A packet-level adaptive forward error correction scheme for wireless networks

Johannes Hund, Andreas Heinrich, Andreas Ziller, Christian Schwingenschlögl, Rolf Kraemer
{johannes.hund.ext,a.heinrich,andreas.ziller,chris.schwingenschloegl}@siemens.com,
kraemer@ihp-microelectronics.com

Abstract—Especially for synchronization-critical wireless networks like ultrawideband impulse radio (UWB-IR), data packets are lost not only due to single bit errors in the payload but also to a large degree because of synchronization errors or preamble failures. Current FEC codes only address bit errors inside a packet. Packets that are lost because of errors in preambles or headers can only be recovered on packet level. In this Paper we propose a low-complexity adaptive packet-level FEC and prove by simulation that it can reduce packet loss with very small overhead.

Index Terms—Erasure Coding, UWB, IEEE 802.15.4a, FEC

I. INTRODUCTION

CURRENT networks, especially wireless networks, suffer from lost packets as one of the main reasons for performance reduction. When packets are lost, the quality is affected either because of the lost data or because the packet needs to be resent. The most popular technique to resend data packets is called Automatic Repeat Request (ARQ). With this technique, every packet gets acknowledged by sending an ACK packet from the receiver to the sender. If the sender does not receive an ACK packet within a certain time after sending the data packet, it assumes the data packet to be lost and re-sends the data packets.

There are two main reasons for packet losses:

1) Failed check sums. Packets include a check sum (mostly CRC-10) that is computed over the payload, a failed check sum indicates that one or more bits of the payload have been incorrectly transmitted. In this case a receiver is aware of a received packet that had to be discarded.

2) Failed packet recognition. This can happen if a packet’s preamble is distorted, so that the receiver fails to synchronize to the sender’s transmission, or because of bit errors in the header or footer which renders the packet unrecognizable. In this case a receiver is not aware of a received packet.

Instead of only reacting to packet loss, proactive techniques are used to avoid lost packets. For the first case, this is done by encoding the packet’s payload with a Forward Error Correction (FEC) code. This FEC can repair bit errors inside the packet’s payload, avoiding to loose the packet. Popular codes to add this resilience are Reed-Salomon Codes, Convolutional codes and Turbo codes.

However, to be able to recover a packet with corrupted bits, the packet must be detected by a successful synchronization, only the payload must be affected by erasure and only to an extend that is recoverable by the used code (described by the Hamming Distance of that code). All other packets are dropped.

Recent research [1] shows that certain wireless communication systems (among those for example Nanotron Chirp-mode systems) are very vulnerable to errors in preambles, especially in harsh environments like industrial automation. We focus on low-latency communication in industrial environments, where communication systems with stringent synchronization requirements such as UWB systems [2] that need to keep preambles short in order to reduce latency will be vulnerable to failed packet recognition.

The remainder of the paper is organized as follows. Section II describes the proposed method to increase redundancy. In section III, we describe the used simulator to prove our assumptions. The results of the simulations are presented in Section IV. Finally, section V discusses our results and shows possible future work.

II. METHOD

We propose a method for combating packet loss with network coding to improve transmission resilience, additionally to applying FEC codes to payload bits. We add an additional packet to protect a burst of packets.

This method is inspired by a RAID 3 hard disc configuration. In those configurations, an array of discs uses one disc to store parity information of the payload on all other discs, which means each bit of this disc serves as a checksum across the same bit on all other discs. So if one disc fails, the stored data can be restored using all remaining discs and the parity information on the last disc. In other words, the last disc contains the output of a level-1 FEC code to protect all other discs from a single disc failure.

We use a very similar approach to protect a burst of packets by appending a single packet that contains information of all packets in the burst to recover a single packet drop. To achieve this, we use a level-1 coding operation (such as parity or in our case xor) across all packets to create the last packet that we call a “control packet”. S. Katti et al. showed in [3], [4] that when n packets are combined using an xor operation, the knowledge of n-1 of them is sufficient to decode the remaining one. This is a simple form of a technique known as network coding [5], [6], where network packets are treated as numbers in algebraic calculations. Error-Correction methods on packet level are also referred to as erasure codes (see also [7]). In our
case we create a vector \( P_x \), representing our control packet, through encoding the packets of the burst:

\[
P_x = P_1 \oplus P_2 \oplus ... \oplus P_n
\]  

(1)

where \( P \) denotes the packets in the burst and \( \oplus \) denotes an xor operation.

Upon receiving a burst of packets, the receiver can use the control packet to restore one lost or dropped packet. Let one single packet \( P_x \) be lost during the transmission (meaning it is either dropped because of unrecoverable bit errors or was not received at all because of synchronization problems). Then the receiver can use all remaining data packets of that burst and \( P_x \) to recover \( P_x \).

A possible implementation could be as follows. To be able to recover one packet, the receiver uses a cache that is the size of one packet, which we will call \( C \). At the start of each burst, this \( C \) is filled with zeros. Upon each successful reception, the received packet gets xored against that cache (2). Receiving \( P_C \), the receiver can restore the lost packet \( P_x \) by xoring \( C \) and \( P_C \) (3).

\[
C = P_1 \oplus P_2 \oplus ... \oplus P_{X-1} \oplus P_{X+1} \oplus ... P_n
\]

(2)

\[
P_x = C \oplus P_C
\]

(3)

The receiver then sends a special ACK packet that we call a control packet acknowledgment (CPA). This CPA is used by the sender as a cumulative acknowledgment over the whole burst to determine which of the packets arrived. This is used to determine the link quality and to selectively resend missed packets.

Additionally, the sender can adjust the resilience to the channel quality, depending on the CPA.

If the channel quality is good, the sender can for example increase the burst size. This reduces overhead for sending resp. waiting for an ACK packet which improves performance if only few packet losses occur. However, packet loss (resp. a lost ACK packet) leads to retransmission of the whole burst. To avoid this, we use a technique called “sliding window”, which adapts the size of said packet burst to the occurred errors, thereby adapting to the channel condition (small burst size for weak channel conditions, larger for better channel conditions). In case of lost packets, the burst size can remain the same. If too many packets are lost, then the sender might even decrease the burst size.

Additionally, the transmission resilience can be increased by changing the packet parameters. For UWB this could mean increasing the number of preamble repetitions to increase synchronization probability and therefore avoid packet losses based on synchronization failure. To avoid packet losses because of too many corrupted bits, the FEC and modulation can be adapted.

III. SIMULATION

We implemented a simulator in C++ to prove our concept using Monte Carlo simulations. This simulator sends and receives packets as described above over a slotted-time link through a channel with a time-independent, fixed Packet Error Rate. Each packet (including ACK packets) take up one whole time slot. The channel drops single packets with a constant probability according to the Packet Error Rate.

The simulator has two operation schemes. One is called *stream mode*, mimicking a UDP (synchronous) transfer, counting just the number of packets transmitted, opposed to an *ARQ mode* that tries to deliver a complete message, thus using acknowledgments and resends and counting the number of packets that have to be sent.

In stream mode, the sender sends out bursts of packets, where the receiver tries to recover losses using the coding packet and counts the received packets. The result for stream mode simulation is the count of successfully delivered packets. In ARQ mode, the receiver tries to recover lost packets using the control packet (if the control packet survived). Then the receiver sends an acknowledgment packet, indicating the last received packet and eventually the known packet losses (according to the control packet). The sender can then resend either the indicated lost packets (selective repeat) or send a burst packets starting one after the last well-received packet. For non-coding transmission, the receiver acknowledges every packet. The sender then resends every unacknowledged packet.

The result for ARQ mode is the completion time, in other words the count of packets (and therefore time slots) that needed to be sent.

The simulator runs simulations for error rates from 0% up to 99% packet loss for both stream and ARQ modes. The message to deliver always consists of 1000 packages. Each simulation is run 50 times, and the mean value of the results is taken.

IV. RESULTS

The simulations proved that the proposed coding scheme increases the number of successfully delivered packets, as Figure 1 shows.

Figure 1. Successful receptions over error rate

However, the scheme causes a certain overhead, as more packets have to be sent. This is visible in the ARQ mode, where the scheme significantly reduces the completion time for channels with a low to moderate packet error rate, as Figure 2 shows. However for channels with a high error rate, the initial adaption of the sliding window causes an overhead that causes the completion time with coding to be higher than the completion time without coding.

Figure 2. Completion times using ARQ with and without coding

We discovered that the efficiency barrier for this coding scheme is roughly between 35% and 40% packet errors, see Fig.3. This means that as long as 2/3 of the packets can be received, this scheme will perform better than traditional ARQ.

Figure 3. Detail of ARQ Completion
V. DISCUSSION

We proposed a novel scheme of packet-based error correction and proved by simulation its advantage over the traditional approach. Especially for networks with a high risk of losing complete packets, as for example UWB IR networks, this scheme shows advantages. Due to the fact that this code protects one packet in a burst of packets, this scheme performs best for channels with an overall packet error rate lower than 35-40%. Note that this denotes the PER after payload decoding, meaning packets with too many corrupted bits than the payload FEC can cover.

A topic for future research is the initial burst length and its adaption. We implemented an initial burst length that is adapted linearly in both directions. The TCP protocol, which inspired our “sliding window” technique, uses an adaption scheme called linear increase, multiple decrease (LMD), which reacts more sensitive to failed transmissions. For some setups, the scheme might perform better using this kind of adaption strategy, or even adapt by multiplication for both increases and decreases. Also the initial burst length is fixed in our simulations. It could be beneficial to fit this initial burst length to the channel conditions by sensing or sounding the channel.

Another point for future work is how the packet’s own resilience can be increased according to channel condition described the CPA, e.g. by choosing more preamble repetitions in IEEE 802.15.4a [2] or increasing the payload FEC and therefore adding resilience to that transmission.

As our results show, this scheme can be beneficial for time-constraint or even real-time applications, where the deterministic overhead of added control packets protects against lost packets or the undeterministic delay of resending packets. But even without the need for determinism, this scheme outperforms ARQ in terms of completion time, except under worst channel conditions.

REFERENCES


