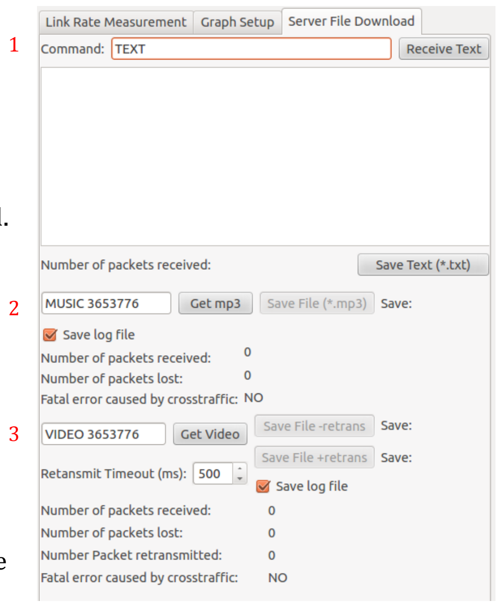
Data Acquisition Tool

As a base for the implementation of the packet reception the Link Rate Measurement Tool was used again. Due to the fact that no strong time constraints have to be met, it was decided to handle incoming data packets by a event driven data receiving function. This reduces the amount of computational power required to receive and to process the data significantly and leaves the user more resources while he is waiting for the arrival of the packets. The application has no restrictions concerning the maximum data size and the maximum count of sequences, because it allocates new memory automatically in a dynamic manner. This was realized by using a specialized object oriented data type QMap, which has in comparison to QList the advantage of automatically sorting the entries in relation to the specified data type. Here the data type used for the QMap as sorting parameter is quint64. For an effective data handling the third and the fourth byte of the incoming UDP packet, which represent the packet sequence number within the RTP header, are converted into a 16 bit unsigned integer number. After deriving the packet number from the RTP header the first 12 byte of the UDP packet, which represent the RTP header are chopped off and the packet payload is stored into the QMap variable at the index (key) derived from the RTP header.
The implemented functions can be accessed by setting up the server connection and then selecting the Server File Download tab. An additional reason to use the Link Rate Measurement Tool as a base for the program was the idea to develop a more powerful tool that has the performance to analyse network traffic for the user more efficiently. Furthermore the graphs could be linked to the server file download to display the average data rate. To acquire the average data rate it would be necessary to monitor the inter arrival time of the packets in relation to their data size. The withdraw of the this implementation is the application requiring harder real time constraints in relation to data handling to achieve a more precise measurement and by this the application would require more computational performance. A trade off would be just to monitor the average bitrate between the sending of the command and the reception of all byte. In the current application no monitoring was implemented yet, to improve the performance figures of the system and to avoid a non event driven data reception.
In the following the single implementations will be discussed further separately.



Command: TEXT — Receive Text

Number of packets received: — Save Text (*.txt)

MUSIC 3653776 — Get mp3 — Save File (*.mp3) — Save:

☑ Save log file

Number of packets received: 0
Number of packets lost: 0
Fatal error caused by crosstraffic: NO

VIDEO 3653776 — Get Video — Save File -retrans — Save:

Retansmit Timeout (ms): 500 — Save File +retrans — Save:

☑ Save log file

Number of packets received: 0
Number of packets lost: 0
Number Packet retransmitted: 0
Fatal error caused by crosstraffic: NO

1) Text Reception

The text reception consists of a command line edit and a text field output. In the command line edit any command can be specified, that will be sent to the server. When sending the command a variable is set, which later determines how the data in the input buffer queue of the network card is served. For the text reception the packet data is simply given out at the text field. Furthermore the number of packets that were received is given out to the user and the received information can by saved as an Utf8 decoded text file. No log file will be created.

2) Download Music File

The music file can be downloaded from the server by specifying the file in the line edit and then clicking on the “Get mp3” button. Incoming data will be handled, in relation to the variable set as operation mode when clicking the button, as an mp3 file. After a complete reception the “Save File” button will be enabled and the data can be stored. Furthermore a log file with the connection report will be stored, when the “Save log file” checkbox is checked by the user. The name of the log file equals the name of the mp3 file, but is extended with the characters “_warn” and stored as a txt file.
(Mp3 file name: “Myfile.mp3” => Log file name “Myfile_warn.txt”)
The number of packets received is derived from the number of packets, which were stored into the QMap variable. The number of packets lost is calculated by subtracting the number of elements stored in QMap from the highest number derived from the RTP packet header sequence numbers. Furthermore the program is able to detect if packets, which probably don’t belong to the data transmission requested by the user, influenced the transmission (cross traffic). This is achieved by comparing the number of received packets since sending the command with the number of packets stored inside the buffer. Furthermore a very high number of lost packets in relation to the number of packets received can pinpoint to errors caused by cross traffic, because it can happen that incoming packets that don’t belong to the transmission are interpreted as RTP packets, because their payload will meet the parameter we analyse of the RTP header. To guarantee that no packets that don’t belong to the transmission will be stored into the data

memory, it would be necessary to check further parameters of RTP header or to scramble the payload and attach a checksum to it. In case of any of the previously discussed errors the notification “Fatal error caused by cross traffic: YES” will be given out to notify the user. Furthermore the information will be stored into the “mp3name_warn.txt” file. In the following the format of the content of the log file can be seen:

```
TRANSMISSION REPORT
Transmission: SUCCESS
Expected sequences: 479
Received sequences: 479
Lost sequences: 0

The MD5 sum of the music file is: a5c148c02208d0c521c553517651131e
```

3) Download Video File
When clicking on the “Get Video” button the video data will be requested from the server and the received packets will be stored in the same manner like the packets of the mp3 file. It was checked, that all packets, which are received are stored into the output file. Furthermore it was verified that no zeros or other data elements are written into the output file. The implementation of the data acquisition function as an event driven function or a sequential function polling the input buffer for data packet reception leads to no difference in terms of the required data. The sequential implementation was done only, because at first it was impossible to decode the original sequence without retransmission, using the decoder downloaded from the website announced in the slides. Furthermore attention was paid to make sure the sequence number of the packets contained by both files increases constantly. The sequence numbers of the missing packets are directly derived from the original file, taking the highest sequence number received as upper limit and searching the data set for empty entries. Each missing packet is requested at the server for retransmission. After requesting the missing packets from the server for retransmission a watchdog timer is setup. In case some packets are lost again, the watchdog timer ISR will search in the current data set for missing packets and request their retransmission again. The user can set the interval of the watchdog timer manually. After receiving all packets the watchdog timer will be automatically disabled and the save buttons will be enabled again. The user output is similar to the Download Music File section, except that the program gives out the number of packets that were already retransmitted in addition to the number of packets received and lost. By clicking on the “Save File –retrans” button the original sequence without retransmission will be saved. A relating log file will be saved equivalent to the mp3 file section, when the checkbox is checked. When pressing the “Save File +retrans” button the file containing the original sequence as well as the retransmitted sequences will be stored. The log file of the retransmitted file contains, in difference to the log file of the file containing just the number of original received sequences, an additional line, mentioning the amount of packets that were retransmitted. In addition the sequence numbers of the retransmitted packets are shown to the user at the Link Rate Measurement tab in the output console.
In the following you can see the output log files as well as the results of the decoder supplied on Moodle.
Decoding of file lost_news.264 (original sequencess) and computing SNR of with news.qcif.yuv file as reference:

```
-------------------- Average SNR all frames ------------------------------
 SNR Y(dB)         : 29.38
 SNR U(dB)         : 37.27
 SNR V(dB)         : 39.16
 Total decoding time : 1.264 sec
--------------------------------------------------------------------------
```

Decoding of file news.264 (original+retransmitted sequence) and computing SNR of with news.qcif.yuv file as reference:

```
-------------------- Average SNR all frames ------------------------------
 SNR Y(dB)         : 37.88
 SNR U(dB)         : 40.92
 SNR V(dB)         : 41.30
 Total decoding time : 1.298 sec
--------------------------------------------------------------------------
```

LOG FILES

lost_news_warn.txt:

```
TRANSMISSION REPORT
Transmission: ERROR
RETRANSMISSION OF LOST SEQUENCES APPLIED: NO

Expected sequences: 457
Received sequences: 426
Lost sequences: 31
```

news_warn.txt:

```
TRANSMISSION REPORT
Transmission: ERROR
RETRANSMISSION OF LOST SEQUENCES APPLIED: YES

Expected sequences: 457
Received sequences: 426
Lost sequences: 31
Retransmitted sequences: 31
```

Remarks: Using the current at the website specified in the slides available decoder leads to the result of being unable to decode the original sequence!

Installation / Run Problems
Please refer to link rate