|B} - 1)^k}{2 - P_{G|G} - P_{B|B}} \right) (1 - P_{G|G}) \cdot (2 - P_{G|G} - P_{B|B})^{k-1}
$$

(2.15)

$$
p^k \cdot (1 - P_{B|B}) \cdot (2 - P_{G|G} - P_{B|B})^{k-1}
$$

The initial state probabilities of being in state $G$ or $B$ are specified by the probability vector $u$.

$$
u = (P_G, P_B)
$$

(2.16)

From now on we consider only the simplified GE model. If packet loss happened, the GE model is in state $B$, hence:

$$
u^n = u \cdot P^n = (0, 1) \cdot P^n
$$

(2.17)

Beginning with state $B$, the probability to stay in state $B$ after $m$ further steps is denoted by $P_{B|B}(m)$. The probability of $P_{B|B}(m)$ is given by the second component of the result of Equation 2.18 We obtain:

$$
P_{B|B}(m) = \frac{1 - P_{G|G} + (1 - P_{B|B}) \cdot (P_{G|G} + P_{B|B} - 1)^m}{2 - P_{G|G} - P_{B|B}}
$$

(2.18)

Equation 2.18 can also be written as:

$$
P_{B|B}(m) = \frac{1 - P_{G|G}}{2 - P_{G|G} - P_{B|B}} + \frac{1 - P_{B|B}}{2 - P_{G|G} - P_{B|B}} \cdot (P_{G|G} + P_{B|B} - 1)^m
$$

(2.19)

Substituting 2.13 and 2.18 into 2.19 leads to

$$
P_{B|B}(m) = \pi_B + \pi_G \cdot (P_{G|G} + P_{B|B} - 1)^m
$$

(2.20)

### 2.2 Automatic Repeat-Request

Before one can recover from a lost packet, the packet loss must be detected. Error detection is the capability of detecting the presence of errors. Redundant information must be added to the original data for error detection schemes.

The automatic repeat-request (ARQ) technique allows a reliable data transmission between sender and receiver through packet retransmission. Automatic repeat-request is dependent on a return channel from the receiver to the sender. The receiver signals his error detection results over the return channel. Provided with the error detection results the sender can retransmit lost packets to the receiver. The signal sent over the return channel as detection result is either an acknowledgment (ACK) for correctly received packets or a negative acknowledgment (NACK) for missing packets.
Stop-and-wait

The simplest kind of automatic repeat-request method is Stop-and-wait ARQ. The sender sends one packet at a time. Before the sender sends the next packet, the receiver has to acknowledge the sender the latest packet sent to him. If the sender doesn’t receive the ACK before a certain time, he will resend the package to the receiver. This procedure is repeated till the sender receives an ACK.

Go-Back-N

The Go-Back-N ARQ sends several packets in advance to the receiver and waits for their ACKs. Those packets are in a so-called window and that window is limited by a window size. The sender remembers also the last acknowledged packet. When an ACK is received, the sender will know that the receiver received all packets up to that packet and will adjust the beginning of a window if the ACK is newer than the last acknowledged packet at the sender. A new beginning of the window allows the sender to send packets not yet transmitted in the new window. If any ACK is missing after a certain time at the sender, all packets in the current window are retransmitted. The receiver will ACK in-order packets and ignore out-of-order packets. The throughput is increased in opposite to Stop-and-wait ARQ because there are less waiting periods.

Selective Repeat

The Selective Repeat ARQ enhances the Go-Back-N ARQ. The receiver has also a window. The window has the same size as the sender window. The receiver accepts and acknowledges out-of-order packets in the current window. In-order packets at the receiver advance the receiver window. The receiver must have therefore a large enough buffer to acknowledge packets from the sender older than the current beginning of the receiver window suggests. In addition, the receiver can help the sender by sending NACKs to avoid the waiting time at the sender till a timeout occurs. The sender will not send all packets of the current window after a timeout, but the unacknowledged ones.

Issue

Automatic repeat-request does not scale with increasing number of receivers. Independent packet losses between different receivers lead to an increase of feedback messages via the return channel and thereby also increase the network load. The retransmitted packets due to feedback messages also contribute to the network load.

Adaptivity

On the other hand the Selective Repeat ARQ is highly adaptive. When packets get lost at the receiver side, the receiver feedbacks only to resend redundant packets for the lost packets. The amount of redundancy sent in the retransmission rounds adapts to the current error rate.

2.3 Forward Error Correction

Contrary to automatic repeat-request, the forward error correction (FEC) technique does not require packet retransmission. There is no need for a return channel. When using forward error correction the transmitter encodes the data to be sent in a redundant way, also known as an error correction code, such that not only error detection is possible on
the receiver side, but also the correction of lost packets to a certain level. This is especially useful when additional retransmission stages are not possible, e.g. due to delay constraints. The design of the code determines the maximum fraction of errors that can be corrected. Different conditions require therefore different forward error correcting codes. When the output of an FEC encoder at the transmitter includes the unmodified input, the code is systematic, otherwise nonsystematic. When applying systematic forward error correction in packet-oriented transmission, the transmitter sends $n$ packets to the receiver. The $n$ packets consist of $k$ data packets and $n-k$ redundant packets. The receiver can reconstruct the original $k$ data packets, if at least $k$ packets out of $n$ packets reach the receiver. The ratio $\frac{n-k}{k}$ indicates the amount of redundancy added.

![Figure 2.4](image)

**Figure 2.4**: Forward error correction example scenario demonstrating error correction capabilities.

Figure 2.4 shows an example of a sender transmitting $n = 5$ packets from sender to receiver. The forward error correction mechanism adds additional $n-k = 2$ redundant packets ($r1$ and $r2$) to the original $k = 3$ data packets ($d1$, $d2$ and $d3$) via the FEC encoder. On the receiver side only two data packets ($d1$ and $d3$) and one redundant packet ($r1$) arrive. Two packets have been dropped ($r2$ and $d2$) on their way to the receiver. The FEC decoder at the receiver is still able to reconstruct the lost data packet ($d2$) since at least $k = 3$ packets have arrived.

### 2.4 Hybrid ARQ

The hybrid automatic repeat-request (HARQ) technique [12] is a combination of ARQ and FEC. There are basically two different versions of HARQ to adapt to network conditions and scale with numerous receivers. The Type I HARQ scheme sends a certain amount of parity packets with the first transmission to the receiver. If the receiver is not able to reconstruct the data because of a high loss rate, ARQ is used to retransmit more parity packets. The Type II HARQ is similar to Type HARQ. The difference is that Type II HARQ scheme sends only information to detect errors on the receiver side in the first transmission, but no information to eventually reconstruct the data. That way less information is send initially.
Chapter 3

Generalized Architecture of Erasure Error Recovery

An approach developed at the Telecommunications Lab of Saarland University introduces a generalized architecture of erasure error correction [16]. It integrates existing techniques like automatic-repeat-request and forward error correction. Furthermore ARQ and FEC can be used in combination. Consequently the generalized architecture supports adaptivity to react on network changes. Also a generic retransmission based scheme will be used rather than focussing on a specific transport protocol. In the following two sections 3.1 and 3.2 a brief overview of the architecture is given. Afterwards, the parameters describing the generalized architecture are presented in section 3.3. At last, the effect of the delay limit is discussed in section 3.4.

3.1 First Transmission

Without loss of generality, one sender wants to transmit data to one or more receivers in a multicast scenario.

\[ N_{blk} = k + N_p \]  

(3.1)
packets to all receivers with the first transmission, see figure 3.2 where:

\[ k > 0, N_p \geq 0 \]  

If \( N_p \) is set to zero, there is no redundancy added in the first transmission. The \( k \) data packets are sent alone to the receivers without protection. The transmission fails when at least one data packet gets lost. If \( k \) is set to one and \( N_p \) is greater than zero, the additional \( N_p \) packets are copies of the source data packet. In this case the transmission is successful if at least one of the \( 1 + N_p \) transmitted packets reaches each receiver. Forward error correction is being used if \( k \) is greater than one and \( N_p > 0 \). Then any \( k \) packets out of \( k + N_p \) packets need to reach each receiver to be successful.

\[ k + N_p \text{ packets} \]

Figure 3.2: The sender multicasts up to \( k + N_p \) packets to every receiver with the first transmission.

3.2 Retransmission

The first transmission might be unsuccessful in restoring the original source data on one or more receivers. Each affected receiver can introduce an additional transmission round by transferring a NACK message via a return channel back to the sender. This NACK message contains information about the missing packets.

 Upon getting the first NACK message, the sender will multicast supplementary redundant data to the receiver to restore the source data. Other NACK messages could arrive at the sender from the other receivers after the first NACK message has arrived and was handled. If the NACK message requires more redundant packets than the first one, the sender will multicast further redundant packets to the receivers. Otherwise the sender will neglect those NACK messages. In the later case the sender is doing feedback suppression.
3.3 Parameters

The receivers might be able to restore the original source data after the additional transmission round. If this is still not the case, there may be some more transmission rounds initiated by another NACK message.

In addition to the feedback suppression at the sender, the receivers are also able to do feedback suppression. If there are many receivers and each one would feedback missing packets by NACK messages, there would be a significant increase in traffic. On the other hand, the receivers may share the same number of needed redundant packets. A receiver transmits his NACK message via multicast so that other receivers can exploit the information present in that NACK message if they see the message on the way back to the sender. The other receivers may e.g. suppress their NACK message if they have seen a NACK message with the equal or larger number of needed redundant packets.

3.3 Parameters

The parameters that describe the generalized architecture can be grouped in parameters describing the current network conditions and parameters that must be set to adapt to the current network conditions.

3.3.1 Network Parameters

The parameters representing the current network conditions are listed in table 3.1.

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$PLR_{\text{target}}$</td>
<td>required maximum packet loss rate</td>
<td>scalar</td>
</tr>
<tr>
<td>$D_{\text{target}}$</td>
<td>maximum delay</td>
<td>scalar</td>
</tr>
<tr>
<td>$N_{\text{recv}}$</td>
<td>number of receivers</td>
<td>scalar</td>
</tr>
<tr>
<td>$\text{RTT}$</td>
<td>average round trip time for each receiver, one way delay is $\text{RTT}_j / 2$</td>
<td>vector</td>
</tr>
<tr>
<td>$t_s$</td>
<td>average time between two data packets at the sender</td>
<td>scalar</td>
</tr>
<tr>
<td>$t_{s,\text{min}}$</td>
<td>minimum packet interval</td>
<td>scalar</td>
</tr>
<tr>
<td>$t_{sw}$</td>
<td>average waiting time at the sender</td>
<td>scalar</td>
</tr>
<tr>
<td>$t_{rw}$</td>
<td>average waiting time at the receiver</td>
<td>scalar</td>
</tr>
<tr>
<td>$t_{lp}$</td>
<td>for each receiver the time between detecting packet loss at the receiver and possibly receiving the required packets in the subsequent retransmission round</td>
<td>vector</td>
</tr>
</tbody>
</table>

Table 3.1: Parameters representing current network conditions

First of all, the parameter $PLR_{\text{target}}$ corresponds to the predefined maximum packet loss rate. The maximum delay on the other hand is expressed by $D_{\text{target}}$. The number of receivers in the multicast scenario is stated by the parameter $N_{\text{recv}}$. The variable $j$ denotes the $j$-th receiver in the following. It is $1 \leq j \leq N_{\text{recv}}$ if not otherwise stated. In the following the variable $j$ serves as an index for all mentioned vectors, which have all $j$ entries, unless otherwise stated.

The parameter $t_s$ denotes the average time interval between two data packets at the transmitter. The value of $t_s$ may be derived from the stream data rate. In addition, there’s a minimal packet interval $t_{s,\text{min}}$ given that can be derived from the channel data rate. The average waiting time at the sender $t_{sw}$ is the time between receiving a NACK message from the receiver and the time the redundant packets required by the NACK message are transmitted by the sender to the receiver, as depicted in figure 3.4.
The average waiting time at each receiver $t_{rw}$ is the time between the detection of packet loss on receiver side and the time the corresponding NACK is sent as depicted in figure 3.5. The parameter $t_{rw}$ is the same for every receiver since the process for all receivers is the same.

The time between detecting packet loss on the $j$-th receiver and possibly receiving the required packets in the subsequent retransmission round can be expressed through the parameters $RTT$, $t_{sw}$ and $t_{rw}$ as:

$$t_{lp}[j] = t_{rw} + \frac{RTT[j]}{2} + t_{sw} + \frac{RTT[j]}{2} = t_{rw} + RTT[j] + t_{sw}. \quad (3.3)$$

### 3.3.2 Parameters of the Generalized Architecture

All adaptable parameters of the architecture are summarized in table 3.2. The parameters $k$, $N_p$ and $N_{rr}$ were already mentioned in the previous introductional sections. The remaining parameters $N_{cc}$ and $N_{rt}$ influence the amount of redundant packets sent in the retransmission rounds by the sender. These vectors are of size $N_{rr}$. If forward error
3.4. Constraints

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>$k$</td>
<td>number of source data packets</td>
<td>scalar</td>
</tr>
<tr>
<td>$N_p$</td>
<td>number of redundant packets sent with the first transmission</td>
<td>scalar</td>
</tr>
<tr>
<td>$N_{rr}$</td>
<td>number of transmission rounds after the first transmission</td>
<td>scalar</td>
</tr>
<tr>
<td>$N_{cc}$</td>
<td>multiplier for redundant packets in every retransmission round</td>
<td>vector</td>
</tr>
<tr>
<td>$N_{rt}$</td>
<td>number of copies for each retransmission packet in every retransmission round</td>
<td>vector</td>
</tr>
</tbody>
</table>

Table 3.2: Parameters of the Architecture

correction is used, that means $k > 1$, and $r$ redundant packets are needed at the receiver side, the value $N_{cc}[q]$ specifies for the $q$-th retransmission round that $r \cdot N_{cc}[q]$ redundant packets will be sent.

Figure 3.6 shows an example with one sender, one receiver and $k > 1$. The receiver indicates it requires $r = 2$ redundant packets in the first retransmission to recover all data packets sent in the first transmission. Since $N_{cc}[1]$ equals to two in this example, the sender will transmit $r \cdot N_{cc}[1] = 2 \cdot 2 = 4$ packets with the first retransmission.

![Figure 3.6: $N_{cc}[1]$ example scenario with one sender and receiver](image)

If $k$ equals to one, the redundant packets sent in the retransmission rounds are copies of the original data packet sent in the first transmission. The value $N_{rt}[q]$ specifies for the $q$-th retransmission round that $N_{rt}[q]$ redundant packets will be sent to the receiver. Those redundant packet get sent with the average time interval $t_{s, min}$ between each other.

Figure 3.7 shows an example with one sender, one receiver and $k$ equal to 1. The receiver indicates he missed the transmitted data packet in the first transmission. Since $N_{rt}[1]$ equals to three in this example the sender will transmit $N_{rt}[1] = 3$ packets, separated by the time interval $t_{s, min}$, with the first retransmission.

Furthermore depending on whether $N_{cc}$ or $N_{rt}$ is used, the other parameter is useless and should therefore have no impact:

$$N_{cc}[q] = 1 \quad \text{for } k = 1 \quad \quad \text{(3.4)}$$

$$N_{rt}[q] = 1 \quad \text{for } k > 1 \quad \text{(3.5)}$$

3.4 Constraints

By adjusting the number of retransmission rounds and the number of redundant packets sent, among others, one tries to target a low or predefined packet loss rate. Sending packets
and the logic behind the generalized architecture costs time. Therefore the strict delay limit will limit the parameters of the architecture to a reasonable scope.

Since all of the receivers share the same parameters related to the architecture, the worst receiver can guarantee the requirements for every other receiver in a multicast scenario. Therefore, without loss of generality, it is assumed the first receiver is the worst receiver. The worst receiver is the one with the largest packet loss rate and largest round-trip time.

We derive now a formula for the end-to-end delay limit for the first receiver. The maximum possible end-to-end delay includes three parts. The first part is the one-way delay in the first transmission, which is \( \frac{RTT[1]}{2} \). Then we need to add the decoding delay at the receiver, which is \( k \cdot t_s \). The third and last part includes \( N_{rr} \) of two-way delays caused by the retransmission round, which is \( N_{rr} \cdot t_{lp}[1] \).

Eventually the maximum possible end-to-end delay for the retransmission packets of the first receiver must satisfy:

\[
\frac{RTT[1]}{2} + k \cdot t_s + N_{rr} \cdot t_{lp}[1] \leq D_{target}
\]  

(3.6)

The end-to-end delay is exemplary depicted in figure 3.8.

---

**Figure 3.7:** \( N_{rt}[1] \) example scenario with one sender and receiver

**Figure 3.8:** The end-to-end delay limit between sender and receiver using initial and additional retransmission rounds.
By adjusting the number of retransmission rounds and the number of redundant packets sent, among others, one tries to target a low or predefined packet loss rate. Due to the imposed and strict delay limit in a network scenario some parameters of the architecture have to be limited in a reasonable scope. The maximum number of retransmission rounds $N_{rr,max}$ is e.g. subject to the strict delay limit.

By solving unequation 3.6 for $N_{rr}$ and since the value of the parameter $k$ is at least 1, the maximum number of retransmission rounds $N_{rr,max}$ is constrained by:

$$N_{rr,max} = \left\lfloor \frac{D_{target} - \frac{\text{RTT}[1]}{2} - 1 \cdot t_s}{t_{lp}[1]} \right\rfloor$$  (3.7)

Similar the length of the parameter $k$ depends on the number of retransmission rounds $N_{rr}$, denoted by $k(N_{rr})$, and the maximal $k$ using $N_{rr}$ retransmission rounds is given by:

$$k_{\text{max}}(N_{rr}) = \left\lfloor \frac{D_{target} - N_{rr} \cdot t_{lp}[1] - \frac{\text{RTT}[1]}{2}}{t_s} \right\rfloor$$  (3.8)

In the following we derive limits for the number of redundant packets in the first transmission and retransmission rounds. The affected parameters are $N_p$, $N_{cc}$ and $N_{rt}$.

The data packets are emitted at the sender in constant time $t_s$. The redundant packets have to fit into the slots between the data packets. They are multiplexed into the subsequent blocks. We simplify now the analysis by assuming that the parameters remain constant for several blocks. Under this assumption all redundant packets for one encoding block have to fit into the transmission period of the block itself instead of the next ones. Figure 3.9 demonstrates the available time slots that can be used for redundant packets. We have $k = 3$ and $t_{s,min}$ is a third of $t_s$, so one encoding block spans 9 $t_{s,min}$ time slots. But there are only 3 time slots consumed out of 9 time slots.

![Figure 3.9: Used and available time slots for an example scenario with $k$ equal to 3. The parameter $t_{s,min}$ is one third of $t_s$ in this scenario.](image)

The maximal number of redundant packets per encoding block is given by:

$$RI_{\text{max}}(k) = \left\lfloor \frac{t_s - t_{s,min}}{t_{s,min}} \cdot k \right\rfloor$$  (3.9)

Using equation 3.9 we can calculate that there are $2 \cdot 3 = 6$ time slots available for redundant packets in the previous example. Knowing the number of available time slots for redundant packets, we can now allocate those to the parameters $N_p$, $N_{cc}$ and $N_{rt}$. 

---

**3.4. Constraints**

By adjusting the number of retransmission rounds and the number of redundant packets sent, among others, one tries to target a low or predefined packet loss rate. Due to the imposed and strict delay limit in a network scenario some parameters of the architecture have to be limited in a reasonable scope. The maximum number of retransmission rounds $N_{rr,max}$ is e.g. subject to the strict delay limit.

By solving unequation 3.6 for $N_{rr}$ and since the value of the parameter $k$ is at least 1, the maximum number of retransmission rounds $N_{rr,max}$ is constrained by:

$$N_{rr,max} = \left\lfloor \frac{D_{target} - \frac{\text{RTT}[1]}{2} - 1 \cdot t_s}{t_{lp}[1]} \right\rfloor$$  (3.7)

Similar the length of the parameter $k$ depends on the number of retransmission rounds $N_{rr}$, denoted by $k(N_{rr})$, and the maximal $k$ using $N_{rr}$ retransmission rounds is given by:

$$k_{\text{max}}(N_{rr}) = \left\lfloor \frac{D_{target} - N_{rr} \cdot t_{lp}[1] - \frac{\text{RTT}[1]}{2}}{t_s} \right\rfloor$$  (3.8)

In the following we derive limits for the number of redundant packets in the first transmission and retransmission rounds. The affected parameters are $N_p$, $N_{cc}$ and $N_{rt}$.

The data packets are emitted at the sender in constant time $t_s$. The redundant packets have to fit into the slots between the data packets. They are multiplexed into the subsequent blocks. We simplify now the analysis by assuming that the parameters remain constant for several blocks. Under this assumption all redundant packets for one encoding block have to fit into the transmission period of the block itself instead of the next ones. Figure 3.9 demonstrates the available time slots that can be used for redundant packets. We have $k = 3$ and $t_{s,min}$ is a third of $t_s$, so one encoding block spans 9 $t_{s,min}$ time slots. But there are only 3 time slots consumed out of 9 time slots.

The maximal number of redundant packets per encoding block is given by:

$$RI_{\text{max}}(k) = \left\lfloor \frac{t_s - t_{s,min}}{t_{s,min}} \cdot k \right\rfloor$$  (3.9)

Using equation 3.9 we can calculate that there are $2 \cdot 3 = 6$ time slots available for redundant packets in the previous example. Knowing the number of available time slots for redundant packets, we can now allocate those to the parameters $N_p$, $N_{cc}$ and $N_{rt}$. 

---

**3.4. Constraints**

By adjusting the number of retransmission rounds and the number of redundant packets sent, among others, one tries to target a low or predefined packet loss rate. Due to the imposed and strict delay limit in a network scenario some parameters of the architecture have to be limited in a reasonable scope. The maximum number of retransmission rounds $N_{rr,max}$ is e.g. subject to the strict delay limit.

By solving unequation 3.6 for $N_{rr}$ and since the value of the parameter $k$ is at least 1, the maximum number of retransmission rounds $N_{rr,max}$ is constrained by:

$$N_{rr,max} = \left\lfloor \frac{D_{target} - \frac{\text{RTT}[1]}{2} - 1 \cdot t_s}{t_{lp}[1]} \right\rfloor$$  (3.7)

Similar the length of the parameter $k$ depends on the number of retransmission rounds $N_{rr}$, denoted by $k(N_{rr})$, and the maximal $k$ using $N_{rr}$ retransmission rounds is given by:

$$k_{\text{max}}(N_{rr}) = \left\lfloor \frac{D_{target} - N_{rr} \cdot t_{lp}[1] - \frac{\text{RTT}[1]}{2}}{t_s} \right\rfloor$$  (3.8)

In the following we derive limits for the number of redundant packets in the first transmission and retransmission rounds. The affected parameters are $N_p$, $N_{cc}$ and $N_{rt}$.

The data packets are emitted at the sender in constant time $t_s$. The redundant packets have to fit into the slots between the data packets. They are multiplexed into the subsequent blocks. We simplify now the analysis by assuming that the parameters remain constant for several blocks. Under this assumption all redundant packets for one encoding block have to fit into the transmission period of the block itself instead of the next ones. Figure 3.9 demonstrates the available time slots that can be used for redundant packets. We have $k = 3$ and $t_{s,min}$ is a third of $t_s$, so one encoding block spans 9 $t_{s,min}$ time slots. But there are only 3 time slots consumed out of 9 time slots.

The maximal number of redundant packets per encoding block is given by:

$$RI_{\text{max}}(k) = \left\lfloor \frac{t_s - t_{s,min}}{t_{s,min}} \cdot k \right\rfloor$$  (3.9)

Using equation 3.9 we can calculate that there are $2 \cdot 3 = 6$ time slots available for redundant packets in the previous example. Knowing the number of available time slots for redundant packets, we can now allocate those to the parameters $N_p$, $N_{cc}$ and $N_{rt}$.
The maximal number of redundant packets in the first transmission, when allowing \( N_{rr} \) retransmission rounds, is given by:

\[
N_{p,\text{max}}(k, N_{rr}) = RI_{\text{max}}(k) - N_{rr}
\]

(3.10)

The values of \( RI_{\text{max}}(k) \) and \( N_p \) limit the entries of \( N_{cc} \) and \( N_{rt} \), respectively, by:

\[
N_p + \sum_{q=1}^{N_{rr}} N_{rt}[q] \leq RI_{\text{max}}(k), \text{ if } k = 1
\]

(3.11)

\[
N_p + \sum_{q=1}^{N_{rr}} N_{cc}[q] \leq RI_{\text{max}}(k), \text{ if } k > 1
\]

(3.12)

Using equations 3.9 and 3.10 and the current instance of \( N_p \), each element of \( N_{cc} \) and \( N_{rt} \), respectively, is limited by:

\[
M(k, N_p, N_{rr}) = RI_{\text{max}}(k) - N_p - N_{rr} + 1
\]

(3.13)

\[
N_{rt}[q] \leq M(N_{rr}, N_p), \text{ if } k = 1
\]

(3.14)

\[
N_{cc}[q] \leq M(N_{rr}, N_p), \text{ if } k > 1
\]

(3.15)
Chapter 4
Mathematical Framework

Given a network scenario with its strict delay limit and maximum packet loss rate, we want to apply the generalized architecture. The parameters of the architecture must meet the strict delay limit and a maximum packet loss rate with a minimum of redundancy information. The question is how to choose the parameters of the architecture.

A mathematical model that comprehends the parameters of the generalized architecture and can calculate the packet loss rate and redundancy information for a specific parameter set in a network scenario is needed. That way the different parameter sets can be compared and the optimal parameters can be derived.

In the following one specific mathematical model developed by the Telecommunications Lab at the Saarland University \[16\] is presented.

### 4.1 Parameters

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
<th>Type</th>
</tr>
</thead>
<tbody>
<tr>
<td>( \pi_G )</td>
<td>steady state probability of being in state G for each receiver</td>
<td>vector</td>
</tr>
<tr>
<td>( \pi_B )</td>
<td>steady state probability of being in state B for each receiver</td>
<td>vector</td>
</tr>
<tr>
<td>( P_G )</td>
<td>probability of an error when in state G for each receiver</td>
<td>vector</td>
</tr>
<tr>
<td>( P_B )</td>
<td>probability of an error when in state B for each receiver</td>
<td>vector</td>
</tr>
<tr>
<td>( P_{G</td>
<td>G} )</td>
<td>probability stay in state G when in state G for each receiver</td>
</tr>
<tr>
<td>( P_{B</td>
<td>B} )</td>
<td>probability stay in state B when in state B for each receiver</td>
</tr>
</tbody>
</table>

**Table 4.1:** Gilbert Elliot parameters

The mathematical framework uses the Gilbert Elliot model. Therefore to each receiver \( j \) are assigned channel state information (CSI). Table 4.1 lists all parameter needed in this chapter related to the Gilbert Elliot model. The channel state information include the probabilities \( P_{G|G} \) and \( P_{B|B} \) as well as \( P_G \) and \( P_B \) of a Gilbert Elliot model for each receiver. Steady state probabilities are expressed by the parameters \( \pi_G \) and \( \pi_B \).

### 4.2 Random Variables

In order to derive equations for the packet loss rate and redundancy information of a parameter set, we use the random variables \( N_{\text{req}}(j), N_{\text{req,max}} \) and \( I_{k}(j,w) \).

The variable \( N_{\text{req}}(j) \) represents the minimal number of redundant packets needed at the \( j \)-th receiver to recover all \( k \) data packets after the first transmission. The value of \( N_{\text{req}}(j) \)
is therefore in the range

$$0 \leq N_{\text{req}}(j) \leq k.$$ \hspace{1cm} (4.1)

If one goes one step further, the random variable $N_{\text{req, max}}$, where $0 \leq N_{\text{req, max}} \leq k$, represents the minimal number of redundant packets needed to recover all missing data packets for all receivers. The random variable $N_{\text{req, max}}$ is given by selecting the biggest $N_{\text{req}}(j)$ value of all receivers, because the receiver with the highest $N_{\text{req}}(j)$ value will also satisfy the needed redundant packets for all other receivers to recover all $k$ data packets after the first transmission.

$$N_{\text{req, max}} = \max(N_{\text{req}}(1), \ldots, N_{\text{req}}(N_{\text{recv}}))$$ \hspace{1cm} (4.2)

The random variable $I_k(j, w)$ represents the number of missing data packets at the $j$-th receiver after experiencing $w$ retransmission rounds. The value of $I_k(j)$ is also in the range

$$0 \leq I_k(j) \leq k.$$ \hspace{1cm} (4.3)

### 4.3 Simplification

To simplify the analysis of the mathematical model some assumptions have been made regarding the retransmission rounds. It is assumed that the feedback channel from receiver to sender is error-free. Also the perfect suppression of NACK messages among all $N_{\text{recv}}$ receivers is assumed.

### 4.4 Packet Loss Rate Calculation

The packet loss rate for the $j$-th receiver can be calculated by the expected number of missing data packets after experiencing the $w$-th retransmission round:

$$\text{PLR}(j, w) = \frac{E(I_k(j, w))}{k} = \frac{\sum_{i=1}^{k} i \cdot Pr(I_k(j, w) = i)}{k}$$ \hspace{1cm} (4.4)

In the following subsections the probability function $Pr(I_k(j, w))$ of a given parameter set is being derived to complete equation 4.4.

#### 4.4.1 Loss of Packets in GE Channel Model

The probability $P(m, d, j)$ expresses the loss of $m$ packets in a sequence of $d$ packets sent for the $j$-th receiver in a Gilbert Elliot channel model. $P(m, d, j)$ can be calculated by a double recursion as proposed in [13].

The probability of $m$ errors in $d$ transmissions for the $j$-th receiver and the non-simplified Gilbert Elliot model is given by:

$$P(m, d, j) = P_G(m, d, j) + P_B(m, d, j)$$ \hspace{1cm} (4.5)

$P_G(m, n, j)$ is the probability of $m$ errors in $n$ transmissions with the end state $G$ for the $j$-th receiver. On the other hand, $P_B(m, n, j)$ denotes the probability of $m$ errors in $n$ transmissions with end state $B$ for the $j$-th receiver.
4.4. Packet Loss Rate Calculation

\[ P_G(m,n,j) = P_G(m,n-1,j) \cdot P_G[G][j] \cdot (1 - P_G[j]) + P_B(m,n-1,j) \cdot (1 - P_B[j]) \cdot (1 - P_G[j]) \]
\[ + P_G(m-1,n-1,j) \cdot P_G[G][j] \cdot P_G[j] + P_B(m-1,n-1,j) \cdot (1 - P_B[j]) \cdot P_G[j] \] (4.6)

\[ P_B(m,n,j) = P_B(m,n-1,j) \cdot P_B[G][j] \cdot (1 - P_B[j]) + P_G(m,n-1,j) \cdot (1 - P_G[G][j]) \cdot (1 - P_B[j]) \]
\[ + P_B(m-1,n-1,j) \cdot P_B[G][j] \cdot P_B[j] + P_G(m-1,n-1,j) \cdot (1 - P_B[j]) \cdot P_B[j] \] (4.7)

In the case of the simplified Gilbert Elliot model, where \( P_G \) is zero and \( P_B \) is set to one, the equations 4.6 and 4.7 can be simplified to:

\[ P_G(m,n,j) = P_G(m,n-1,j) \cdot P_G[G][j] + P_B(m,n-1,j) \cdot (1 - P_B[j]) \cdot (1 - P_G[j]) \] (4.8)

\[ P_B(m,n,j) = P_B(m-1,n-1,j) \cdot P_B[B][j] + P_G(m-1,n-1,j) \cdot (1 - P_G[G][j]) \cdot P_B[j] \] (4.9)

The initial conditions for the recursion are

\[ P_B(0,0,j) = \frac{1 - P_B[B][j]}{1 - P_B[B][j] + 1 - P_G[G][j]} \] (4.10)

\[ P_G(0,0,j) = \frac{1 - P_B[B][j]}{1 - P_G[G][j] + 1 - P_B[B][j]} \] (4.11)

and

\[ P_B(m,0) = P_G(m,0) = 0, m \neq 0 \] (4.12)

\[ P_B(m,n) = P_G(m,n) = 0, m > n \] (4.13)

4.4.2 Special Aggregates

Let the set of all receivers be defined by:

\[ Z = \{1, \ldots, N_{\text{recv}}\} \] (4.14)

Further on the \( m \)-th way of choosing \( h \) elements out of the set \( Z \) without repetition is denoted by the subset \( Z(h,m) \). In total there are \( \binom{N_{\text{recv}}}{h} \) ways of choosing \( h \) elements out of the set \( Z \). The function \( e(a,m) \) identifies the \( a \)-th element in the subset \( Z(h,m) \), where \( 1 \leq a \leq h \). Then the subset \( Z(h,m) \) can be written as:

\[ Z(h,m) = \{e(1,m), \ldots, e(h,m)\} \] (4.15)
Let’s have a look at an example. Let $Z$ be the set $\{1, 2, 3\}$. We want to choose $h = 2$ elements out of the set $Z$. There are $m = \binom{3}{2} = 3$ ways to choose those two elements:

$$Z(2, 1) = \{e(1, 1), e(2, 1)\} = \{1, 2\}$$
$$Z(2, 2) = \{e(1, 2), e(2, 2)\} = \{1, 3\}$$
$$Z(2, 3) = \{e(1, 3), e(2, 3)\} = \{2, 3\}$$

The complement of the subset $Z(h, m)$ is denoted by $\overline{Z}(h, m)$. It contains $N_{\text{recv}} - h$ elements and is itself a subset of $Z$. The function $\overline{\tau}(b, m)$ identifies the $b$-th element in the subset $\overline{Z}(h, m)$, where $1 \leq b \leq N_{\text{recv}} - h$. Then the subset $\overline{Z}(h, m)$ can be written as:

$$\overline{Z}(h, m) = \{\overline{\tau}(1, m), \ldots, \overline{\tau}(N_{\text{recv}} - h, m)\}$$ (4.16)

In the above example the corresponding complements would be:

$$\overline{Z}(2, 1) = \{\overline{\tau}(1, 1)\} = \{3\}$$
$$\overline{Z}(2, 2) = \{\overline{\tau}(1, 2)\} = \{2\}$$
$$\overline{Z}(2, 3) = \{\overline{\tau}(1, 3)\} = \{1\}$$

We define $Z^j$ as a set with all receivers without the $j$-th receiver as:

$$Z^j = Z \setminus \{j\}$$ (4.17)

Similar to the previous definitions of $Z(h, m)$ the $m$-th way of choosing $h$ elements out of the set $Z^j$ is denoted by the subset $Z^j(h, m)$. In total there are $\binom{N_{\text{recv}} - 1}{h}$ ways of choosing $h$ elements out of $Z^j$. The function $\epsilon^j(a, m)$ identifies the $a$-th element in the subset $Z^j(h, m)$, where $1 \leq a \leq h$. Then the subset $Z^j(h, m)$ can be written as:

$$Z^j(h, m) = \{\epsilon^j(1, m), \ldots, \epsilon^j(h, m)\}$$ (4.18)

For instance, let $Z$ be the set $\{1, 2, 3, 4\}$. Then, if $j$ equals to two, $Z^2$ is the set

$$Z^2 = \{1, 2, 3, 4\} \setminus \{2\} = \{1, 3, 4\}$$ (4.19)

We want to choose $h = 2$ elements out of the set $Z^2$. There are $m = \binom{4-1}{2} = 3$ ways to choose those two elements:

$$Z^2(2, 1) = \{\epsilon(1, 1), \epsilon(2, 1)\} = \{1, 3\}$$
$$Z^2(2, 2) = \{\epsilon(1, 2), \epsilon(2, 2)\} = \{1, 4\}$$
$$Z^2(2, 3) = \{\epsilon(1, 3), \epsilon(2, 3)\} = \{3, 4\}$$

The complement of the subset $Z^j(h, m)$ is denoted by $\overline{Z}^j(h, m)$. It contains $N_{\text{recv}} - h - 1$ elements and is itself a subset of $Z$. The function $\overline{\nu}(b, m)$ identifies the $b$-th element in the subset $\overline{Z}^j(h, m)$, where $1 \leq b \leq N_{\text{recv}} - h - 1$. Then the subset $\overline{Z}^j(h, m)$ can be written as:
4.4. Packet Loss Rate Calculation

\[
Z^j(h, m) = \{\tau^j(1, m), \ldots, \tau(N_{\text{recv}} - h - 1, m)\}
\] (4.20)

In the above example the corresponding complements would be:

\[
Z^2(2, 1) = \{\tau(1, 1)\} = \{4\}
\]
\[
Z^2(2, 2) = \{\tau(1, 2)\} = \{3\}
\]
\[
Z^2(2, 3) = \{\tau(1, 3)\} = \{1\}
\]

4.4.3 Loss in First Transmission

The probability of \(N_{\text{req}}(j)\) being \(i\) is denoted by:

\[
P_{\text{req}}^i(j) = Pr(N_{\text{req}}(j) = i) = P(N_p + i, N_{\text{blk}}, j) \quad (4.21)
\]

The probability of \(N_{\text{req}}(j)\) being less than \(i\) is denoted by:

\[
P_{\text{req}}^{<i}(j) = Pr(N_{\text{req}}(j) < i) = \sum_{g=0}^{N_p+i-1} P(g, N_{\text{blk}}, j) \quad (4.22)
\]

We assume now that \(N_{\text{req,max}}\) is \(i\). That means \(h\) receivers lost \(N_p + i\) packets and the other \(N_{\text{recv}} - h\) receivers lost less than \(N_p + i\) packets, which is expressed by the function:

\[
P_{N_{\text{req,max}}}^i(h) = \left(\binom{N_{\text{recv}}}{h}\right) \prod_{a=1}^{h} P_{\text{req}}^a(e(a, g)) \prod_{b=1}^{N_{\text{recv}}-h} P_{\text{req}}^{<i}(\tau(b, g)) \quad (4.23)
\]

The general case of any \(h\) receivers losing \(N_p + i\) packets and the other \(N_{\text{recv}} - h\) receivers losing less than \(N_p + i\) packets is expressed by:

\[
P_{N_{\text{req,max}}}^i(h, j) = \left(\binom{N_{\text{recv}}-1}{h}\right) \prod_{a=1}^{h} P_{\text{req}}^a(e(a, g)) \prod_{b=1}^{N_{\text{recv}}-h} P_{\text{req}}^{<i}(\tau(b, g)) \quad (4.25)
\]

We consider now all receiver except for the \(j\)-th receiver. Similar to equation 4.23 we define the probability of \(h\) receivers losing \(N_p + i\) packets and the other \(N_{\text{recv}} - h - 1\) receivers losing less than \(N_p + i\) packets by:

\[
P_{N_{\text{req,max}}}^i(h, j) = \sum_{g=0}^{N_p+i-1} P(g, N_{\text{blk}}, j) \quad (4.26)
\]

We divide now the calculation of \(P_{\text{req}}(i, c, j)\) into two cases and will afterwards merge both into one function using a helper function.

In the first case the number of missing packets are no more than \(N_p + c\) for any receiver among all \(N_{\text{recv}} - 1\) receivers in \(Z^j\), which is equal to the probability of \(N_{\text{req,max}} = i\) and \(N_{\text{req}}(j) = c\) with \(i = c\). Using [4.25] this probability can be expressed as:

\[
P_{\text{req}}(c, c, j) = Pr(N_{\text{req,max}} = c, N_{\text{req}}(j) = c)
\]
\[
= P(N_p + c, N_{\text{blk}}, j) \cdot \sum_{h=0}^{N_{\text{recv}}-1} P_{N_{\text{req,max}}}^c(h, j) \quad (4.26)
\]
In the second case at least one receiver among all $N_{recv} - 1$ receivers in $Z^j$ lost $N_p + i$ packets and all other receivers lost less than $N_p + i$ packets. Using (4.25) this probability can be expressed as:

$$P_{req}(i,c,j) = Pr(N_{req,max} = i, N_{req}(j) = c)$$

$$= P(N_p + c, N_{blk}, j) \cdot \sum_{h=1}^{N_{recv} - 1} P_{N_{req,max}}^i(h, j) \quad (4.27)$$

To integrate (4.26) and (4.27) into one expression for $P_{req}(i,c,j)$, we define the helper function:

$$f_{cmr}(x_1, x_2) = \begin{cases} 0 & \text{if } x_1 = x_2 \\ 1 & \text{if } x_1 > x_2 \end{cases} \quad (4.28)$$

Using the helper function (4.28) we can write $P_{req}(i,c,j)$ as:

$$P_{req}(i,c,j) = P(N_p + N_{blk}, j) \cdot \sum_{h=f_{cmr}(i,c)}^{N_{recv} - 1} P_{N_{req,max}}^{max(i,c)}(h, j) \quad (4.29)$$

If there are $b$ packets lost out of $k + N_p$ packets in the first transmission, let $P_d(i, b)$ denote the probability of $i$ data packets lost out of $b$ packets lost:

$$P_d(i, b) = \frac{\binom{k}{i} \cdot \binom{N_p - i}{b - i}}{\binom{k + N_p}{b}} \quad (4.30)$$

In equation (4.30) it is assumed that all $k + N_p$ packets have the same loss probability.

### 4.4.4 Loss in Retransmissions

In the retransmission phase we will use the convention that state transitions occur at the beginning of a time slot of unit length $t_s$ and then a packet is transmitted. Since the packets are retransmitted during a time slot of unit length $t_s$, the value of $t_{lp}[j]$ can be expressed in a discrete form by the number of packet intervals, in a vector $T_{lp}$, where the individual entries are given by:

$$T_{lp}[j] = \left\lceil \frac{t_{lp}[j]}{t_s} \right\rceil \quad (4.31)$$

Using equation (2.20) the transition probability beginning with state $B$ to stay in state $B$ within $m$ steps is given by the probability $P_{B|B}(m,j)$.

$$P_{B|B}(m,j) = P_B[j] + P_G[j] \cdot (P_G|G[j] + P_B|B[j] - 1)^m \quad (4.32)$$

During the $w$-th retransmission round the number of time slots between the $j$-th receiver detecting lost packets in the first transmission and possibly receiving the $h$-th retransmission of those packets, where $1 \leq h \leq N_{rt}[w]$, is at least $w \cdot T_{lp}[j] + h - 1$. Therefore for the $h$-th retransmission of the redundant packets for one encoding block during the $w$-th retransmission round, the probability of the GE channel being in state $B$ for the $j$-th receiver is $P_{B|B}(w \cdot T_{lp}[j] + h - 1, j)$. Let $P_B(w,j)$ be the loss probability of each retransmission packet for the $j$-th receiver in the $w$-th retransmission round. The probability $P_B(w,j)$ is given by:

$$P_B(w,j) = \prod_{h=1}^{N_{rt}[w]} P_{B|B}(w \cdot T_{lp}[j] + h - 1, j) \quad (4.33)$$
Let $P_{\text{recv}}(r, s, w, j)$ be the probability of $r$ packets received in case of $s$ packets retransmitted in the $w$-th retransmission round for the $j$-th receiver.

\[
P_{\text{recv}}(r, s, w, j) = \binom{s}{r} \cdot (1 - P_B(w, j))^r \cdot (P_B(w, j))^{s-r} \quad (4.34)
\]

Assume $b$ packets, where $k + N_p \geq b > N_p$, are lost out of $k + N_p$ packets in the first transmission for the $j$-th receiver and in the first transmission round there are at most $s$ redundant packets, where $k \geq s \geq b - N_p$, required at the sender. Then let the upper-band of the probability of decoding failure for the $j$-th receiver after experiencing $w$ retransmission rounds be:

\[
P_{f,\text{upper}}(w, b, s, j) \leq \sum_{r_1=0}^{r_{1,\text{max}}} \sum_{r_2=0}^{r_{2,\text{max}}} \cdots \sum_{r_w=0}^{r_{w,\text{max}}} (P'_{\text{recv}}(1) \cdot P'_{\text{recv}}(2) \cdot \ldots \cdot P'_{\text{recv}}(w)) \quad (4.35)
\]

Where:

\[
s_1 = s \quad (4.36)
\]
\[
s_i = b - N_p - \sum_{q=1}^{i-1} r_q \quad (4.37)
\]
\[
r_{1,\text{max}} = b - N_p - 1 \quad (4.38)
\]
\[
r_{i,\text{max}} = b - N_p - \sum_{q=1}^{i-1} r_q - 1 \quad (4.39)
\]

\[
P'_\text{recv}(i) = \begin{cases} 
0 & \text{if } s_i \leq 0 \\
P_{\text{recv}}(r_i, \min(N_{\text{cc}}[i] \cdot s_i, \max(0, RI_{\text{max}}(k) - \sum_{j=1}^{i} N_{\text{cc}}[j] \cdot s_j), i, j)) & \text{otherwise}
\end{cases}
\]

Using equations 4.30, 4.29 and 4.34 the probability distribution function of $I_k(j,w)$ will satisfy:

\[
Pr(I_k(j,w) = i) \leq \sum_{k=b-N_p}^{k=N_p+i} \sum_{i=0}^{N_p} P_{d}(i,b) \cdot P_{req}(s,b-N_p,j) \cdot P_{f,\text{upper}}(w,b,s,j) \quad (4.40)
\]

### 4.4.5 Final PLR-Equation

Finally, we can complete 4.4 using 4.40:

\[
\text{PLR}(j,w) = \frac{E(I_k(j,w))}{k} \leq \frac{\sum_{i=1}^{k} i \cdot Pr(I_k(j,w) = i)}{k} \leq \frac{\sum_{i=1}^{k} i \cdot \sum_{b=N_p+1}^{b=N_p+i} \sum_{i=0}^{N_p} P_{d}(i,b) \cdot P_{req}(s,b-N_p,j) \cdot P_{f,\text{upper}}(w,b,s,j)}{k} \quad (4.41)
\]
Chapter 4. Mathematical Framework

4.5 Calculation of Redundancy Information

We have finished the derivation of the packet loss rate equation and continue now to derive an equation to express the redundancy information when using a certain parameter set. The calculation of redundancy information can be separated in two parts: the redundancy added in the first transmission and the redundancy added in the retransmissions.

\[ RI = 'RI first transmission' + 'RI retransmissions' \]

Redundancy is added in the first transmission if \( N_p \) is greater than zero. The redundancy information given in the first transmission is then simply given by:

\[ 'RI first transmission' = N_p / k \] (4.42)

Now we’ll have a look at the redundancy information added in the retransmission rounds. Further on it is assumed the retransmission packets in the \( w \)-th retransmission round have the same average loss probability to simplify the analysis. The average loss probability for any receiver in the \( w \)-th retransmission round is given by:

\[ P_{B|B}(w) = \sum_{j=1}^{N_{recv}} P_{B|B}(w,j) / N_{recv} \] (4.43)

Let \( P_{los}(i,l,w) \) be the probability of \( i \) retransmission packets lost with a total of \( l \) retransmission packets retransmitted at the sender in the \( w \)-th retransmission round. Then the probability of \( i \) redundant packets being lost out of \( l \) redundant packets is given by:

\[ P_{los}^i(l,w) = \binom{l}{i} (P_{B|B}(w))^i \cdot (1 - P_{B|B}(w))^{l-i} \] (4.44)

Based on (4.44) we define now two basic probabilities in the case of \( l \) redundant packets retransmitted at the sender in the \( w \)-th retransmission round. First the probability of \( i \) redundant packets being lost out of \( l \) redundant packets is given by:

\[ P_{los}(i,l,w) = P_{los}(i,l,w) \] (4.45)

Second the probability of less than \( i \) packets being lost out of \( l \) redundant packets is given by:

\[ P_{los}^<_i(l,w) = \sum_{m=0}^{i-1} P_{los}(m,l,w) \] (4.46)

Then let \( P_{los,max}(i,l,h,g,w) \) be the probability of \( h \) receiver losing \( i \) redundant packets and the other receivers losing less than \( i \) redundant packets among \( g \) receivers after retransmitting \( l \) redundant packets in the \( w \)-th retransmission round. Using (4.45) and (4.46) \( P_{los,max}(i,l,h,g,w) \) is given by:

\[ P_{los,max}(i,l,h,g,w) = \binom{g}{h} \cdot (P_{los}^i(l,w))^h \cdot (P_{los}(l,w))^{g-h} \] (4.47)

Using equation (4.47) the probability of \( h \) receivers requiring the maximum number of redundant packets of \( i \) for the \( w \)-th retransmission round among all of the \( N_{recv} \) receivers is recursively given by:

\[ P_{req,max}(i,h,1) = P_{N_{recv, max}}^i(h) \] (4.48)

\[ P_{req,max}(i,h,w) = \sum_{m=1}^{N_{recv}} \sum_{g=h}^{N_{recv}} P_{req,max}(m,g,w-1) \cdot P_{los,max}(m,g,w-1,h,g,w) \] (4.49)
4.6. Optimization Problem

Let $P'_{\text{req}, \text{max}}(i, w)$ be the probability of the maximum number of redundant packets required of $i$ among all of the $N_{\text{recv}}$ receivers for the $w$-th retransmission round. $P'_{\text{req}, \text{max}}(i, w)$ is recursively defined by:

\[
P'_{\text{req}, \text{max}}(i, 1) = P_{N_{\text{req}, \text{max}}}
\]

\[
P'_{\text{req}, \text{max}}(i, w) = \sum_{m=i}^{k} \sum_{g=1}^{N_{\text{recv}}} P_{\text{req}, \text{max}}(m, g, w - 1) \cdot \left( \sum_{h=1}^{g} P_{\text{los}, \text{max}}((N_{\text{cc}}[w-1] \cdot m + i, N_{\text{cc}}[w-1], h, g, w - 1) \right)
\]

Finally we can construct the redundancy information equation:

\[
\text{RI} = \frac{N_{p}}{k} + \frac{1}{k} \sum_{w=1}^{N_{rr}} \sum_{i=1}^{k} i \cdot N_{\text{rt}}[w] \cdot N_{\text{cc}}[w] \cdot P'_{\text{req}, \text{max}}(i, w)
\]

4.6 Optimization Problem

In this section we will show the optimization problem for finding the optimal parameter set for the generalized architecture when using the mathematical framework presented so far. The equations to calculate the packet loss rate \[4.41\] and redundancy information \[4.52\] can also be seen as functions. The packet loss rate equation \[4.41\] for the $j$-th receiver is a function of $k$, $N_{p}$, $N_{cc}$ and $N_{rt}$, which is denoted by:

\[
f_{j, \text{PLR,UP}}(k, N_{p}, N_{cc}, N_{rt}, N_{\text{recv}})
\]

The equation to calculate the redundancy information \[4.52\] is a function of $k$, $N_{p}$, $N_{cc}$, $N_{rt}$ and $N_{\text{recv}}$, which is denoted by:

\[
f_{RI}(k, N_{p}, N_{cc}, N_{rt}, N_{\text{recv}})
\]

The optimization problem of finding suitable parameters for the architecture, which satisfy a certain packet loss requirement for the first receiver under strict delay constraint with minimal total needed redundancy information, can then be written as:

\[
RI_{AHEC, \text{opt}} = \arg \min f_{RI}(k, N_{p}, N_{cc}, N_{rt}, N_{\text{recv}})
\]

Subject to:

\[
f_{PLR,\text{UP}}(k, N_{p}, N_{cc}, N_{rt}, N_{\text{recv}}) \leq \text{PLR}_{\text{target}}
\]

\[
\frac{\text{RTT}_1}{2} + k \cdot t_s + N_{rt} \cdot t_{ip}[1] \leq D_{\text{target}}
\]
Chapter 5

Methods of Resolution

In this chapter we present and analyze possible methods for finding the solution of the optimization problem within the allowed parameter space.

5.1 Full Search

All the parameters of the architecture, see table 3.2, have discrete values. Given certain network parameters, section 3.4 introduces limits for all parameters of the architecture. Therefore, it is possible to do a full search over all parameter values that come into question to find the optimal parameter set.

5.1.1 Algorithm

Listing 1 shows an algorithm in pseudo code that does full search to find the optimal parameter set. The variable \( \text{opt set} \) will hold the optimal parameter set at the end of the algorithm. Lines 2 to 4 iterate over the constraints for \( N_{rr} \), \( k \) and \( N_p \) as mentioned in section 3.4. Lines 5 to 7 iterate over all possible \( N_{rt} \) and \( N_{cc} \) combinations, respectively. For the special case that \( N_{rr} \) equals to zero, this code snippet would set \( N_{cc} \) and \( N_{rt} \) to [] and iterate once through the code block beginning with line 7. In line 8 the packet loss rate for the current parameter set gets calculated. The function \( \text{calculate_pll} \) calculates the packet loss rate for the first receiver using the current parameter set according to equation 4.41 in section 4.4.5. The next line compares whether the new parameter set ensures the given target loss rate \( PLR_{\text{target}} \). If this is the case, the redundancy information for the new parameter set is calculated in line 10. The function \( \text{calculate_ri} \) computes the redundancy information of the current parameter set according to equation 4.52 in section 4.5. In line 11 there is a check whether the calculated redundancy information for the new parameter set is better than the currently noted optimal parameter set. When the check is true, there’s a new optimum candidate and it’s saved as the optimal parameter set in line 12.

5.1.2 Complexity

In the following we analyze the complexity of the full search space. We analyze the amount of iteration steps to traverse the whole search space. One iteration step is an evaluation of a new potential optimal parameter set that is different from the previous one. The search space without retransmissions depends only on the parameters \( k \) and \( N_p \). The parameter \( k \) traverses the values 1 till \( k(0) \) and the parameter \( N_p \) traverses the values 0 till \( N_{p,max}(0) \). This results for the first transmission in the search space complexity:
Listing 1 Full Search Algorithm

1: opt_set = nil

2: for \( N_{rr} \) in 0..\( N_{rr,max} \) do
3:     for \( k \) in 1..\( k_{max}(N_{rr}) \) do
4:         for \( N_p \) in 0..\( N_{p,max}(k, N_{rr}) \) do
5:             \( M_{max} = M(k,N_p,N_{rr}) \)
6:             combinations = combinations_for(\( M_{max}, N_{rr} \))
7:         for \( N_{cc}, N_{rt} \) in combinations do
8:             plr = calculate_plr()
9:             if plr \( \leq \) PLR\(_{target} \) then
10:                ri = calculate_ri()
11:               if ri \( \leq \) opt_set[ri] then
12:                   opt_set = get_cur_parameter_set() \( \triangleright \) new optimum candidate found
13:           end if
14:       end if
15:     end for
16: end for
17: end for
18: return opt_set

\[
\sum_{k=1}^{k_{max}(0)} (N_{p,max}(k, 0) + 1) \quad (5.1)
\]

Next we calculate the search space complexity for the retransmission rounds. The search space in the retransmission rounds also depends on the parameters \( k \) and \( N_p \). If forward error correction is being used, the parameter \( k \) traverses 2 to \( k(N_{rr}) \), so \( k(N_{rr}) - 1 \) steps, for \( N_{rr} \) retransmission rounds, otherwise \( k \) is being set to 1, so there’s only one step. The parameter \( N_p \) traverses the values 0 till \( N_{p,max}(k, N_{rr}) \) using the current value of \( k \).

The search space complexity in the retransmission rounds depends additionally on all possible \( N_{rt}[q] \) and \( N_{cc}[q] \) combinations for \( q \) in 1 to \( N_{rr} \). As of section 3.4 the individual entries of \( N_{rt} \) and \( N_{cc} \) are limited by \( M(k,N_p,N_{rr}) \). Also the sum over all entries is limited by \( R_{I_{max}}(k) - N_p \).

We try to come up with an equation to calculate the number of combinations of the above mentioned constraints. The number of all possible combinations of \( N_{rt}[q] \) and \( N_{cc}[q] \), respectively, for \( q \) in 1 to \( N_{rr} \) is:

\[
comb(k,N_p,0) = 1
\]

\[
comb(k,N_p,N_{rr}) = \sum_{i_1=1}^{l_1} \sum_{i_2=1}^{l_2} \cdots \sum_{i_{N_{rr}}=1}^{l_{N_{rr}}} 1 \quad (5.2)
\]
5.1. Full Search

Where:

\[ l_1 = \min(M(k, N_p, N_{rr}), R_{l_{max}}(k) - N_p - (N_{rr} - 1)) \] (5.3)

\[ l_j = \min(M(k, N_p, N_{rr}), R_{l_{max}}(k) - N_p - \left(\sum_{y=1}^{j-1} i_y\right) - (N_{rr} - j)) \] (5.4)

There is a special case if \( N_{rr} \) is equal to zero. Then, \( N_{cc} \) and \( N_{rt} \) are set to [], so there is exactly one combination.

We got now a measure to count the number of iterations for \( N_{cc} \) and \( N_{rt} \), respectively, by means of the function \( \text{comb} \). So, if \( k \) is equal to one, the search space complexity for \( N_{rr} \) retransmission rounds is given by:

\[ \sum_{N_p=0}^{N_{p,\text{max}}(1,N_{rr})} \text{comb}(1, N_p, N_{rr}) \] (5.5)

If \( k \) is greater than one, the search space complexity for \( N_{rr} \) retransmission rounds is given by:

\[ \sum_{N_p=0}^{N_{p,\text{max}}(k,N_{rr})} \sum_{k=2}^{k_{\text{max}}(N_{rr})} \text{comb}(k, N_p, N_{rr}) \] (5.6)

The equations 5.5 and 5.6 can be combined to calculate the search space complexity for any \( k \) for \( N_{rr} \) retransmission rounds by:

\[ \sum_{N_p=0}^{N_{p,\text{max}}(k,N_{rr})} \sum_{k=1}^{k_{\text{max}}(N_{rr})} \text{comb}(k, N_p, N_{rr}) \] (5.7)

Using equations 5.1 and 5.7 and the upper limit of \( N_{rr,\text{max}} \) retransmission rounds we get the overall search space complexity:

\[ \begin{bmatrix} \sum_{k=1}^{k_{\text{max}}(0)} (N_{p,\text{max}}(k, 0) + 1) \\ \sum_{N_{rr}=0}^{N_{rr,\text{max}}} \sum_{k=1}^{k_{\text{max}}(N_{rr})} \sum_{N_p=0}^{N_{p,\text{max}}(k,N_{rr})} \text{comb}(k, N_p, N_{rr}) \end{bmatrix} \]

\[ = \sum_{N_{rr}=0}^{N_{rr,\text{max}}} \sum_{k=1}^{k_{\text{max}}(N_{rr})} \sum_{N_p=0}^{N_{p,\text{max}}(k,N_{rr})} \text{comb}(k, N_p, N_{rr}) \] (5.8)
5.2 Greedy

Searching the whole search space is not quite efficient, but the most obvious approach. We come now to a special class of algorithms, the greedy algorithms[7]. A greedy algorithm is characterized by stepwise choosing the next step which looks the best at the moment. Often greedy algorithms are fast but do not solve the problem optimally. They make a locally optimal choice. It doesn’t need, however, to be the global optimum.

Guoping Tan and Thorsten Herfet propose a more efficient greedy algorithm [11] than the previously mentioned full search to find the optimal solution to the optimization problem. The first optimization is to allow only the maximal value \( k_{\text{max}}(N_{rr}) \) for \( k \) at any retransmission stage besides \( k = 1 \). The second optimization is to allow only certain \( N_{cc} \) and \( N_{rt} \) combinations. This approach is a greedy algorithm because of the latter optimization, which we will discuss in this section in more detail. Moreover, we have to apply and adjust their greedy approach to fit the new constraints mentioned in section 3.4.

5.2.1 First Transmission

We begin with the first transmission without retransmissions and try to reach the target loss rate with minimal redundancy information needed. In the first transmission only the parameters \( k \) and \( N_p \) can be adjusted. The redundancy added in the first transmission is equal to \( N_p/k \) as of section 4.5.

No redundancy is needed if \( N_p \) is equal to zero and the target loss rate gets achieved. When the target loss rate is not reached, we need to introduce redundant packets by increasing \( N_p \). Ideally, we increase \( N_p \) from 0 till \( N_{p,\text{max}}(k,0) \), where \( k \) is set to \( k_{\text{max}}(0) \) to minimize the fraction \( N_p/k \), and check whether the target loss rate has been reached in each step. If the target loss rate gets reached by any value of \( N_p \) and \( k \), we found a new optimum candidate and stop the search for the first transmission. Otherwise, we will try till \( N_p \) equals to \( N_{p,\text{max}}(k,0) \) and stop looking for an optimal parameter set in the first transmission.

The search for optimal parameter sets continues using retransmission rounds.

5.2.2 Retransmission

For the retransmission rounds we still iterate over \( N_{rr} \), \( k \), \( N_p \) and at the end over \( N_{rt} \) and \( N_{cc} \), respectively, but with two modifications. The first modification is to skip all values of \( k \) that are between 1 and \( k_{\text{max}}(N_{rr}) \).

The other modification is related to the combinations of \( N_{rt} \) and \( N_{cc} \). Once the parameters \( N_{rr} \), \( k \) and \( N_p \) have been chosen, we initialize \( N_{rt} \) and \( N_{cc} \) by setting each entry of the vectors to one and apply a greedy search. The greedy search always increases the minimum redundancy information at each stage. This is done by increasing the last entry of the \( N_{rt} \) or \( N_{cc} \) vector by one, so the new vector \( N_{rt} \) and \( N_{cc} \), respectively, is given by:

\[
N_{rt} = N_{rt} + \underbrace{(0, \ldots, 0, 1)}_{N_{rr}-1}, \text{ if } k = 1
\]

\[
N_{cc} = N_{cc} + \underbrace{(0, \ldots, 0, 1)}_{N_{rr}-1}, \text{ if } k > 1
\]

At each stage we check whether the target PLR requirement is satisfied. If it is the case, we might have a new optimum that is better than the last known one. Otherwise, we increase the last entry again as long as \( M(k,N_p,N_{rr}) \) allows it.

5.2.3 Algorithm

Listing 2 page 45 and listing 3 page 46 show an algorithm in pseudo code that does greedy search to find the optimal parameter set. Listing 2 shows the algorithm for the first
transmission as described in the previous section. Lines 4 to 6 initialize the parameters for the first transmission. Lines 7 to 10 do the search for an optimal parameter set till either one is found or the \( N_{p,max}(k,0) \) budget has been exhausted. The last lines, 12 to 15, save the new optimal parameter set if one has been found. The value of \( N_p \) needs to be decreased by one in line 14, since the previously mentioned loop beginning with line 7 has increased \( N_p \) by one when leaving the loop.

Listing 3 goes on with finding the optimal parameter set when using retransmission rounds to comply with the target packet loss rate. Line 3 contains the first modification compared to the full search algorithm in listing 1. The parameter \( k \) traverses only the values 1 and \( k_{max}(N_{rr}) \). Then, in line 5 and 6, the parameters \( N_{cc} \) and \( N_{rt} \) get initialized to contain the value one for each round. Line 7 calculates the maximal value the last entry of \( N_{rt} \) and \( N_{cc} \), respectively, may reach. The packet loss rate gets calculated for those vectors in the next line. If the packet loss rate isn’t fulfilled, the while loop from line 9 till 17 will perform a greedy search by incrementing the last entries of the vector \( N_{cc} \) or \( N_{rt} \) and checking whether the packet loss rate can be fulfilled. The function \( fplusone \) does increment the last entries of the vectors, depending on whether \( k \) equals to 1 or \( k_{max}(N_{rr}) \). Lines 18 till 23 set the optimal parameter set if a new optimum candidate was found during the greedy search.

### Listing 2 Greedy Algorithm, First Transmission

```plaintext
1: opt_set = nil
2: \( \triangleright \) without retransmissions
3: \( N_{rr} = 0 \)
4: \( N_{rt} = N_{cc} = [] \)
5: \( N_p = 0 \)
6: \( k = k_{max}(N_{rr}) \)

7: repeat
8:     plr = calculate_plr()
9:     \( N_p = N_p + 1 \)
10: until plr \( \leq \) PLR_{target} or \( N_p > N_{p,max}(k,0) \)

11: \( \triangleright \) new optimum candidate found
12: if plr \( \leq \) PLR_{target} then
13:     opt_set = get_parameter_set()
14:     opt_set[\( N_p \)] = \( N_p - 1 \)
15: end if
```

### 5.2.4 Complexity

In the following we analyze the complexity of the greedy search. We analyze the amount of iteration steps the greedy search will take in the worst case. The worst case can serve as a comparison to the iteration steps taken by the full search.

The algorithm will increase \( N_p \) at worst \( N_{p,max}(k_{max}(0),0) + 1 \) times in the first transmission. In the retransmission rounds the number of iterations depends on the parameters \( N_{rr} \), \( N_p \) and the value of \( M(k,N_p,N_{rr}) \). The parameter \( N_{rr} \) iterates from 1 to \( N_{rr,max} \). Then the parameter \( k \) is chosen to be one and \( k_{max}(N_{rr}) \). Depending on \( k \) the parameter \( N_p \) traverses \( N_p \) from 0 to \( N_{p,max}(1,N_{rr}) \) or to \( N_{p,max}(k_{max}(N_{rr}),N_{rr}) \). The final step is the greedy search, which has in the worst case, depending on \( k \), \( M(1,N_p,N_{rr}) \) or \( M(k_{max}(N_{rr}),N_p,N_{rr}) \) steps.
Listing 3 Greedy Algorithm, Retransmissions

1: ⊳ with retransmissions
2: for \( N_{rr} = 1..N_{rr,max} \) do
3:   for \( k \) in \([1,k_{max}(N_{rr})]\) do
4:     for \( N_p \) in \(0..N_{p,max}(k,N_{rr})\) do
5:       \( N_{cc} = \text{Array.new}(\text{size} \to N_{rr}, \text{value} \to 1) \)
6:       \( N_{rt} = \text{Array.new}(\text{size} \to N_{rr}, \text{value} \to 1) \)
7:       \( m = M(k,N_p,N_{rr}) \)
8:       plr = calculate_plr()
9:     while plr > PLR_{target} and \( m > 1 \) do
10:       if \( k == 1 \) then
11:         f_{plus_one}(N_{rt}, N_{rr})
12:       else
13:         f_{plus_one}(N_{cc}, N_{rr})
14:       end if
15:       plr = calculate_plr()
16:       \( m = m - 1 \)
17:     end while
18:   if plr \leq PLR_{target} then
19:     ri = calculate_ri()
20:     if ri < opt_set[ri] then
21:       opt_ri = get_parameter_set() ⊳ new optimum found
22:     end if
23:   end if
24: end for
25: end for
26: return opt_set

The worst case complexity is then given by:

\[
\begin{align*}
& (N_{p,max}(k_{max}(0),0) + 1) + \\
& \sum_{N_{rr}=1}^{N_{rr,max}} \sum_{N_p=0}^{N_{p,max}(1,N_{rr})} M(1,N_p,N_{rr}) + \\
& \sum_{N_p=0}^{N_{p,max}(k_{max}(N_{rr}),N_{rr})} M(k_{max}(N_{rr}),N_p,N_{rr}) \\
& \text{first transmission} \quad k = 1 \\
& \sum_{N_p=0}^{N_{p,max}(k_{max}(N_{rr}),N_{rr})} M(k_{max}(N_{rr}),N_p,N_{rr}) \\
& \text{first transmission} \quad k = k_{max}(N_{rr})
\end{align*}
\]
5.3 Improved Greedy

In this section we try to come up with an improved greedy search that acts more aggressive in finding the optimal parameter set.

5.3.1 First Transmission

After the search for the optimal parameter set in the first transmission, as of section 5.2.1, the search will continue with the retransmissions independent of the result in the first transmission phase. But there is one case when the search using retransmission rounds doesn’t need to be done. When the search in the first transmission finds an optimal set with $N_p$ equal to zero, the redundancy information is equal to $N_p/k = 0$. There is no redundancy needed. This is the case when the network parameters are good enough to reach the target loss rate without redundant packets. If redundant packets are not needed, there is a global optimum found. The search for optimal parameters using retransmission rounds is obsolete.

5.3.2 Retransmission

Once the parameters $N_{rr}$, $k$ and $N_p$ for the retransmission rounds have been chosen, the search described in section 5.2.2 will do a greedy search on the parameter $N_{cc}$ or $N_{rt}$. This search will be executed independent of the currently known optimal parameter set candidate. When a current optimal parameter set candidate exists, the redundancy information of that candidate is the current standard value that new candidates need to fall below. So, if at the beginning of the greedy search the first candidate already exceeds the current standard value of redundancy information, the currently running greedy search doesn’t have to be continued, since the next parameter sets in the greedy search increase the redundancy information. Further on, the greedy search in section 5.2.2 searches for the optimal parameter set beginning with only one retransmission round as far as $N_{rr,max}$ retransmission rounds. Sending more redundant packets in later retransmission rounds leads to a lower redundancy information than sending redundant packets at the beginning. The first retransmission rounds might already be able to correct packet losses. Staving off redundant packets to the last retransmission round, by beginning the $N_{rr}$ iteration with $N_{rr,max}$ and going down to one retransmission round, is another enhancement.

5.3.3 Algorithm

Listing 4 shows an algorithm in pseudo code that does an improved greedy search, compared to the greedy search in the previous section 5.2, to find the optimal parameter set. Listing 5 shows the algorithm for the first transmission as described in the previous section 5.3.1. The main difference between the improved algorithm and the algorithm in listing 2 are the lines 15 to 17. They check for a global optimum and stop the search, if there is a global optimum found.

Listing 6 shows the algorithm for the retransmission rounds. Line 9 to 12 test whether the RI value at the beginning of the greedy search already exceeds the current RI standard value. If the value exceeds the RI standard value, the currently running greedy search is interrupted in line 11 otherwise the search continues in line 13. The search for the optimal parameter set, beginning with the maximal number of available retransmission rounds, is set up in line 2 and 3. Line 29 reduces the retransmission rounds by one.
Listing 4 Improved Greedy Algorithm, First Transmission

1: opt_set = nil

2: \[\text{// without retransmissions}\]
3: \[N_{rr} = 0\]
4: \[N_{rt} = N_{cc} = []\]
5: \[N_p = 0\]
6: \[k = k_{max}(N_{rr})\]

7: repeat
8: \[p{}lr = \text{calculate}_\text{plr}()\]
9: \[N_p = N_p + 1\]
10: until \[p{}lr \leq PLR_{target} \text{ or } N_p > N_{p,max}(k, 0)\]

11: \[\text{// new optimum candidate found}\]
12: if \[p{}lr \leq PLR_{target}\] then
13: \[\text{opt}_\text{set} = \text{get}_\text{parameter}_\text{set}()\]
14: \[\text{opt}_\text{set}[N_p] = N_p - 1\]
15: if \[N_p == 0\] then
16: \[\text{return result}\]
17: end if
18: end if
5.3. Improved Greedy

Listing 5 Improved Greedy Algorithm, Retransmissions

1: $N_{tr} = N_{tr,max}$
2: while $N_{tr} > 0$ do
3:     for $k$ in $[1, k_{max}(N_{tr})]$ do
4:         for $N_p$ in $0..N_{p,max}(k, N_{tr})$ do
5:             $N_{cc} = \text{Array.new}(\text{size} \rightarrow N_{tr}, \text{value} \rightarrow 1)$
6:             $N_{rt} = \text{Array.new}(\text{size} \rightarrow N_{tr}, \text{value} \rightarrow 1)$
7:             $m = M(k, N_p, N_{tr})$
8:             $ri = \text{calculate}_ri()$  $\triangleright$ RI constraint
9:             if $ri > \text{opt_set}[ri]$ then
10:                break
11:         end if
12:     end for
13:     end for
14:     $N_{tr} = N_{tr} - 1$
15: end while

16: return $\text{opt_set}$
Chapter 5. Methods of Resolution
Chapter 6
Analysis

In this chapter we analyze practically some aspects of the mathematical framework and of the methods of resolution presented.

6.1 Performance of Algorithms

In the following we compare the performance of the algorithms presented in chapter 5. We select a specific scenario and compare the number of iteration steps, see section 5.1.2 on page 41, for each algorithm. For the full search and the greedy algorithm we got an equation as of sections 5.1.2 and 5.2.4 to compute the worst case number of iteration steps. We also demonstrate the actual iteration steps taken by the greedy search and improved greedy search using the implementation developed during the writing of this thesis, see the next chapter.

6.1.1 General

Figures 6.1, 6.2 and 6.3 show the performance of the different algorithms for the example scenario listed in table A.1 on page 69 in the appendix. The value of $D_{target}$, starting with 50 millisecond, is being increased stepwise by 10 millisecond.

Figure 6.1 shows the performance of the full search versus the greedy search in the worst case. The iteration steps taken by a full search get very early very high in opposite to the worst case of a greedy search. With the increasing delay target $D_{target}$ the end-to-end delay equation 3.6 in section 3.4 allows in every step the maximal value of $k$ to increase and there is more time for redundant packets. The greedy search, on the other hand, will still iterate over two values of $k$, which are 1 and $k_{max}(N_{rr})$. Only the value of $M(k, N_{p}, N_{rr})$ will increase slowly, representing the number of steps taken to find the right value for $N_{cc}$ and $N_{rt}$. The greedy search, on the other hand, suffers soon under the possible combinations for $N_{cc}$ and $N_{rt}$, respectively.

Figure 6.2 shows the actual number of iteration steps needed by the greedy search to find an optimal parameter set versus the greedy search in the worst case. In the beginning, when there is no room for many redundant packets and retransmission rounds, the equation to calculate the worst case complexity of the greedy search 5.2.4 fits nearly the actual number of iterations used by the greedy search. With increasing delay target, the equation for the worst case doesn’t seem to be very accurate. The actual number of iteration steps is significantly lower than the worst case in many simulation runs done during the thesis. A more accurate equation, e.g. using amortized analysis [7], would be needed.

Figure 6.3 shows the actual number of iteration steps taken by the greedy search versus the improved greedy to find the optimal parameter set. In the beginning the results are the same.
Figure 6.1: The number of iteration steps needed by the full search to get the optimal parameter set versus the worst case greedy search for an example scenario.

Figure 6.2: The number of iteration steps needed by the greedy search in the worst case versus the actual number of iteration steps measured for an example scenario.
6.1. Performance of Algorithms

**Figure 6.3:** The actual number of iteration steps needed by the greedy search versus the number of iteration steps needed for the improved greedy search for an example scenario.

**Figure 6.4:** The iteration steps (theoretical and actual) needed by all algorithms to find the optimal parameter set for an example scenario. The capability of the channel is good enough to abandon redundant packets such that the improved greedy search returns in the first step already the global optimum.
This is the case because at low $D_{\text{target}}$ values no retransmission is needed. Both algorithms perform the same since there is no difference in the algorithm in the first transmission, except for the check for a global optimum in the case of the improved greedy. As soon as retransmissions are possible with higher delay targets, the improved greedy performs better.

### 6.1.2 Global Optimum

For the sake of completeness we look also at an example scenario where the improved greedy algorithm finds a global optimum after the first step and doesn’t continue, as previously discussed in section 5.3.1. The network conditions are summarized in table A.3 on page 70. Figure 6.4 on page 53 shows the iteration steps taken by all algorithms. The axes showing the iteration steps is logarithmically scaled, so the one and only iteration step taken by the improved greedy search has the value 0 in the figure.

### 6.2 Optimization Results

In the following we use the improved greedy algorithm to discuss the performance of the generalized architecture in a few scenarios.

#### 6.2.1 Adaptivity

We consider the example specified by the parameters in table A.4 on page 70 in the appendix. Further on we look at different application delay targets and packet loss rates for one to five receivers. The delay target, beginning with 20 milliseconds, gets increased by approximately 5 milliseconds till 100 milliseconds are reached. For the packet loss we consider packet loss rates from two till 10 percentage in steps of one percentage. For each delay target and packet loss alternative we determine the optimal parameter set according to the improved greedy search.

Figure 6.5, page 55, shows the result of the improved greedy search for the previously specified search space by grouping each parameter set in a certain group. For a delay target of 10 milliseconds there are no results (see the black dots). There exists no scheme that satisfies the given scenario.

At the next delay target step of 16 milliseconds there exists a first scheme to deliver the data according to the specified target packet loss rate. For the first two packet loss rates, two and three percentage, the proposed optimal parameter sets work without retransmission rounds (see the red dots). Retransmission rounds would exceed the delay target.

With increasing delay targets till around 30 milliseconds the generalized architecture is capable of correcting increasing packet loss rates.

Around 36 milliseconds there is a switch in the choice of how to correct errors. The delay target is now big enough to allow retransmission rounds (see the cyan and blue dots). This example shows exemplarily how the generalized architecture is capable of adapting to varying network conditions.

Figure 6.6, page 56, denotes the redundancy information used by the optimal parameter sets for the given example scenario with one receiver. The blue dots mark the corresponding redundancy information value for a given packet loss rate and target delay. When there are no retransmission rounds possible to correct the given packet loss, the first transmission must include enough redundant packets. This results in a high redundancy information value in the region of low target delays. With increasing target delays and usage of retransmission rounds the generalized architecture will resign from sending many redundant packets with the first transmission and reduce therefore the RI value. Also, with increasing packet loss rates the optimal parameter sets will need to send more redundant packets overall resulting in an increasing RI value along the $P_e$ axis.
6.2. Optimization Results

![Diagram](image)

**Figure 6.5:** The generalized architecture is capable of adapting to various network scenarios. Depending on the delay target, the packet loss rate and the number of receivers the optimal parameter sets are calculated. The example scenario shows that no retransmission rounds are used when the delay target does not permit retransmission rounds. Retransmission rounds are used to minimize redundancy when the delay target increases.

6.2.2 Physical Limit

The research of the generalized architecture, see the papers [11], [8], [10] and [9] among others, has not considered so far a physical limit of the link the data is transferred over to the receivers. The data rate was theoretically infinite.

Section 3.4 introduced new constraints for the number of redundant packets available to the generalized architecture beginning with equation 3.9. It is the first attempt to give the generalized architecture a sense of the free space available to redundant packets. Previous attempts have e.g. not known theoretical limits for the parameter $N_{cc}$. This is reflected in the discussion of the greedy search, where the number of redundant packets for the last retransmission round is theoretically unlimited and practically only limited by a reference redundancy information standard value. On the other side this thesis restricts the redundant packets available to the parameter $N_{cc}$. There is now a theoretical limit for the parameter $N_{cc}$.

The data rate provided by the link plays now an additional role. Figure 6.7 on page 57 shows the result of the improved greedy search following the same conditions as the example scenario in the previous section 6.2.1 except for the data rate. The data rate has been lowered to 8 MBit/s. The generalized architecture has now less bandwidth for redundant packets. Thus, there are even more cases where the generalized architecture doesn’t find optimal parameter sets for the given scenario (see the black dots). The generalized architecture finds optimal parameter sets not until around 26 milliseconds.
6.3 Tables

The presented methods of resolution are by far not practicable in real-world applications to calculate the optimal parameter set. The execution time varies from under one second till theoretically infinity depending on the parameters. Real-world applications on the other hand require to find solutions in the range of a few hundred milliseconds to adjust to the varying network conditions.

A solution for real-world applications would e.g. consist of saving the computed solutions for different network conditions in tables in the memory. It is then a matter of space how many solutions for different scenarios can be saved and how precise those solutions should be. How to efficiently encode those information is another issue.

In the following we present some results that should be considered in the further research of this topic.

Delay Equation

We consider now the example scenario given by the parameters in table A.5 on page 71. The parameter $D_{\text{target}}$ is increased every step by 0.1 millisecond. Table 6.1 shows an extract of the optimal parameter sets and their needed redundancy information when iterating over varying delay targets using the improved greedy search. The last column contains row numbers to reference the individual rows in the next sections.

After row 6 all calculated optimal parameter sets are the same till row 10. Row 11 contains again a different optimal parameter set. There are successive blocks of redundant optimal parameter sets throughout the whole result set. That redundant data takes unnecessary space and eats computation time.

It seems like the value by which $D_{\text{target}}$ is increased in every step should be larger. The
difference of the values for $D_{\text{target}}$ between row 7 and 10 is 2.5 millisecond. This time difference is nearly equal to the value of $t_s$.

A change in the optimal set might also occur earlier than changing the value of $D_{\text{target}}$ by $t_s$ seconds. This is the case when the delay target allows an additional retransmission round to be taken. This effect is also shown in table 6.1, page 58. From row 14 to 15 there is a third retransmission round added. The redundant parameter sets beginning with row 11 until before the additional retransmission round, which is row 14, are the same for the $D_{\text{target}}$ range of 1.5 milliseconds, which is smaller than $t_s$.

The effects in table 6.1 can be explained by the delay equation 3.6 and the resulting limits $N_{rr,\text{max}}$ and $k_{\text{max}}(N_{rr})$, see equations 3.7 and 3.8 in section 3.4, for the parameters $k$ and $N_{rr}$. We assume now $N_{rr,\text{max}}$ won’t change and we are at the beginning of a newly added retransmission round and have calculated the current optimal parameter set. The optimal parameter set will not change as long as there is not enough time for at least a new redundant packet. New redundant packets are only possible when $k_{\text{max}}(N_{rr})$ increases by one. All arguments in the equation $k_{\text{max}}(N_{rr})$ do not change except for $D_{\text{target}}$. As soon as $D_{\text{target}}$ gets increased by at least the value of the denominator $t_s$, the value of $k_{\text{max}}(N_{rr})$ will increase by one. There is enough time for an additional redundant packet. Every time the delay target gets increased by $t_s$ seconds, a new optimal parameter set must be found. So once we have a new optimal parameter set, this optimal parameter set will last for the range of $t_s$ seconds of $D_{\text{target}}$.

Next we assume we are at the beginning of a newly added retransmission round and have calculated the current optimal parameter set. When we determine new optimal parameter sets by increasing the delay target by $t_s$ seconds, we would miss the turning point when there is again a new transmission round added. Additional retransmission rounds come into question when $N_{rr,\text{max}}$ will increase by one. This is the case as soon as $D_{\text{target}}$ gets increased by at least the value of the denominator $t_{\text{lp}}[1]$. The constraints for $N_{rr,\text{max}}$ and $k_{\text{max}}(N_{rr})$ can be used to determine the accuracy with which to iterate over delay target values. This will save the number of iteration steps and remove redundancy, since an optimal parameter set is the optimal set for a certain range.
Table 6.1: Extract of optimal parameters for an example scenario when varying $D_{\text{target}}$. The table shows redundant parameter set entries throughout the whole table.

$D_{\text{target}}[s]$ | $k$ | $N_p$ | $N_{rr}$ | $N_{cc}$ | $N_{rt}$
---|---|---|---|---|---
0.0827 | 15 | 3 | 1 | [5] | 1 | 1
0.0828 | 15 | 3 | 1 | [5] | 1 | 2
0.0829 | 4 | 0 | 2 | [1,5] | 1,1 | 3
0.0830 | 4 | 0 | 2 | [1,5] | 1,1 | 4
0.1163 | 16 | 1 | 2 | [1,5] | 1,1 | 5
0.1164 | 16 | 1 | 2 | [1,5] | 1,1 | 6
0.1165 | 17 | 1 | 2 | [1,5] | 1,1 | 7
0.1166 | 17 | 1 | 2 | [1,5] | 1,1 | 8
0.1189 | 17 | 1 | 2 | [1,5] | 1,1 | 9
0.1190 | 17 | 1 | 2 | [1,5] | 1,1 | 10
0.1191 | 18 | 1 | 2 | [1,5] | 1,1 | 11
0.1192 | 18 | 1 | 2 | [1,5] | 1,1 | 12
0.1205 | 18 | 1 | 2 | [1,5] | 1,1 | 13
0.1206 | 18 | 1 | 2 | [1,5] | 1,1 | 14
0.1207 | 7 | 0 | 3 | [1,1,4] | 1,1,1 | 15
0.1208 | 7 | 0 | 3 | [1,1,4] | 1,1,1 | 16
0.1230 | 7 | 0 | 3 | [1,1,4] | 1,1,1 | 17
0.1231 | 7 | 0 | 3 | [1,1,4] | 1,1,1 | 18
0.1232 | 8 | 0 | 3 | [1,1,4] | 1,1,1 | 19
0.1232 | 8 | 0 | 3 | [1,1,4] | 1,1,1 | 20

of $D_{\text{target}}$. Similar conclusions can be determined, when iterating over one of the other parameters, that affect the value of $N_{rr,max}$ or $k_{\text{max}}(N_{rr})$. 
Chapter 7

Software Implementation

We have looked so far at the generalized architecture, a mathematical framework to express the generalized architecture and its optimization problem. The previous chapter showed algorithms to find the optimal parameter set that satisfies the optimization problem. During the writing of this thesis it was important to research the parameter space, the behavior of the different equations in the mathematical framework and the proposed algorithms. Therefore an implementation of the mathematical framework as of chapter has been developed.

This chapter might help as a reference in the development of other implementations and to understand the pitfalls of the theoretical foundation when it comes to real-world implementations.

7.1 Overview

The implementation used during this thesis is written in the programming language Ruby. Ruby is a general purpose object-oriented programming language and is dynamic and reflective. It has been chosen at the beginning of this thesis as it allows very fast prototyping and is the primary programming language of the author. The prototype has allowed the author to test theories, explore design alternatives and confirm performance.

7.2 Features

In the following subsections some useful information regarding the implementation of the mathematical framework and the methods of resolution are discussed.

7.2.1 Testing

The technique test-driven development has been used during the development. Test-driven development (TTD) is a software technique to develop software by creating software tests before the actual implementation. It allows to get rapid feedback changes during development.

Tests were not only useful to spot simple software bugs, like e.g. failed execution, but also to verify the results during the switch to C extensions, see section 7.2.5. Furthermore it should be thought about saving sample values for all functions returning numerical values to find significant variations during time. It will help saving time to figure out which functions introduced new deviations in results.
7.2.2 Modularity

To analyze and explore the equations in the mathematical framework, see chapter 4, it was also important to keep equations in their own methods in the implementation. Table 7.1 shows the important equations of the mathematical framework in the first column and their location in the corresponding implementation in the second column. Each equation is handled in a different method. The implementation is very modular and allows therefore to substitute equations through different implementations very easily.

<table>
<thead>
<tr>
<th>Equation</th>
<th>Method</th>
</tr>
</thead>
<tbody>
<tr>
<td>Loss in first transmission</td>
<td></td>
</tr>
<tr>
<td>4.21 ( P_{req}(j) )</td>
<td>( p_{req,j} )</td>
</tr>
<tr>
<td>4.22 ( P_{req}(j) )</td>
<td>( p_{req,l,j} )</td>
</tr>
<tr>
<td>4.23 ( P_{Nreq,max}(h) )</td>
<td>( p_{Nreq,max,i\of h} )</td>
</tr>
<tr>
<td>4.24 ( P_{Nreq,max} )</td>
<td>( p_{Nreq,max,i} )</td>
</tr>
<tr>
<td>4.25 ( P_{Nreq,max}(h,j) )</td>
<td>( p_{nreq\of h\and j} )</td>
</tr>
<tr>
<td>4.30 ( P_{d}(i,b) )</td>
<td>( p_{d} )</td>
</tr>
<tr>
<td>4.29 ( P_{req}(i,c,j) )</td>
<td>( p_{req} )</td>
</tr>
<tr>
<td>Loss in retransmissions</td>
<td></td>
</tr>
<tr>
<td>4.32 ( P_{B</td>
<td>B}(m,j) )</td>
</tr>
<tr>
<td>4.33 ( P_{B}(w,j) )</td>
<td>( p_{b,w,j} )</td>
</tr>
<tr>
<td>4.34 ( P_{recv}(r,s,w,j) )</td>
<td>( p_{recv} )</td>
</tr>
<tr>
<td>4.35 ( P_{f\upper}(w,b,s,j) )</td>
<td>( p_{f\upper} )</td>
</tr>
<tr>
<td>4.31 ( PLR(j,w) )</td>
<td>( plr_AHEC )</td>
</tr>
<tr>
<td>Redundancy information</td>
<td></td>
</tr>
<tr>
<td>4.43 ( P_{B</td>
<td>B}(w) )</td>
</tr>
<tr>
<td>4.41 ( P_{loss}(i,l,w) )</td>
<td>( p_{loss} )</td>
</tr>
<tr>
<td>4.42 ( P_{los}(i,l,w) )</td>
<td>( p_{los_{eq,i}} )</td>
</tr>
<tr>
<td>4.46 ( P_{los}(l,w) )</td>
<td>( p_{los_{lt,i}} )</td>
</tr>
<tr>
<td>4.47 ( P_{los,max}(i,l,h,g,w) )</td>
<td>( p_{los_{max}} )</td>
</tr>
<tr>
<td>4.52 ( RI )</td>
<td>( ri )</td>
</tr>
</tbody>
</table>

Table 7.1: Equations and their corresponding methods in the implementation. Each equation is computed in a different method.

7.2.3 Recursion

Recursion is a technique to define a function that calls itself within its own definition. The function \( P(m,d,j) \) in section 4.4.1 has a recursion. The function \( P(m,d,j) \) uses a double recursion to calculate the loss of \( m \) packets in a sequence of \( d \) packets sent for the \( j \)-th receiver in a GE channel. Implementing the function \( P(m,d,j) \) as it is given in this thesis is far too expensive and was a bottleneck of the implementation in an early stage. Every recursive call creates information that needs to be put on a stack in the application process, e.g. information where to continue execution after the function call. Depending on the depth of the recursion, the invocation of recursive functions should be avoided where possible or at least minimized. Deep recursion may cause trouble like eating the stack space available to the process and produce large execution times to release the function call information on the stack. Recursion can often be avoided by iteration. Iteration is a technique to solve a problem stepwise, often using loop constructs. In the current implementation iteration could be applied to the equation \( P_{req,max}(i,h,w) \) for instance.
7.2.4 Caching

A cache stores data computed earlier. It is used to prevent expensive fetching or computation as long as reading the cache is less costly. Caching was used in the implementation to prevent reoccurring, expensive computation on one hand and to speedup computation. The data generated by the Gilbert Elliot model, for instance, is used very frequently through the equation \( P(m,d,j) \). It is not effective to compute those values every time \( P(m,d,j) \) gets used but to compute it once during initialization of the mathematical framework in the implementation and store the data for further usage. The stored values can be dropped again when \( k \) or \( N_p \) or the parameters of the Gilbert Elliot model change. That way it can be quite effective to iterate e.g. over different delay targets without recomputing the Gilbert Elliot model data.

Caching was also used to save computation steps during the calculation of \( P(m,d,j) \). The mathematical framework uses the function \( P(m,d,j) \) to compute for every receiver \( j \) the values for \( P(0,N_{blk},j) \) till \( P(N_{blk},N_{blk},j) \). Intermediate results computed in the \( m \)-th step of \( P(m,d,j) \) can be reused in the \( m+1 \)-th step.

7.2.5 C Extension

After the first prototype has been finished there were some performance issues. The methods to calculate packet loss rate and redundancy information for a given parameter set call at the end of the function call stack very often the equation \( P(m,d,j) \). It was essential to get a faster implementation to calculate the probability of losing certain packets in a sequence of packets. Another issue was a missing reference implementation for the factorial of a non-negative integer. Ruby and most other dynamic programming languages allow to extend their libraries through extensions written in other languages. Most commonly the programming language C is being used to write those extensions. Using so called C extensions is more efficient since the code is no longer interpreted but compiled to object code.

The author of this thesis embraced C extensions to replace critical paths in the implementation. Once a working prototype existed, it was also obvious to implement new equations directly as C extensions. Figure 7.1, page 62, shows the parts of the implementation written in the programming language Ruby and the parts moved to the C extension.

Factorial

As previously mentioned an efficient implementation for the factorial of a non-negative integer was missing. The math library that comes with Ruby doesn’t offer an implementation. W.H. Press and others discuss in their book\[20\] the implementation of the factorial of a non-negative integer, among others, in the programming language C. The implementation of the factorial of a non-negative integer is based on the implementation in the previously mentioned book.

7.3 Issues

The current implementation has shown some flaws at the end of the writing of this thesis, which will be discussed in the following subsections.

7.3.1 Interpreted Language

The usage of an interpreted language such as Ruby during the development and prototyping has some limitation. The main issue is performance. Once a working prototype has been developed, the performance of large searches like the full search was very time consuming.
At least the implementation in Ruby combined with a C extension has shown significant improvements in opposite to an internal implementation in MATLAB [3] at the Telecommunications Lab. MATLAB is a numerical computing environment and programming language. Another problem is that the current implementation cannot be used efficiently in other languages. There needs to be an extra process running Ruby and the Ruby implementation. The programming language C fits here better. Programmers can create interfaces such that the mathematical framework routines and the proposed algorithms can be used from high-level languages like Ruby, MATLAB, Java [4], Python [5] and others.

At least the parts of the implementation that are implemented in the C extension, see section 7.2.5, can be reused in future developments of an C implementation.

On the other side, even if a faster implementation would exist in e.g. C, it would still not be efficient in real-world applications to compute the solution for a given scenario immediately. Real-world applications would anyway use tables to store solutions for specific scenarios, as already mentioned in section 6.3.

### 7.3.2 Parallel Computing

The algorithms in chapter 5 are written down as sequential instructions to be executed by an implementation. On the other hand, if one looks e.g. on the full search algorithm in listing 1 on page 42, it is obvious that the calculation of packet loss and redundancy information for one parameter set is independent of another parameter set. Computation could be done in parallel to shorten the overall execution time.

Parallel computation is done in software in general via threads. A thread [17] is a sequence of executing instructions. This instructions can run independently of other threads. Threads
are capable of sharing data with other threads within a process. Every process consists of at least one thread. Ruby uses so called *green threads* to emulate a multithreaded environment. It doesn’t need to rely on the native operating system capabilities which enables Ruby to be ran in environments that do not have native thread support. The single Ruby process is simply doing multithreading by time-division multiplexing. The process switches between different threads. The downside is that green threads cannot assign work to multiple processors. If one wants to try out different algorithms that exploit parallel computation, one would need to emulate the parallel computation by creating multiple Ruby processes.
Chapter 8

Conclusions and Outlook

8.1 Summary

The media transport over packet-switched networks is an important topic nowadays. Packet losses impair the transport and affect the user experience. Redundancy is needed to enhance the user experience, but at the same time the media should be delivered in the time the application requires. Furthermore the needed redundancy should be minimal.

Chapter 2 gave an overview over the errors happening in packet-switched networks and how to recover from errors. Automatic repeat-request and forward error correction are introduced as measures for error recovering.

The research group at the Telecommunications Lab of Saarland University has developed a generalized architecture of erasure error recovery. Chapter 3 presented a version of the generalized architecture. The generalized architecture has been extended by new constraints that respect the current capability of the network.

A mathematical framework for the generalized architecture has been presented in chapter 4. It shows how to model the packet loss rate and redundancy information of the generalized architecture. Furthermore an optimization problem has been proposed. The goal is to find the optimal parameters for the generalized architecture to satisfy the optimization problem. In chapter 5 methods of resolution to find the parameter set solving the optimization problem have been proposed. The proposed methods include the full search of the parameter space, a greedy algorithm and an enhanced greedy algorithm. The proposed greedy algorithm originates from the Telecommunications Lab of Saarland University and has been adjusted to the mathematical framework in this thesis. Equations to calculate the complexity of the full and greedy search have been given.

In chapter 6 some aspects of the mathematical framework and of the methods of resolution has been analyzed. The last chapter 7 describes the implementation used during this thesis to analyze the mathematical framework and provides assistance for the future development.

8.2 Future Work

The development and study of the generalized architecture is an ongoing topic at the Telecommunications Lab of Saarland University. Beyond this thesis, the following research might be useful:

- In this thesis we have shown only a few approaches to find the optimal parameter set for the given optimization problem. Further study is needed to try other approaches or extend the current algorithms to be more efficient. The complexity of the packet loss rate and redundancy information equation must also be considered. The rising
of multi-core processors and computation on graphic processing units is especially interesting related to the parallelization of the presented methods of resolutions.

- The mathematical framework presented in chapter 4 is quite sophisticated in terms of complexity. It is an ongoing challenge to model the architecture in a realistic way and to be simple in its nature such that it’s appropriate for practical systems in real-world applications. With the increasing details in the architecture the complexity seems to increase exponentially. Anytime there is a boundary reached where real-world applications do not need or cannot even support that detail level, e.g. because the current hardware architecture does not support that much precision.

- We have dealt so far with a single (virtual) link from sender to each receiver. In real-world scenarios the data between sender and receivers traverses often more than one intermediate link. This is for example the case for the internet. The intermediate links are not only behind one another, but also different independent paths between sender and receiver are possible. In addition each link has different network characteristics. Research is needed how to distribute redundant packets on the different links.

- The implementation of all parts of the mathematical framework and the methods of resolution in the programming language C should be considered. A library written in C facilitates the usage of a common implementation. It might be used in applications written in the programming language C, but also in all other high-level programming languages as long as they allow to write extensions in C. For instance, the application MATLAB is very popular among engineers and allows to call C routines as if they were built-in functions.

- The development and standardization of a media-oriented transport protocol in the future could be another goal. The development of the generalized architecture on real hard- and software could show how well the architecture does in real-world applications and at least approve the theoretical foundation. The work towards establishing a standard of the generalized architecture is also desirable, e.g. as a Request for Comments (RFC). The availability of a standard could lead to more attention and a widespread community working with the architecture to enhance and correct it.
Bibliography


Appendix A

Analysis Data

$RT = \text{data rate}$

$MDR = \text{multimedia data rate}$

$EPL = \text{encoding packet length}$

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$PLR_{\text{target}}$</td>
<td>0.000001</td>
<td>$P_e$</td>
<td>0.1</td>
</tr>
<tr>
<td>$DR$</td>
<td>$4 \cdot 1024 \cdot 1024 \text{ Bit/s}$</td>
<td>$MDR$</td>
<td>4 MBit/s</td>
</tr>
<tr>
<td>$t_{sw}$</td>
<td>0.0025 s</td>
<td>$t_{rw}$</td>
<td>0.0025 s</td>
</tr>
<tr>
<td>$t_{s,\text{min}}$</td>
<td>0.000862 s</td>
<td>$t_s$</td>
<td>0.00258636 s</td>
</tr>
<tr>
<td>$RTT$</td>
<td>0.16 s</td>
<td>$t_{lp}[1]$</td>
<td>0.055 s</td>
</tr>
<tr>
<td>$EPL$</td>
<td>$1356 \cdot 8 \text{ Bit/s}$</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Table A.1: Example scenario used to apply different methods of resolution
<table>
<thead>
<tr>
<th>$D_{\text{target}}$</th>
<th>Full Search</th>
<th>Greedy (worst case)</th>
<th>Greedy (actual)</th>
<th>Improved Greedy</th>
</tr>
</thead>
<tbody>
<tr>
<td>0.05</td>
<td>99</td>
<td>19</td>
<td>13</td>
<td>13</td>
</tr>
<tr>
<td>0.06</td>
<td>195</td>
<td>27</td>
<td>14</td>
<td>14</td>
</tr>
<tr>
<td>0.07</td>
<td>323</td>
<td>35</td>
<td>16</td>
<td>16</td>
</tr>
<tr>
<td>0.08</td>
<td>483</td>
<td>43</td>
<td>17</td>
<td>17</td>
</tr>
<tr>
<td>0.09</td>
<td>709</td>
<td>75</td>
<td>42</td>
<td>37</td>
</tr>
<tr>
<td>0.1</td>
<td>1148</td>
<td>165</td>
<td>60</td>
<td>38</td>
</tr>
<tr>
<td>0.11</td>
<td>2166</td>
<td>321</td>
<td>75</td>
<td>44</td>
</tr>
<tr>
<td>0.12</td>
<td>3968</td>
<td>541</td>
<td>87</td>
<td>49</td>
</tr>
<tr>
<td>0.13</td>
<td>6810</td>
<td>825</td>
<td>99</td>
<td>54</td>
</tr>
<tr>
<td>0.14</td>
<td>10948</td>
<td>1175</td>
<td>113</td>
<td>57</td>
</tr>
<tr>
<td>0.15</td>
<td>16838</td>
<td>1631</td>
<td>150</td>
<td>67</td>
</tr>
<tr>
<td>0.16</td>
<td>24343</td>
<td>2092</td>
<td>168</td>
<td>72</td>
</tr>
<tr>
<td>0.17</td>
<td>40677</td>
<td>2788</td>
<td>192</td>
<td>77</td>
</tr>
<tr>
<td>0.18</td>
<td>70122</td>
<td>3610</td>
<td>213</td>
<td>77</td>
</tr>
<tr>
<td>0.19</td>
<td>120848</td>
<td>4562</td>
<td>236</td>
<td>84</td>
</tr>
<tr>
<td>0.20</td>
<td>202731</td>
<td>5652</td>
<td>267</td>
<td>93</td>
</tr>
<tr>
<td>0.21</td>
<td>298143</td>
<td>6815</td>
<td>299</td>
<td>99</td>
</tr>
<tr>
<td>0.22</td>
<td>479171</td>
<td>8267</td>
<td>331</td>
<td>99</td>
</tr>
<tr>
<td>0.23</td>
<td>771845</td>
<td>9680</td>
<td>359</td>
<td>113</td>
</tr>
</tbody>
</table>

**Table A.2:** Iteration steps taken by the different algorithms for an example scenario

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$D_{\text{target}}$</td>
<td>0.3 s</td>
<td>$PLR_{\text{target}}$</td>
<td>0.01</td>
</tr>
<tr>
<td>$DR$</td>
<td>12 MBit/s</td>
<td>$MDR$</td>
<td>4 MBit/s</td>
</tr>
<tr>
<td>$EPL$</td>
<td>1356 · 8 Bit/s</td>
<td>$RTT$</td>
<td>0.16 s</td>
</tr>
<tr>
<td>$t_{sw}$</td>
<td>0.0025 s</td>
<td>$t_{rw}$</td>
<td>0.0025 s</td>
</tr>
<tr>
<td>$t_s$</td>
<td>0.0025863 s</td>
<td>$t_{s,\text{min}}$</td>
<td>0.0008621 s</td>
</tr>
<tr>
<td>$t_{lp[1]}$</td>
<td>0.165 s</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table A.3:** Example scenario where the improved greedy algorithm returns a global optimum

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$DR$</td>
<td>12 MBit/s</td>
<td>$MDR$</td>
<td>4 MBit/s</td>
</tr>
<tr>
<td>$EPL$</td>
<td>1356 · 8 Bit/s</td>
<td>$RTT$</td>
<td>0.015 s</td>
</tr>
<tr>
<td>$t_{sw}$</td>
<td>0.0025 s</td>
<td>$t_{rw}$</td>
<td>0.0025 s</td>
</tr>
<tr>
<td>$PLR_{\text{target}}$</td>
<td>0.000001</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

**Table A.4:** Example scenario to show adaptivity of the generalized architecture
Table A.5: Example scenario to show adaptivity of the generalized architecture with lower data rate

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Value</th>
<th>Parameter</th>
<th>Value</th>
</tr>
</thead>
<tbody>
<tr>
<td>$PLR_{target}$</td>
<td>0.000001</td>
<td>$P_e$</td>
<td>0.1</td>
</tr>
<tr>
<td>$DR$</td>
<td>4 MBit/s</td>
<td>MDR</td>
<td>4 MBit/s</td>
</tr>
<tr>
<td>$EPL$</td>
<td>$1356 \cdot 8$ Bit/s</td>
<td>$RTT$</td>
<td>0.16 s</td>
</tr>
<tr>
<td>$t_{s,\text{min}}$</td>
<td>0.00086212158203125 s</td>
<td>$t_s$</td>
<td>0.00258636474609375 s</td>
</tr>
<tr>
<td>$t_{sw}$</td>
<td>0.0025 s</td>
<td>$t_{rw}$</td>
<td>0.0025 s</td>
</tr>
<tr>
<td>$t_{lp}[1]$</td>
<td>0.055 s</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>
Index

A

ACK ................. see acknowledgment
acknowledgment .................. [9]
ARQ ............. see automatic repeat-request
automatic repeat-request .......... [9]

B

BEC ............. see binary erasure channel
binary erasure channel ........... [16]

C

C extension .................. [61]
cache .................. [61]
capacity .................. [16]
channel .................. [15]
channel state information ....... [31]
code rate .................. [16]
codes .................. [16]
CSI ................ see channel state information

D
data rate ................. [69]

E

encoding packet length .......... [69]
EPL ............... see encoding packet length
erasure channel ............... [16]
error correction code ........... [20]

F

feedback suppression .......... [24]
full search .................. [41]
full search algorithm ........... [41]

G

Gilbert Elliot model .......... [18]
Go-Back-N ARQ ............... [20]
greedy .................. [44]
greedy algorithm ............. [44]

H

HARQ, see hybrid automatic repeat-request
hybrid automatic repeat-request ...... [21]

I

improved greedy algorithm ....... [17]
iteration .................. [60]

J

Java ................ [62]

M

MATLAB ................ [62]
MDR .............. see multimedia data rate
multimedia data rate ........... [59]

N

NACK ........ see negative acknowledgment
negative acknowledgment ........ [19]
nonsystematic ............... [21]

P

Python ................ [52]

R

recursion .................. [80]
redundancy information ........ [16]
Request For Comments .......... [86]
return channel ............... [19]
RFC .......... see Request for Comments
RI ........ see redundancy information
RT ........ see data rate
Ruby .............. [59]

S

Selective Repeat ARQ .......... [20]
Stop-and-wait ARQ .......... [20]
systematic ........................................ 21

T

tables ........................................... 56
TDD .......... see Test-driven development
Test-driven development .................... 59
thread .......................................... 62
Type I HARQ .................................... 21
Type II HARQ ................................. 21

W

window size ................................. 20
worst receiver ......................... 28