IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received July 7, 2016, accepted July 16, 2016, date of publication July 20, 2016, date of current version August 26, 2016.

Digital Object Identifier 10.1109/ACCESS.2016.2593605

Protocols and Mechanisms to Recover Failed Packets in Wireless Networks: History and Evolution

SHERAZ ALI KHAN[1], MUHAMMAD MOOSA[2], FARHAN NAEEM[2], MUHAMMAD HAMAD ALIZAI[3], AND JONG-MYON KIM[4] (Member, IEEE)

[1]Embedded Ubiquitous Computing Systems Laboratory, Department of Electrical, Electronics, and Computer Engineering, University of Ulsan, Ulsan 44617, South Korea
[2]EmNets Laboratory, University of Engineering & Technology Peshawar, Peshawar 25120, Pakistan
[3]SBA School of Science and Engineering, Lahore University of Management Sciences, Lahore 54792, Pakistan
[4]University of Ulsan, Ulsan 44617, South Korea

Corresponding author: J.-M. Kim (jmkim07@ulsan.ac.kr)

This work was supported in part by the Korea Institute of Energy Technology Evaluation and Planning within the Ministry of Trade, Industry, and Energy, Republic of Korea, under Grant 20162220100050 and in part by the National Research Foundation of Korea within the Ministry of Science, ICT and Future Planning through the Leading Human Resource Training Program of Regional Neo Industry under Grant NRF-2016H1D5A1910564.

ABSTRACT The emergence of multihop wireless networks and the increase in low-latency demands of error tolerant applications, such as voice over internet protocol, have triggered the development of new protocols and mechanisms for recovering failed packets. For example, recovering partially corrupt packets instead of retransmission has emerged as an effective way to improve key network performance metrics, such as goodput, latency, and energy consumption. A number of similar and interesting solutions have been proposed recently to either reconstruct or process corrupt packets on wireless networks. The proliferation of multimedia services on 3G and long term evolution networks, and the stringent quality of service requirements for these applications have given birth to robust codes and new error tolerant mechanisms for packet delivery. Despite years of active research in the field, we lack a comprehensive survey that summarizes recent developments in this area and highlights avenues with potential for future growth. This survey tries to fill in this void by providing a comprehensive review of the evolution of this field and underscoring areas for future research.

INDEX TERMS Wireless networks, packet recovery, error tolerant, 3G, LTE, internet of things.

LIST OF ACRONYMS

3G	3rd Generation
ACK	Acknowledgment
ACR	Adaptive Coding Rate
AIR	Adaptive Incremental Redundancy
AL-FEC	Application Level FEC
AMR-WB	Adaptive Multi-Rate Wideband
AP	Access Point
API	Application Programming Interface
ARQ	Automatic Repeat Request
ATSC	Advance Television Systems Committee
BER	Bit Error Rate
BSD	Berkeley Software Distribution
CD	Compact Disc
CN	Corruption Notification
CPC	Complimentary Punctured Convolutional
CPR	Corrupt Packets Recycling
CPS	Cyber Physical Systems
CRC	Cyclic Redundancy Check
CRTP	Compressed RTP
CSMA/CA	Collision Sense Multiple Access with Collision Avoidance
CTCP	Compressed TCP
CTP	Collection Tree Protocol
CTS	Clear to Send
DMA	Digital Media Adapter
DSL	Digital Subscriber Loop/Line
DVB	Digital Video Broadcasting
DVB-S2	Digital Video Broadcasting-Satellite 2
DVD	Digital Video Disc
EARQ	Extended ARQ

ECC	Error Correction Code
EEC	Error Estimation Codes
ESP	Encapsulating Security Payload
FEC	Forward Error Correction
HARQ	Hybrid ARQ
HSDPA	High Speed Downlink Packet Access
HSUPA	High Speed Uplink Packet Access
ICMP	Internet Control Message Protocol
IETF	Internet Engineering Task Force
IGMP	Internet Group Management Protocol
IoT	Internet of Things
IP	Internet Protocol
IPv4	Internet Protocol Version 4
IPv6	Internet Protocol Version 6
IR	Incremental Redundancy
ITU-T G.hn	International Telecommunication Union Telecommunication Standardization Sector Gigabit Home Networking
LAN	Local Area Network
LDPC	Low Density Parity Check Codes
LT	Luby Transform
LTE	Long-Term Evolution
MAC	Media Access Control
MANET	Mobile Adhoc Networks
MBMS	Multimedia Broadcast and Multicast Services
MDPC	Multi Dimensional Parity Check Code
MMS	Multimedia Messaging Service
MRD	Multiple Radio Diversity
MRQ	ARQ with Memory
NACK	Negative Acknowlegment
NAK	Negative Acknowledgment
NASA	National Aeronautics and Space Administration
PDU	Protocol Data Unit
PHY	Physical
PP-ARQ	Partial Packet ARQ
PPR	Partial Packet Recovery
QoS	Quality of Service
RAM	Random Access Memory
RBAR	Receiver-Based AutoRate
RCPC	Rate Compatible Punctured Convolutional
ROHC	Robust Header Compression
RS	Reed-Solomon
RTMA	Real Time Multimedia Applications
RTP	Real-time Transport Protocol
RTS	Request to Send
SIP	Session Initiation Protocol
SNR	Signal-to-Noise Ratio
SPaC	Simple Packet Combining
TCP	Transmission Control Protocol
TTL	Time to Live
TVA	Transmit-Verify-Acknowledge
UART	Universal Asynchronous Receiver Transmitter
UDP	User Datagram Protocol
VOIP	Voice Over Internet Protocol
VOIP	Voice Over Internet Protocol
WiMAX	Worldwide Interoperability for Microwave Access
WLAN	Wireless LAN
WMN	Wireless Mesh Networks
WSN	Wireless Sensor Networks

I. INTRODUCTION

Wireless networks present challenging network conditions. In comparison to wired links, wireless links are generally more susceptible to *packet failure*. The problem is further aggravated in multihop wireless networks such as wireless sensor networks (WSN), wireless mesh networks (WMN) and mobile ad hoc networks (MANETs) where a packet has to traverse multiple wireless links before it reaches its intended destination [1], [2].

There are two main facets of packet failure: (i) *packet corruption* — packet is received albeit with bit-errors, and (ii) *packet loss* — packet is not received at all and is lost at PHY layer, e.g., due to strong interference. Traditionally, both these aspects of packet failure have been treated synonymously, i.e. retransmit the packet whenever a failure[1] event occurs. However, over the past decade, research on multihop wireless networks has shown that packet retransmission is not the most attractive way to salvage failed packets either due to *corruption* or *loss* [3], [4]. Depending upon network conditions, which vary across different deployments, retransmitting failed packets could degrade network performance metrics such as throughput, delay and energy consumption.

To cope with this peculiar problem in wireless networks, an assortment of packet recovery techniques have been developed over the past several years, which try to discern packet corruption from packet loss. These techniques try to recover corrupt packets locally at the receiver without requiring their complete retransmission by the sender. The packet recovery mechanisms include packet reconstruction using forward error checksums (FEC) [5], combining multiple erroneous copies of the same packet [6], or even completely disregarding bit-errors [4] for error tolerant applications.

In this survey, we try to revisit the history and evolution of this field: *protocols and mechanisms to recover failed packets*. This evolution has been triggered mainly by the growing number of error tolerant applications, such as voice and video, which are more sensitive to network delays than to packet failures. For this purpose we have broadly categorized this evolution into three main stages.

- **Retransmission - zero tolerance for errors:** This is the traditional mechanism for packet recovery in both wired and wireless networks. At this stage of its evolution, the protocol stack was treated as error sensitive assuming that wireless protocols cannot handle errors in packets. Hence, when failed, a packet must be retransmitted by

[1] Throughout the paper, packet failure refers to unsuccessful delivery either due to packet loss or due to packet corruption.

the sender. A failure is typically detected when the packet is not acknowledged within a certain time period. We discuss this phase of evolution and different techniques employed for packet recovery in Section II.

- **Error Correction - when retransmissions are too expensive:** In challenging network conditions, typically observed in multihop wireless networks, retransmissions are very expensive in terms of energy and bandwidth. In such networks, error correction techniques are employed at different layers of the network stack. Hence, at this stage, although the protocols are still considered error sensitive, i.e., they cannot process erroneous packets as is, but are rather enabled to recover partially corrupt packets at the receiver before processing. We discuss variants of these techniques employed at different layers of the stack in Section III.
- **Error tolerance: accepting erroneous packets while disregarding errors:** Finally, error tolerant protocols have been developed that can process erroneous packets. The need for such protocols is driven by the proliferation of multimedia and real time applications that can tolerate errors but require timely delivery of information. Hence, protocols leading this stage of evolution primarily cater to the needs of error tolerant applications. UDP-Lite is the prime example of an error tolerant protocol that has been standardized by the IETF. We discuss UDP-Lite and other error tolerant approaches in Section IV.

We conclude our discussion in Section V by providing a tabular categorization of the existing approaches before highlighting possible directions for future research in this field.

II. RETRANSMISSION: ZERO TOLERANCE FOR ERRORS

Packet retransmission [7] is the classical approach to recover failed packets in a medium where packet errors are very common due to interference and collisions. At this stage of its evolution, the protocol stack is treated as an error-sensitive entity, which cannot handle packet failures. Hence, the sender must always retransmit a packet if it is not acknowledged within a certain time period. This technique, more commonly known as ARQ (Automatic Repeat reQuest) is a fundamental method employed in many protocols with slight variations in its behavior, depending upon the underlying network conditions, to achieve reliable data transfer. Although different flavors of ARQ exist, the fundamental idea remains the same: sender retransmits a packet based on the acknowledgments (positive or negative) from the receiver. Hence, the responsibility of delivering a correct packet solely rests with the sender based on the feedback from the receiver.

Packet retransmission relies on strong error detection techniques at the receiver that provide necessary feedback to the sender. These techniques employ algorithms called checksums to verify the integrity of a received packet. A correctly received packet is positively acknowledged (ACK) whereas packet failure is either negatively acknowledged by the receiver (NACK) or assumed by the sender upon the expiry of a retransmission timer when no ACK/NACK is received. This mechanism mandates the appropriate selection of a re-transmission period to avoid contention on the medium and efficiently utilize bandwidth.

In the ensuing text, we describe algorithms employed to detect packet errors that are introduced during their transmission. In this section we revisit the traditional mechanisms for error detection in order to formulate the background essential for the remaining discussion in the paper. We conclude this section by presenting *ARQ and its kin*.

A. CHECKSUMS: ERROR DETECTION BEFORE RETRANSMISSION

Checksums are small datum computed over blocks of data that are appended to the original data and used for detecting errors introduced during transmission. The procedure through which a checksum is calculated is called checksum function. Checksums usually provide for simple data integrity checks, designed particularly for low cost computations yet with some caveats, i.e., some errors may completely elude the check. Simple examples of checksum functions include, summation, 1's complement, 2's complement and other logical operations such as XOR. To understand how checksums operate, we discuss two of the basic checksums, i.e., parity bit and parity word, followed by standard checksums that are in vogue today.

Parity Bit (also known as *check bit*) is a very basic checksum and can only detect single bit errors. It works by appending a single bit to a block of data depending upon the number of "1" bits in the data. The appended bit is either a 0 or 1 and is so chosen to keep the total number of "1" bits in the data and checksum even, in an even parity system, or odd in an odd parity system. The mechanism is simple to implement as it only requires the XOR of all the bits in a block of data. However, it may result in false-positives when multiple bits in the data are in error.

Parity Word is an improved adaptation of the parity bit checksum and it extends the parity bit error detection mechanism to large blocks of data, which are broken down into a fixed number of n-bit words. For each word the XOR of its n-bits is calculated and the result is stored into a parity word, which is appended and transmitted along with the data. The receiver, upon receiving the data computes the XOR of all the words including the *parity word*. An erroneous transmission is detected if the result of the XOR operation is not n zeros.

IP checksum is among the numerous error detection mechanisms that the internet protocol suite has adopted at its various layers to ensure reliable data transfer over communication networks. These mechanisms range from CRC at lower layers of the IP protocol stack to IP and TCP checksums at its higher layers, each making up for the limitation of the other. IP checksum [8] is used to ensure the integrity of IPv4 headers and is calculated by determining the 16-bit one's

complement of the one's complement sum[2] of all the 16-bit words in an IPv4 header. This checksum does not cover TCP data or header, hence there is a similar checksum in TCP header that covers all the 16-bit words in the TCP header, the IP addresses of source and destination, the protocol value field and the length of the TCP segment.

IP and TCP checksums fare well in detecting multi-bit errors, however they cannot detect common errors like reordering of bytes,[3] insertion of zeros, deletion of zeros or multiple errors that can sum to zero. These weaknesses have been addressed in Adler's checksum [9], Fletcher's checksum [9], [10] and CRC [11] by considering the positions of bits in the sequence. However, this robustness comes at the cost of increased computational complexity.

Fletcher's checksum [10] is a position-dependent checksum and was proposed by John Fletcher in the late 1970s. Its error-detection capabilities are comparable to CRC, and it requires much less in computational effort, hence making it a useful alternative to CRC. Fletcher's checksum is sensitive to the order of blocks (bytes) in the data word over which it is calculated, and hence it can detect any error in the reordering of bytes. This is achieved by calculating two checksum values, i.e. S_1 and S_2. The first checksum, S_1, is the modular sum of all the data blocks, i.e., D_0 to D_N, into which the data word has been divided. The second checksum, S_2, is the sum of all the values taken by the first checksum, S_1, as each block of the data, D_i for $i = 1 \ldots N$, is added to it. Both sums, S_1 and S_2 are initialized to zero and use the same modulus i.e. modulo 255 for 8-bit blocks and modulo 65, 535 for 16-bit blocks. The Fletcher checksum is formed by concatenating S_1 and S_2. Fletcher checksum gets its sensitivity for the order of data blocks due to the accumulation of S_2.

The size of the Fletcher checksum is dependent upon the size of the data block. An 8-bit data block results in two 8-bit checksums combined into a 16-bit Fletcher checksum. Similarly, 16-bit and 32-bit data blocks result in 32-bit and 64-bit Fletcher checksums, respectively.

Adler's Checksum employs the same algorithm as Fletcher's checksum but ostensibly improves its effectiveness by selecting a prime number modulus for both the checksums [9]. The use of a prime number modulus is purported to ensure that even the least detectable patterns, which dominate the overall performance of an error detection technique, are detected. However, studies have demonstrated [12] that contrary to popular belief Fletcher's checksum is more effective than Adler's checksum in all cases except Adler-16 over short data word lengths. This slight improvement in error detection of Adler-16 comes at a relatively higher computation cost due to its usage of a prime number modulus.

Cyclic Redundancy Check or simply CRC was initially proposed in 1961 by Peterson and Brown [13]. It is a polynomial based arithmetic block code, an entirely different technique in comparison to simple checksums due to its complexity and robustness. CRC has many variants and it usually falls into a category of its own. Although computationally expensive, CRCs are easy to implement and analyze, and can successfully detect common transmission errors [11].

CRC is based on cyclic codes, which are linear blocks of code wherein the cyclic shift of each codeword produces another codeword [14]. Cyclic codes are usually generated using shift registers. The cyclic shift of any codeword $w(x)$ by i positions is also a codeword:

$$w^{(i)}(x) = x^i w(x) mod(x^n - 1) \quad (1)$$

Likewise, as mentioned earlier these cyclic codes are linear, therefore the sum of their shifts are also codewords. Hence for any polynomial $p(x) = p_0 + p_1 x + ... + p_m x^m$, the polynomial $p(x)w(x)$ is a codeword:

$$p(x)w(x) mod(x^n - 1) = \sum_{i=0}^{m} (p_i x^i w(x) mod(x^n - 1)) \quad (2)$$

The CRC algorithm works by inserting redundant check-bits into each data packet to be transmitted. These check-bits are remainders of polynomial division, of base 2, performed over blocks of data. A polynomial in CRC refers to the bits representing the divisor. For example the divisor bits 1011 represent the polynomial x^3+x+1. A polynomial upon which both the sender and receiver agree is called the *generator polynomial*, which is then used to generate the CRC check-bits for the data to be transmitted. Many types of CRC are standardized based on the generator polynomial. The most common ones are summarized in table 1.

TABLE 1. Common CRC standards.

Name	No. of bits	Polynomial
CRC-8	9	$x^8 + x^7 + x^6 + x^4 + x^2 + 1$
CRC-16	17	$x^{16} + x^{15} + x^{11} + x^9 + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$
CRC-32	33	$x^{32} + x^{26} + x^{23} + x^{22} + x^{16} + x^{12} + x^{11} + x^{10} + x^8 + x^7 + x^5 + x^4 + x^2 + x + 1$
CRC-64	65	$x^{64} + x^{62} + x^{57} + x^{55} + x^{53} + x^{52} + x^{47} + x^{46} + x^{45} + x^{40} + x^{39} + x^{38} + x^{37} + x^{35} + x^{33} + x^{32} + x^{31} + x^{29} + x^{27} + x^{24} + x^{23} + x^{22} + x^{21} + x^{19} + x^{17} + x^{13} + x^{12} + x^{10} + x^9 + x^7 + x^4 + x + 1$

Whether wired or wireless, studies [15]–[17] have shown that the error detection and correction capabilities of CRC are unaffected by the medium of transmission. CRC, on account of its exceptional reliability in error detection, is the most widely used mechanism for verifying data integrity in communication networks. Most of the protocols in the TCP/IP stack such as IP, ICMP, IGMP, UDP, and TCP use CRC algorithms to detect errors.

Mandel and Mache [15], [16] have shown that CRC generator polynomials can be used to detect not just simple single-bit errors but also *burst errors*. They further show that the

[2]A one's complement sum is obtained by adding the carry to the 16-bit sum of all the 16-bit words.

[3]Since the checksum is basically a 16-bit sum, it doesn't change with the order of the bytes.

limitation of CRC, i.e., being only able to detect errors and not correct them, lies in the improper selection of generator polynomial for a particular network environment. An appropriate generator polynomial can be selected for a given network environment by studying network properties like channel condition and error patterns. They proposed a new protocol, called TVA (Transmit-Verify-Acknowledge) [17], to practically demonstrate the error correction capabilities of CRC, which selects the generator polynomial based on properties of the network.

Granted, CRC is capable of correcting single bit errors or more, nevertheless error detection remains its primary function. CRC, in conjunction with IP and TCP checksums, forms the mainstay of the TCP/IP protocol suite when it comes to the correct identification of corrupt packets. Higher layer checksums (such as IP and TCP checksums) help detect any data corruption introduced by intermediate devices (e.g. routers as they work at the IP layer) or end hosts whereas CRC is used to verify data integrity at lower layers during local media transmission (e.g. ethernet switches employ CRC as they work only at lower layers). It is quite obvious that checksums are used as secondary checks on top of CRC but they have their own utility and help catch software errors introduced in buffer mismanagement or even hardware errors introduced by network adapters and DMA, which are very common.

Despite their limitations, experimental studies suggest that IP and TCP checksums are adequate for IP header integrity [18]. Notwithstanding these experimental findings IP checksum is considered superfluous and its need debated for long in the presence of other checksums, especially when at times CRC and IP checksums do not agree [19] thereby causing issues. The IP checksum was eventually deemed unnecessary and abandoned in IPv6, arguing that TCP checksums combined with link layer CRC are adequate, and provide sufficient data integrity checks during transmission.

B. ARQ: AND ITS KIN

ARQ is a timer based retransmission scheme and by far the most widely used mechanism to recover failed packets. Its simplest version is programmed at the link layer. In this scheme the sender waits for feedback from the receiver for a certain time duration before retransmitting a packet. This feedback can either be an ACK or NACK to inform the sender whether the packet was correctly received or otherwise, respectively.

The most notable use of ARQ is in the sliding window protocol in TCP [20] and in IEEE 802.11 and 802.15.4 based wireless data links for in-order and reliable delivery of packets. In these schemes, packets (or frames) are assigned sequence numbers allowing the receiver to ACK correctly received packets and identify the missing ones. The receiver is required to acknowledge all packets (or frames) that it receives. The sender and receiver can both manage buffer overrun by controlling the flow of data using the protocol's window function. Sliding window protocol in TCP makes use of three types of ARQ.

Stop-and-Wait ARQ is the simplest form of sliding window protocol where the size of both transmission and receiving windows is set to one. Sender sends one packet (or frame) at a time and waits for its acknowledgement by the receiver. The packet (or frame) is resent if its ACK is not received from the receiver within a certain time duration. The stop-and-wait ARQ scheme has two main problems.

First, duplicate frames cannot be recognized. An example scenario can be when the sender doesn't receive the ACK, it times out, and sends the frame again. However, the receiver cannot determine whether this new frame is the next frame in the order or a duplicate. To resolve this problem, a one-bit sequence number is added to each frame that indicates the next expected frame.

Second, a sender can receive two ACKs for the same frame whereas it should get only one as the stop-and-wait ARQ allows for only one outstanding (un-ACKed) frame i.e. the sender waits for an ACK before sending the next frame. This might be the result of excessive latency in the transmission medium causing the sender to resend a frame before its ACK is received. This would result in the receiver having two copies of the same frame and sending two ACKs to the sender, who is expecting only a single ACK.

Go-Back-N ARQ allows the sender to send multiple frames without waiting for an acknowledgement. The number of N outstanding frames in Go-Back-N ARQ is determined by the window size between the sender and the receiver. In other words, when compared to stop-and-wait ARQ, Go-Back-N ARQ uses a transmission window of size N and a receiving window of size 1.

The Go-Back-N ARQ is inefficient on links that are more susceptible to packet loss as the receiver won't accept any out of order frame and would cause the re-transmission of all the frames even if a single frame is lost. Other than efficiency issues on some links, this version of ARQ can result in a predicament for the transmitter. An example scenario would be when the transmitter sends all the frames in the transmit window without waiting for any acknowledgement. Once the frames have been transmitted, any acknowledgement after that must contain enough information to determine which frames have been received correctly and which haven't. Unfortunately for the transmitter, these acknowledgements do not provide such information, hence the predicament.

Selective-Repeat ARQ is the most general and widely used form of sliding window protocol where both transmission and receiving windows are of size N. The transmitter sends frames according to a pre-specified window size without waiting for ACKs of individual frames. The receiver can request the sender to resend selective frames from its window and it should be capable enough to store frame that are delivered out of order. The receiver can reject individual frames without requiring the transmitter to resend all the frames as in Go-Back ARQ, thus making this form of ARQ well suited for links with low reliability.

The utility of ARQ stems further from the fact that it has been used in combination with (or as a backup for) other error detection and correction schemes such as EARQ, HARQ and MRQ. We defer a more detailed discussion on HARQ and its variants to Section III-D and EARQ and MRQ to Section III-C.

III. ERROR CORRECTION: WHEN RETRANSMISSIONS ARE TOO EXPENSIVE

Retransmission of a packet is a rudimentary yet reliable technique for recovering from packet failure but it has several disadvantages. Real-time services and applications requiring low latency cannot use ARQ schemes because it becomes too late for the data to be useful by the time the sender retransmits it. In some cases, the original data is no longer available to be transmitted if errors occur after transmission. Moreover, in battery-driven wireless devices, retransmission can be very expensive both in terms of energy and bandwidth. Thus it is but a natural progression in the evolution of packet recovery techniques that we recover the corrupt packets. The need for error correction mechanisms is rooted in systems and applications where transmitter immediately forgets the data it sent or where there is no back-channel (simplex communication: TV or radio broadcasting) to send feedback to the transmitter in order to use ARQ.

Error Correction Code (ECC) — also known as Forward Error Correction (FEC) or Channel Coding — is a technique used to add redundant data to a packet that can be used by the receiver to correct errors introduced during transmission. Hamming [21] pioneered this field by introducing hamming codes to correct errors. Majority of error correction techniques work in tandem with other basic forms of packet recovery methods, such as ARQ, to ensure error-free delivery of packets. For example, Hybrid-ARQ combines FEC and ARQ, i.e., packet retransmission is requested only when FEC fails to recover it.

In the last few years, besides error correction codes, several other methods have emerged for recovery of corrupt packets ranging from the simple ones, which combine multiple corrupted copies of a packet (Section III-C) to the more complex ones, which use confidence values from physical layer (Section III-B.2). In the following text, we discuss a variety of these error correction mechanisms.

A. ERROR CORRECTION CODES

Claude Shannon's paper "A Mathematical Theory of Communication" [22], published in 1948, marked the inception of the field of *information theory* (also called coding theory or channel coding). Shannon's work focused on finding out ways to encode information to be sent over a noisy channel that could corrupt it. He proved in his publication that reliable communication is possible over such channels only if the transmitted information is sent at a rate, which is less than the channel's capacity. Though Shannon established the existence of such codes, which could reliably transmit data over data corrupting communication channels, he did not propose explicit guide on how to achieve such a feat.

The purpose of channel coding stretches from finding codes to transmit information quickly and reliably to ones, which can detect and correct errors in the transmitted information. In this section we focus on error correction codes [23] also known as *erasure codes*, which have long been used in wireless networks for both error detection and correction. The central idea behind error correction codes is to add redundant bits to the original data, using some algorithm, which can be useful in reconstructing or recovering the original data.

Richard Hamming invented the hamming code [21], which paved the way for the evolution of more efficient codes. FEC [5], [24] is a good example of an error correction code, which is extensively used in real-time wireless applications [25] and wireless networks [26].

A key measure of efficiently encoding blocks of data using correction codes is *information rate* or *code rate*. In simple words, code rate is the original data (non-redundant) divided by total data (original data plus redundancy). For example, in case of Hamming(3,1), the total number of transmitted bits are 3, whereas original data is only 1 bit, which means that the code rate for hamming(3,1) is $1/3 \approx 0.333$. This means that only 33% of the total block is useful information, rest is redundant data inserted for corrections lest the data gets corrupted during transmission.

Correction codes can be generalized into two categories in terms of their evolution over time;

- *Simple codes*, which have low computational overhead but a higher communication overhead, which makes them suitable only for wired communication with dedicated channel between transmitter and receiver e.g. Universal Asynchronous Receiver Transmitter (UART).
- *Efficient codes*, which use complex algorithms that entail a high computational cost but their low transmission overhead means higher data rates rendering them more suitable for wireless communication.

1) SIMPLE CODES

The simplest error correction code is the *repetition code*. As the name suggests, repetition code is a multiple repetition of message (or bits) over a channel. The receiver can then reconstruct the correct message either by using the majority-vote over each bit or simply selecting a message that occurs most often. Repetition codes are very simple to implement but introduce a very high communication overhead, which is highly undesirable in bandwidth restricted wireless communications. Moreover, the assumption that only a few copies of a message will be corrupted is not entirely valid in wireless networks.

A more logical and refined way of coding is the multi-dimensional parity-check code (MDPC). It is more robust and provides higher data rates at almost no extra computational cost in comparison to repetition code. It operates by arranging the message into a matrix; rows and columns, and calculates a parity digit for each row and column. For example, using this

scheme the original message 1234, is transmitted as follows:

$$\begin{bmatrix} 1 & 2 \\ 3 & 4 \end{bmatrix} \tag{3}$$

To the original message in Eq. 3 parity digits are added by summing rows and columns as given below:

$$\begin{bmatrix} 1 & 2 & \mathbf{3} \\ 3 & 4 & \mathbf{7} \\ \mathbf{4} & \mathbf{6} & \end{bmatrix} \tag{4}$$

Thus for a 4 digit message "1234", an 8 digit encoded message is sent as "12334746". The receiver rearranges the message into a matrix and can detect and correct single digit errors by adding up the rows and columns.

Hamming codes, a family of error correction codes, succeed the repetition code. Hamming(7,4) or just Hamming refers to the first error correction code of this family, which was invented by Richard W. Hamming in 1950 [21]. It can correct single bit errors whereas detect 2 bit errors. (7,4) implies that the codeword (or a block of message) is 7 bits long of which 4 bits are original data and 3 bits are redundant. This nomenclature for codes was also devised by Hamming and it has become a standard for other error correction codes as well. Redundant bits are used to create a minimal Hamming distance[4] of 3 bits. This means that each codeword has a 3 bit Hamming distance from the other and the received bits can be corrected successfully, if they are not altered by a Hamming distance greater than 2.

The Hamming code works by creating a set of 3 parity bits for 4 bits of data. These 3 redundant parity bits help in detecting and correcting single bit errors in the data. The addition of a further overall parity bit ensures the integrity of the 3 redundant parity bits. Table 2 shows how parity bits overlap each other and the check data bits. For example, p_3 covers even parity for bits 4, 5, 6 and 7 of which bit no. 4 is p_3 itself. Further more, from perspective of column, it is evident that Bit no. 6 is covered by p_2 and p_3 but not by p_1.

TABLE 2. Hamming code with (7,4) repetition.

Bit #	1	2	3	4	5	6	7
Transmitted Bit	p_1	p_2	d_1	p_3	d_2	d_3	d_4
p_1	Yes	No	Yes	No	Yes	No	Yes
p_2	No	Yes	Yes	No	No	Yes	Yes
p_3	No	No	No	Yes	Yes	Yes	Yes

Repetition codes, MDPC and Hamming codes are computationally very simple and highly desirable in digital transmissions where data is less prone to errors and bandwidth is not a major constraint e.g. UART and Random Access Memory (RAM). These codes typically offer single bit-error correction and double-bit error detection. However, these codes, because of very high redundancy, are not scalable and thus not a feasible option in wireless communication.

[4]Hamming distance refers to the minimal number of bits that need to be changed before reaching the next code word. The (3,1) repetition would mean, 3 bits need to be flipped before we reach the next codeword.

2) EFFICIENT CODES

Wireless channels are more noisy in comparison to their wired counterparts, hence they incur higher bit error rates and necessitate the use of a more evolved set of error correction codes for reliable data transfer. A number of efficient codes with lower communication overhead have been developed specifically for wireless communications. The algorithms for these codes are however very complex and their complete description is beyond the scope of discussion in this paper. In the following text we only present their generalized description for comparison.

Reed-Solomon Code: One of the first and most notable among efficient codes is the Reed-Solomon (RS) [27]. Reed-Solomon (RS) code, invented by S. Reed and Gustave Solomon in the 1960s, quickly found its applications in commercial consumer electronics such as CDs, DVDs, in data transmission technologies such as DSL, WiMAX and in video broadcast systems such as DVB (Digital Video Broadcasting) and ATSC (Advance Television Systems Committee). The commercial success and popularity of the RS code was a result of its unique encoding scheme, which enabled it to recover from both multiple random errors in symbols/blocks, and burst errors, where a sequence of symbols/blocks is lost.

The Reed-Solomon code views the data to be encoded as a polynomial. We know for a fact from elementary algebra that a polynomial of degree $k-1$ is uniquely determined by k distinct points. The polynomial uniquely determined by the data to be encoded (which form the coefficients of the polynomial) is transmitted using its samples (evaluations) at various values of x. The Reed-Solomon scheme ensures reliable data transfer by encoding k input numbers (the polynomial coefficients)

$$a_0, a_1, a_2, ..., a_{k-1}$$

into n numbers (samples or evaluations of the polynomial at n different values of x)

$$\alpha_0, \alpha_1, \alpha_2, ..., \alpha_{n-1}$$

for transmission, where $n \geq k$. The redundancy introduced by this scheme, for the reliable transmission of k numbers, is $n-k$. The receiver would be able to recover the original k input numbers (a_i), if it receives at least k encoded numbers (α_i), no matter which k encoded numbers (α_i) are received. Redundancy and reliability come naturally to RS codes i.e. to increase reliability we just need to send more (redundant) samples of the polynomial [28]. Generally an RS codeword is given as follows:

Data	Parity
k	2t

A popular Reed-Solomon code is the RS(255,223) with 8-bit symbols (denoted by s). Each codeword contains 255 (denoted b n) code word bytes, of which 223 (denoted by k) bytes are data and 32 bytes are parity. For this code, the decoder can correct any 16 (denoted by t where $2t = n-k$) symbol errors in the code word i.e. errors in up to 16 bytes anywhere in the codeword can be automatically corrected.

In many applications especially those involving deep space satellite communications such as NASA's Voyager mission, Reed Solomon codes have been used in concatenation with *convolutional codes* [29][5] Convolutional codes do not divide source data into blocks, instead read and transmit bits continuously. The transmitted bits are linear functions of past source bits, hence convolutional encoders require memory while encoding data for transmission. Convolutional codes are extensively used as error correcting codes for radio satellite communications [30].

The bit error rate improvement provided by RS code is phenomenal but it still falls short of Shannon's limit. In 1993, *Turbo codes* [31] were proposed by Claude Berrou, Alain Glavieux and Punya Thitimajshima with the potential of affording a bit error rate approaching Shannon's limit. Turbo codes are popularly used in deep space communication and other telecommunications technologies such as 3G mobile communication(cellular communication). The essence of turbo codes is in the use of data blocks. It sends three sub-blocks of bits: first sub-block being $m-$bits of payload, second sub-block is $n/2$ parity bits of payload and third sub-block is $n/2$ permutation of payload. The second and third sub-blocks are computed using recursive systematic convolution code (RSC code) [31]

LDPC (Low-Density Parity-Check codes) [32], [33] were invented by Robert Gallager in 1962 long before turbo codes. However, only recently LDPC codes are replacing turbo codes in many areas of digital communication. This is mainly because of the late implementation of LDPC codes for commercial applications. It provides highly efficient information transfer over noisy channels. LDPC codes are being used in a wide variety of communication applications ranging from the long distance satellite TV transmissions i.e. Digital Video Broadcasting-Satellite 2 (DVB-S2), to the short range home networking i.e. the ITU-T G.hn standard.[6] Variants of LDPC codes are also used in 10GBase-T Ethernet and in IEEE 802.11 protocol family as an optional part of 802.11n and 802.11ac standards.

Full explanation of LDPC is beyond the scope of this paper. We give here a short description of LDPC and briefly differentiate it from other codes. LDPC is a linear block code that is based on sparse bipartite graphs. It has a parity check matrix $\mathbf{H}$, with sparse rows and columns. The parity check matrix describes linear relations that components of codewords must satisfy. An LDPC code, where each row has the same weight j, which is the number of 1's in each row, and each column has the same weight k, which is the number of 1's in each column is called a *regular* Gallager code [30]. The LPDC parity check matrix must satisfy two conditions: $j \ll N$ and $k \ll M$, where N is the block length and M is the number of constraints on a single bit. This condition is only met when the matrix is sufficiently large. LDPC codes perform near Shannon's limit, if large block lengths are used.

Raptor Codes are an extended form of the Luby Transform codes or simply LT-codes, which were invented by Michael Luby in 1998 [34]. They have been adopted into different standards by the IETF [35], [36] and the Long Term Evolution (LTE) for multimedia broadcast and multicast services (MBMS) [37]. Raptor being a *fountain code*, has the property that its encoder can generate as many encoding symbols as are required from the given source symbols. Moreover, the decoder for Raptor codes, can theoretically recover all the original source symbols from any subset of the encoding symbols, provided the number of encoding symbols in the subset is either equal or slightly larger than the number of source symbols. Fountain codes can work in practice only if they have reasonably fast encoders and decoders, e.g., the encoder and decoder for Raptor code work in linear time. Moreover, the decoders for fountain codes must be able to recover the original symbols with reasonably high probability [34].

B. PARTIAL PACKET RECOVERY

In wireless networks the receiver tests a received packet for errors and usually discards it even if a small number of bits are corrupt in the packet; whereas, in most cases intermediate level of data corruption occurs [38]. This leads to more frequent retransmissions in wireless networks because the link or channel condition is not as reliable as in wired networks, which causes the protocol to be very inefficient. This inefficiency is avoided by exploiting partial packet recovery (PPR) in which a corrupt packet is not discarded immediately but rather used, for example, for correlating with the other retransmissions of the same packet if retransmissions also fail to provide a completely correct packet.

We divide partial packet recovery approaches into two classes i.e. PHY layer and link layer approaches. While PHY layer approaches require changes in the networking hardware, most link layer approaches mainly update the driver and relevant software.

1) PHY LAYER APPROACHES

After ECC, one of the most effective ways to detect and correct errors is at the PHY layer of the protocol stack. PHY layer provides a wealth of information and is the best place to recover corrupt packets, being the first layer to handle a packet at the receiver node. However, the majority of the techniques at this layer understandably require changes in the networking hardware, which is sometimes undesirable (e.g. compatibility issues) or not even possible (e.g. existing deployments).

Physical layer operates by grouping data bits into symbols and then modulating those symbols into a signal for transmission over the communication channel. These symbols at physical layer can be very useful in detecting and correcting corrupted bits. A number of techniques [39]–[42] have been developed to utilize these symbols in correcting erroneous

[5] A type of error correction code in which $m-bit$ symbols are encoded in $n-bit$ symbols, where m/n is code rate such that $n \geq m$.

[6] G.hn is an ITU telecommunication standard to enable high speed networking in homes for audio, video and data transmission over existing coaxial, telephone and power lines.

packets. Among these, PPR [39], SOFT [40] and ZigZag [41] are the most widely recognized solutions that try to utilize PHY layer information for correcting packets.

Partial Packet Recovery (PPR) [39] system identifies corrupt bits in a packet and then requests the retransmission of those corrupt bits only. This is contrary to other approaches, such as FEC, which requires a complete packet retransmission upon failure, thus wasting network capacity. PPR does so by introducing a Soft-PHY interface, which allows the receiver to determine the bits in a packet that are likely to be received correctly. Using this interface, the PHY layer informs upper layers about how closely the received decoded symbol matches the original symbol mapped by the PHY. The higher layers can use these hints from PHY layer to decide which bits are to be retransmitted. PPR further introduces the concept of *postamble* decoding to correctly decode bits from packets, whose preambles are corrupted due to noise or packet collisions. The idea is to replicate the preamble and header of a packet in a postamble and trailer of the packet, respectively. This allows the the receiver to correctly decode these portions of the packet. Finally PPR introduces PP-ARQ: a modified ARQ, which only requests the retransmission of those portions of a packet that are likely to be received incorrectly. Overall, PPR modifies the PHY layer (and the hardware) extensively to achieve the aforementioned functionality.

SOFT [40] uses the confidence measure of PHY layer's decision on each bit to recover corrupt packets. Existing PHY layers compute their confidence for each bit, however, this information is not utilized because of the lack of relevant interfaces between the PHY and link layers. The confidence measure of PHY layer alone is not enough for the higher layers to establish whether a particular bit is a '0' or a '1'. Hence, for uplink, SOFT exploits the deployment of overlapping access points to achieve these confidence measures from multiple access points. In this scheme, all the access points that overhear a packet transmission communicate their confidence measures to the intended access point over wired ethernet. Hence, the larger capacity of the wired ethernet is utilized to reduce wireless retransmissions. For downlink, a host can utilize the same confidence measures over multiple corrupt copies of the same packet. SOFT is a very efficient mechanism to overcome *dead spots* and significantly improve delivery rates in noisy environments.

ZigZag [41], [42] combats the hidden terminal problem [43]–[45] that results in successive collisions at the receiver in 802.11 networks. CSMA/CA[7] is a reliable technique to avoid collisions in 802.11 networks but fails in one particular scenario, where two nodes can communicate with a third one (Access Point), but cannot sense each other, this case is known as the *hidden node* or *hidden terminal problem*. The consequences of two nodes simultaneously sending packets to an access point, because both nodes are hidden from each other, are drastic; packets collide indefinitely, choking bandwidth and performance of the network.

ZigZag exploits two important characteristics of IEEE 802.11. *First*, an unnoticed collision of packets, transmitted by the hidden terminals tends to result in more collisions at the receiver. *Second*, since the sender jitters every transmission with a short random interval, successive collisions at the receiver start after a random number of interference-free bits. This asynchronous nature of retransmission is exploited by ZigZag and correct bits are collected, decoded and combined piece by piece from repeated collisions of the same packets. Consider, for example, an AP receiving collided packets from two hidden terminals. Both the senders will retransmit the packets resulting in a second collision. Due to IEEE 802.11 jitters, both these collisions will have a different starting offset. AP finds the chunk of bits that experienced interference in one collision but were correctly received in the other due to the random jitter. It uses this chunk (say chunk 1) to bootstrap its decoder. After decoding the correctly received chunk 1 with the normal decoder, it moves on to subtract this chunk from the second collision to compute chunk 2 and this process continues until both the packets are correctly decoded.

ZigZag achieves the same throughput as if the collided packets were received in separate time-slots. It only requires modification at the receiver and is thus useful in typical wireless LANs with multiple hosts connected to a single AP. The benefits of ZigZag are not bound to hidden terminal problem only, but extend to mesh networks, to avail more concurrent transmissions. ZigZag is backwards compatible because changes are only made in the AP and not in the sending nodes. It does not modify the MAC protocol and behaves as IEEE 802.11 in the absence of collisions, however when packets collide, ZigZag wakes up out of the blue, and decodes partially corrected packets. Hardware changes need to be made in the Access Points, because ZigZag makes heavy use of physical symbols in decoding the partially corrupted packets, which are not accessible in the hardware of conventional Access Points.

2) LINK LAYER APPROACHES

The majority of software based approaches for partial packet recovery operate at the link layer.

Maranello [3] proposes a block-checksum based retransmission mechanism for partial packet recovery. If a corrupt packet is received, the receiver computes checksums over blocks of consecutive bytes in that packet. These checksums are appended with a NACK to the sender. The sender recomputes checksums using the original copy of that packet. It then re-sends only those blocks for which the checksums do not match and repeats this partial retransmission until an ACK is received. The prime advantage of Maranello is that it does not introduce any additional overhead in the case of correct transmissions. The only overhead is the checksums that are appended with the NACKs of corrupt packets [46]. These appended checksums can increase the size of a Maranello-NACK upto 96 bytes as opposed to the standard

[7] A sender first senses the medium before sending anything and abstains from sending packets if the medium is busy.

14 bytes, hence resulting in larger airtime, which in turn may lead to collisions with frames from hidden terminals at some lower transmission rates [47]. Nevertheless, Maranello evaluation results [46] show significant improvement in throughput as compared to the standard IEEE 802.11. Although Maranello is compatible with current IEEE 802.11 based devices, since it observes the timing and back-off standards, its implementation on other platforms is impeded by the fact that Maranello NACK generation is a time critical operation, which requires firmware modification [47].

ZipTx [6] is another software based solution that modifies the wireless LAN card driver. Hence, similar to Maranello, it can also be deployed in existing IEEE 802.11 based WLANs. ZipTx disables the CRC functionality thus allowing the hardware to pass-on even the corrupt packets. For corrupt packets with low BER, ZipTx uses error correction codes to recover corrupt bits. Since error correction codes require twice as many symbols as the number of incorrect coding symbols, ZipTx recovers packets with high BER through retransmission. It uses known pilot bits in each packet to identify packets with high BER. This dual strategy allows ZipTx to gain significant throughput improvements. In contrast to other partial packet recover approaches, which use fixed modulation and coding schemes, ZipTx operates on top of adaptive modulation and coding employed by IEEE 802.11 based WLANs to maximize their error correction capabilities. ZipTx is the first approach that combines the gains of partial packet recovery and adaptive modulation and coding. However, ZipTx increases latency as it uses coding and aggregative feedback schemes. It also incurs a relatively high computational cost on channels with high BER [47].

C. CORRELATION BASED PARTIAL PACKET RECOVERY

Correlation based techniques recover a packet using its multiple corrupt copies that are received over multiple paths. The underlying assumption is that most of the errors introduced into these packets are either due to multipath fading (path-dependent errors) or noise (location-based errors). These techniques also employ partial packet recovery, but, it is only based on multiple corrupt receptions of a packet without any additional *soft* or coding information from the PHY layer.

The idea of combining two or more corrupt packets to reconstruct the original one was first proposed as an extension to the ARQ scheme [48], i.e., *Extended ARQ (EARQ)*. In this scheme, the receiver stores the corrupt packet and requests its retransmission. If the second copy is also corrupt, which is most likely if errors are due to path loss, then both copies (and more if available) are XOR-ed to locate bit errors. If the errors in both packets are not in the same position, then the packets are combined and the original packet is reconstructed by removing the erroneous bits form both copies. Otherwise, this process is repeated until the correct copy can be reconstructed.

EARQ is very useful in adverse networking conditions, where high noise and interference make it difficult for the sender to deliver a completely correct packet. In contrast to ARQ, where retransmissions are requested until a packet is correctly received, EARQ can reconstruct a packet before receiving its entirely correct copy. EARQ achieves better transmission efficiency for small packet sizes as its complexity increases exponentially with increasing packet size [48]. *MRQ (ARQ with memory)* [49] improves EARQ by using block-by-block error correction and retransmission, thereby avoiding the need for retransmission of an entire packet.

Multi Radio Diversity (MRD) [50] utilizes the spatial diversity of multiple receivers, and packet combining to improve loss resilience in WLANS. This is achieved by using multiple APs for uplink, and multiple antennas at the receiver for downlink. The underlying idea is that packet losses on different receiving antennas are statistically independent, and hence, a correct packet can be constructed from its multiple receptions on different receivers (mutiple APs or only antennas). Even if all the received copies of packets are corrupted, which could be the case for extremely noisy channels, MRD might still be able to reconstruct the original packet by coordination. As a backup, MRD also utilizes a light-weight retransmission scheme to ensure correct packet delivery. Nonetheless, MRD algorithms require the clients to simultaneously communicate with multiple APs deployed in overlapping coverage area, a requirement, that is currently not supported by widespread WLAN schemes.

Simple Packet Combining (SPaC) [51] is a light weight technique for low-power WSN, where channel utilization is typically low and packet corruption is mostly due to path fading and attenuation. The main challenge is to attempt packet recovery within the limited energy budget of WSN. SPaC simply buffers corrupt packets and when two or more corrupt copies are available, it attempts to combine them to recover the original one. For at most two copies within a certain time frame, it uses incremental CRC, while for more than two copies, majority vote over packet bits is employed to reconstruct a packet.

D. HYBRID TECHNIQUES

Error correction codes are both inefficient and insufficient in extremely poor channel conditions. A fall back mechanism improves performance of transmission when error correction schemes are not effective. Schemes, that apply error corrections combined with retransmission as a fall back, are called *Hybrid-ARQ* or simply *H-ARQ*.

Hybrid-ARQ (H-ARQ) [52] combines FEC [24], [53] with ARQ (Section III-A and Section II-B, respectively). In H-ARQ, FEC works as the primary mechanism to correct errors, while ARQ is used as a fall back mechanism. In normal ARQ technique, parity bits (such as CRC) are added to original data to detect errors, but HARQ can omit parity bits (CRC) if error correction code used by FEC mechanism can detect errors. One example of FEC code that can detect and correct errors is the Reed-Solomon code.

There are three different versions of H-ARQ (Type I, Type II and Type III), each with its own advantages and pitfalls depending on channel conditions. All the H-ARQ

techniques avail multiple techniques to reduce overhead or redundant bits sent with the original data for error detection or correction. For example, Type I H-ARQ [54] employs Adaptive Coding Rate[8] (ACR) [55], which decides the number of redundant bits (error correction codes and CRC) to be added to the transmission frame based on channel conditions. It reduces the transmission overhead whenever it can. The aim of ACR is varying the coding rate to correct errors. Coding rate is changed as soon as channel conditions are changed. However, Type II H-ARQ [56] uses Adaptive Incremental Redundancy (AIR) [57] for variable rate retransmissions and, Rate Compatible Punctured Convolutional Codes (RCPC Codes) [58] for adaptively changing the code rate without requiring the retransmission of entire blocks.

In principle, however Type I, Type II and Type III H-ARQ have a very simple working mechanism. In Type I H-ARQ, the receiver end decodes error correction codes and recovers the data corrupted over transmission, when channel conditions are good. In bad channel conditions, the receiver cannot recover corrupted data with error correction codes. The situation is detected using the error detection code and a request for retransmission is sent. A single data unit in Type I, always consists of original data, error correction code and error detection code.

The Type II H-ARQ works more efficiently as it sends the original data with error detection codes(CRC) only; Error correction codes (FEC) are sent separately when required. If the original data is received error free (detected using CRC) then error correction codes are not sent at all. In good channel conditions, data units (original data plus error detection code) are received correctly, eliminating the need of error correction codes. Thus TYPE II achieves high data throughput in good channel conditions as compared to Type I (Type I always sends both error correction and error detection codes without any exceptions).

Type III H-ARQ [56] is a modification of TYPE II and the only way it differs from Type II is due to its use of Complementary Punctured Convolutional (CPC) [59] codes for changing the coding rate. The CPC codes are self decodable, which relieves the decoder from relying on previous transmissions of the same data unit for decoding. Type III, as compared to type II, is more complex in implementation but with advantage of not requiring larger buffers at the receiver side. Nevertheless, the working mechanism of Type II and Type III are exactly similar.

Different types of H-ARQ are used in widely adopted technologies such as HSDPA [60], HSUPA [60], mobile WiMAX [61] and LTE networks [62].

IV. ERROR TOLERANCE: ACCEPTING ERRONEOUS PACKETS WHILE DISREGARDING ERRORS

Errors are bound to occur in data transmission and are more likely to appear in wireless media. Several methods, as discussed in the preceding sections, have been developed and incorporated into protocols to detect and recover corrupt packets. Error detection and correction mechanisms are either bandwidth choking or computationally demanding, and present a range of issues for battery driven devices. Protocols and mechanisms for packet recovery have evolved to the next stage, i.e., *error tolerance*. Error tolerant techniques take an entirely different approach towards dealing with transmission errors. Instead of recovering from errors in data, using redundant information that is sent alongside data, these methods tolerate errors. This is done either by preventing errors from happening at all (if possible), as in Header Compression techniques (Section IV-A), or by Error Estimation (Section IV-B), or by enabling the protocols and applications to function properly even in the presence of errors by attempting unconventional recovery mechanisms as discussed in Section IV-C.

The fact that these techniques allow protocols and applications to function even in the presence of errors, gives them an evolutionary advantage over other methods, which try to recover from errors. In the following text, we discuss the prominent approaches that represent this stage of evolution.

A. HEADER COMPRESSION TECHNIQUES

Header compression schemes, though originally aimed at reducing overhead, enable error-tolerant communications, i.e., by reducing the header size it decreases the chances of bit errors in vital portions of the packet. The need for compressing headers stemmed from the low speed serial links that provided Internet connectivity to home users in the late 80's. Such terminal connections required interactive responses from users.

The first header compression algorithm CTCP [63] was proposed in 1991, which compressed IP and TCP headers for better bandwidth utilization and line-efficiency[9] over serial links. Afterwards, many compression schemes [63]–[67] have been standardized for various protocol streams, such as for compressing TCP/IP header [63], for IP/UDP/RTP [64], for IP header compression [65], and UDP lite [67], to compress control information by exploiting redundancy in headers. For example, typical headers of UPD and TCP can be compressed down to 4–7 octets, from the original size of 20 octets. This high level of compression serves many purposes, such as:

- Improving line-efficiency for small data packets.
- Using small packets for delay sensitive low data rate traffic, i.e., voice and video.
- Decreasing header overhead and utilize bandwidth more efficiently.
- Reducing packet loss rate and bit errors rate (BER) over lossy channels.

Transmitting lesser header information means lowering the probability of errors that occur in headers. Header com-

[8]Code rate is the ratio of data bits in transport frame to the redundant bits inserted by channel coding

[9]Line-efficiency of a protocol is the ratio of data to header+data in a datagram.

pression, by significantly decreasing overhead due to control information is of phenomenal importance for battery driven, low data rate devices, such as in IEEE 802.15.4 based networks. Apart from reducing the chances of errors in headers, it also brings about energy efficiency, which is a major constraint in IEEE 802.15.4 based devices, as lesser number of bits have to be transmitted.

One of the more advanced schemes, robust header compression (ROHC) [68], can reduce the header size of UDP, Real-time transport protocol (RTP) and IP datagrams down to 3–4 bytes. In the case of both IPv4 and IPv6, it can reduce the TCP and IP header size down to 8 bytes from their original sizes of 40 and 60 bytes, respectively [62]. The ROHC protocol is widely used in LTE networks [62] and exploits the information redundancy in the headers of TCP/IP, UDP and RTP protocols. It provide different profiles for header compression, which specify how certain packet streams are going to be compressed over certain links. Hence, first the stream of packets is classified and then it is compressed using an appropriately selected profile. Unlike CTCP, which is protocol specific, the ROHC framework includes profiles for many protocol streams such as IP/UDP, IP/ESP, IP-only, IP/UDP-Lite/RTP, IP/UDP-Lite, IP/UDP/RTP, and IP/TCP, for bandwidth limited links, such as in cellular telephony, with high error rates and long round trip times.

On the downside, since header compression changes the communication behaviour of all participating nodes, therefore it needs to be deployed at all the nodes. Moreover, all the senders are required to maintain their current state of connections with all their communication partners in order to eliminate redundancies.

B. ERROR ESTIMATION

Error correction codes, discussed in section III-A, have the ability to reconstruct erroneous packets at the receiver end. Wireless networks are evolving into leveraging and taking benefit from partially correct packets. One such novel scheme, i.e., the *Error Estimation Codes (EEC)* [69] enable the receiving device (which may not be the destination of a packet) to estimate the number of corrupted bits in the partially correct packet without knowing the exact position of the corrupt bits. It can be viewed as generalized CRC, where CRC can only provide information, if BER exceeds zero. On the contrast, EEC can estimate the exact number of flipped bits or some predefined threshold.

Error Estimation Codes (EEC) add small redundancy overhead as compared to Error Correction Codes (ECC). Software implementation of EEC can support all data rates of 802.11a/g on typical hardware, while ECC typically require hardware support to increase execution speed, yet they are slower than EEC and can barely achieve data rates of WiFi networks. The low overhead and support for high data rates, and the ability of EEC to accurately estimate the number of errors in a packet, make it better than ECC. WiFi routers that implement EEC are known as BER-aware devices. By enabling EEC, routers can decide wether or not to request the retransmission of a corrupt packet, by looking at the accepted threshold of errors. This results in fewer retransmissions of corrupt packets, i.e., only those packets are retransmitted, which can't be dealt with on the higher layers. EEC can assist in better WiFi rate adaptation, by selecting data rate that provides good throughput based on tradeoff between data rates and packet BER.

Signal-to-noise ratio and symbol confidence on physical interface can be also analyzed to estimate BER. *RBAR* [70] estimates SNR (relation between SNR and BER is well-known) to adapt to a rapidly changing communication channel in WiFi networks. It estimates SNR on exchange of RTS/CTS at the beginning. However, it is not possible to accurately compute BER using RTS/CTS mechanism as SNR changes during packet transmission after the RTS/CTS is computed. Most of the contemporary methods for rate adaptation are based on SNR, but EEC comfortably outperforms the best SNR methods deployed.

SoftPHY interface information is used by SoftRate [71] to compute BER and select the transmission bit rate accordingly. It has advantage over SNR based rate adaptation schemes due to its agility in adapting to bit rates in just a single packet time duration. The quick response of this protocol due to calculations made over each packet for BER, even on ones that have no errors at all. Correct packets indicate the ideal bit rates for transmission. Regular Wi-Fi devices do not expose physical layer symbols/SoftPHY to upper layers, which renders it as an unlikely candidate for use in Wi-Fi networks. Providing the same functionality in low cost networks, such as wireless sensor networks, is undesirable due to the extra cost for changes in the required hardware.

The most notable use of BER is to adapt to transmission or receiving bit rate in response to channel variations in wireless networks. For example, Wi-Fi networks usually have the choice of selecting data rates, thus using BER information it can adapt to different data rates to reduce BER. Lower data rates mean less BER, and less BER is the only way to avoid errors. Multihop networks can choose to select different routes to optimize BER instead of optimizing delay, wherever necessary.

C. ERROR RESILIENT PROTOCOLS

Real Time Multimedia Applications (RTMA), such as video and voice over wireless IP networks, are proliferating at an ever increasing rate. The interactive nature of these multimedia applications makes them less tolerant to delays and more accepting towards corrupt data. Audio and video codes that can cope with errors in payloads have been designed, and they only require headers to be delivered free of errors. Such applications can take advantage of protocols that allow the delivery of damaged packets.

UDP-lite [72], [73] is probably the first protocol to ignore the payload errors and ensure only the integrity of the protocol header. It can deliver erroneous packets, i.e., packets with correct headers but damaged payloads, to applications and let them deal with the errors in the payload. In contrast to

UDP, UDP-lite can provide the high data rates and flexibility required for RTMA, on error-prone networks, especially wireless links. It divides its entire PDU length into sensitive and insensitive parts, i.e., the header fields and payload, respectively. Checksum is applied only to the sensitive part. The ''length'' field in the typical UDP header is renamed as ''coverage'', and it shows the number of bytes that need to be labeled as sensitive. UDP-lite was designed and proposed for Multimedia applications that prefer damaged data over delays and retransmissions. Studies [74], [75] have shown that UDP-lite improves the quality and performance of RTMAs by reducing packet delays, packet loss and the inter-arrival time of packets.

To support UDP-lite, the lower layers, especially the link layer must support partial error detection. This would enable the lower layers to let the corrupt packets reach the upper layer, where UDP-lite can receive them. Moreover the link layer must be able to detect frames that carry UDP-lite PDUs, so that they are not discarded if errors are detected in the insensitive part of the UDP-lite PDUs. This is unlike the the traditional link-layer behavior of the widely available wireless devices. UDP-lite has two major drawbacks;

- It uses a different protocol identifier[10] than UDP, which implies that applications and network devices using UDP-lite must have a UDP-lite stack installed at both the sending and receiving. Thus making UDP-lite backwards incompatible with traditional UDP stacks in network devices.
- It does not handle corrupt packets at the application layer. All the packets received by the application layer from UPD-lite are accepted as normal packets, i.e., the application is not aware of any corruption in data. This could be disastrous for protocols that use communication signalling, such as session initiation protocol(SIP).

UDP-liter [76] is a successor to UDP-lite and has been named so because it leverages the concept of UDP-lite, and is ''lighter'' than its predecessor. UDP-liter makes minor changes in UDP-lite and elegantly solves its drawbacks. UDP-liter does not change the ''length'' field in the UDP header, and the checksum field provides checksum for the entire packet. A conventional UDP receiver can process the packets as it would without the need for a different protocol identifier. UDP-liter works like UDP and uses checksums to discard erroneous packets. It has a run-time option, which if enabled, allows it to automatically start ignoring the checksum results and pass on the erroneous packets to the upper layer. Shared protocol identifier and similar headers mean that only the receiving node needs to be aware of UDP-liter. However, a modification in the BSD socket API enables it to activate the run-time option and receive a corruption notification (CN). The CN allows the API to determine whether to use the corrupt packet with a different algorithm or discard it. It also makes the application aware of corrupt packets, thus fixing the inability of UDP-Lite in handling corrupt packets at the application layer.

UDP-liter is designed to let the applications choose, how to deal with a packet. Applications make use of the options through calls to the *socket()* function call. This way, they can choose to use UDP-liter in the conventional UDP mode for their transmissions or take advantage of the services offered by UDP-liter. For conventional UDP behavior, UDP-liter simply discards packets with failed checksums, and for UDP-liter services, it passes the packets to the upper layer with a corruption notification (CN), which is carried by the parameter, *pCorrupted*. With the *pCorrupted* parameter, an application can determine whether a packet has been corrupted or not, and therefore handle it accordingly. *Refector* [4] takes the UDP-Liter approach to the next level by delivering damaged packets with errors even in their headers. Refector is the first protocol that is tolerant to header errors. Errors in headers are intelligently investigated and heuristically corrected, based on previous knowledge from protocol state. Just like UDP-Lite, Refector is a higher layer protocol, however unlike UDP-lite, which works on layer 4 only, Refector works on both layer 3 and layer 4. Corrupted destination port number (layer 4) and corrupted destination address (layer 3) are repaired by matching them to the protocol state and selecting the one closest, using Hamming code.[11] Refector also exploits the fact that the integrity of all the header fields is not necessary for correctly delivering a packet to its destination, e.g., TTL. Hence error correction is not applied to such fields.

CPR or *corrupt packets recycling* proposes a similar approach of heuristically repairing corrupt packets [77], with emphasis on forwarding partially corrupt packets over multiple hops in WSN. The protocol design does not require to completely recover the header, and does not attempt to recover the payload at all. This is due to the nature of applications and network configurations of WSN nodes. This approach works mainly because of the way the CTP, i.e., *collection tree protocol* works [78]. *CTP* is the defacto routing protocol for WSN applications. It implements addressing techniques to traverse different paths to gateways. Since all the packets are destined to a single destination (gateway or sink), even corrupt headers with wrong addresses can be routed to the gateway. This is the focal point of CPR, i.e., to forward partially corrupt packets over multiple hops.

CPR can heuristically correct or ignore other CTP fields (Pull bit, origin bit, etc) that are required by CTP for efficient routing. However, CPR can take decisions without considering these fields at the expanse of a fractional loss in performance. Interestingly, CPR has no proposition for correcting damaged payloads in packets. The packet payload fields are marked as ''don't care'', and concealed during the entire process of recycling and forwarding packets. Correcting the payload errors is left to methods implemented on the gateways. In most WSN applications, the data collected by a

[10] A protocol identifier or protocol code is number assigned to each transport layer protocol that identifies the protocol used for upper layers.

[11] Protocol state includes information like addresses and ports open from previous communication.

sensor is not 100% correct, and the receiving devices don't rely on a single reading from a single device. Therefore, data, which is received from multiple devices over time, is aggregated and manipulated at the gateway to gather meaningful information. This level of sophistication and redundancy, in WSN data, makes it resilient to payload errors. The goal of CPR is to recycle packets to avoid retransmission, improve data delivery rate and eventually improve latency, bandwidth usage, and battery life of WSNs. Preliminary results of CPR implementation show staggering improvements in data delivery rates, up to 4 times without any transmission overheads. Though, the concealed payload part of the CPR is a bit skeptical but there are some applications that can directly benefit from this method especially, where the data delivery rate and battery life take priority over the reception of 100% correct data. Passive Collection Streams like [79] and [80], which are WSNs that focus on maximizing network lifetime, can directly benefit from CPR.

A *probabilistic approach* [81] of repairing entirely corrupt packets, exploits patterns in such packets. This approach targets outdoor wireless sensor networks, where radio link performance deteriorates significantly, thereby incurring retransmissions and energy burden on the nodes. While symbols and code words at the physical layer are used by other protocols to construct data, this approach finds a noticeable pattern in packet corruption, over specific time chunks, and across specific links to the finest level of data transmissions, i.e symbols, code words and pseudo-noise chip sequence (physical layer). The distribution of errors and the amount of corruption are also taken into consideration before applying probability to symbols received at the lowest level, to infer the likelihood of the originally sent symbols. Since data generated by sensors is not accurate per se, probabilistically recovered packets are acceptable in sensor networks.

D. APPLICATION LAYER

The primary objective of the packet recovery methods discussed so far, with the exception of UDP-Lite, UDP-Liter and Refector, is to ensure the delivery of correct and reliable data to the application layer. The corrupt packets are not allowed to reach the application layer. The lower layers of the protocol stack provide reliable transmission and end to end connectivity for the upper most layer, which only generates user data. However, effective packet recovery and error resilience by the lower layers, requires fundamental changes in the protocol standards. Specific applications such as audio video calls and multimedia streaming, require real time data processing and quality of service. The use of error-correction techniques, on the entire network stack, to achieve reliable packet delivery; or the introduction of mechanisms to ensure quality of service for audio and video services, can introduce enormous overhead and complications over the entire network. Instead recovery mechanisms at the application layer can be more transparent and independent from the lower layers.

The authors in [82] propose forward error correction for the headers of video streams at the application layer. They propose to encode only the header of the video stream using Hamming code and embed this redundant data into the original bitstream. The original header is left untouched, which enables the standard video codecs that do not support application layer error correction, to decode the video stream. The redundancy introduced by the proposed method increases the video bit-rate by 1 *kbps*. The use of error correction techniques at the application layer relaxes the entire network stack from adjusting parameters such as FEC coding mode, transmission power and the rate of ARQ to adjust the Quality of Service.

In [83], an application level incremental redundancy or IR scheme is proposed for streaming multimedia to a wireless client over a lossy packet network. This scheme uses a type III H-ARQ scheme (Section III-D) at the application layer that works end-to-end to ensure QoS support for multimedia, without any QoS support from the network. In this scheme unequal redundancy is inserted into the data at the source, which can be varied depending upon the network conditions that are determined using the NAK (Negative acknowledgement) feedback from the destination. The client at the destination must be able to receive corrupt packets at the application layer, i.e., use the UDP-Lite protocol stack, and it must also be able to sense data corruption at the application level to send back the appropriate feedback. The source, which is a streaming media server, ensures QoS for the client by: 1) Determining the network conditions using NAK feedback from the destination, 2) Using its knowledge of the importance, dependencies and different deadlines of different media packets, it sets up a transmission policy that is optimum in a rate-distortion sense, i.e., different media packets are afforded different error protection so as to minimize the end-to-end distortion while maintaining the expected rate of transmission. The performance and effectiveness of this scheme can however be limited by header corruption. Thus, for very high BER, it requires header protection support at the transport, network or link layers.

In [37], a discussion of similar application level forward error correction (AL-FEC) used in LTE networks for multimedia broadcast and multicast services (MBMS), is provided. The AL-FEC in LTE networks uses Raptor codes (Section III-A) to address the limitations of FEC, when it is used by other layers of the network protocol stack, i.e. it ensures reliable and scalable operation in the backdrop of different packet loss rates. The use of Raptor codes for AL-FEC makes up for the data loss due to packets that are lost or rejected at the lower layers of the protocol. The overall scheme tries to ensure that every packet is sent at most once by introducing sufficient redundancy, in the form of FEC overhead, to deal with different packet loss rates at the mobile receiver.

Audio codecs use error concealment approach to leverage the benefits of application layer. The AMR-WB+ [84] codec, which is designed for use by MBMS and multimedia

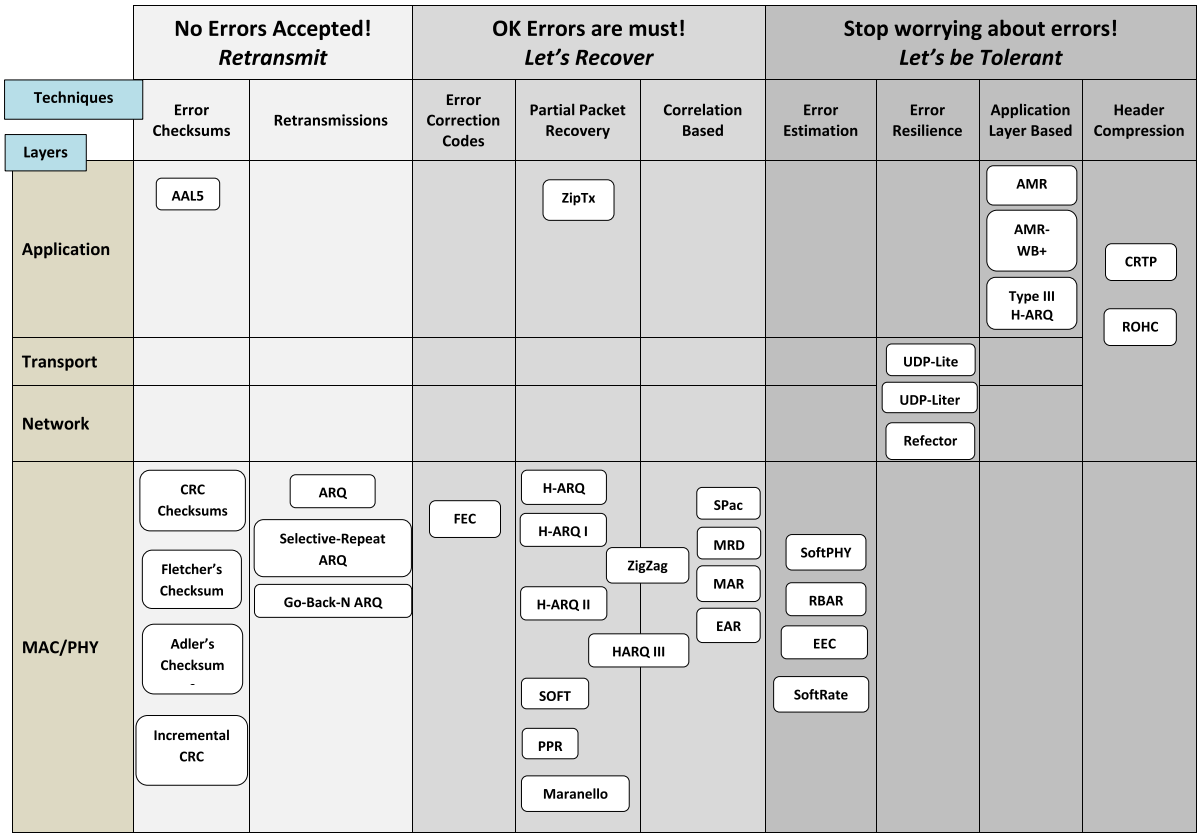


FIGURE 1. A taxonomy of network protocols and techniques for error detection, correction and packet recovery.

messaging services (MMS) in 3G and LTE networks, uses both frame interleaving and forward error correction (FEC) to achieve robustness against packet loss. The AMR-WB+ can use either the generic FEC as defined in [85] or audio redundancy coding defined in [86] to make the RTP packets more resilient to losses. However, both these approaches increase the bit rate of the audio stream. In audio redundancy coding, new payloads are constructed by combining previously transmitted audio frames with new frames, using a sliding window before transmission. This mechanism increases the buffer requirements at the receiver and can also lead to more delays, as the receiver would require more memory to store the redundant frames and would wait longer for their arrival. Moreover, a packet loss can result in the loss of several consecutive frames, which adversely affects the efficiency of error concealment and leads to audible distortions in the reconstructed audio. In order to improve the quality of audio over lossy wireless links, interleaving of audio frames is used. The idea is simple, consecutive frames are not bundled into a single packet, rather they are distributed over multiple packets, so that a single packet loss would not result in the loss of consecutive audio frames. Interleaving increases the quality of audio, but adversely affects the end to end delay and buffering requirements. Therefore in AMR-WB+, frame interleaving is optional and is negotiated by the communicating nodes during session setup.

V. CONCLUSION

In this survey, we looked at the evolution of protocols and mechanisms for packet recovery in wireless networks. We believe that during its evolution the protocol stack passed through three distinct phases, i.e., from an error intolerant entity in the beginning, into one that has accepted the reality of errors and is more tolerant towards them. In our opinion, the current evolution of these protocols and mechanisms (Fig. 1) was driven in large part by multimedia applications such as voice and video services over wireless IP networks, which are more tolerant to errors and less tolerant to delays thus forcing the protocol stack to take measures that would reduce retransmissions and ensure timely delivery of packets, even if they are corrupt. This has resulted in a growing role for the application layer to recover packets that would otherwise be lost at the lower protocol layers. Future evolution of these mechanisms would be driven in large part by potential future applications and uses of the internet. The internet of things or cyber physical systems (CPS) is the next big thing, which is going to result in a greater integration of the physical world and services with the internet. The IoT and cloud computing technologies, which predominantly use wireless networks to collect and process data and affect behavior of the physical environment, would require ever more robust, secure, scalable and reliable protocols for data delivery. The availability of cheap, energy efficient computing platforms

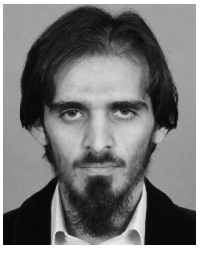
for wireless nodes and the ever increasing volume of traffic over wireless networks is going to encourage the proliferation of protocols that are more tolerant to errors at the lower layers of the protocol stack and instead rely more on innovative coding techniques to reconstruct failed packets at the application layer. Application layer error correction works best with header protection at the lower layers of the network. Hence, mechanisms for header protection at the lower layers of the protocol stack is another potential area for future investigation.

REFERENCES

[1] S. Alfredsson, "TCP in wireless networks: Challenges, optimizations and evaluations," Faculty Econ. Sci., Commun. IT, Karlstad Univ., Karlstad, Sweden, Tech. Rep. ISSN 1403-8099; 2005:13, 2003.
[2] M. H. Alizai, *Exploiting Wireless Link Dynamics*. Aachen, Germany: Shaker Verlag GmbH, 2013.
[3] B. Han *et al.*, "Maranello: Practical partial packet recovery for 802.11," in *Proc. 7th USENIX Conf. Netw. Syst. Design Implement. (NSDI)*, 2010, p. 14.
[4] F. Schmidt, M. H. Alizai, I. Aktaş, and K. Wehrle, "Refector: Heuristic header error recovery for error-tolerant transmissions," in *Proc. 7th Conf. Emerg. Netw. Experim. Technol. (CoNEXT)*, New York, NY, USA, 2011, pp. 22:1–22:12.
[5] M. Luby, L. Vicisano, J. Gemmell, L. Rizzo, M. Handley, and J. Crowcroft, *Forward Error Correction (FEC) Building Block*, document RFC 3452, 2002.
[6] K. C.-J. Lin, N. Kushman, and D. Katabi, "ZipTx: Harnessing partial packets in 802.11 networks," in *Proc. 14th ACM Int. Conf. Mobile Comput. Netw. (MOBICOM)*, 2008, pp. 351–362.
[7] E. Rozner, A. P. Iyer, Y. Mehta, L. Qiu, and M. Jafry, "ER: Efficient retransmission scheme for wireless LANs," in *Proc. ACM CoNEXT Conf. (CoNEXT)*, 2007, pp. 8:1–8:12.
[8] *Internet Protocol—DARPA Inernet Programm, Protocol Specification*, document RFC 791, Internet Engineering Task Force, Sep. 1981.
[9] P. Deutsch and J.-L. Gailly, *Zlib Compressed Data Format Specification Version 3.3*, document RFC 1950, 1996.
[10] J. Fletcher, "An arithmetic checksum for serial transmissions," *IEEE Trans. Commun.*, vol. COM-30, no. 1, pp. 247–252, Jan. 1982.
[11] C. Partridge, J. Hughes, and J. Stone, "Performance of checksums and CRCs over real data," in *Proc. SIGCOMM Comput. Commun. Rev.*, Oct. 1995, pp. 68–76.
[12] T. C. Maxino and P. J. Koopman, "The effectiveness of checksums for embedded control networks," *IEEE Trans. Dependable Secure Comput.*, vol. 6, no. 1, pp. 59–72, Jan./Mar. 2009.
[13] W. W. Peterson and D. T. Brown, "Cyclic codes for error detection," *Proc. IRE*, vol. 49, no. 1, pp. 228–235, Jan. 1961.
[14] J. S. Sobolewski, "Cyclic redundancy check," in *Encyclopedia of Computer Science*. Chichester, U.K.: Wiley, 2003, pp. 476–479.
[15] T. Mandel and J. Mache, "Investigating CRC polynomials that correct burst errors," in *Proc. ICWN*, 2009, pp. 632–637.
[16] T. Mandel and J. Mache, "Selected CRC polynomials can correct errors and thus reduce retransmission," in *Proc. WITS (DCOSS)*, Jun. 2009, pp. 1–6.
[17] T. Mandel and J. Mache, "Practical error correction for resource-constrained wireless networks: Unlocking the full power of the CRC," in *Proc. 11th ACM Conf. Embedded Netw. Sensor Syst. (SenSys)*, New York, NY, USA, 2013, pp. 3:1–3:14.
[18] J. Stone, M. Greenwald, C. Partridge, and J. Hughes, "Performance of checksums and CRCs over real data," *IEEE/ACM Trans. Netw.*, vol. 6, no. 5, pp. 529–543, Oct. 1998.
[19] J. Stone and C. Partridge, "When the CRC and TCP checksum disagree," *ACM SIGCOMM Comput. Commun. Rev.*, vol. 30, no. 4, pp. 309–319, Oct. 2000.
[20] J. Postel, *Transmission Control Protocol*, document RFC 793, Internet Engineering Task Force, Sep. 1981.
[21] R. W. Hamming, "Error detecting and error correcting codes," *Bell Syst. Tech. J.*, vol. 29, no. 2, pp. 147–160, Apr. 1950.
[22] C. E. Shannon, "A mathematical theory of communication," *Bell Syst. Tech. J.*, vol. 27, no. 3, pp. 379–423, Oct. 1948.
[23] S. Lin and D. J. Costello, *Error Control Coding*. Englewood Cliffs, NJ, USA: Prentice-Hall, 2004.
[24] P. K. McKinley, C. Tang, and A. P. Mani, "A study of adaptive forward error correction for wireless collaborative computing," *IEEE Trans. Parallel Distrib. Syst.*, vol. 13, no. 9, pp. 936–947, Sep. 2002.
[25] Y. Wang and Q.-F. Zhu, "Error control and concealment for video communication: A review," *Proc. IEEE*, vol. 86, no. 5, pp. 974–997, May 1998.
[26] M. Elaoud and P. Ramanathan, "Adaptive use of error-correcting codes for real-time communication in wireless networks," in *Proc. 17th Annu. Joint Conf. IEEE Comput. Commun. Soc. (INFOCOM)*, vol. 2. Mar./Apr. 1998, pp. 548–555.
[27] I. S. Reed and G. Solomon, "Polynomial codes over certain finite fields," *J. Soc. Ind. Appl. Math.*, vol. 8, no. 2, pp. 300–304, Jun. 1960.
[28] J. Gao. (Feb. 2007). *Reed-Solomon Code*. [Online]. Available: http://www3.cs.stonybrook.edu/~jgao/CSE590-fall09/reed-solomon.pdf
[29] R. Johannesson and K. S. Zigangirov, *Fundamentals of Convolutional Coding*. New York, NY, USA: Wiley, 1999.
[30] D. J. C. MacKay, *Information Theory, Inference and Learning Algorithms*. Cambridge, U.K.: Cambridge Univ. Press, 2003.
[31] C. Berrou, A. Glavieux, and P. Thitimajshima, "Near Shannon limit error-correcting coding and decoding: Turbo-codes. 1," in *Proc. IEEE Int. Conf. Commun. (ICC)*, vol. 2. May 1993, pp. 1064–1070.
[32] R. G. Gallager, "Low-density parity-check codes," *IRE Trans. Inf. Theory*, vol. 8, no. 1, pp. 21–28, Jan. 1962.
[33] S. J. Johnson and S. R. Weller, *Low-Density Parity-Check Codes: Design and Decoding*. New York, NY, USA: Wiley, Jan. 2003, pp. 1–18.
[34] A. Shokrollahi, "Raptor codes," *IEEE Trans. Inf. Theory*, vol. 52, no. 6, pp. 2551–2567, Jun. 2006.
[35] M. Luby, A. Shokrollahi, M. Watson, and T. Stockhammer, "Raptor forward error correction scheme for object delivery," IETF (Internet Engineering Task Force), Fremont, CA, USA, Tech. Rep. RFC 5053, 2007.
[36] M. Luby, A. Shokrollahi, M. Watson, and T. Stockhammer, *RaptorQ Forward Error Correction Scheme for Object Delivery*, document RFC 6330, Internet Engineering Task Force, 2007.
[37] C. Bouras, N. Kanakis, V. Kokkinos, and A. Papazois, "Application layer forward error correction for multicast streaming over LTE networks," *Int. J. Commun. Syst.*, vol. 26, no. 11, pp. 1459–1474, 2013.
[38] D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris, "Link-level measurements from an 802.11b mesh network," in *Proc. Conf. Appl., Technol., Archit., Protocols Comput. Commun. (SIGCOMM)*, 2004, pp. 121–132.
[39] K. Jamieson and H. Balakrishnan, "PPR: Partial packet recovery for wireless networks," in *Proc. Conf. Appl., Technol., Archit., Protocols Comput. Commun. (SIGCOMM)*, 2007, pp. 409–420.
[40] G. R. Woo, P. Kheradpour, D. Shen, and D. Katabi, "Beyond the bits: Cooperative packet recovery using physical layer information," in *Proc. MOBICOM*, 2007, pp. 147–158.
[41] S. Gollakota and D. Katabi, "Zigzag decoding: Combating hidden terminals in wireless networks," in *Proc. ACM SIGCOMM Conf. Data Commun. (SIGCOMM)*, 2008, pp. 159–170.
[42] S. Khurana, A. Kahol, and A. Jayasumana, "Effect of hidden terminals on the performance of IEEE 802.11 MAC protocol," in *Proc. 23rd Annu. Conf. Local Comput. Netw. (LCN)*, Oct. 1998, pp. 12–20.
[43] Y.-C. Cheng, J. Bellardo, P. Benkö, A. C. Snoeren, G. M. Voelker, and S. Savage, "Jigsaw: Solving the puzzle of enterprise 802.11 analysis," in *Proc. Conf. Appl., Technol., Archit., Protocols Comput. Commun. (SIGCOMM)*, New York, NY, USA, 2006, pp. 39–50.
[44] P. C. Ng, S. C. Liew, K. C. Sha, and W. T. To, "Experimental study of hidden-node problem in IEEE802.11 wireless networks," in *Proc. Sigcomm Poster*, 2005, pp. 1–2.
[45] S. Khurana, A. Kahol, and A. P. Jayasumana, "Effect of hidden terminals on the performance of IEEE 802.11 MAC protocol," in *Proc. 23rd Annu. IEEE Conf. Local Comput. Netw. (LCN)*, Washington, DC, USA, Oct. 1998, pp. 12–20.
[46] C.-J. M. Liang, N. B. Priyantha, J. Liu, and A. Terzis, "Surviving Wi-Fi interference in low power ZigBee networks," in *Proc. 8th ACM Conf. Embedded Netw. Sensor Syst.*, 2010, pp. 309–322.
[47] J. Xie, "Design, implementation, and evaluation of an efficient software partial packet recovery system in 802.11 wireless LANS," Ph.D. dissertation, Dept. Comput. Sci., Florida State Univ., Tallahassee, FL, USA, 2012.
[48] S. S. Chakraborty, E. Yli-Juuti, and M. Liinaharja, "An ARQ scheme with packet combining," *IEEE Commun. Lett.*, vol. 2, no. 7, pp. 200–202, Jul. 1998.

[49] P. Sindhu, "Retransmission error control with memory," *IEEE Trans. Commun.*, vol. 25, no. 5, pp. 473–479, May 1977.
[50] A. Miu, H. Balakrishnan, and C. E. Koksal, "Improving loss resilience with multi-radio diversity in wireless networks," in *Proc. 11th Annu. Int. Conf. Mobile Comput. Netw.*, 2005, pp. 16–30.
[51] H. Dubois-Ferrière, D. Estrin, and M. Vetterli, "Packet combining in sensor networks," in *Proc. 3rd Int. Conf. Embedded Netw. Sensor Syst. (SenSys)*, New York, NY, USA, 2005, pp. 102–115.
[52] H. T. Nguyen, H. H. Nguyen, and T. Le-Ngoc, "Power-efficient cooperative coding with hybrid-ARQ soft combining for wireless sensor networks in block-fading environment," *Int. J. Sensor Netw.*, vol. 4, nos. 1–2, pp. 3–12, 2008.
[53] L. Rizzo and L. Vicisano, "RMDP: An FEC-based reliable multicast protocol for wireless environments," *SIGMOBILE Mobile Comput. Commun. Rev.*, vol. 2, no. 2, pp. 23–31, Apr. 1998.
[54] J. Roman, F. Berens, M. Kirsch, and S. Tanrikulu, "Hybrid ARQ schemes for future wireless systems based on MC-CDMA," in *Proc. IST*, 2009, pp. 1–5.
[55] S. Kallel, "Efficient hybrid ARQ protocols with adaptive forward error correction," *IEEE Trans. Commun.*, vol. 42, no. 234, pp. 281–289, Feb./Mar./Apr. 1994.
[56] Y. J. Guo, *Advances in Mobile Radio Access Networks*. Norwood, MA, USA: Artech House, 2004.
[57] S. Lin, D. J. Costello, and M. J. Miller, "Automatic-repeat-request error-control schemes," *IEEE Commun. Mag.*, vol. 22, no. 12, pp. 5–17, Dec. 1984.
[58] J. Hagenauer, "Rate-compatible punctured convolutional codes (RCPC codes) and their applications," *IEEE Trans. Commun.*, vol. 36, no. 4, pp. 389–400, Apr. 1988.
[59] S. Kallel, "Complementary punctured convolutional (CPC) codes and their applications," *IEEE Trans. Commun.*, vol. 43, no. 6, pp. 2005–2009, Jun. 1995.
[60] H. Holma and A. Toskala, Eds., *HSDPA/HSUPA for UMTS: High Speed Radio Access for Mobile Communications*. New York, NY, USA: Wiley, 2007.
[61] F. Wang, A. Ghosh, C. Sankaran, P. J. Fleming, F. Hsieh, and S. J. Benes, "Mobile WiMAX systems: Performance and evolution," *IEEE Commun. Mag.*, vol. 46, no. 10, pp. 41–49, Oct. 2008.
[62] A. Larmo, M. Lindström, M. Meyer, G. Pelletier, J. Torsner, and H. Wiemann, "The LTE link-layer design," *IEEE Commun. Mag.*, vol. 47, no. 4, pp. 52–59, Apr. 2009.
[63] V. Jacobson, *Compressing TCP/IP Headers for Low-Speed Serial Links*, document RFC 1144, Internet Engineering Task Force, Feb. 1990.
[64] S. Casner and V. Jacobson, *Compressing IP/UDP/RTP Headers for Low-Speed Serial Links*, document RFC 2508, Internet Engineering Task Force, Feb. 1999.
[65] M. Degermark, B. Nordgren, and S. Pink, *IP Header Compression*, document RFC 2507, Internet Engineering Task Force, Feb. 1999.
[66] L.-E. Jonsson and G. Pelletier, *RObust Header Compression (ROHC): A Compression Profile for IP*, document RFC 3843, Internet Engineering Task Force, Jun. 2004.
[67] G. Pelletier, *RObust Header Compression (ROHC): Profiles for User Datagram Protocol (UDP) Lite*, document RFC 4019, Internet Engineering Task Force, Apr. 2005.
[68] C. Bormann *et al.*, *RObust Header Compression (ROHC): Framework and Four Profiles: RTP, UDP, ESP, and Uncompressed*, document RFC 3095, Internet Engineering Task Force, Jul. 2001.
[69] B. Chen, Z. Zhou, Y. Zhao, and H. Yu, "Efficient error estimating coding: Feasibility and applications," *SIGCOMM Comput. Commun. Rev.*, vol. 40, no. 4, pp. 3–14, Oct. 2010.
[70] G. Holland, N. Vaidya, and P. Bahl, "A rate-adaptive MAC protocol for multi-hop wireless networks," in *Proc. 7th Annu. Int. Conf. Mobile Comput. Netw.*, 2001, pp. 236–251.
[71] M. Vutukuru, H. Balakrishnan, and K. Jamieson, "Cross-layer wireless bit rate adaptation," *SIGCOMM Comput. Commun. Rev.*, vol. 39, no. 4, pp. 3–14, Oct. 2009.
[72] L.-A. Larzon, M. Degermark, S. Pink, L.-E. Jonsson, and G. Fairhurst, *The Lightweight User Datagram Protocol (UDP-Lite)*, document RFC 3828, Internet Engineering Task Force, Jul. 2004.
[73] L.-A. Larzon, M. Degermark, S. Pink, L.-E. Jonsson, and G. Fairhurst, *The Lightweight User Datagram Protocol (UDP-Lite)*, document Internet RFC 3828, Jul. 2004.
[74] A. Singh, A. Konrad, and A. D. Joseph, "Performance evaluation of UDP lite for cellular video," in *Proc. NOSSDAV*, 2001, pp. 117–124.
[75] L.-A. Larzon, M. Degermark, and S. Pink, "Efficient use of wireless bandwidth for multimedia applications," in *Proc. IEEE Int. Workshop Mobile Multimedia Commun. (MoMuC)*, Nov. 1999, pp. 187–193.
[76] P. P.-K. Lam and S. C. Liew, "UDP-Liter: An improved UDP protocol for real-time multimedia applications over wireless links," in *Proc. IEEE ISWCS*, Sep. 2004, pp. 314–318.
[77] M. H. Alizai, M. Moosa, D. Han, O. Gnawali, and A. A. Syed, "Recycling corrupt packets over multiple hops," in *Proc. 12th Eur. Conf. Wireless Sensor Netw. (EWSN)*, Feb. 2015, pp. 242–249.
[78] O. Gnawali, R. Fonseca, K. Jamieson, M. Kazandjieva, D. Moss, and P. Levis, "CTP: An efficient, robust, and reliable collection tree protocol for wireless sensor networks," *ACM Trans. Sensor Netw.*, vol. 10, no. 3, 2013, Art. no. 16.
[79] A. Mainwaring, D. Culler, J. Polastre, R. Szewczyk, and J. Anderson, "Wireless sensor networks for habitat monitoring," in *Proc. 1st ACM Int. Workshop Wireless Sensor Netw. Appl. (WSNA)*, New York, NY, USA, 2002, pp. 88–97.
[80] G. Werner-Allen *et al.*, "Deploying a wireless sensor network on an active volcano," *IEEE Internet Comput.*, vol. 10, no. 2, pp. 18–25, Mar./Apr. 2006.
[81] F. Hermans, H. Wennerström, L. McNamara, C. Rohner, and P. Gunningberg, "All is not lost: Understanding and exploiting packet corruption in outdoor sensor networks," in *Wireless Sensor Networks* (Lecture Notes in Computer Science), vol. 8354, B. Krishnamachari, A. L. Murphy, and N. Trigoni, Eds. Switzerland: Springer, 2014, pp. 116–132.
[82] C.-H. Pan, I.-H. Lee, S.-C. Huang, C.-C. Cheng, C.-J. Lian, and L.-G. Chen, "Application layer error correction scheme for video header protection on wireless network," in *Proc. 7th IEEE Int. Symp. Multimedia (ISM)*, Dec. 2005, pp. 499–505.
[83] J. Chakareski and P. A. Chou, "Application layer error-correction coding for rate-distortion optimized streaming to wireless clients," *IEEE Trans. Commun.*, vol. 52, no. 10, pp. 1675–1687, Oct. 2004.
[84] J. Sjoberg, M. Westerlund, A. Lakaniemi, and S. Wenger, *RTP Payload Format for the Extended Adaptive Multi-Rate Wideband (AMR-WB+) Audio Codec*, document RFC 4352, Internet Engineering Task Force, Jan. 2006.
[85] J. Rosenberg and H. Schulzrinne, *An RTP Payload Format for Generic Forward Error Correction*, document RFC 2733, 1999.
[86] C. Perkins *et al.*, "RTP payload for redundant audio data," IETF (Internet Engineering Task Force), Fremont, CA, USA, Tech. Rep. RFC 2198, 1997.

SHERAZ ALI KHAN received the B.Sc. degree in computer information systems engineering from the University of Engineering & Technology Peshawar, Pakistan, in 2004, and the M.S. degree in nuclear engineering from the Pakistan Institute of Engineering & Applied Sciences, in 2007. He is currently pursuing the Ph.D. degree with the Embedded Ubiquitous Computing System Laboratory, University of Ulsan, South Korea. He is an Assistant Professor with the Institute of Mechatronics Engineering, University of Engineering & Technology Peshawar. His research interests include development of efficient techniques, architectures, and embedded platforms for fault diagnosis and prognosis in rotary machines that operate in a networked environment.



MUHAMMAD MOOSA received the B.S. and M.S. degrees in computer systems engineering from the University of Engineering & Technology Peshawar. He is a Graduate Research Assistant with the EmNets Laboratory, University of Engineering & Technology Peshawar. He is currently involved in research projects in the area of wireless sensor networks and error tolerant communications with particular interest in WSN applications, protocols, and architecture and evaluation tools.

FARHAN NAEEM received the B.S. and M.S. degrees (Hons.) in computer systems engineering from the University of Engineering & Technology Peshawar. He is a Research Assistant with the EmNets Laboratory, University of Engineering & Technology Peshawar. He is currently involved in multiple research projects in the area of wireless sensor networks, delay tolerant networks, and error tolerant communications.

MUHAMMAD HAMAD ALIZAI received the Ph.D. and M.Sc. degrees from RWTH Aachen University, in 2012 and 2007, respectively. He was a Research Assistant with the ComSys Group, RWTH Aachen University, Germany. He is an Assistant Professor with the Department of Computer Science, Lahore University of Management Sciences. He is particularly interested in applications, protocols, architectures, and evaluation tools for future networks. His research interests are in mobile applications, Internet of Things, sensornets, and delay tolerant networking.

JONG-MYON KIM (M'05) received the B.S. degree in electrical engineering from Myongji University, Yongin, South Korea, in 1995, the M.S. degree in electrical and computer engineering from the University of Florida, Gainesville, in 2000, and the Ph.D. degree in electrical and computer engineering from the Georgia Institute of Technology, Atlanta, in 2005. He is a Professor of IT Convergence with the University of Ulsan, South Korea. His research interests include multimedia processing, digital watermarking, multimedia specific processor architecture, parallel processing, and embedded systems. He is a member of the IEEE Computer Society.