Effectively Capturing Attention Using the Capture Effect

Michael König
Distributed Computing Group
ETH Zürich, Switzerland
mikoenig@ethz.ch

Roger Wattenhofer
Distributed Computing Group
ETH Zürich, Switzerland
wattenhofer@ethz.ch

ABSTRACT

We propose a new class of wireless transmission schemes decoupling synchronization headers from payloads to create new transmission primitives involving a second sender. By transmitting a synchronization header only we can let nearby nodes receive fragments of a packet without having to receive that packet's synchronization header, and by using the capture effect we can overwrite portions of the payload of longer ongoing packets.

We explore two scenarios potentially benefiting from such schemes. A) First, we consider crossing a network chasm over which all links are of poor quality: by broadcasting a fabricated packet header on the receiving side of the chasm all receiving nodes are informed to record the packet to the best of their ability. B) Second, we investigate the insertion of short high-priority packets into longer lower-priority transmissions from a different sender. This has the advantage that high-priority senders do not need to wait for the medium to become free but can begin sending at once, while receivers lose only the affected portion of their already incoming low-priority packets.

Further, we examine two techniques to reduce the amount of symbol decoding errors caused by using a mismatching synchronization header: 1) careful transmission timing and 2) correction of deterministic symbol decoding errors. In scenario A) these techniques improve the chance of every part of a packet being received successfully by some node on the receiving side of the chasm from 5% to up to 30%. In scenario B) we reach successful decoding of the injected packet in up to 70% of cases.

In proof of concept implementations on a testbed of TelosB nodes we confirm the soundness of our methods.

CCS Concepts

•**Networks** → **Link-layer protocols; Wireless access networks;** Sensor networks; •**Hardware** → Wireless devices; Wireless integrated network sensors;

Sensys 2016 November 14–16, 2016, Stanford, CA, USA

ISBN 978-1-4503-4263-6…$15.00

DOI: http://dx.doi.org/10.1145/2994551.2994560

Keywords

Capture Effect, Packet Injection, Power Control, Concurrent Transmission, Medium Access Control, Wireless Sensor Networks

1. INTRODUCTION

Wireless networks have been around for a long time, and almost always the participating nodes adhere to a simple setup: There is a sender which transmits data to a single receiver or a set of receivers. More recently, this setup increasingly includes multiple antennas on both sides (multiple-input and multiple-output, MIMO). However, conceptually, we still have a single sender trying to transmit a packet to a (set of) receiver(s).

In this paper, we introduce a third party, a "third man" in the form of a second sender. This second sender tries to capture the attention of the receiver(s) using the capture effect. The second sender ignores the standard rules of wireless communication, and blatantly transmits *during* the transmission of the first sender. The second transmitter could transmit during the payload of the original packet, or during its header. We want the second sender to be stronger than the first sender, either by being closer to the receiver(s), or by transmitting with more power.

We present two scenarios benefiting from such unorthodox behavior. However, we believe that the general idea of having a second sender may have potential beyond our two examples.

Our first scenario addresses networks with partially low density, e.g., where a few nodes failed and the network was separated into two or more parts that can hardly communicate with each other. Consider the case of a spread out multi-hop wireless network containing a "chasm", i.e., a virtual or physical gap between two parts of the network across which links are very poor, as illustrated in Figures 1 and 2. Even if the network still forms a connected graph via routes around the chasm using only stable links, these detours incur a large latency and medium usage overhead due to the additional hops packets have to make.

By sending across the chasm we might collect several partially incorrectly received packets at multiple nodes at the receiving side. These erroneous copies may then be pieced together to obtain a completely correct copy of the sent packet. However, if already the synchronization header is not received correctly at a node, that node will not recognize the incoming packet and hence will not be able to record it. By employing a node broadcasting a fake synchronization header on the receiving side, we are able to ensure that

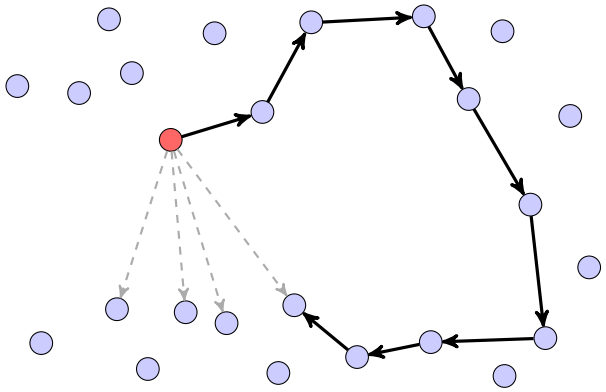

Figure 1: Chasm scenario: connection from the red node to bottom half of the graph via A) stable edges around the right side versus B) unstable crossing edges (dashed and in gray).

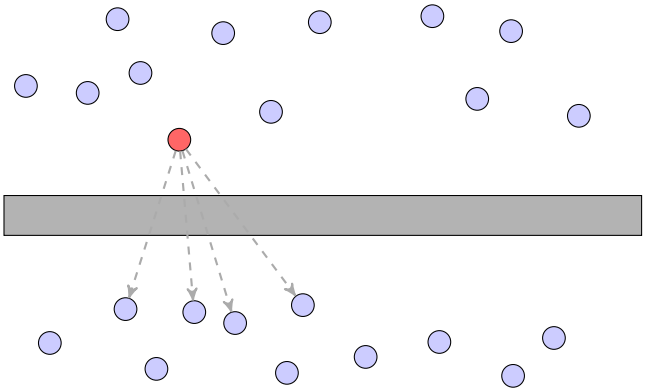

Figure 2: Alternative chasm scenario: the network is partitioned by a wall without any stable edges crossing it.

every node which is a candidate for receiving at least parts of the packet is listening. This improves the chances of correctly receiving every part of the packet in some cases from less than 5% to about 30%.

The second scenario we discuss makes use of what we call the packet-in-packet communication primitive: Here, the second sender does not transmit during the header, but during the payload of the first sender's packet. As such, the second sender may inject a short packet into concurrently ongoing transmissions using the capture effect. Doing so, the time and control message overhead for avoiding collision between high-priority and low-priority traffic can be reduced to a minimum. This is particularly beneficial when the high-priority traffic consists of short spontaneous bursts.

Naively sending a high-priority packet into an ongoing transmission with a sufficient power differential for the capture effect to occur (about 4-5 dB on TelosB) results in the inserted packet being decoded correctly in only about 5% of cases. However, with proper improvements, we can bring this value up to 70%.

In particular, we apply two techniques to improve the success rates in both scenarios: 1) improving transmission synchronization and 2) repairing misinterpreted symbols where possible.

1) By improving transmission synchronization we increase the chance of the symbols of the strong/injected and the weak/base packet to overlay closely enough, such that the symbols have a high chance of being decoded properly. In the chasm scenario this results in a complete packet transmission in 25% of cases. For packet-in-packet this improves the correct decoding rate to about 40%.

2) Among the remaining bad cases we observe many to experience a deterministic mapping of values on the injected symbols. By reversing this mapping we are able to repair some of the bad cases, raising the total correct decoding rate to 30% in the chasm scenario and to 70% in the packet-in-packet scenario.

We implemented proofs of concept of these techniques based on Contiki for both scenarios on the popular sensor node TelosB and tested them on the office building testbed FlockLab [14].

2. RELATED WORK

In recent years, wireless sensor networks have been an area of active research. The inherent multi-hop topology as well as the unique properties of the wireless medium enabled the inception of several new communication primitives. In particular, most recently, ways to induce constructive interference through concurrent forwarding of a single transmission have been explored [4, 3]. Constructive interference can boost signal quality and allows rapid network flooding without the overhead of a carefully planned flooding schedule. Another example is the deliberate use of the capture effect, which was previously thought of as a nuisance. Yet it is now counted on more and more to resolve collisions without experiencing destructive interference, which most classical models would predict [22, 11]. Our work mainly falls into this field of work on medium access primitives. However, one of our scenarios also deals with the topic of unreliable links.

Lu et al. [15] proposed the *Flash* flooding protocol, in which a flooding schedule is forgone in favor of letting every reached node simply broadcast a few times. The capture effect enables correct reception at nodes receiving packets from different neighbors at sufficiently different strengths. The protocol also implements fallback mechanisms to ensure flooding is able to proceed at nodes which experience only destructive interference due to the arriving signals being too similar in strength. This approach was shown to reach flooding latencies close to the theoretical optimum, reducing previous latency values by up to 80%.

Flury et al. [5] proposed *Slotos*, which among other concepts features an alarm propagation system, in which alarms are propagated as a simple recognizable waveform. In this system, whenever a node would notice the alarm waveform, it would start sending this waveform itself. Due to the capture effect and only requiring a short glimpse of the alarm waveform, destructive interference is not an issue, and the alarm is able to spread quickly and reliably. Clearly, the downside of this approach is its inability to carry meaningful data beyond the presence of the alarm itself. Gotzhein et al. [7] presented *Black Burst Synchronization*, employing a similar approach, offering multi-hop synchronization via collision-resistant "black bursts".

Dutta et al. [2] discovered that automatic acknowledgments of multiple nodes having received the same packet will

in fact be synchronized so well as to cause constructive interference. Automatic acknowledgments are a common wireless transceiver feature that allows an immediate response to correctly received packets (as verified by the packet's checksum). As the acknowledgment is completely handled by the hardware, this feature avoids any variable desynchronizing delays the microprocessor software may cause.

Ferrari et al. [4] took this idea of synchronizing to the end of a previously received packet one step further and proposed *Glossy*: instead of sending an acknowledgment, which does not allow for the transmission of meaningful data, they prepare a full response packet during the reception of the incoming packet. By immediately issuing its transmission when the transceiver indicates the completion of the reception of the incoming packet the variable delays microprocessor software can introduce is minimized. As the packet contents sent by the different responders still have to be the same to allow constructive interference, this approach is best-suited for flooding. The constructive interference leads to improved signal strengths, making weaker and longer links more viable. Further, the rapid forwarding leads to exceptionally short flooding durations. Several recent protocols are based on the rapid data dissemination capability of Glossy [3, 12, 1, 24] and various studies on its reliability and the conditions for creating constructive interference have been conducted [25, 23].

For the scenarios examined in this paper, a Glossy-like mechanism is not applicable, as 1) the presence of a common reference packet for all senders is not guaranteed and in fact impossible in the case of the chasm and 2) these scenarios aim to have the two senders start sending at different times rather than simultaneously.

Unreliable links have been studied from various different angles. Zuniga et al. [26] proposed an analytical model predicting the behavior of links in the transitional region between stable and nonexistent communication. On the other hand, adapting behavior to deal with intermittently missing links has been proposed: Su et al. [20] developed a set of routing algorithms dealing efficiently with intermittent outages in links, while Seth et al. [19] came up with a system to facilitate communication in rural areas piggybacking on vehicles.

Santhapuri et al. [17] proposed a hardware feature allowing a transceiver to disengage from the reception of an ongoing transmission and lock onto a newly started stronger one. To do so they proposed the hardware keep scanning for syncwords even during transmission reception, similar to the way we detect injected packets. However, the approaches we present in this paper do not rely on such hardware support, but instead make do with the capabilities of run-of-the-mill transceivers. Further, in most cases of the packet-in-packet scenario we are able to recover after the injected packet and correctly receive the remaining tail end of the original packet.

To the best of our knowledge, cleanly injecting and decoding packets in packets as we study in this paper has not been attempted yet.

3. CONCEPTS

3.1 Preliminaries

In the following we will discuss the basics of wireless receiver operation relevant to our undertaking. The details

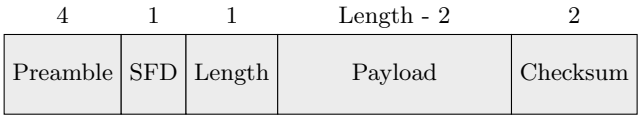

Figure 3: IEEE 802.15.4 PHY layer packet format plus checksum footer (technically part of the MAC layer) [8]. The values on top designate the length of the respective segment in bytes. The preamble consists of only zero symbols and the SFD (Start of Frame Delimiter) is a pair of constant symbols. The checksum is a simple 16-bit CRC value of the payload.

of the parts of a transmission vary slightly between wireless standards, but the presented primitives are only loosely tied to the IEEE 802.15.4 standard [8] we used. See Figure 3 for an overview of the IEEE 802.15.4 packet format.

When a packet is transmitted it is usually prepended a preamble and a synchronization word (sometimes called start of frame delimiter, SFD). Together, the preamble and the synchronization word form the synchronization header. The preamble is a predefined pattern of symbols used to let listening receivers synchronize to the correct phase of symbols and chips. In IEEE 802.15.4 it consists of '0' symbols only. The synchronization word is a constant string of symbols to mark the start of the packet, which signals receivers to start recording the following symbols. If a receiver does not hear the synchronization word of a transmission, it will not decode the signal and perceive it as noise.

At the head of a packet there usually is a length field specifying the number of bytes in the payload, and thus also the duration of the transmission. Receivers use this field to know how many symbols to decode. A common feature of receivers is to "lock on" to a transmission once they received its header including the length: they commit to decoding that packet in its entirety no matter how bad the signal quality may get. This is intended to allow the application to still make use of the non-broken parts of a packet even if part of the packet reception was disrupted.

Finally, a packet usually contains a checksum over its length and payload to allow telling whether the whole packet was received correctly. However, if there are any errors, the checksum does not help in determining their location.

The capture effect occurs when multiple signals arrive at a receiver concurrently and the strongest signal is stronger than the noise plus the other signals by some threshold. In this case, the receiver will decode the strongest signal without error, completely ignoring the other signals. When a receiver is already receiving a packet, it does not scan for further synchronization headers. Thus, an overlapping packet may not be recognized as a packet, even if it is strong enough to induce the capture effect. Instead, a stronger but later packet essentially writes its data over the payload of the weaker packet as it is being received. This means that the overwritten data of the weaker packet is lost, but it also means that the stronger packet is not lost. In many cases the receiver can find the stronger packet's data (and headers) either directly in the weaker packet's payload (if the two packets' symbol phases were well-synchronized) or after descrambling the received symbols (see Section 6).

```
1b1f21ffdedededededededededededededededede
1b1f229fd3dedededededeed2deddedededededede8ede
1b1f21ffdeded4dedededededededededededededede
1b1f21ffde2ede1ed77eddd92ed2ded0deded7dede6e94
```

Figure 4: An example of a single short packet after being received by 3 different receivers, exhibiting seemingly independent decoding errors. The first line shows the packet as it was sent.

3.2 The Chasm

Consider the situation described in Figures 1 and 2: traditionally, to send a packet to the other side of the chasm, one would use the multi-hop route using the stable edges. This route, however, causes a high latency for the packets and incurs a noticeable overhead in medium usage. Worse, sometimes this route may not even exist, and without being able to send across the chasm the network graph would be disconnected.

In the chasm scenario we isolate this situation and explore what is possible using the weak links only, hence avoiding the detour. We assume the weak links to have a near zero chance of transmitting a complete packet correctly, while still being capable of correctly decoding some symbols now and then. If after a cross-chasm transmission the nodes on the receiving side exchange what symbols they got, they may be able to piece together the complete packet and verify its correctness using the checksum field.

For this setup to be viable, errors in the transmission should not occur at or near the sender. Instead, they should occur in the decoding stage at each receiver, independently of the other receivers. We found this assumption to be correct in our setup by comparing the received symbols of multiple receivers for the same packet. An illustrative example can be seen in Figure 4.

As mentioned above, the receivers will in fact only record incoming data if a correct synchronization header is received. We cannot make the assumption that this will always be the case for our receivers in the chasm scenario, since the symbols of the synchronization header are just as likely to not be received correctly as any of the data symbols. In other words, we will not get any information about the packet from the portion of receivers which failed to recognize the synchronization header.

To solve this problem we propose having another node on the receiving side of the chasm send out a synchronization header as well as a large packet length field, but not transmit any packet payload, just before the cross-chasm packet arrives (see Figure 5). This header should be easily recognizable by all the receivers due to their proximity and cause them to start recording symbols.

Note that this does require knowledge of when the next cross-chasm packet will arrive. This may be determined from a pre-determined schedule or be agreed upon in a previous cross-chasm packet. It also implies maintaining a synchronization error low enough to ensure the cross-chasm packet is sent during the fake packet. In IEEE 802.15.4 the duration of a maximum length payload is about 5.1 ms. This dictates a maximum absolute synchronization error of 2.5 ms for short cross-chasm packets. The longer the cross-chasm packets are, the more stringent the synchronization needs

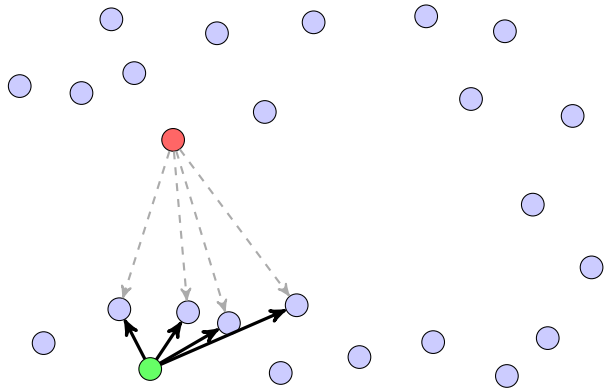

Figure 5: By having the green node send out a synchronization header first we can ensure that all the nodes with a chance of receiving parts of the cross-chasm packet are indeed listening and recording symbols.

to be, up to a symbol level synchronization at maximum length. An alternative would be to try and detect packets by monitoring the energy levels on the channel. However, this approach is prone to false positives from noise and non-chasm transmissions. Due to the large overhead incurred, we propose to employ this primitive only when the detour alternatives are prohibitively long or nonexistent.

In our implementation we pre-assign all nodes their respective roles (cross-chasm sender, wakeup sender and receivers) and establish a proof of concept for the ideas above. We consider short cross-chasm packets only and no other traffic within the network. Integrating the transmission primitive into a more general and adaptive protocol is left to future work.

3.3 Packet-In-Packet

For this scenario imagine a network of nodes being used both for low-priority bulk data transmission and short, irregularly occurring high-priority messages. One traditional approach would be to use a schedule guaranteeing time slots for high-priority traffic. This comes with a large overhead to the low-priority traffic even during times without high-priority traffic. To avoid the overhead of scheduling one could instead use an opportunistic channel access scheme: trying to send as soon as the channel appears free (using clear channel assessment, CCA). This approach has trouble guaranteeing traffic prioritization in busy networks as low-priority traffic may starve out high-priority traffic. Additionally, the hidden and exposed terminal problems become problematic [9]. To improve traffic prioritization, one could impose time-slotting on the network. By then delaying low-priority messages by a constant time high-priority messages are allowed to access the channel first. This, however, still does not solve the hidden and exposed terminal problems.

Further, traditional approaches have in common that they require high-priority messages to wait at least until all currently ongoing conflicting low-priority messages have finished. The packet-in-packet primitive we propose makes use of the capture effect with the aim to allow high-priority messages to be sent regardless of any ongoing low-priority messages, i.e., immediately upon traffic emergence. If a

low-priority message is already being received when a high-priority message arrives, it will overwrite and be decoded as part of the low-priority message's payload. We say, the high-priority message is *injected* into the low-priority message. Again, we present a proof of concept implementation of this basic primitive. Integrating it into a proper protocol or a MAC layer is left to future work.

For this scheme to work we need to be able to reliably induce the capture effect, i.e., we need high-priority traffic to always be at least 4-5 dB stronger than any low-priority traffic at the intended receivers. This can be accomplished one of two ways: 1) placing the receiver significantly closer to the high-priority sender than to the low-priority sender, or 2) using transmit power control, which requires the used links to be capable of operating at different transmit power settings. If this condition is fulfilled, no additional overhead is imposed on low-priority traffic beyond the data lost as a direct result from high-priority data taking its place. Otherwise, the additional hops necessitated by the unavailability of certain links too weak or too strong to induce the capture effect may offset this primitive's usefulness.

In addition, this scheme relies on low-priority packets to be at least 2-3 times longer than high-priority packets to be able to operate efficiently. This is because a tail portion of an injected high-priority message will be lost if its transmission does not finish before the end of the low-priority base packet it was injected into. This is a result of the aforementioned behavior of receivers to record exactly as many symbols as specified in the packet header of the packet whose synchronization header was recognized. The shorter the high-priority messages are in relation to the length of the low-priority packets, the less frequently such a loss occurs.

Detecting the potential presence of an injected packet is trivial using a checksum covering the whole payload as is commonly attached as a packet footer: if the checksum check fails, there may be an injected packet. To find the injected packet, one may simply search the payload for the symbol sequence of the synchronization header. Similarly, the length of the injected packet can be determined, and thus the exact portion of the payload that was overwritten is known. To verify the suspected injected packet, its own regular footer checksum can be used. Extracting further injected packets from the same base packet is possible following the same procedure. Note that we cannot distinguish the injection of a packet from the unlikely coincidence of a regular payload containing a synchronization header and valid packet length together with a matching packet checksum.

Depending on the link quality one may assume the remaining non-overwritten symbols of the packet to be correct in spite of the lack of a correct checksum. However, to ascertain the correctness in the case of an injected packet, additional checksums across parts of the payload or forward error correcting codes could be used. Naturally, the data that was overwritten will need to be transmitted again at a later time.

4. EXPERIMENT SETUP

4.1 Hardware

We employed the popular Tmote Sky sensor node (also known as "TelosB") [16] as testing hardware. It features the TI MSP430 16-bit CPU and the TI CC2420 wireless transceiver. The SFD ("start of frame delimiter") pin of the CC2420 is connected to a timer capture pin on the MSP430. This enables capturing the timer register at the exact time the CC2420 finishes sending or receiving the SFD byte at the start of a transmission. This is used in MAC layer timestamping, which we use for clock synchronization (see Section 5.1).

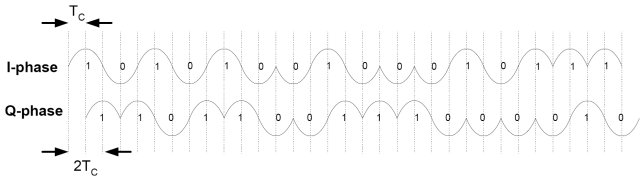

Figure 6: An example of the modulation of the pseudo-random chip sequence of 32 chips for the '0' symbol on the I and Q phases. The figure is taken from the IEEE 802.15.4 standard [8]. ($T_C = 0.5$ µs, i.e., the chip rate is 2 MChips/s)

The CC2420 conforms with the IEEE 802.15.4 standard [8] and provides features such as automatically computing a packet's checksum and inserting it into the packet footer. The packet structure was shown in Figure 3.

On the physical layer, each byte is represented by two symbols: first, one representing the least-significant 4 bits of the byte, followed by a symbol representing the most-significant 4 bits. The IEEE 802.15.4 standard assigns a pseudorandom chip sequence of 32 chips to each of the 16 possible symbol values. Each chip directly corresponds to an interval of a certain waveform on the carrier medium. Figure 6 shows an example encoding of the zero symbol, 16 microseconds in length. When receiving a symbol, the receiver will determine the received symbol to be the one whose pseudorandom chip sequence correlates the most with the received signal. We would expect the pseudorandom chip sequence design to cause a desynchronized waveform (as would be caused by a poorly synchronized injected packet) to be decoded to an "arbitrary" (pseudorandom) wrong symbol; however, as shown in Section 6, this is not always the case.

Our experiments are conducted on the FlockLab testbed [14] spread over the floor of an office building using a channel whose band experienced low to medium outside interference. We observed instances of the motivating scenarios described in earlier sections in this testbed while using each node's maximum possible transmit power. Unfortunately, the wireless environment conditions proved very prone to fluctuations caused by changing conditions such as the closing or opening of doors or changes in temperature or humidity. To obtain reproducible results, we had to rely on the transmit power control options of the CC2420. The available output powers on the CC2420 range from −30 dBm to 0 dBm with a noticeably larger degree of granularity in the higher power options.

4.2 The Chasm

We choose a sending node and a set of receiving nodes to represent the two sides of the chasm, such that there are stable links with roughly equal qualities between the sender and each receiver when the sender uses its maximum transmit power. Additionally, these links should falter at approximately the same transmit power value. We then designate a node amongst the receiving nodes to be responsible for

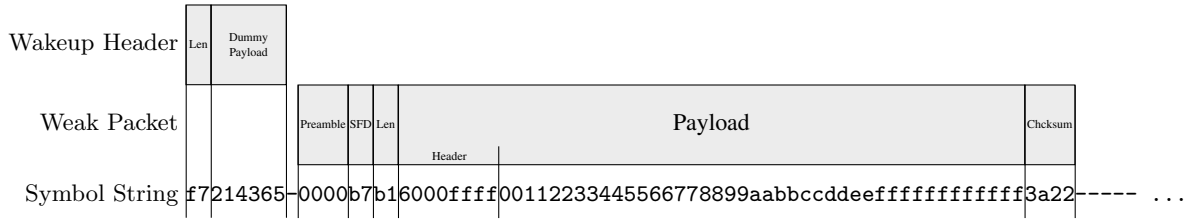

Figure 7: The symbol string the receiver would ideally receive in the chasm scenario, each letter corresponding to one symbol representing 4 bits. The symbols are given in order, i.e., each byte's least significant 4 bits come first. '-' denotes a symbol slot where no node was sending anything; the values of these slots will be determined by the noise in the environment.

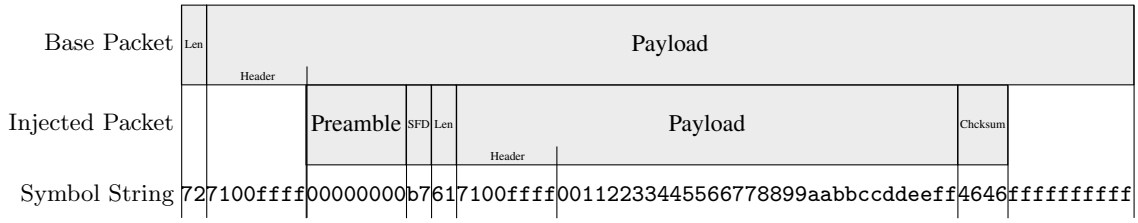

Figure 8: The symbol string the receiver would ideally receive in the packet-in-packet scenario. The individual packet fields of both the outer base packet and inner injected packet are shown. Note that the preamble, SFD and checksum of the outer packet are missing as they are already removed by the transceiver by the time software can read the symbol string.

sending the wakeup synchronization header. It should have a good link to all the other receiving nodes at its maximum transmit power. In our experiments, this node will not participate in receiving the packets from the sender. However, it might occasionally be able to, if it happens to receive the synchronization header from the sender correctly. For the results presented in this paper we used 4 receiving nodes plus the wakeup header sending node, but in practice any number of receivers is imaginable. Even a single receiver may profit from the wakeup header, although it is unlikely to receive the whole packet correctly.

The test procedure consists of several rounds in which, first, the sender broadcasts a packet at maximum transmit power for synchronization purposes. This will allow the node responsible for later broadcasting the wakeup synchronization header to update its clock and to know approximately when the weak packets will arrive. In practice, this shortcut for announcing weak packet arrival times is not available or there would be no reason to use this scheme in the first place. Instead, to achieve initial synchronization, one would need to either use a route around the chasm or rely on getting a single packet through by normal transmission, which may happen with a small percentage chance (see Figure 15).

The main part of a round then consists of several packets of length 29 (58 symbols without headers) being sent by the sender at a weaker power $P < P_{max}$. 256 microseconds before every second of these packets the wakeup node will send a synchronization header and a packet header specifying the maximum packet length (127) at maximum transmit power. If a receiving node receives a synchronization and packet header, it will store the entire received raw symbol string, no matter which of the two possible headers it received. The symbol string is then evaluated manually, ignoring the CC2420's automated checksum test. By alternating between sending weak packets with and without the wakeup synchronization header, we can directly measure the advantage gained through our method, independent of fluctuations in the environment.

To send a synchronization and packet header only, the wakeup node starts transmitting a packet of maximum length. However, it then deliberately does not supply the CC2420 with enough data bytes to continue transmitting beyond the first 3 bytes. This causes a buffer underflow to occur, automatically causing the CC2420 to stop transmitting. The receivers of this packet will, however, not notice the ceasing of the transmission and continue recording symbols to the best of their ability as discussed in Section 3.1. The chosen advance time of 256 microseconds corresponds to 16 symbols. This allows the wakeup sender to finish sending its synchronization and packet headers (8 symbols) and the 3 bytes of dummy data (6 symbols) before the weak packet from the sender arrives. Figure 7 illustrates the symbol strings sent and what is received in the ideal case.

Each round ends with every receiver reporting to the sender how many packets it received when no wakeup synchronization header was sent. These packets are sent at maximum transmit power to be able to cross the virtual chasm. Using this feedback the sender can adapt its choice of P to be able to concentrate on the interesting transmit power values: values such that no single receiver is likely to receive the packet completely all by itself. In practice, the range of interesting values settles after only a few dozen rounds, i.e., the remaining rounds all repeatedly scan a window of the same few values for P.

We noticed that over 95% of the packets that were received without a wakeup header were received completely without error. Effectively, if a link happened to be stable enough to receive the complete synchronization header, it also was stable enough to receive the packet completely without error 95% of the time. To reduce the filtering at the synchroniza-

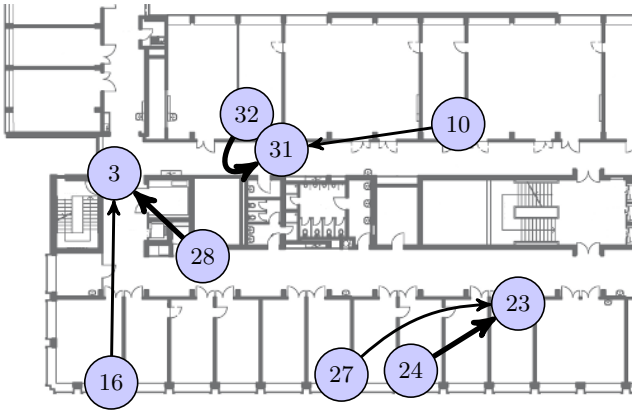

Figure 9: A subset of the wireless sensor node testbed we used for our practical tests with three particular node triplets used for packet-in-packet experiments highlighted. The second, stronger senders are indicated by thicker arrows.

tion header, we decided to use a shorter preamble of only 2 instead of 4 bytes, shortening the synchronization header to 3 instead of 5 bytes. In the interesting transmit power range, this dramatically increased the number of packets received without a wakeup header, but lowered the ratio of packets received without any error to 0-25%.

4.3 Packet-In-Packet

We choose triplets of 2 sending nodes (B and I) and 1 receiving node (R), where node B will send the base packet and node I will send the injection packet. These triplets are selected such that at node R node I's signal is at least 5 dB stronger than node B's, to facilitate the capture effect. Further, we also ensure that node B's link still has a high packet reception ratio ($> 98\%$) in normal operation without packet injections. Finally, both senders should be able to hear the receiver for synchronization purposes. If necessary we use the transmit power control options of the CC2420 to achieve this constellation. See Figure 9 for a few example triplets.

The test procedure again consists of several rounds which start with a packet broadcasted for synchronization purposes. Then, node B begins transmitting a packet of length 40 bytes at a designated time after the synchronization packet. Node I aims to begin transmission 320 microseconds after the base packet. This delay corresponds to 20 symbols or 10 bytes, ideally placing the second signal at the fourth payload byte of the base packet.

Both packets contain a short header of 4 bytes we use to store packet metadata such as sender, receiver and packet type. After, the inserted packet's payload contains 16 bytes repeating each of the 16 possible symbols twice, which will be useful for demonstrating symbol mapping, see Section 6. The base packet's payload is filled with the symbol of value 15 (or '`f`'). This choice of packet contents was made for presentation purposes in this paper. In practice, any data can be used as our tests show the packet contents not to influence the success in decoding the injected packet. Figure 8 shows the symbol string the receiver would receive in the ideal case, and details the locations and sizes of the packets' fields.

Without closer synchronization we would expect to correctly decode the injected packet's symbols in on average at most 1 of 16 cases. In the other cases we would expect the chip string to be misaligned with the symbol boundary so much that the waveform is interpreted to be a different one of the 16 pseudorandom chip sequences.

4.4 Baseline Results

Using the packet-in-packet scenario as example, we present the results one would obtain without applying either of the two improvements discussed in the next two sections.

In practice, the success rate is only 4.8%, slightly less than the predicted $\frac{1}{16} = 6.25\%$. For intuition, examine Figure 10, which shows a representative sample of 17 received symbol strings. Every letter corresponds to a symbol in symbol transmission order, i.e., for every byte the 4 least-significant bits come first. Unexpected symbol values are marked in blue. Since not a single preamble was detected, all non-'`f`' symbols are classified as errors in the body of the underlying packet.

It is easy to see the lack of tight synchronization in the variation of the first and last affected symbol of the underlying packet. However, a certain regularity in the erroneous symbols can be observed. In some cases we can exploit this to salvage some of these scrambled lines, see Section 6.

At the bottom of Figure 10 the distribution of correctly received symbols can be found. The red line for the injected packet is almost completely constant at around 5% for the duration of the injection. This means, when an injected packet's preamble was correctly decoded, the remainder of the packet was nearly always also completely without error. The uneven distribution of the base packet's successes is caused by the varying densities of '`f`' symbols caused by the injection. This is due to certain of the injected packet's symbols appearing to be more likely to be mapped to an '`f`' than others. Finally, note the "tail" of the base packet sometimes experiencing symbol errors even after the transmission injection has ceased.

5. TRANSMISSION SYNCHRONIZATION

5.1 Clock Synchronization

There are two clocks accessible to the MSP430 CPU on the TelosB: a 32 kHz (2^{15} Hz) quartz crystal and a digitally controlled oscillator (DCO), which also serves as the CPU's clock. While the DCO is considerably less reliable than the quartz crystal in both long-term clock speed stability and short-term jitter, it supports frequencies up to 5 MHz. Using Contiki we regularly adjust the DCO frequency to stay as close as possible to 2^{22}Hz $\approx$ 4MHz.

As the quartz crystal exhibits low jitter but has a low resolution and the DCO is very jittery but of higher resolution, we combine the two into a single clock. This way, we attempt to get the best from both worlds: high long-time stability but a high resolution. Effectively, we are resetting the DCO clock every quartz crystal tick, such that the lowest 7 bits of the DCO timer register directly extend the precision of the quartz crystal timer register. This approach was previously proposed by Schmid et al. [18]. For the upcoming calculations we also include another 8 "decimal" digit bits in the timestamp. Finally, these combined timestamps are stored in 64-bit integers, as 32-bit integer timestamp values would roll over every 4 seconds.

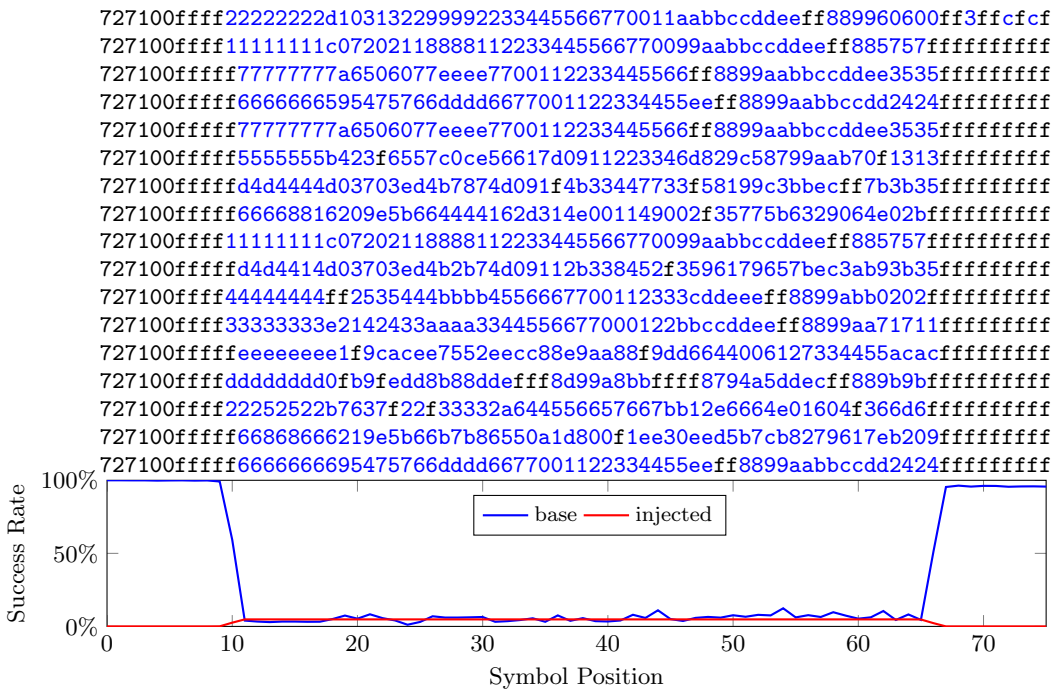

Figure 10: Loose Synchronization, No Symbol Remapping. Sample and correctness rate on a per symbol basis.

To transmit synchronization information we use MAC layer timestamping [6]. This technique generally involves capturing the local time at the start of an arriving or departing packet and storing that same timestamp in the footer of an outgoing packet. The captured local time of incoming packets can then be compared to the time in the packet to obtain an accurate value for the clock difference between the nodes at the start of the packet.

As the quartz crystals of different nodes exhibit slightly different tick rates, clocks of two once synchronized nodes will slowly drift apart at an almost constant rate. To counteract this drift, we keep a moving window of the information of several past synchronization points and perform linear regression to compute the current drift speed and extrapolate future time values. For details on this procedure please refer to the work of Lenzen et al. [13]. We found values of 4-8 stored synchronization points to be reasonable in both accuracy and speed of adapting to drift speeds changing due to environmental factors such as temperature changes.

As a result, our synchronization error remains within about ± 0.2 microseconds in 70% of cases and within ± 0.5 microseconds in 95% of cases when synchronizing at least once every couple of seconds.

5.2 Transmission Timing

Recently, a way of very accurately timing multiple transmissions using reference packets has been popularized by Glossy [4]. Unfortunately, this approach is unsuitable for us as we do not want to send packets simultaneously, but rather one at some arbitrary point in time after the other has started.

Hence, we need to rely on using the available clocks to find the point in time at which to issue the transmission command. A naive busy wait would repeatedly compute the current time after drift compensation and compare it to the target transmission time. However, 64-bit integer operations on a 16-bit CPU are comprised of hundreds of instructions and often dozens of branches. Such a busy wait loop cycle would require a rather long and unpredictable time, wasting a lot of precision in the process. Ideally, we'd want to transmit as precisely as a single DCO cycle.

To reach this goal, we employed several techniques. First, to avoid the costly multiplication instructions in every loop iteration, we apply our linear regression model "backwards" to compute the local time value corresponding to the target "global" transmission time target. This local time value can then be converted into the expected values of the 16-bit timer registers, allowing for much cheaper integer comparisons. Next, we compute the exact cycle count of the resulting inner loop, now consisting of a MOV, a CMP and a JLO instruction. Instructions on the MSP430 require different amounts of cycles, but the amounts are known at compile time. Hence, we know our final inner busy-wait loop is 8 DCO cycles in length. This allows us to align entering the loop with the target time by using a jump table in front of the loop to execute up to 7 NOP instructions prior to the loop. These techniques are described in detail in [10].

We found that in 65% of cases the amount of DCO cycles the CC2420 takes to begin transmitting after receiving the corresponding strobe is constant. This means, in these 65% of cases we get to send exactly at the time we want at DCO precision. In over 99% of cases the transmission time is off by at most 1 DCO cycle. However, for the problem at hand even an error of 1 cycle, corresponding to 0.25 microseconds, is likely to severely disrupt transmission synchronization.

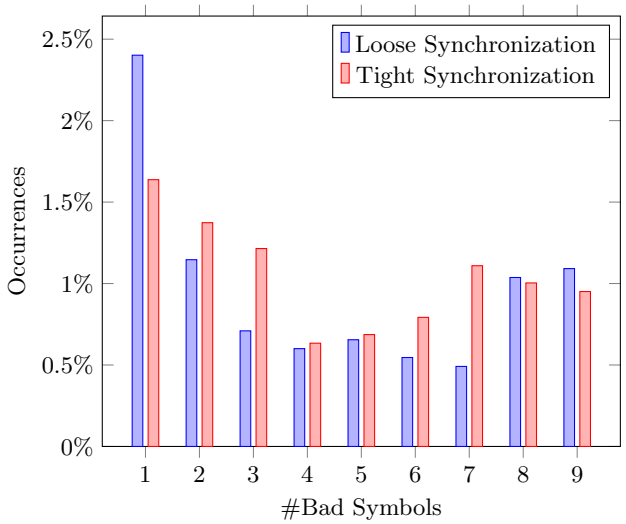

Figure 13: Distribution of the number of misinterpreted symbols after the injected packet. Not pictured: in 91% of cases none of the symbols are broken.

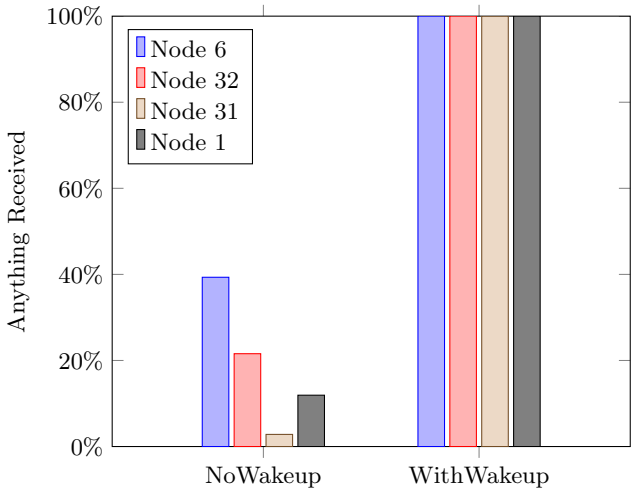

Figure 14: Percentage of received packets with and without the wakeup synchronization header at all 4 receivers of a test run.

5.3 Packet-In-Packet Improvements

Now 38.4% of cases allow correct decoding of the injected packet. Figure 11 shows another representative sample of received symbol strings. Blue indicates errors in the base packet (as before), green indicates the recognized and correctly decoded injected packet, and red indicates errors part of the injected packet (although there are no such errors in this sample).

Again, the injected packet barely experiences any internal symbol errors, and its symbol decoding success rate remains constant, this time at around 40%. The base packets' apparent peaks in decoding successes during the injection period is in fact misleading, as the injected packet's correctly decoded '`f`' symbols are interpreted as also matching the base packet and occur at static offsets.

Note that the tail of the base packet is still experiencing decoding errors. Figure 13 shows the distribution of the number of symbol errors in the tail. While both with loose and tight synchronization the tail is decoded completely correctly in about 91% of cases, loose synchronization appears to more often only incur a single symbol error. There is also a tendency visible for all of the remaining 8-9 tail symbols to be corrupted. This may be explained by the demodulation of correct but slightly misaligned symbols sometimes throwing the synchronization of the demodulator off permanently.

6. MAPPING SYMBOLS

6.1 Mapping Symbols

In the last sample (Figure 11) a large degree of determinism in the symbol decoding error can be observed in many of the broken packets. In fact, there appears to be a one-to-one mapping between actual and misinterpreted symbol values in most cases, albeit there appear to exist 7 different mappings. Based on the symbols which were received at the location of the preamble of the inner packet we can identify which mapping took place. If we denote by a the value of the preamble symbols and by s the correct symbol value, the 7 one-to-one mappings $m(s, a)$ for $1 \leq a \leq 7$ are:

$$m(s, a) = s - (s \bmod 8) + ((s + a) \bmod 8)$$

We can exploit the bijectiveness of these mappings to reverse them to reconstruct the original symbols in the majority of cases.

There are no simple one-to-one symbol mappings for the remaining cases of $8 \leq a \leq 15$. This is easy to see when examining the pairs of symbols in the payload of the injected packet not being recognizable as pairs anymore. However, it is likely that some information can still reliably be decoded, as certain pairs of subsequent symbols should also follow an exploitable deterministic mapping, judging by the interplay of the predefined chip sequences when time-shifted.

To detect injected packets after such a symbol mapping within the base packet, we not only search for the synchronization word (preamble + SFD) as before, but also for the synchronization word with each of the 7 mappings applied. If one is found, the respective mapping is then used to decode the length, payload and checksum of the injected packet.

7. RESULTS

7.1 The Chasm

The wakeup synchronization header sent at maximum power by a close neighbor unsurprisingly increases the number of cases in which any symbols were recorded to essentially 100% (see Figure 14).

Figure 15 shows how often every symbol in the packet was received correctly at least once. For lower transmit power settings – emulating the case of weak links – in the range from 7 to 10, corresponding to transmit powers from -15 dBm to -10 dBm [21], our scheme raises the chance of receiving a complete copy of the packet from 5% to 30%. Without the symbol remapping only 25% are achieved.

As the transmit power increases regular transmissions without wakeup reach $> 99\%$ reliability. This is because frequently one of the receivers receives a perfect copy of the

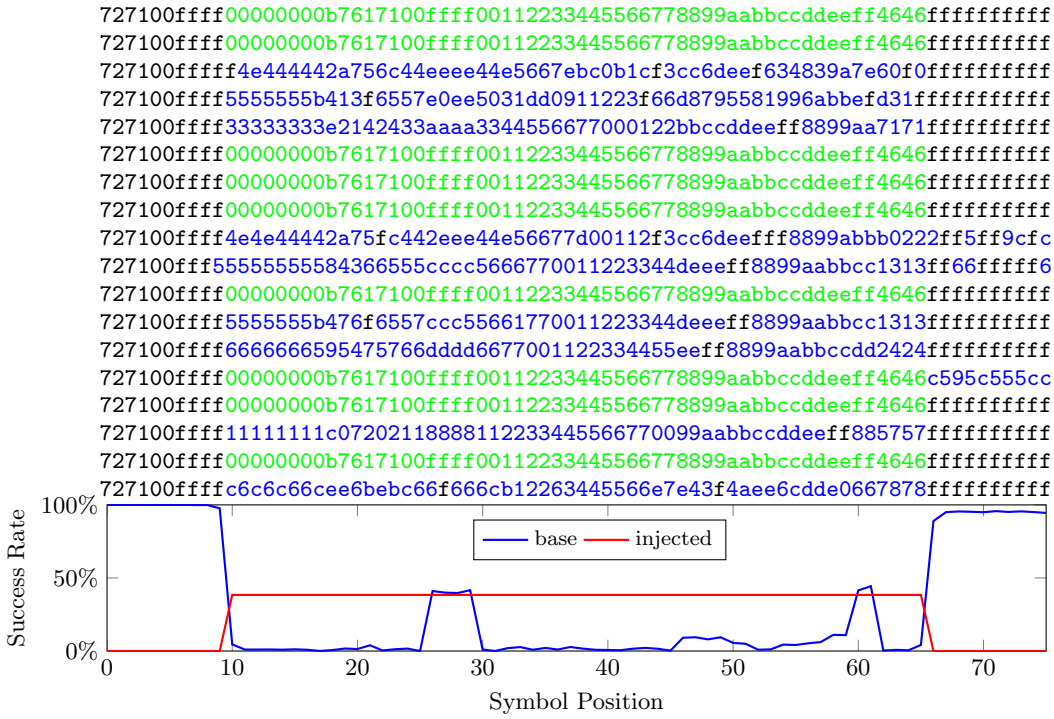

Figure 11: Tight Synchronization, No Symbol Remapping. Sample and correctness rate on a per symbol basis.

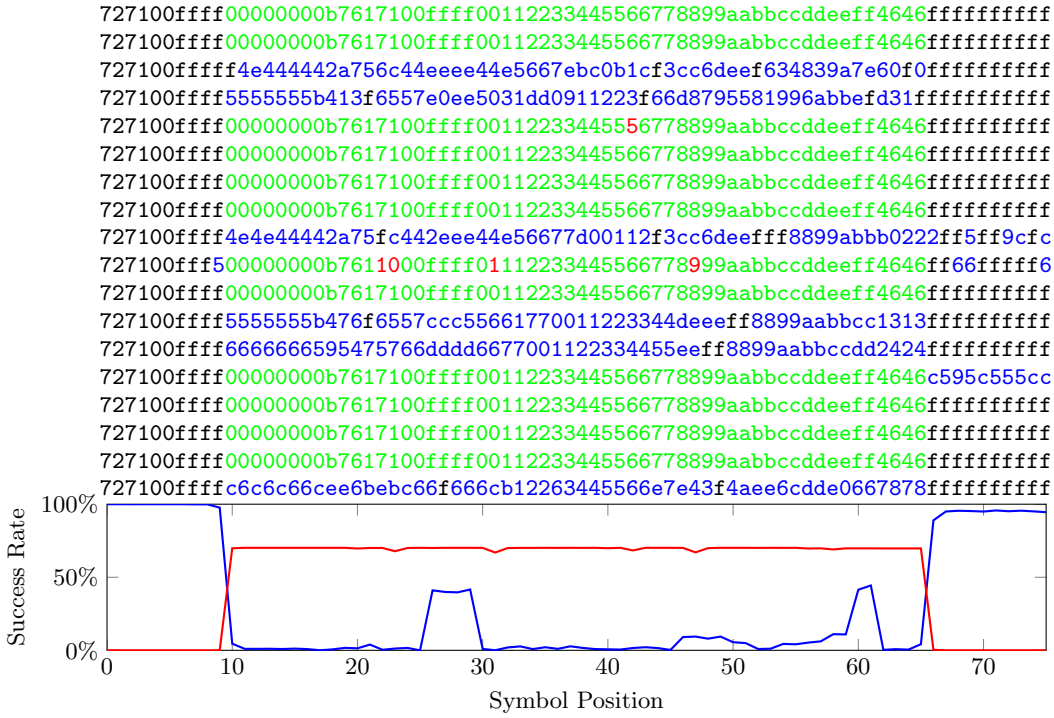

Figure 12: Tight Synchronization, With Symbol Remapping. Sample and correctness rate on a per symbol basis.

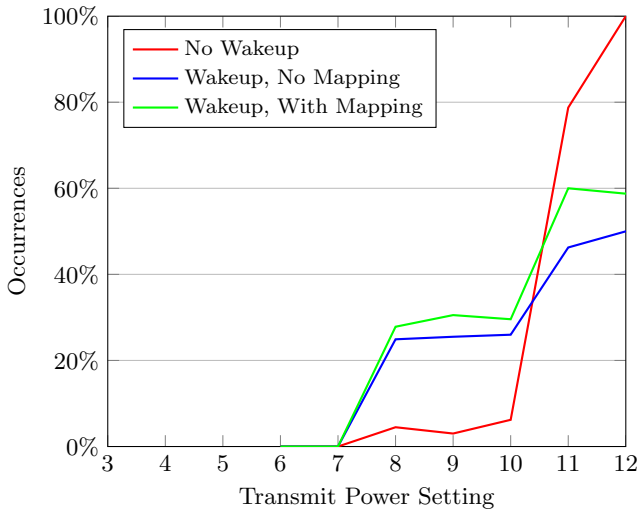

Figure 15: Percentage of cases in which at least 1 complete copy of the sent weak packet could be reassembled. Up to transmit power 10, our approach of adding a wakeup synchronization header significantly improves the chances.

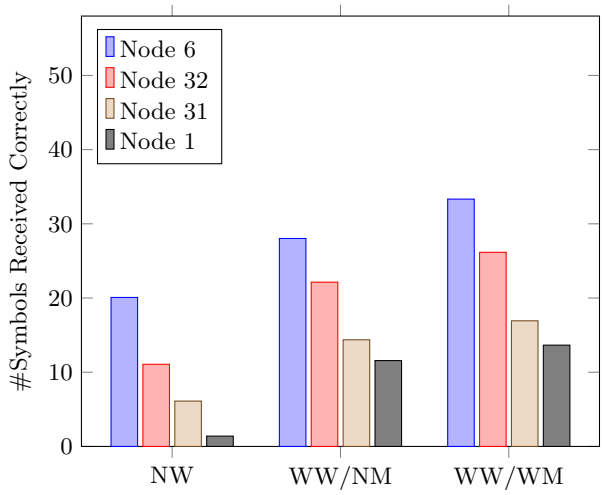

Figure 16: Average number of correct symbols received (out of 58) with and without the wakeup synchronization header at all 4 receivers of a test run. NW/WW = No/With Wakeup, NM/WM = No/With Symbol Mapping.

packet by itself, while our approach still suffers from the penalties induced by using a mismatching synchronization header. Hence, we do not recommend using our approach for any and all links, but rather only for chasm-like scenarios in which only weak links are available.

Figure 16 shows the number of correct symbols received on average within the transmit power setting range of 6 to 10 at each of the individual receivers. While the node with the best link (node 6) only gained about 65% of additional correct symbols, overall the number of correct symbols received more than doubled. As which symbols are correctly received for a receiver is independent of the other receivers (see Section 3.1), it stands to reason such an increase directly improves the chances of being able to assemble a complete correct copy of the packet.

The test run used for these plots contained 3800 samples, 3000 of which were taken within the transmit power setting range of 6 to 10.

7.2 Packet-In-Packet

In the packet-in-packet scenario, applying both techniques increases the success rate for decoding the injected packet completely without error to 64.3%. Tolerating up to 3 symbol errors, a success rate of 70% is reached. Figure 12 shows the same sample as in Figure 11, but with symbol remapping applied. It is easy to see that some of the erroneous lines from before have been corrected. However, in some instances a few symbols were corrupted (marked in red). Peculiarly, judging from the success rate graph at the bottom, some locations in the packet are particularly prone to symbol errors not following the mapping.

Figure 17 shows the distribution of the number of symbol errors in the decoded injected packet given that its synchronization word was intact and recognizable. While the number of errors appears to be higher when applying symbol mapping, this is merely due to the fact that in these cases additional synchronization words could be salvaged. These additional instances appear to be particularly prone to symbol errors, although the extent is low enough to still preserve most of the packet data. Further, tighter synchronization appears to reduce the occurrences of these errors.

The total amount of recovered packets can be seen in Figure 18, subdivided by the combination of applied techniques. Enforcing a limit on the number of symbol errors hurts the performance of the cases employing symbol remapping, as is to be expected. However, allowing up to 3 symbol errors almost completely closes the gap to the number of cases in which the synchronization word of the injected packet was at all detected. Recalling our injected packet being 46 symbols in length (when excluding the synchronization word), an error of 3 symbols appears rather acceptable.

The test run used for these plots used the node triplet $R = 23, I = 24, B = 27$ and contained 1900 samples for loose and tight synchronization each.

8. CONCLUSION AND FUTURE WORK

We explored the possibilities unfolding when mixing and matching synchronization headers and packet payloads as is made possible by inducing the capture effect. Correctly interpreting the resulting sometimes jumbled symbol string is not trivial and certainly not as efficient as with a matching synchronization header. However, we showed that some scenarios may nevertheless profit: A) bridging a chasm which is only crossed by weak links and may otherwise only be crossed taking a long detour – or not at all, if the network graph is not connected otherwise, and B) propelling high-priority packets across a network without any waiting or any control message overhead, barely disturbing low-priority traffic. The resulting gains we observed in our experiments are best summarized by Figures 15, 16 and 18.

Integrating such new transmission primitives into existing systems and MAC layers will no doubt carry an additional overhead we did not have to deal with in our proof of concept implementations. Exploring the tradeoffs of such

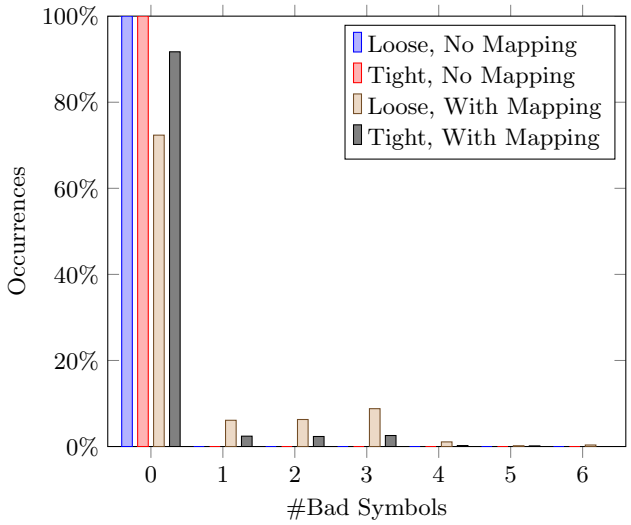

Figure 17: Distribution of the number of bad symbols within the injected packet once the injected packet's preamble has been read correctly.

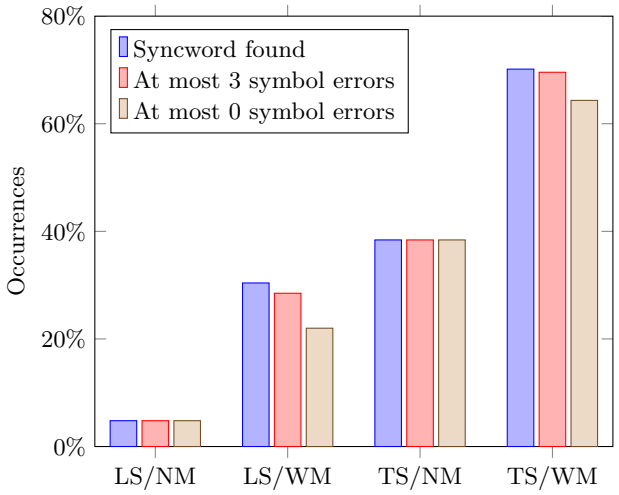

Figure 18: Success rates for decoding the injected packets using the different techniques. LS/TS = Loose/Tight Synchronization, NM/WM = No/With Symbol Mapping.

integrations promises to be an interesting topic for future work.

Although we conducted all our experiments using the IEEE 802.15.4 standard, we believe these primitives to, in principle, be feasible in most wireless standards using temporally separated symbols for modulation. We also expect this style of communication to be able to benefit many other scenarios beyond the two presented in this paper.

The results of our solution to the chasm scenario undoubtedly leave room for improvements. One aspect worth further research is the use of multiple senders, ideally synchronized well enough cause constructive interference in some subset of the receivers. Another promising direction is the use of forward error correcting codes to improve complete reception ratios at the cost of bandwidth. Such a scheme may also benefit the packet-in-packet primitive.

We see the main application of the packet-in-packet primitive in enabling high-priority messages to be sent and received at almost any time, in spite of ongoing lower-priority transmissions. This significantly reduces the latency these high-priority messages would otherwise incur, possibly accumulating further at every hop. Another imaginable application are data aggregation algorithms, using planned data insertions from many different nearby nodes to quickly form a single packet containing the aggregated data. To reduce the overhead of individual packets in this controlled scenario, one could shorten the synchronization word and omit or shorten the checksum footer.

In the remaining failure cases often some of the preamble symbols are recognizable, but there is no reversible one-to-one symbol mapping. It may be interesting to explore the interplay of the chip sequences defined by IEEE 802.15.4 under varying degrees of desynchronization in future work. We believe that computing the original string of symbols no matter the time shift is generally not possible. However, by smartly choosing the symbol combinations used to carry the data of the injected packet, always retrieving all the data correctly may be possible. In the best case, this may even allow dropping the stringent synchronization requirement at the cost of perhaps doubling the injected packet's length.

9. ACKNOWLEDGMENTS

We would like to thank Silvia Santini as well as the anonymous reviewers for their comments and suggestions.

10. REFERENCES

[1] M. Doddavenkatappa and M. C. Chan. P^3: a practical packet pipeline using synchronous transmissions for wireless sensor networks. In *IPSN'14, Proceedings of the 13th International Symposium on Information Processing in Sensor Networks, April 15-17, 2014, Berlin, Germany*, pages 203–214. IEEE/ACM, 2014.

[2] P. Dutta, R. Musaloiu-Elefteri, I. Stoica, and A. Terzis. Wireless ack collisions not considered harmful. In *7th ACM Workshop on Hot Topics in Networks - HotNets-VII, Calgary, Alberta, Canada, October 6-7, 2008*, pages 19–24. ACM SIGCOMM, 2008.

[3] F. Ferrari, M. Zimmerling, L. Mottola, and L. Thiele. Low-power wireless bus. In *The 10th ACM Conference on Embedded Network Sensor Systems, SenSys '12, Toronto, ON, Canada, November 6-9, 2012*, pages 1–14. ACM, 2012.

[4] F. Ferrari, M. Zimmerling, L. Thiele, and O. Saukh. Efficient network flooding and time synchronization with glossy. In *Proceedings of the 10th International Conference on Information Processing in Sensor Networks, IPSN 2011, April 12-14, 2011, Chicago, IL, USA*, pages 73–84. IEEE, 2011.

[5] R. Flury and R. Wattenhofer. Slotted programming for sensor networks. In *Proceedings of the 9th International Conference on Information Processing in Sensor Networks, IPSN 2010, April 12-16, 2010, Stockholm, Sweden*, pages 24–34. ACM, 2010.

[6] S. Ganeriwal, R. Kumar, and M. B. Srivastava. Timing-sync protocol for sensor networks. In *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems, SenSys 2003, Los Angeles, California, USA, November 5-7, 2003*, pages 138–149. ACM, 2003.

[7] R. Gotzhein and T. Kuhn. Black burst synchronization (BBS) - A protocol for deterministic tick and time synchronization in wireless networks. *Computer Networks*, 55(13):3015–3031, 2011.

[8] Institute of Electrical and Electronics Engineers. *IEEE Standard 802.15.4: Wireless Medium Access Control (MAC) and Physical Layer (PHY) Specifications for Low-Rate Wireless Personal Area Networks (LR-WPANs)*.

[9] A. Jayasuriya, S. Perreau, A. Dadej, and S. Gordon. *Hidden vs exposed terminal problem in ad hoc networks*. PhD thesis, ATNAC, 2004.

[10] M. König and R. Wattenhofer. Maintaining constructive interference using well-synchronized sensor nodes. In *International Conference on Distributed Computing in Sensor Systems, DCOSS 2016, Washington, DC, USA, May 26-28, 2016*, pages 206–215. IEEE, 2016.

[11] M. König and R. Wattenhofer. Sharing a medium between concurrent protocols without overhead using the capture effect. In *Proceedings of the International Conference on Embedded Wireless Systems and Networks, EWSN 2016, Graz, Austria, 15-17 February 2016*, pages 113–124. Junction Publishing, Canada / ACM, 2016.

[12] O. Landsiedel, F. Ferrari, and M. Zimmerling. Chaos: versatile and efficient all-to-all data sharing and in-network processing at scale. In *The 11th ACM Conference on Embedded Network Sensor Systems, SenSys '13, Roma, Italy, November 11-15, 2013*, pages 1:1–1:14. ACM, 2013.

[13] C. Lenzen, P. Sommer, and R. Wattenhofer. Pulsesync: An efficient and scalable clock synchronization protocol. *IEEE/ACM Trans. Netw.*, 23(3):717–727, 2015.

[14] R. Lim, F. Ferrari, M. Zimmerling, C. Walser, P. Sommer, and J. Beutel. Flocklab: a testbed for distributed, synchronized tracing and profiling of wireless embedded systems. In *The 12th International Conference on Information Processing in Sensor Networks, IPSN 2013, Philadelphia, PA, USA, April 8-11, 2013*, pages 153–166. ACM, 2013.

[15] J. Lu and K. Whitehouse. Flash flooding: Exploiting the capture effect for rapid flooding in wireless sensor networks. In *INFOCOM 2009. 28th IEEE International Conference on Computer Communications, 19-25 April 2009, Rio de Janeiro, Brazil*, pages 2491–2499. IEEE, 2009.

[16] J. Polastre, R. Szewczyk, and D. E. Culler. Telos: enabling ultra-low power wireless research. In *Proceedings of the Fourth International Symposium on Information Processing in Sensor Networks, IPSN 2005, April 25-27, 2005, UCLA, Los Angeles, California, USA*, pages 364–369. IEEE, 2005.

[17] N. K. Santhapuri, J. Manweiler, S. Sen, R. R. Choudhury, S. Nelakuditi, and K. Munagala. Message in message (MIM): A case for shuffling transmissions in wireless networks. In *7th ACM Workshop on Hot Topics in Networks - HotNets-VII, Calgary, Alberta, Canada, October 6-7, 2008*, pages 25–30. ACM SIGCOMM, 2008.

[18] T. Schmid, P. Dutta, and M. B. Srivastava. High-resolution, low-power time synchronization an oxymoron no more. In *Proceedings of the 9th International Conference on Information Processing in Sensor Networks, IPSN 2010, April 12-16, 2010, Stockholm, Sweden*, pages 151–161. ACM, 2010.

[19] A. Seth, D. Kroeker, M. A. Zaharia, S. Guo, and S. Keshav. Low-cost communication for rural internet kiosks using mechanical backhaul. In *Proceedings of the 12th Annual International Conference on Mobile Computing and Networking, MOBICOM 2006, Los Angeles, CA, USA, September 23-29, 2006*, pages 334–345. ACM, 2006.

[20] L. Su, C. Liu, H. Song, and G. Cao. Routing in intermittently connected sensor networks. In *Proceedings of the 16th annual IEEE International Conference on Network Protocols, 2008. ICNP 2008, Orlando, Florida, USA, 19-22 October 2008*, pages 278–287. IEEE Computer Society, 2008.

[21] Texas Instruments. *2.4 GHz IEEE 802.15.4 / ZigBee-ready RF Transceiver*. CC2420 Data Sheet.

[22] K. Whitehouse, A. Woo, F. Jiang, J. Polastre, and D. Culler. Exploiting the capture effect for collision detection and recovery. In *Proceedings of the 2nd IEEE Workshop on Embedded Networked Sensors*, pages 45–52. IEEE Computer Society, 2005.

[23] D. Yuan and M. Hollick. Let's talk together: Understanding concurrent transmission in wireless sensor networks. In *38th Annual IEEE Conference on Local Computer Networks, Sydney, Australia, October 21-24, 2013*, pages 219–227. IEEE Computer Society, 2013.

[24] D. Yuan, M. Riecker, and M. Hollick. Making 'glossy' networks sparkle: Exploiting concurrent transmissions for energy efficient, reliable, ultra-low latency communication in wireless control networks. In *Wireless Sensor Networks - 11th European Conference, EWSN 2014, Oxford, UK, February 17-19, 2014, Proceedings*, volume 8354 of *Lecture Notes in Computer Science*, pages 133–149. Springer, 2014.

[25] M. Zimmerling, F. Ferrari, L. Mottola, and L. Thiele. On modeling low-power wireless protocols based on synchronous packet transmissions. In *2013 IEEE 21st International Symposium on Modelling, Analysis and Simulation of Computer and Telecommunication Systems, San Francisco, CA, USA, August 14-16, 2013*, pages 546–555. IEEE Computer Society, 2013.

[26] M. Zuniga and B. Krishnamachari. Analyzing the transitional region in low power wireless links. In *Proceedings of the First Annual IEEE Communications Society Conference on Sensor and Ad Hoc Communications and Networks, SECON 2004, October 4-7, 2004, Santa Clara, CA, USA*, pages 517–526. IEEE, 2004.