A Heuristic Header Error Recovery Scheme for RTP

Florian Schmidt, David Orlea, Klaus Wehrle
Communication and Distributed Systems Group
RWTH Aachen University, Germany
Email: {schmidt,wehrle}@comsys.rwth-aachen.de, david.orlea@rwth-aachen.de

***Abstract*—Streaming applications often tolerate bit errors in their received data well. This is contrasted by the enforcement of correctness of the packet headers and payload by network protocols. We investigate a solution for the Real-time Transport Protocol (RTP) that is tolerant to errors by accepting erroneous data. It recovers from header errors by leveraging the known state of a stream, passing potentially corrupted payloads to the codecs. It is a receiver-based solution that requires neither support from the sender nor changes to the RTP specification. Evaluations show that our header error recovery scheme can recover from almost all errors, with virtually no erroneous recoveries, up to bit error rates of about 10%.**

I. Introduction

Wireless communication is playing an ever-increasing role in Internet connectivity. The ubiquity of notebooks, tablets, and smartphones leads to increasingly common use of wireless connections for the last hop. One fundamental problem of wireless communication is the higher unreliability of the link compared to wired communication, which leads to a higher bit error rate. This high bit error rate requires large numbers of retransmissions of data, because protocol standards that were defined with wired link characteristics in mind require full packet retransmission if even a single bit error occurs.

At the same time, video and audio traffic has greatly increased. Many codecs are in principle error-tolerant, being able to correct or at least mask errors. However, especially for live streaming and bidirectional communication, they require high timeliness of data to reduce harmful delay. For this class of traffic, partially erroneous packets arriving in time are helpful, while correct packets that arrive too late (due to packet drops and retransmissions) are practically useless. Therefore, providing those streaming applications with partially erroneous data is beneficial to their overall performance [1].

It therefore stands to reason to support error-tolerant applications by introducing error tolerance concepts to the standard Internet communication. Previous solutions, most prominently UDP-Lite [2], introduced error tolerance for a single protocol. They typically focus on *payload error tolerance*, that is, they allow errors in the payload by using checksums to secure only packet headers. For large payload sizes, as in video streaming, this works well because headers only form a small part of each packet. Conversely, for communication such as Voice over IP (VoIP), packets are typically small, and the headers are, in relation, large, sometimes larger than the payload. This limits the effectiveness of payload-only error tolerance.

We therefore focus on how to introduce *tolerance also to header errors*. We accept erroneous packets, even if the errors are within the header area, and heuristically repair these errors to identify the correct data stream the packet belongs to. In previous work [3], we showed that an approach that we termed *Refector* (from Latin: repairer, mender) is feasible for UDP and IP, and that it can significantly reduce packet loss. However, for many application scenarios, UDP and IP are not enough. Many streaming applications, first and foremost VoIP, employ the application-layer Real-time Transport Protocol (RTP) [4] for timestamping, sequencing, and payload format layout.

Our main contribution is a heuristic header error recovery scheme for RTP that enables error-tolerant media codecs to receive packet payloads even if there are errors in the RTP header. We identify which stream a packet belongs to by looking at the header values expected for the next packet in each stream, and then repair the header contents to those expected ones. Thus, we can repair errors for static parts of the header as well as dynamic parts that change every packet. Our system only needs to be deployed on the receiver's side, neither requires any support from the sender nor changes RTP's behavior, and as such is easily and incrementally deployable. Recovery works well even at very high bit error rates up to 10%, recovering most packets and almost never recovering incorrectly. Our main envisioned application scenarios are VoIP and audio conferencing, in which due to small payload sizes, header recovery will produce high relative gains over payload-only error tolerance such as in UDP-Lite, and in which many codecs support bit error tolerance. However, the basic concept is also applicable to other scenarios that use RTP.

II. System Design

We will first explain the concept of heuristic header recovery, followed by a short introduction to RTP. Then, we discuss the details of the recovery process for this protocol.

A. Heuristic Header Error Recovery

Our scheme leverages the fact that at any given time, a protocol has expectations about the contents of headers of received messages. For example, RTP encapsulates media data into one or multiple so-called streams. For every received packet, RTP requires the packet's header contents to match values expected for one of the streams. If it cannot match the packet to any of them, the standard behavior is to discard it.

To heuristically repair header errors, instead of discarding erroneous packets, we assign them to the stream whose expected header values best match the received values. As similarity metric, we employ Hamming distances, which have several advantages: they are computationally inexpensive to calculate, and their similarity metric is independent of the positions of bit errors.

However, if a packet was corrupted in a way that makes it resemble a different stream's headers more closely than

the original one, it will be assigned to the wrong stream. This problem of *misattribution* is inherent to the system. Since our solution focuses on error-tolerant media codecs, assigning data to the wrong stream is not immediately fatal (it will rather appear to such a codec as if it received a highly corrupted payload); it is, however, undesirable, because the correct stream loses data, while another stream will have to cope with unrelated data. Our main goal is therefore to maximize correct identification, while reducing misattribution to very rare occurrences.

After recovery, the packet headers can be repaired by replacing header values with those expected by the stream the packet was assigned to. This is not strictly necessary, but generally advisable: it allows standard, unchanged protocol routines to process the packet properly.

B. The Real-time Transport Protocol

The Real-time Transport Protocol (RTP) is used for a wide range of streaming and conferencing scenarios. For example, many VoIP solutions combine RTP as the streaming protocol with session protocols such as SIP [5] into a telephony system.

RTP uses so-called profiles that define encodings for media codecs. They can even, to a certain extent, modify the size and existence of header fields. For this work, we will focus on the baseline profile [6] standardized together with RTP itself. In this scenario, a streaming setup comprises one or more RTP *sessions*. For example, a videoconference system is expected to use two sessions concurrently, one for video and one for audio, to separate the two media types. Each session comprises one or more *streams* that identify logical units of data. For example, each stream applies sequence numbers to its packets independently of other streams within the same session. To identify streams within a session, each stream uses a *synchronization source identifier* (SSRC) as unique ID.

Different RTP sessions are typically managed by different underlying protocol connections and will use different transport-layer ports. This means that "cross-talk" between sessions can be ignored for the purposes of RTP error tolerance.[1] Thus, our heuristic repair only has to correctly identify streams within a session, not sessions within a complete RTP setup.

C. Header Field Categorization

To support our header recovery scheme, we categorize the fields of the RTP header into three classes. Due to space constraints, we will not explain each header field's categorization in detail, but merely give examples for each category.

Static fields are fields that do not change in the lifetime of a stream. They are either the same for all RTP packets (e.g., the version field), or different, but immutable, for each stream (e.g., the SSRC). These fields are trivial to repair, because we know their values for each stream at any given point in time. We simply need to calculate the Hamming distance of the received header values to the static values of each stream.

Predictably dynamic fields change from packet to packet within a stream, but allow for prediction. To repair these fields, we need to learn their behavior to predict the possible values. For example, the sequence number is incremented by 1 with every packet. To match a received value to a stream, we need to match it against the next expected value for those fields.

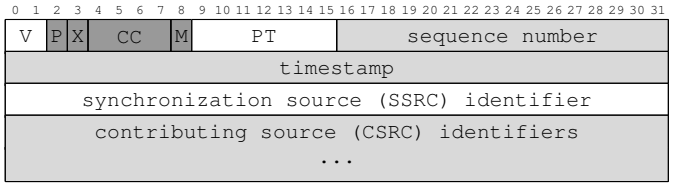

Fig. 1. A Real-time Transport Protocol header, with fields classified as static (white), predictably dynamic (light gray), and unpredictably dynamic (dark gray). Most fields are recoverable with our header recovery scheme; only seven bits are classified as unrecoverable, and even those are recoverable in many standard situations.

Unpredictably dynamic fields change from packet to packet within a stream irregularly. For example, the marker bit is used to signal events. Its use is codec dependent and can denote events such as the beginning of talk spurts or the last packet of a multi-packet video frame. These fields cannot be predicted, and therefore, errors in those fields are unrecoverable. Note that this does not lead to outright drops of the packet, but rather to potentially incorrect values in those fields.

D. RTP Stream Identification in Corrupted Packets

If a corrupted packet is received by the RTP library, the most straightforward solution is to check whether the SSRC matches one of the ongoing connections. If none of them matches perfectly, finding the closest match via Hamming distance might be the next step. Still, if the SSRC is strongly corrupted, this will be problematic. However, the matching can be improved by taking into account more header fields. Since each RTP stream uses its own progression of sequence numbers and timestamps, these can be included in the overall decision. To facilitate this, we learn additional state information from every correctly received packet, and use it to predict the correct stream for erroneous packets.

Whenever a correct packet is received, the *learner* saves its header contents for future use. For each ongoing stream, it saves the last correctly received packet in this fashion. Furthermore, it calculates the sampling rate, that is, the difference between two consecutive packets in their timestamps. As the RFC notes, this sample rate is expected to be static in audio streams. That way, whenever a corrupted packet is received, the second component, the *predictor*, can match the received header field contents against that stream's SSRC, sequence number incremented by 1, and timestamp incremented by the sampling rate. Thus, a much larger area of the header can be compared, and finding the best match becomes more stable.

This will work well until more than one corrupted packet is received in sequence. In that case, a simple incrementation is not effective, because any further packets will not have the chance to exactly match this information. So we save, for every stream, a *bad packet counter* that tracks how many corrupted packets were assigned to that stream since its last correct packet was received. We then use this value as a multiplier to the increments for sequence number and timestamp. While completely lost packets or those misattributed to the wrong stream will still cause a slight desynchronization between

[1] Identification of the correct receiver port, even under errors, is a different problem outside of the scope of this paper; however, it has been shown [3] that such identification is feasible.

received and expected information, a number of corrupted packets received in succession will not.

E. User–Kernel interface

As explained in Section II-D, our approach learns from correct packets to correctly identify corrupted packets with errors in header fields. Because RTP does not employ any checksumming, it relies on lower layers to detect errors. Furthermore, to even receive erroneous packets in the first place, it will need to instruct the operating system's network stack to let those pass. We therefore extend the standard socket interface for user–kernel space interaction as described in [3].

When our RTP library sets up a connection, it signals to the OS that it can handle erroneous packets by setting a socket option. After that, whenever the network stack hands an erroneous packet to the RTP library, it signals this as ancillary information with the packet. In our Linux implementation, we (backwards-compatibly) extend the `recvmsg` syscall for this. To receive erroneous packets in the first place, the network stack's error handling has to be changed. In this paper, we abstract from this problem and assume a solution such as in [3].

III. Implementation

For this work and to evaluate our concepts, we implemented the learner–predictor scheme for heuristic header error recovery into the oRTP [7] library (version 0.16.5), an open-source library that was easily adaptable for our purposes. The implementation follows a minimally invasive approach that interferes with the standard behavior of the RTP packet handling as little as possible. This is advantageous for such tasks as statistics collection that can be used to inform the sender about current reception conditions via RTCP [4] to potentially decide on reactions to improve streaming quality.

Whenever a correct packet is received, the learner takes the header of the RTP packet and saves it to its list of current streams, indexed by SSRC. In addition, it calculates the sampling rate between the received and the last saved packet, and resets the bad packet counter to 0.

Whenever an erroneous packet is received, the predictor iterates over the list maintained by the learner and matches the received header to the expected headers of each stream as described in Section II-D. After the predictor has decided which stream the packet most likely belongs to, it will attempt header repair by copying the saved header from the predictor's list over the received header, updating the sequence number and timestamp value accordingly. Thus, subsequent routines of the oRTP library do not have to be changed to introduce error handling, since the header is now guaranteed to be coherent. Finally, it will increment the bad packet counter of that stream.

The ease of this approach and the minimal changes in the oRTP library suggest that similar changes in other RTP implementations should be similarly easy and fast to implement.

IV. Evaluation

Since the main advantage of our heuristic header error recovery scheme is that erroneous packets can be assigned to a stream, our evaluation focuses on two packet-delivery-related metrics: (1) How often can a packet be delivered to the correct stream? And (2) how often does the heuristic approach misidentify the stream the packet belongs to, and misattributes it to the wrong stream? To answer these questions, we will present several setups that we evaluated. We will start with a description of the evaluation setup before discussing results.

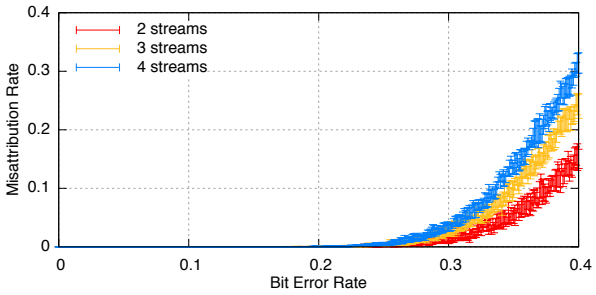

Fig. 2. Misattribution rates for two, three, and four concurrent streams with no cutoff (i.e., the best match is always taken, even if its Hamming distance is large; packets are never dropped). For each bit error rate (increments of 0.001 from 0 to 0.5), the mean and 95% confidence intervals are shown. Even at a bit error rate of 20%, we witnessed virtually no misattributions.

A. Experimental Setup

To eliminate influences from layers below RTP that could skew our evaluation results, we exchanged the standard network socket interface of the oRTP library with Unix Domain Sockets. These allow data exchange between processes in a similar way to network communication, without additional protocols being used. Thus, RTP packets could be exchanged between two instances of the library running on the same machine, without changing the behavior of the implementation. To introduce errors into the RTP packets, we connected the two instances via a simple packet destroyer. It introduces bit errors into a data stream with a defined probability p. For each bit, it rolls a random number between 0 and 1, and flips the bit if the number is lower than p, implementing a Bernoulli process.

We investigated three scenarios that differed in the number of concurrent streams in the RTP session. While a single-stream session is arguably the most common use case for RTP, especially in the case of VoIP telephony, this case is also not very interesting to evaluate. In fact, in a single-stream scenario, since there is no risk of misattribution, our repair technique will be able to correctly assign *every* packet, regardless of error rate. To investigate the possible downsides of our scheme, we therefore looked at more challenging scenarios with several concurrent streams. In every experiment, the sender sent 10 000 packets for each stream. Each stream's first two packets were not corrupted to populate the list of known streams. This models a scenario with successful connection setup and subsequent link quality degradation. Experiments were repeated 10 times for every data point. This is important because oRTP follows the RFC's advice to randomize SSRCs, initial timestamps, and sequence numbers. This influences the robustness of the heuristics that depend on how large the Hamming distances between values of different streams are. Error bars in graphs denote 95% confidence intervals. In some cases we removed error bars to preserve lucidity.

B. Misattribution

As a first step, we ran our experiments for two, three, and four concurrent streams. In this setup, every packet was

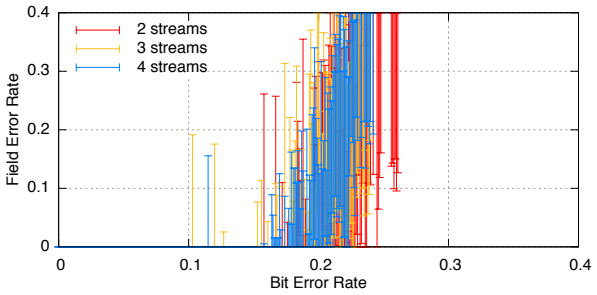

Fig. 3. Errors in header fields due to incorrect repairing. The high variance is due to error propagation that leads to incorrect repair of header fields in a large number of packets after a single misattribution, especially at high BERs.

assigned to the stream it most likely belonged to, without discarding any packets. The results from this experiment are shown in Figure 2. As expected, misattribution increases with bit error rate (BER) and the number of concurrent streams. High BERs lead to more corruption in the header; as an extreme case, at 50% BER, a header will form a random bit sequence. As the number of concurrent streams in an RTP session increases, the Hamming distances between streams will decrease on average, making it harder to distinguish them.

Our heuristic header error recovery scheme produces almost no misattributions up to BERs of 20%. For comparison, BERs of more than 1% can lead to almost 100% packet loss in standard systems, and even if data reaches the codec, most voice codecs will start showing noticeable degradation at 20% BER [8].

C. Field Errors

Misattribution is only one type of error that can occur in heuristic recovery. Packets are not only assigned to a stream, but also repaired to the values that this stream expects in the next packet's header. Thus, header fields can be wrongly repaired. The occurrences of these *field errors* are shown in Figure 3. The high error rates and large uncertainties occur due to error propagation. As an example, consider two concurrent streams A and B that expect s_A and s_B as next sequence numbers. If packet n belongs to stream A, but is misattributed to B, its sequence number will be incorrectly repaired to s_B. In addition, packet $n+1$'s sequence number will now also be repaired incorrectly. If it belongs to A, it contained $s_A + 1$, which will be repaired to s_A. If it belongs to B, it contained s_B, which will be repaired to $s_B + 1$. This shift by 1 will continue until the stream "resynchronizes" after receiving a correct packet. At high BERs, most packets are corrupted, so that it can take a long time until this error is corrected.

Two facts are of note: (1) The field error rate overestimates the impact of errors. In the sequence number case, while a large number of them might be incorrect, they are simply shifted. Interruptions in the regular pattern only occur at the time of misattribution and resynchronization. These are the only points in time playback would be negatively affected. (2) Because these errors are a secondary effect of misattribution, they do not occur at BERs below 10%. Again, this is much higher than the typically tolerable BER for media transmissions.

D. Reduction of Misattribution

The last two sections showed that our heuristic header error recovery scheme only produces errors under such high BERs that packets would typically get lost before these occur. However, in cases such as short-term interference, the RTP header could experience a much higher BER than the lower-level protocols. In this case, packets would reach the RTP library without problems, but the high header BER could lead to misattributions. We therefore investigated when misattribution occurred, so that we could further reduce it. To be of practical use, we only examined information that was available to the RTP library. One such information is the Hamming distance to the best match. Low distances mean a very close match, while high ones mean only rough resemblance. Thus, the probability that a misattribution occurs should be higher whenever a high distance to the closest match is observed.

The results of this investigation are shown in Figure 4a. Indeed, misattributions are virtually nonexistent at distances of less than 20. For comparison and reference, a minimum RTP header has 96 bits. 20 bits translate into more than 20% BER, which matches our results from Figure 2, in which we witnessed almost no misattributions until that BER.[2] Considering this, we changed our scheme to drop all packets that show a high Hamming distance to the best match. The goal is to reduce misattributions at high BERs, and to drop those suspicious packets instead. Figure 4b shows misattribution rates for four different Hamming distance cutoffs in a 4-stream scenario. We focus on the 4-stream scenario because it produced the highest misattribution in our initial experiments. The results show that cutoffs are very effective at reducing misattributions. Even at a lenient cutoff of 24, misattributions are reduced by almost two orders of a magnitude; cutoffs of 20 and, even more so, 18 almost completely eliminate them. Unfortunately, we now also drop suspicious packets that would have been assigned to the correct stream. Figure 4c shows that we indeed increased the drop rate compared to the misattribution rate of Figure 2. For the strictest cutoff value of 18, drops start to occur at about 4% BER, and reach 10% drop rate at 10% BER. The less strict cutoff of 20 only has a drop rate of about 1% at 10% BER, and the most lenient cutoff of 24 rarely drops any packets until the BER is in excess of 15%.

Judging from these results, we suggest a cutoff of 20 bits as a good tradeoff. This should effectively prevent most misattributions in most situations (single-digit number of concurrent streams, independently of BER), while still only regularly dropping packets when the BER reaches excessively high values above 10%.

V. Related Work

While the concept of heuristically recovering from header errors is relatively unique, related work can be roughly separated into two fields: optimizing the reception and retransmission of data, and ignoring errors in packet payloads.

Considering the first group, Maranello [9] improves ARQ by partitioning packets, calculating and sending partial checksums, and letting the receiver identify corrupt blocks that are selectively retransmitted. PPR [10] uses soft information (per-bit error probabilities) to recognize erroneous parts in packets and selectively retransmit those. Soft information can also be

[2]Note that Figure 4a shows the absolute number of misattributions in our experiments, which is why the numbers decrease again at high Hamming distances since these are less likely to occur.

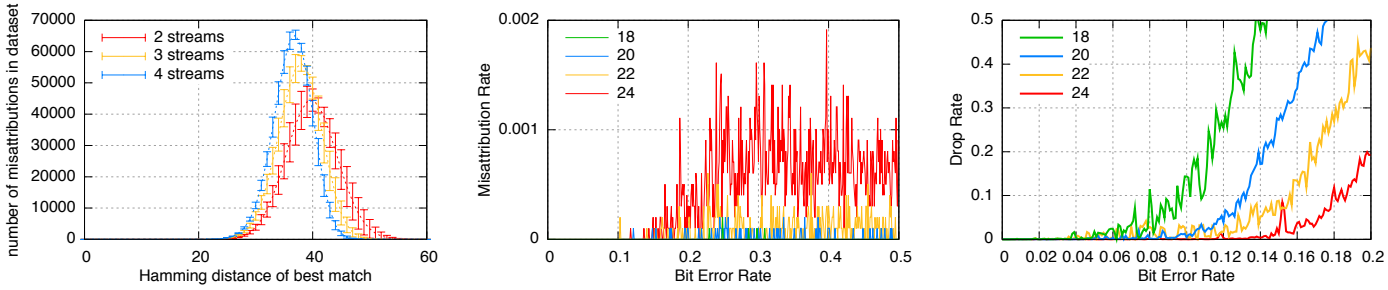

Fig. 4. Misattribution occurs due to high bit error rates. (a) A side effect of high BERs is that even the best match often shows a high Hamming distance to its expected header values. Hamming distance can be used as an estimator for risk of misattribution. (b) The result is a dramatic reduction in misattribution. Even with 4 streams at a cutoff value of 24 (one quarter of the 96 bits of a standard RTP header), misattribution stays below 0.2%, regardless of BER, and becomes exceedingly rare at stricter cutoffs. (c) The tradeoff is an increased drop rate. However, even strict cutoffs only show minimal drop rates until BERs$>5\%$.

used to reconstruct a correct packet from several receptions, either due to spatial diversity, or from retransmissions. Examples include SOFT [11] (spatial diversity), ZigZag [12] (retransmissions), and MRD [13] (both). All of these approaches either require deployment on all nodes, or special hardware that allows access to physical layer soft information, or both.

The second group is most prominently represented by UDP-Lite [2], a UDP derivative that redefines the "length" header field as "checksum coverage". This means that both sides need to understand UDP-Lite, as it is a different transport layer protocol. UDP-Liter [14] solves this by enabling applications to receive packets with UDP checksum errors. However, UDP-Liter does not provide mechanisms to recover from header bit errors. Packets with errors in the UDP header are lost, as are packets with errors on lower layers.

With respect to heuristic recovery, Jiang [15] and Schmidt et al. [3] proposed solutions for header bit errors. Both rely on Hamming distances as similarity metric. Jiang [15] focuses on header recovery for some static fields in the 802.11 MAC header, such as MAC addresses, as opposed to sequence number fields, which are not considered. Schmidt et al. [3] aim at recovering headers in the IP and UDP protocol. Again, the focus is on the static fields that form the bulk of information in those protocol headers. In contrast, this paper proposes a simple but effective way to deal with dynamic header fields that follow regular patterns, such as the RTP timestamp field.

VI. Conclusion

In this paper, we presented a heuristic header error recovery scheme for RTP that even in case of header errors identifies the stream a packet belongs to and repairs those fields. We showed that our scheme is robust up to bit error rates of 10%, very rarely assigning packets to wrong streams or repairing incorrectly, while still keeping packet drop rates low. This holds true even more at more modest BERs that are realistically tolerable by media codecs.

One field of future work is to make the static Hamming cutoffs from Section IV-D dynamic, adapting to both the number of concurrent streams and the Hamming distance in their SSRCs, timestamps, and sequence numbers, to further optimize the tradeoff between misattribution and drop rate.

Overall, we consider the work presented in this paper a feasible, simple, and effective approach to support an error-tolerant application in the reception of partially erroneous data.

Acknowledgments

This research was funded in part by the DFG Cluster of Excellence on Ultra High-Speed Mobile Information and Communication (UMIC).

References

[1] F. Hammer, P. Reichl, T. Nordström, and G. Kubin, "Corrupted speech data considered useful: Improving perceived speech quality of voip over error-prone channels," *Acta acustica*, vol. 90, pp. 1052–1060, 2004.

[2] L.-Å. Larzon, M. Degermark, S. Pink, E. Jonsson, and E. Fairhurst, "The lightweight user datagram protocol (UDP-Lite)," IETF, RFC 3828, Jul. 2004. [Online]. Available: http://www.rfc-editor.org/rfc/rfc3828.txt

[3] F. Schmidt, M. H. Alizai, I. Aktas, and K. Wehrle, "Refector: Heuristic header error recovery for error-tolerant transmissions," in *Proc. ACM CoNEXT*, Dec. 2011, pp. 1–12.

[4] H. Schulzrinne, S. Casner, R. Frederick, and V. Jacobson, "RTP: A Transport Protocol for Real-Time Applications," IETF, RFC 3550, Jul. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3550.txt

[5] J. Rosenberg, H. Schulzrinne, G. Camarillo, A. Johnston, J. Peterson, R. Sparks, M. Handley, and E. Schooler, "SIP: Session Initiation Protocol," IETF, RFC 3261, Jul. 2002. [Online]. Available: http://www.ietf.org/rfc/rfc3261.txt

[6] H. Schulzrinne, "RTP Profile for Audio and Video Conferences with Minimal Control," IETF, RFC 3551, Jul. 2003. [Online]. Available: http://www.ietf.org/rfc/rfc3551.txt

[7] "oRTP, A Real-Time Transport Protocol (RTP, RFC3550) library," [Online] Available http://linphone.org/eng/documentation/dev/ortp.html.

[8] S. Nguyen, C. Okino, C. Loren, and W. Walsh, "Space-Based Voice over IP Networks," in *Proc. IEEE Aerospace Conference*, March 2007.

[9] B. Han, A. Schulman, F. Gringoli, N. Spring, B. Bhattacharjee, L. Nava, L. Ji, S. Lee, and R. Miller, "Maranello: practical partial packet recovery for 802.11," in *Proc. NSDI*. USENIX Association, 2010.

[10] K. Jamieson and H. Balakrishnan, "PPR: Partial Packet Recovery for Wireless Networks," in *Proc. SIGCOMM*, August 2007, pp. 315–326.

[11] G. Woo, P. Kheradpour, D. Shen, and D. Katabi, "Beyond the bits: cooperative packet recovery using physical layer information," in *Proc. MOBICOM*. ACM, 2007, pp. 147–158.

[12] S. Gollakota and D. Katabi, "Zigzag decoding: combating hidden terminals in wireless networks," in *Proc. SIGCOMM*. New York, USA: ACM, 2008, pp. 159–170.

[13] A. Miu, H. Balakrishnan, and C. E. Koksal, "Improving loss resilience with multi-radio diversity in wireless networks," in *Proc. ACM MOBICOM*, 2005, pp. 16–30.

[14] P.-K. Lam and S. Liew, "UDP-Liter: an improved UDP protocol for real-time multimedia applications over wireless links," in *Proc. Wireless Communication Systems, 2004*, Sep. 2004, pp. 314–318.

[15] W. Jiang, "Bit Error Correction without Redundant Data: a MAC Layer Technique for 802.11 Networks," in *Proc. WiNMee*, Apr. 2006.