An Error Protection Protocol for user-transparent bridging of Fast Ethernet data transmission over the optical fading channel in an Aeronautical environment

Bernhard Epple[*a], Hennes Henniger[a], Clara Serrano Solsona[a]
[a] German Aerospace Center (DLR), Institute of Communications and Navigation, 82234 Wessling, Germany;

ABSTRACT

This paper presents an extensive insight into error protection techniques for free space optical links, focusing in particular in aeronautic stratospherically applications. The long distances present in these scenarios along with challenging atmospherically conditions present significant obstacles that degrade link performance. Thus it is imperative to apply highly efficient error protection scheme to avoid unacceptably high loss rates. The goal was to design a point-to-point data link layer error protection protocol that allows user-transparent bridging of Fast Ethernet data transmission over the optical fading channel in an high altitude inter platform link environment.

Keywords: error protection, protocol, ARQ, Ethernet

1. INTRODUCTION

High Altitude Platform Stations (HAP) promises a low-cost and quick access to modern telecommunication services. Such a perspective has been appealing to all civil and military users in remote and rural areas where there is little or even no modern telecommunication infrastructure available. In the future it is planed to use high altitude platform networks to provide services like voice, video-streaming, video on demand, reliable file transfer or web browsing to the end users. These HAP networks are seen as a cost effective alternative to expensive satellite networks. For connecting a large number of users a great amount of backhaul traffic has to be exchanged between the HAPs. Backhaul inter platform links bring up the need to provide sufficient throughput for backhaul transport networks. Free-space optical communication technology satisfies this increasing traffic demand.

One of the biggest challenges facing free-space optical deployment is its optical signal propagation in different atmospheric conditions. This effect will cause variable link degrading due to variable attenuation and slow fading. Large difference of channel coherence time and bit duration in the FSO channels causes problems if working with physical-layer coding to mitigate link blockings and fades. Interleaving over some tens of milliseconds will be needed in order to avoid erasure of whole codewords during fades. For high data-rates these interleavers are technically not feasible. Looking at the channel from the packet transmission point of view it can be observed that only a small number of packets instead of a large number of bits is lost during fades. This is the result of the inherent longer transmission time of packet-symbols instead of bit-symbols. Lost packets can be recovered with proactive (forward error correction, FEC) and reactive recovery schemes (automatic repeat request, ARQ). In general reactive schemes are more effective because only the lost data is transmitted again. Forward error correction is working with fixed overhead which is not always really in use. In this paper a single hop link layer ARQ system is investigated. First a standard ARQ system working on transport layer is discussed for comparison.

[*] Bernhard.Epple@dlr.de; phone +49 (0) 8153 2816; fax +49 (0) 8153 2844; www.dlr.de/kn/

2. REALIBLE TRANSMISSION BASED ON TRANSPORT LAYER RETRANSMISSION

The free-space optical channel can in general be seen as a channel with high loss probabilities. Long signal fades cause burst errors within the communication stream and cause higher loss rates than generally experienced on wired media. In the wired world the Transmission Control Protocol (TCP) combined with the Internet Protocol (IP) can be seen as the standard transport protocol suite and is used by many applications. Unfortunately TCP behaves very badly on the FSO channel. The burst errors present on the FSO channel cause TCP/IP to misinterpret these as channel congestion and TCP/IP is reducing its transmission rate until it finally stops sending new data. After the end of the error burst the so called slow-start algorithm, which is implemented in TCP to avoid a new congestion of the channel, keeps the protocol stack at a low transmission rate even if the FSO channel is nearly error free. In most cases a new fade will cause burst errors and cause the TCP/IP stack to slow down transmission before it has fully recovered from the previous error burst. Therefore the error bursts will cause the TCP/IP communication to die. This bad behavior of TCP in the free space optical fading channel has been practically proven in [1,2]. This issue has been addressed by extensions for TCP/IP like explicit congestion control and header checksums but these are not implemented or activated on most platforms. Another problem for TCP/IP is the long link distances combined with the high data rates present in most FSO scenarios. This has two reasons. First, most estimations within TCP/IP are done on a per round trip time basis. That means parameter selection is done based on the time needed for sending a packet to the receiver and receiving the acknowledged from the receiver. This means that the first round of data is send using standard parameters which generally are very conservative and do not really match the current channel. If the product of delay and bandwidth is high, this can be a lot of data that is send with the wrong parameters. The second reason for a poor performance of TCP/IP over longer link distances is that it belongs to the family of Go-Back-N Automatic Repeat request protocols. If one packet is detected by the sender to be lost, the sender will go back in history of the transmitted data stream to this packet and restart the transmission at this packet. The earliest point when the receiver can detect the loss of one packet is after one round trip, therefore the sender is at least retransmitting the data sent during this round trip just because one packet got lost. Combined with the high error rates in the FSO channel it is obvious that this behavior leads to a very inefficient transmission of data. As it has been explained TCP/IP is not suitable to be used to ensure reliable data transfer on links were long outage times are expected. For enabling wide spread applications that make use of TCP/IP in the FSO environment it is necessary to design a underlying data link layer ARQ system which satisfies the following requirements: First it transports data reliable over the FSO fading channel. This means that the residual data loss must be less than about 1 % so that higher layer protocols like TCP can deal with it. Second it must be transparent to higher layer protocols. That means it fulfills Quality of Service (QoS) requirements given by the transported services. Normally voice gives the most demanding requirements in terms of delay and delay jitter. For voice the delay must not to be longer than 400 ms with a jitter of less than 1ms. It has to be seen that the p2p FSO link will only be a part of the communication network connecting users. Therefore the optical link should not exhaust service requirements in order to keep some margin for other links in the system.

3. RELIABLE SINGLE HOP DATA LINK LAYER TRANSMISION SYSTEM

In this chapter the implementation of a Fast Ethernet based reliable transmission system for connecting two network segments is discussed. This system must to be transparent for higher layer protocols and should fulfill the QoS requirements to transport voice.

3.1 System Setup

The system is implemented as experimental system that allows for as much as possible flexibility during the development process. Therefore it has been decided to implement the protocol as C/C++ software on standard PCs. For accessing data on the data link layer from within C/C++ the packet capture (pcap) library[3] can be used. Pcap gives programs direct access to the data link layer and offers routines for capturing, filtering and injecting packets. In the developed system three Ethernet interfaces are used. One is connected to the local area network (LAN) from which the traffic should be bridged over to another network. The other two Ethernet cards are used as unidirectional communication lines for the bridged traffic. The advantage of splitting the bidirectional traffic of the LAN into two unidirectional streams is that the two communication directions can be implemented using different technologies on the physical layer. For example an unmanned aerial vehicle (UAV) equipped with a video camera will produce a heavily asymmetric type of traffic. The streamed video will require a high bandwidth in the direction from the UAV towards the control center. From the control center to the UAV only a few commands will be sent, so a low bandwidth connection is sufficient. By using the split communication path it is possible to implement the direction from UAV to control center

using high bandwidth FSO and the direction from control center to UAV using easier to implement radio frequency technology. The different transport media will be transparent to the bridged traffic. The Selective-Repeat Protocol is implemented on top of the pcap library. It receives the captured packets from the library, processes it and forwards newly created packets to the library for output on the appropriate interface. For output to the LAN a play-out buffer is implemented which holds back the packets intended for output to the LAN for a certain amount of time.

Fast Ethernet uses a continuous link integrity test which is implemented with so called fast link pulses (FLP). As fading can interrupt the channel for a long time it is possible that link integrity tests fail and the transmission is blocked. To avoid this media converters are needed which locally response to FLPs, for feigning a stable link.

The complete system setup is illustrated in Fig. 1. In the following some special implementation issues are discussed.

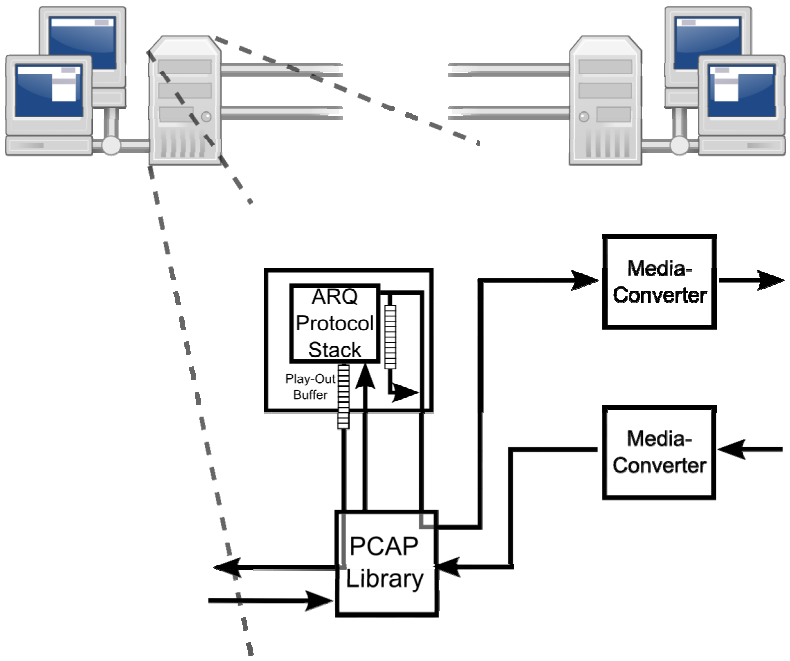

Fig. 1: System setup of the transmission system.

3.2 ARQ mechanism

The retransmission of lost data is controlled by a Selective-Repeat ARQ protocol. Selective-Repeat ARQ has been chosen due to the large delay-bandwidth product that is expected in the given scenario (1.7 ms @500 km × 100 Mb/s = 170 kb). For large delay-bandwidth products Go-back-N and Stop-and-Wait are known to give poor performance[2]. RFC 1072[4] defines the bandwidth delay product as being large if it exceeds 100 kb which is the case in the given scenario. Generally the way in which data is acknowledged can be done in three ways:

Either data has to be explicitly acknowledged by the receiver (by so called ACK packets) before the sender stops retransmitting packets after a certain timeout. Another way is that the sender expects the receiver to send not acknowledgment packets (NCK). Further it is possible that the receiver is sending ACK and NCK packets depending on the packets received.

For realtime traffic it is important that lost data is recovered as fast as possible. As any other packets, ACK and NCK packets can also get lost. If this is the case with the use of ACK packets, the sender will wait for the ACK until the integrity timeout t_{t0} has passed. Then it will unnecessarily retransmit a packet. This retransmission does not introduce additional delay because the receiver already has stored this packet and will simply drop the duplicate. On the other hand, if a NCK packet gets lost, the sender does not get informed about a lost packet which retransmission should be requested by this NCK packet. Therefore the sender will not retransmit this packet until it receives the NCK after another

try. Therefore the loss of a NCK packet adds additional delay for the transmission of the data. To reduce the additional delay it has been decided to use explicit acknowledgement of data by sending ACK packets.

The integrity timeout used for triggering retransmission of not acknowledged data must be larger than the round trip time (which is equal to twice the propagation delay t_p). This is because first the data is transmitted from sender to receiver and then the receiver must wait for an acknowledgement packet (ACK) which again takes at least the duration of the propagation delay. A zero processing time is assumed in this case.

$$t_{to} \geq 2 \cdot t_p \tag{1}$$

A suitable time for t_{t0} is for example 4 ms.

In other implementations if NCKs are used the sender first has to receive a NCK before it retransmits data. If a NCK is lost, the receiver has to wait for a timeout for this packet and retransmit the NCK. This adds additional delay to the retransmission of data.

3.3 Blocking Recovery Mechanism

When designing a protocol for the optical channel, it has to be considered that turbulences can temporary cause total blocking of the channel. Turbulences cause the optical intensity of the laser beam to fluctuate. This creates speckle patterns in the propagated beam. The strength of the fluctuations increases with the propagation distance. It reaches a maximum and then saturates. For inter-platform links the long distance up to 500 km or even more are feasible. If the receiver has a collecting aperture that is large enough, it can average several intensity speckles over its aperture and, in this way, reduce the fluctuations of the received signal. As the size of the receiver aperture increases, more power is collected and additionally scintillation is reduced. But normally receiver size is limited on aeronautical vessels. The size of the intensity speckles generally increases with the propagation distance till the beam wave reaches the saturation regime. In the saturation regime, the speckle sizes tend to decrease as the wave propagates. As transmit power and receiver sensitivity are technically limited and also because of pointing accuracies the minimum beam divergence is limited the receiver has to deal with relatively low received power causing a low signal to noise ratio (SNR). Additionally the received power fluctuates. So the SNR drops temporary. With low SNR the probability of bit substitution errors is high. Furthermore low SNR causes the digital recovered signal to jitter which can lead to a loss of lock of the data and clock recovery. In this case not only wrong data occurs but also symbols are lost until a new synchronisation is locked. Therefore the channel can get temporary totally blocked by atmospheric fading. This is a similar behaviour as it can be observed if for example a bird obscures the laser beam. For voice transmissions these complete link blockings will cause noticeable cracks in the voice stream or completely interrupt the call. If the blocking duration is longer than 400 ms, which is the maximum delay allowed for voice transmission, the data loss will be noticed by the end-users. If the blocking duration is shorter than 200 ms the lost data can be recovered and delivered to the end-user before he notices the loss. If the blocking duration is between 200 and 400 ms, the end-users might notice the data loss. Retransmission of frames is only useful if the time between the retransmissions is larger than the channel coherence time τ. If the retransmission is done in the same channel state as in which the data was lost in the first place, it is most likely that the data will get lost again. So retransmission can be seen as time diversity scheme and in the ideal case the channel's error behavior will be independent at the inertial transmission time and at the time of retransmission. In this case a single retransmission is adequate. The delay of the ARQ system t_{ARQ} can then be calculated by the following: At least after the integrity timeout t_{to} the sender realizes that data is lost. Normally the integrity timeout t_{to} is smaller than the channel coherence time τ. In order to retransmit the data the second time in a channel state which is not correlated with state present at the originally transmission time the protocol hast to wait at least for a time τ before it starts retransmission. Now it takes the propagation delay t_d until the data is arriving; this time correctly.

$$t_{ARQ} = \tau + t_p \tag{2}$$

In this case the processing delay is neglected. So for example a typical value for τ = 20 ms, t_{to}= 4 ms, and $t_{p(500\ km)}$ = 1.7 ms one comes up with 21.7 ms delay. It can be clearly seen that slow fading is the primary factor which causes delay in an ARQ system. If a significant processing delay in the order of τ is introduced by the protocol stack it has to be considered additionally. Only if channel coherence time is in the order of milliseconds real-time services can be applied. If the fading is too slow the delay introduced by each retransmission might be longer than the delay allowed by service

requirements. That makes ARQ suitable for mobile scenarios where faster fading can be observed. Techniques shortening the outage time are appropriate. For example if a physical layer error correction code is used which enables packet transmission (with a low residual packet error probability) even if the short term bit error probability on the physical channel is 1×10^{-3} (which corresponds to a quite low signal to noise ratio at the receiver) outage probability is less and further fades are quasi shortened. Normally in a fade the power is decreasing steadily and after having reached a minimum it is increasing steadily. So physical layer coding can help to reduce the power level below which the system is not working properly and producing extensive errors. Because of the steady "V"-like characteristics of a fade also the fade duration is shortened if physical layer coding is used.

For being able to correct errors caused by long signal interrupts, the data flow has to be delayed prior forwarding it to the end-user. For holding back the data for a while, a play-out buffer is implemented. If an uncorrelated channel is assumed between inertial- and re-transmission the play-out buffer should be t_{ARQ}. For the not ideal case the buffer should be a few times larger than t_{ARQ} in order to support several retransmissions. Nevertheless the overall data transmission delay should not be longer than the maximum allowed delay by the service.

3.4 Link Status Monitor

In a networking scenario it is necessary to check the integrity of the channel continuously. The channel can very frequently get broken by slow fading (some tens of milliseconds), transmitter and receiver alignment problems (some tens of milliseconds) or handover between network nodes (some seconds). In order to detect link losses and stop transmitting or to report the broken link to other network nodes, at least every ten milliseconds a link integrity test should be done. Therefore it is recommended that at least every ten milliseconds a so called keep-alive packet is sent. The arrival of a keep-alive packet can be used to notify that the link is working. If the transmitter was silent and nothing is received which requires sending NCK or ACK at least a link integrity packet should be sent on the return channel every ten milliseconds. This enables the other side to verify the channel availability even if no data was transmitted. Additionally channel state information can be generated using received power either of the data signal or of the beacon signal. Doing this one has to clarify if the fading behaviour is not depending on the propagation direction. For horizontal links this is more or less the case, for satellite up- and downlinks not.

3.5 Packet Format

The used pcap library operates on the link layer on which Ethernet frames are used to transmit data, therefore the captured data will be the data in the format of Ethernet frames and all the injected data should have the format of Ethernet frames. One exception of this rule is the 4 Byte cyclic redundancy checksum (CRC) at the end of the Ethernet frames. Pcap is only capturing valid frames and therefore not forwarding the CRC to the program. If a frame is injected into the Ethernet pcap is automatically adding the correct CRC at the end of the frame, so the program again does not have to care about the CRC. Pcap on it turn does not care about the content of the frames, so it will not filter frames that have for example an MAC address not present in the network. So in principle the library allows to inject any type of data to the network which will be automatically secured by CRC. On the network segment this data can be captured and validated using again the pcap library. It has to be kept in mind that most of today's Ethernet hardware will filter out such data packets because of the random data in the address and type fields; so this approach will only work reliably on direct connections as they are setup in the presented system. Fig. 2 shows the format of the Ethernet frame as they are present on the LAN side of the bridge. These packets are captured by pcap and forwarded to the ARQ protocol.

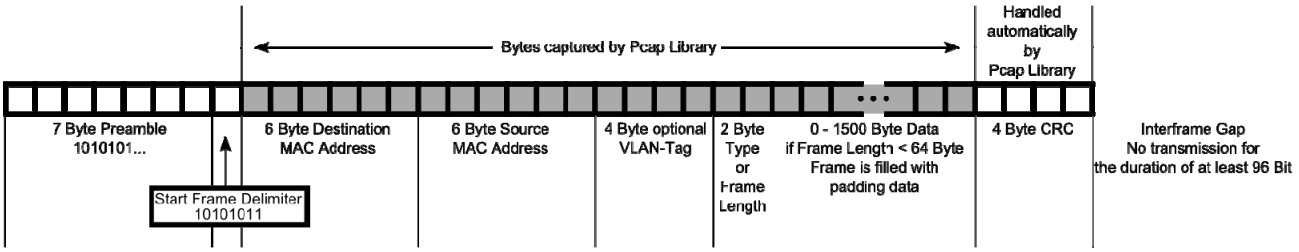

Fig. 2: Format of an Ethernet frame. The data from the Ethernet frames that has to be wrapped by the ARQ protocol is marked by grey blocks.

The ARQ protocol treats each Ethernet frame as a continuous series of bytes and wraps it into its own packet format. These generated frames will look to other hardware or software like normal Ethernet frames, but they can not correctly

be interpreted as such frames. The used packet format is explained in the following. For technical reason the size of an Ethernet frame is defined to be never less than 64 byte (including CRC), so the smallest junk of data that has to be wrapped into the packets from the ARQ protocol is 60 bytes large. The maximum size of an Ethernet frame is defined to be 1522 byte (including CRC). So the larges data junk wrapped into the protocol packet format will be 1518 bytes large. The size injected by the pcap library has also fit within the given limits. The protocol has to add some additional information to the packets so that packets larger than 1522 byte would be created. For this reason the Ethernet packets are split into two halves for sending over the bridge and reassembled at the other end before injecting them into the p2p Ethernet LAN connection. To prevent the construction of Ethernet frames smaller than the given limit, frames are only split if the resulting ARQ packets are larger than 64 byte. Due to the additional data added by the ARQ protocol, this is the case for Ethernet frames of at least 98 bytes of size. Additional information that has to be added to each packet is a 4 byte sequence number for identifying the packets and a 2 byte type value. The type value is needed for two reasons. First, some Ethernet cards read the type field of Ethernet frames and interpret it as the size of the packet if it has a value of less than 1536 (0x0600 in hexadecimal representation). If the supposed length value does not match the real length of the frame, this frame is dropped by the cards. Second, operating systems send from time to time some Ethernet frames for discovering services available on the network. These packets will get captured by pcap on the two connections within the bridge and might be misinterpreted as ARQ protocol packets. All the frames created by the operating systems have a valid Ethernet type entry in the type field so they can be filtered by this field. The ARQ packets will also carry the type field with a value larger than 0x0600 that is not used by known services. It has been decided to use 0xC0DE as type value. This code has always to be at position 17 and 18 in the packet for marking a valid Ethernet frame. Finally one bit is needed for signalling if an ARQ packet carries the first or the second half of an Ethernet frame. For performance reasons and the ease of implementation it is always best to use complete bytes in data structures, so the rest of the byte carrying the flag is reserved for future purposes.

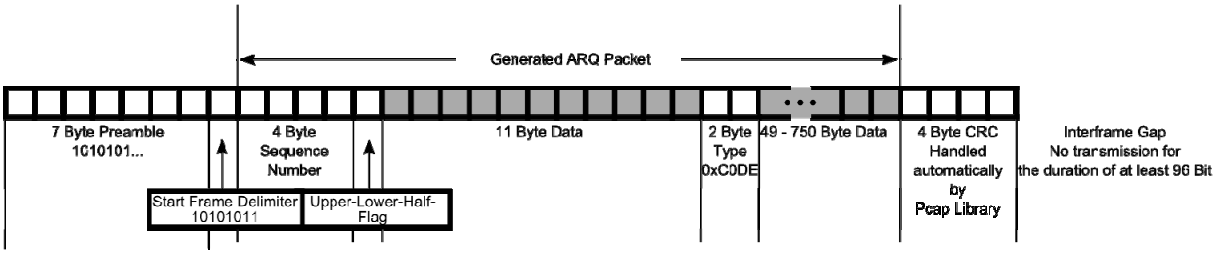

Fig. 3: Packet format used by the ARQ protocol. If this is compared with Fig. 2 it can be seen that the resulting frames can no longer be treated as normal Ethernet frames as the fields required for Ethernet frames are no longer present. Therefore these packets are only valid on the point to point link between the two parts of the bridge. The data wrapped into these ARQ packets are marked by grey blocks. These packets have a size between 71 and 772 byte.

3.6 Acknowledgments Packets

For Selective-Repeat-ARQ protocols the acknowledgement packets have a special functionality. They are not only used for acknowledging one received packet, they also carry information about non-contiguous blocks of data that have been received and queued. For detecting packet losses, the receiver keeps track of the expected sequence number of the next packet. If the sequence number of a received packet is larger than the expected sequence number, this indicates the loss of the packet with the expected sequence number. With Selective-Repeat-ARQ all the received data is stored in the receiving buffer. The receiving buffer is organized as a contiguous block of data. If a packet loss is detected a gap in the contiguous data is inserted as a placeholder for this packet. By this approach the buffered data gets segmented into blocks of contiguous data. For preventing the sender to unnecessarily retransmit the already received data, the information about the contiguous data blocks are transmitted in the acknowledgement packets. A block is characterized by its left edge, which is the sequence number of the first packet belonging to this data block, and by its right edge which is the sequence number belonging to the first lost packet that broke up the contiguous junk of data. ACK packets are part of the bidirectional data flow between sender and receiver, so they carry also a sequence number for identifying their position in the stream. Other information needed in ACK packets by the protocol is the ACK number which is the sequence number of the last packet received prior the first data loss has been detected. This packet should be the next packet transmitted by the sender. Like other packets byte 17 and 18 are used to carry the type 0xC0DE for marking the ACK packet as part of the ARQ traffic. As with other packets the pcap library automatically adds a 4 byte CRC at the end of the injected Ethernet frame. The number of blocks that are created during a transmission varies over time and depends on the error distribution on the channel. For implementation issues it is mostly common to set the number of

blocks carried in the ACK packets to a fixed value. Since each block consist of two 4 byte numbers marking its edges, it is possible to fit six blocks into an Ethernet frame of the minimum size of 64 byte, if the given protocol overhead is included in the calculation. It has been decided to use packet of the minimum size as ACK packets and therefore space for six data block is reserved in the ACK packets. If data loss would produce more data segments at the receiver, the receiver can not acknowledge this data to the sender and the sender will unnecessarily retransmit some data. The format of the ACK packets used by the implemented SR-ARQ is given in Fig. 4.

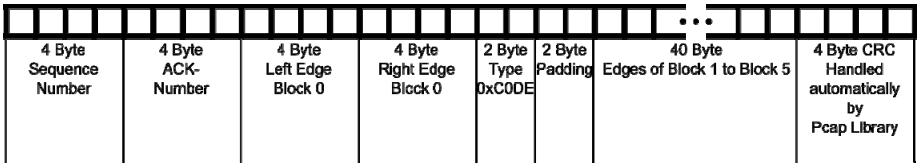

Fig. 4: Format of acknowledgement packets used by the SR-ARQ protocol. The format is layed out to fit with in the minimum packet size of 64 bytes for Ethernet frames. Since the resulting packet has a size of 62 byte, two byte of the frame are filled with padding data for reaching the required size of 64 byte.

3.7 Software Structure

The software is structured in an object oriented and multithreaded way for utilizing the power of today's multi-core processors. The object orientation allows for an easy replacement of the protocol without changing any other code than the protocol itself. So the presented system can be used for implementing other protocols than the Selective-Repeat ARQ and it can also be used for the comparison of the performance of different protocols. All the protocol logic is implemented in the main class named ARQProtocol. This main class owns all the implemented queues for storing and processing packets. ARQProtocol is also the class that receives all the packets captured by the underlying pcap library. For processing the packets and managing the retransmission of lost data, several threads are implemented. The execution of these threads (start and stop) is managed by the main class. Every thread operates on the data stored in one of the buffers. If no data is available in the assigned buffers, the thread goes to sleep. If some data has been processed by a thread, the thread inserts the result of its processing steps into the buffer for the next task. This approach allows for parallel processing of data. For example while one thread is creating an ARQ packet from an Ethernet frame captured from the adjacent LAN, another thread can inject an Ethernet frame received via the bridge into this LAN. An activity diagram of the software is shown in Figure 5. This figure also contains additional markers for illustrating the multithreaded structure of the software.

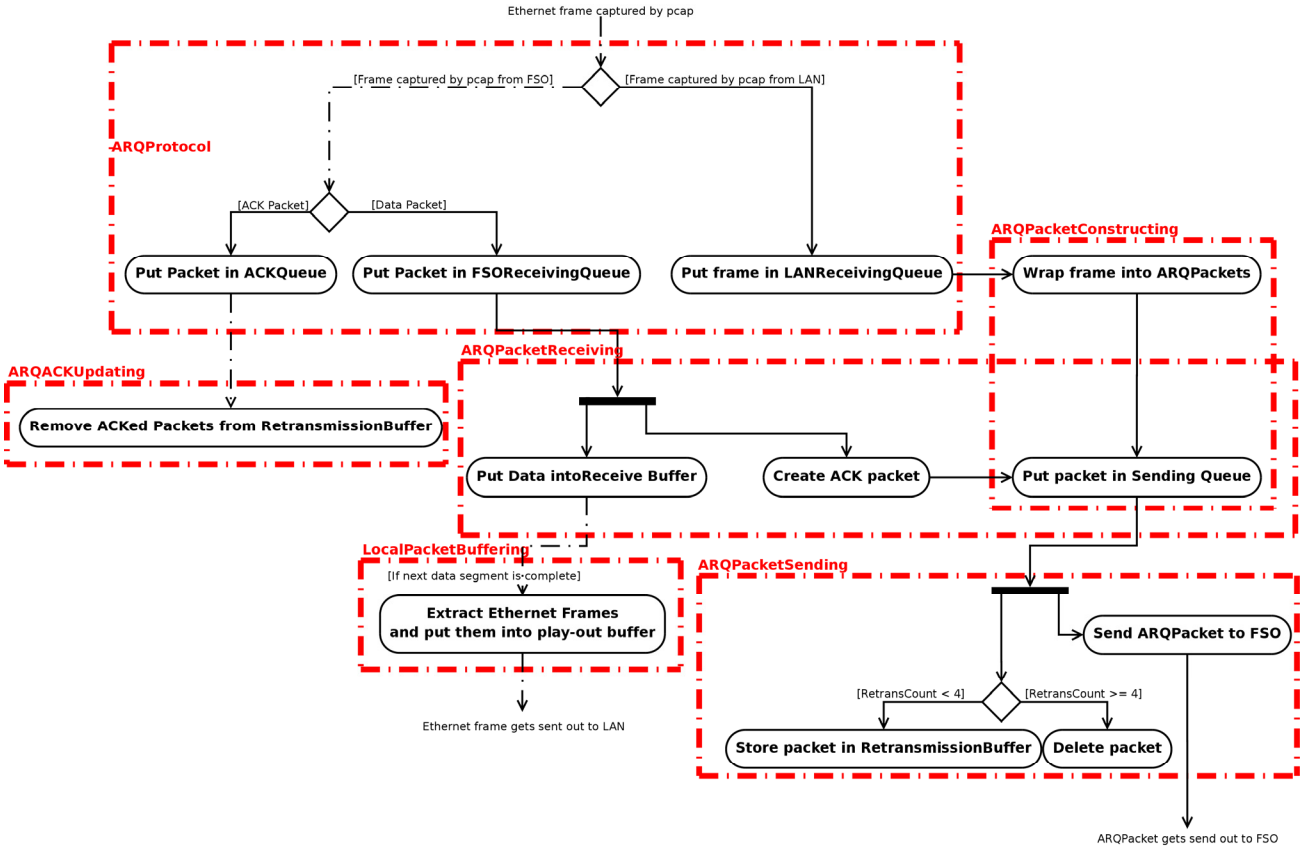

Fig. 5: Activity diagram of the implemented ARQ protocol stack. For illustrating the parallel design of the software the steps done by different threads are grouped by dash-dotted rectangles. The communication between these threads is done via queues or similar data structures.

CONCLUSION

In this paper it has been shown how a link layer ARQ protocol has to be designed for addressing the mitigation of link outages present in long-distance free space optical links, like stratospheric inter platform links. The design of the presented system is kept as flexible as possible so it can easily be adopted for future purposes and different link scenarios. The implementation is already running in a lab environment and it is planned to evaluate its performance in a FSO demonstration in the next year.

REFERENCES

[1] G. W. Johnson, et. al., *Characterization of Gigabit Ethernet Over Highly Turbulent Optical Wireless Links*, International Society for Optical Engineering: International Symposium on Optical Science and Technology, Seattle, Washington, July 2002.

[2] B. Epple, *Performance Optimization of Free-Space Optical Communication Protocols based on results from FSO Demonstrations*, Proceedings of the SPIE, Vol. 6709, 2007.

[3] Tcpdump/libpcap, http://www.tcpdump.org/

[4] V. Jacobson, R. Braden, *RFC 1072 TCP Extensions for Long-Delay Paths*, http://www.ietf.org/rfc/rfc1072.txt, 1988.