Anti-jamming Timing Channels for Wireless Networks

Wenyuan Xu
CSE Department
University of South Carolina
Columbia, SC, 29201
wyxu@cse.sc.edu

Wade Trappe
WINLAB
Rutgers University
North Brunswick, NJ, 08902
trappe@winlab.rutgers.edu

Yanyong Zhang
WINLAB
Rutgers University
North Brunswick, NJ, 08902
yyzhang@winlab.rutgers.edu

ABSTRACT

Wireless communication is susceptible to radio interference, which prevents the reception of communications. Although evasion strategies have been proposed, such strategies are costly or ineffective against broadband jammers. In this paper, we explore an alternative to evasion strategies that involves the establishment of a timing channel that exists in spite of the presence of jamming. The timing channel is built using failed packet reception times. We first show that it is possible to detect failed packet events in spite of jamming. We then explore single sender and multi-sender timing channel constructions that may be used to build a low-rate overlay link-layer. We discuss implementation issues that we have overcome in constructing such jamming-resistant timing channel, and present the results of validation efforts using the MICA2 platform. Finally, we examine additional error correction and authentication mechanisms that may be used to cope with adversaries that both jam and seek to corrupt our timing channel.

Categories and Subject Descriptors

C.2.0 [**Computer-Communication Networks**]: General—*Security and protection*

General Terms

Security, Reliability, Experimentation

Keywords

Jamming, Radio Interference, Timing channel

1. INTRODUCTION

Wireless technologies depend on the opportunities provided by a broadcast medium. However, the factors that facilitate tetherless, ubiquitous communication also can be used to disrupt such communication, thereby undermining the availability of a wireless networks. There are many strategies that may be employed to disrupt wireless connectivity. Some strategies, such as deassociation attacks or association flooding [1], represent network-oriented attacks that have been employed against commodity wireless systems (e.g. 802.11). Although such availability threats are powerful, they can (and should) be addressed through authentication. An alternative strategy to disrupt wireless communications is to directly interfere with communications by jamming the communication channel [2]. In fact, intentionally interfering with wireless communications is simple: an RF source (e.g. a waveform generator) may broadcast energy on a broad spectral band to disrupt signal reception [2], or the medium access control may be cleverly exploited to disrupt link-layer decoding [3, 4], or medium access control mechanisms may be bypassed to prevent other devices from accessing the channel [1, 3].

RF interference has traditionally been addressed at the physical layer through modulation approaches (such as spread spectrum) with sufficiently powerful anti-jam margins to make disruption difficult. Unfortunately, such physical layer approaches are typically representative of military wireless systems and, for reasons of cost, are generally not employed in commodity wireless platforms[1].

For commodity networks, several defense strategies have been proposed to repair network connectivity in the presence of interference. In [5], protocols are presented that allow an ad hoc/sensor network to adjust its operating channel to evade interference. Unfortunately, the interference is limited to jammers operating on one or a few channels, and it should be noted that it is harder for an entire network to adjust its channel allocations than for an aggressive adversary to scan the spectrum and interfere on subsequent channels. Similarly, constantly adapting channels, as proposed in [6], might emulate a link-layer frequency hopping strategy, but also cannot survive in the presence of broadband adversaries. Spatial evasion strategies [7] are costly and require complex reparation protocols to prevent network partitioning.

Such *evasion* strategies are limited and, although it is impossible to overcome a broadband adversary with unlimited transmission power, it is nonetheless desirable to have mechanisms to communicate in the presence of broadband interference when the adversary has transmission power limitations. In this paper, we propose an alternative to evasion strategies. Rather, we seek to communicate in the presence

WiSec'08, March 31–April 2, 2008, Alexandria, Virginia, USA.

[1]Although many commercial wireless systems use spreading this is typically for multipath resistance and not as an anti-jam mechanism.

of a powerful RF adversary[2]. Our objective is to create a low bit-rate overlay that exists on top of the conventional physical/link-layers and that would survive in the presence of a persistent broadband interferer. Our low bit-rate channel is constructed as a timing-channel that exists *in spite* of the jammer being present. We emphasize that this timing channel is not a panacea– it restores the *availability* of a communication link in the presence of interference, but other security issues, such as authentication, must be addressed through complimentary methods.

We begin the paper in Section 2 by providing an overview of our timing channel overlay. In Section 3, we show that detecting failed packet receptions is feasible, and thus our timing channel can be constructed. We next examine a simple single sender scenario in Section 4. Then we extend our timing channel to a multiple sender scenario, in Section 5. Finally, using our multi-sender timing channel, we explore the construction of an overlay link-layer in Section 6, and discuss mechanisms that enhance the reliability of our overlay. We wrap up the paper by discussing related work in Section 7, and provide concluding remarks in Section 8.

2. TIMING CHANNEL OVERVIEW

In order to introduce the idea behind our timing channel, let us consider a parable. Suppose Alice and Bob are socializing and, suddenly, the malicious Mr. X interrupts them and prevents their conversation. The natural question that arises is "What should Alice and Bob do to continue their conversation?" One option (corresponding to evasion strategies) would be for them to excuse themselves from the conversation and meet up with each other later in a different location. If this is not possible, then another approach they could take is to convey information in their attempts to communicate. Even though Mr. X would prevent their sentences from being understandable, the fact that Bob would see Alice trying to communicate (e.g. via her lip movements) could be used to convey information.

Applying this idea to the context of wireless networks, although a jammer might prevent the successful reception of packets, the fact that there was an attempted, incoming packet could be exploited to convey information. For example, assuming Bob can detect failed packet receptions and that Alice knows the jammer is blocking Bob's reception, then Alice can convey information to Bob in the time between her packet transmissions, thereby establishing a timing channel with Bob.

In a conventional layering of the network stack, the physical and data link layers are responsible for providing links of reliable packets upon which network functions operate. Since jamming disrupts these two layers yet, as we shall see, still allows for the detection of packet failure events, we envision that it is possible to create a low-rate overlay between the normal data link layer and the network layer that would allow for the resumption of network services.

The components for this overlay, which we have called the 4-Ounce Overlay (in the spirit of our motivational Tai Chi Chuan quote), are depicted in Figure 1. In particular, the 4-Ounce Overlay must replace the functionality of the physical layer (to create a bitpipe between a sender and a receiver), and the link layer (to create a reliable link for sending error-free packets between the sender and receiver).

[2]We are motivated by the Tai Chi Chuan philosophy, 'A strength of four ounces can defeat a force of a thousand pounds.'

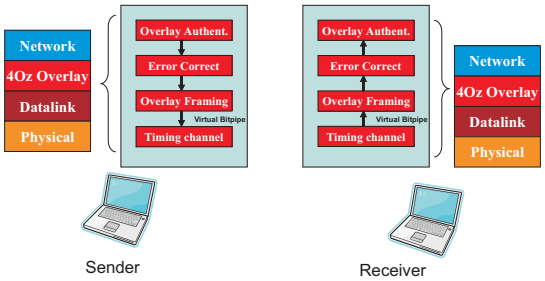


Figure 1: The 4-Ounce Overlay exists between the data link and network layers, and involves the creation of a timing channel, framing, error correction, and authentication services.

As such, the 4-Ounce Overlay first involves creating an overlay physical layer using the timing channel created by failed packet reception events. The successful conveyance of information via a timing channel depends on two crucial steps: (1) a node must detect a failed packet reception, in spite of an interference signal; (2) it is necessary to map the presence of failed packets into the information that needs to be delivered.

3. DETECTING FAILED PACKET EVENTS

Even before commencing with the first stage of creating a timing channel, the senders must detect the jamming. It has been shown by Xu et al. and Wood et al. that jamming detection and mapping jammed regions is possible through use of network utility metrics, such as packet delivery rate combined with received signal strength statistics [3,8]. Once jamming has been detected, our legitimate sender S and receiver R commence with creating the timing channel. To form the timing channel, R must detect failed packets transmitted by S. As we shall see, regardless of the type of interference employed, detecting packet failures is generally possible as long as S is able to employ transmission power levels comparable to the jammer's interference power.

Interference Models: Various radio interference schemes have been proposed in [3]. Generally, interference may be classified as active or reactive, depending on whether the jammer reacts to transmissions or not. In this paper, we examine two typical jammer models (an active and a reactive type) that represent both intentional and unintentional jamming scenarios: (1) constant jammers, and (2) reactive jammers. Constant jammers, whether intentional or unintentional, continuously emit random bits or a radio signal into the channel, thereby corrupting packet reception. We assume, once transmitters detect jamming, that they bypass any MAC-layer mechanisms preventing them from transmitting when the channel is occupied. Reactive jammers, by their nature, are malicious– they remain silent when there is no communication, and start transmitting a jamming signal as soon as they sense packets in the ether. Although there are other variations, these two cases represent the two extremes of jamming and thus serve as the ideal basis for illustrating the feasibility of detecting failed packet reception. In particular, we note that other jammers, such as the deceptive jammer of [3] would have a procedure for detect-

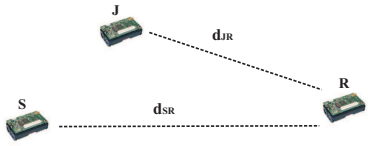


Figure 2: Experiment setup.

ing failed packet reception that is identical to the constant jammer case.

Detecting Packet Presence: We now examine the likelihood for a receiver to detect that its neighbor has tried to transmit a packet when there is a constant jammer or a reactive jammer interfering with its communication.

Reactive Jammer: In reactive jammer scenarios, the jammer stays quiet when the channel is idle, but starts transmitting a radio signal as soon as it senses activity on the channel. In parallel, receiver nodes will start to receive the packet as soon as they decode the preamble. Since the jammer must detect preamble prior to issuing its jamming signal, the receivers will have already entered their receive state prior to the emission of the jamming signal. However, shortly after entering the receive state, the packets will be garbled by the reactive jamming signal, and will be dropped at the lower layer as a result of the failure to pass CRC checking. We may thus detect the failure of packet receptions by using CRC failures, and timestamp CRC failures to facilitate the recording of packet arrival times for establishing our timing channel. As a proof of concept, we conducted a set of experiments using the MICA2 platform. The devices we used each had a 433MHz Chipcon CC1000 radio, and the operating system running on each mote was TinyOS version 1.1.7 [9]. Our experiments involved a transmitter S and a receiver R with a reactive jammer J present as shown in Figure 2. S and R were placed 4 feet apart, and the jammer J was placed $(1, 2, 3, 4, 6, 8)$ feet away from R. 2000 packets were sent as constant rate traffic. The resulting Packet Delivery Ratio (PDR) and Packet Detection Ratio (DR) for jamming scenarios under various distance separation between R and J are shown in Table 1. Without the presence of the reactive jammer, the packet delivery ratio was 99.90%. However, introducing the reactive jammer caused the packet delivery rate to drop to almost 0%, and the receiver was able to detect the failure of packet reception with a 99% effectiveness. We note, when the jammer is more than 4 feet away from the receiver R, the reactive jammer is no longer effective in destroying the communication between R and S.

Constant Jammer: The reactive jammer allows for moderately easy detection of failed receptions. A constant jammer, however, poses a more difficult challenge as the jammer continuously corrupts the channel, thereby preventing the detection of packet preambles. Hence, timestamping CRC failures is not possible since the receiver state machine

d_{JR} (feet)	1	2	3	4	6	8
PDR (%)	0.10	0.70	0.15	98.65	100	99.85
DR (%)	99.80	99.80	99.60	99.95	99.65	99.95

Table 1: The resulting Packet Delivery Ratio (PDR) and Packet Detection Ratio (DR) under various distance separation between R and J.

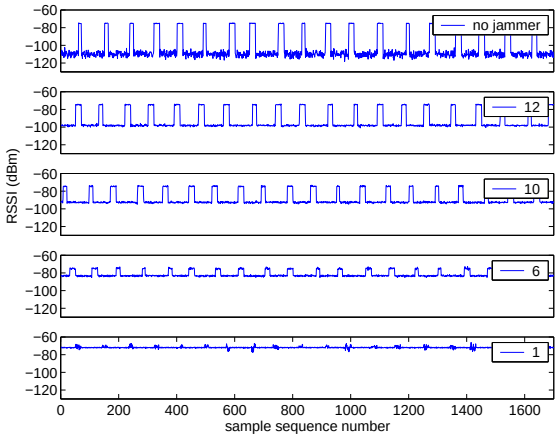


Figure 3: RSSI readings as a function of time in different scenarios. The upper plot corresponds to no jammer scenario, and the lower plots correspond to jamming scenarios where the distances between the jammer J and the receiver R are shown as legend captions in feet.

will not enter the receive state at all (no CRC failures will be reported). Instead, an alternative method must be employed where, once the receiver has detected the presence of jamming, it tracks the ambient received signal strength (which is the combination of the received signal strength from both the jammer and the sender). By searching the received signal strength for short duration spikes (corresponding to the duration of a packet), it is possible to detect a failed packet reception. We may thus use the timestamping of these events to synthesize the timing channel.

The effectiveness of this detection scheme depends on the relationship between the jamming power and the sender transmit power. To understand the effect that a jammer would have on the received signal strength levels, we performed an experiment with Berkeley MICA2 motes. Our experiments involved a single mote R merely measuring signal levels, a mote S sending packets periodically, and a constant jammer J that constantly transmits random bits into the air. We set the transmission power of J and S to $-20dBm$, and then changed J's distance to the receiver R to study the effect that J would have on the received signal strength. We used the same experiment setup as in reactive jammer scenario. While we moved the constant jammer J towards the receiver R, the separation distance between J and R ranged from 12 ft to 1 ft (to eliminate near field effects, we carefully placed J to be at least 4ft from S). The receiver R obtains the RSSI values by posting `RSSIADC.getData()` on the port `TOS_ADC_CC_RSSI_PORT` as fast as the ADC can sample. The reported RSSI values in Figure 3 in dBm are converted from the raw values following the analog-to-digital conversion of the received voltage levels [10].

In Figure 3, the uppermost plot corresponds to the scenario where there is only S sending messages while J remains quiet. The lower four plots are the RSSI reading when J is $(12, 10, 6, 1) feet$ away from R. From these results, we observe that the measured RSSI time series with the jammer exhibits periodic variations with each spike representing a packet event. This observation suggests that pattern recog-

nition techniques can be employed to identify the packet arrival time. As the jammer J is placed closer to R, the height of the spike becomes smaller and we note that when J is only 1 ft away from R with the sender S being 4 ft from R, (i.e. J has a larger impact on the RSSI reading of R than S does), we can still observe small spikes. This implies that, even if the jammer is a moderately high-powered source of interference, it is still feasible to detect the presence of packets using the radio receiver. Finally, we note that, from the jammer's point of view, it is not in its best interest to jam with high power. In most cases, the objective of a malicious jammer is to interfere with the legitimate network communication with low energy consumption or with low probability of being localized. Thus, an adversary seeking to avoid localization would use a minimal jamming power when trying to interfere with wireless communication.

4. ONE SENDER COMMUNICATION

We now examine coding methods that will map the packet arrivals/failure events into messages via packet inter-arrival times. We start our discussion by presenting a formalized coding construction for the single sender scenario, and then validate our construction through a system implementation.

4.1 Coding Constructions

Consider the two party setup where we have a sender S communicating to a single receiver R. The sender S will send a stream of k packets (over a time period $[0, T]$) to convey a symbol $s \in \mathcal{S}$, where $\mathcal{S}$ is a message alphabet. Often, we shall consider $s \in \{0, 1\}$. In order to map packet arrival times to messages, we introduce a definition:

DEFINITION 1. *We define a single-sender arrival code* $\mathcal{C} = \{\mathbf{c}_1, \mathbf{c}_2, \cdots, \mathbf{c}_m\} \subset [0, T]^k$ *where* $m = |\mathcal{S}|$. *Each*

$$\mathbf{c}_j = (c_{1j}, c_{2j}, \cdots, c_{kj})$$

has the monotonicity property $c_{ij} < c_{lj}$ *when* $i < l$.

The codeword $\mathbf{c}_j$ may be thought of as the arrival times associated with a sequence of k packets. Associated with $\mathcal{C}$ is its inter-arrival code $\Delta_{\mathcal{C}}$, which is composed of codewords $\Delta_{\mathcal{C}} = \{\delta_1, \delta_2, \cdots, \delta_m\} \subset [0, T]^{k-1}$, where

$$\delta_j = (c_{2j} - c_{1j}, c_{3j} - c_{2j}, \cdots, c_{kj} - c_{k-1,j}).$$

We note that there is an implicit requirement $\|\delta_j\|_1 < T$ because $c_{kj} < T$.

For a single-user arrival code, we define the minimal difference distance (MDD) of $\mathcal{C}$ by

$$d^*_{\mathcal{C}} = \min_{j,l} \|\delta_j - \delta_l\|_2$$

where $\|\mathbf{v}\|_2$ denotes the standard Euclidean norm. The minimal difference distance of a code is the closest its codewords get to each other. One objective, then, is to design codes with large MDD for a given k, m and T. This will allow us to correctively decode communications should errors occur (e.g. possible calibration drift/jitter). On the other hand, we want to design codes so that the number of symbols that can be sent out for a given duration T is maximized.

Let's look at a simple example, $k = 2$, $m = 2$, where we have $\mathcal{S} = \{0, 1\}$, and $\mathcal{C} = \{\mathbf{c}_1, \mathbf{c}_2\}$. Since we have a period of T time units to convey either 0 or 1, we must have that $\delta_{1j} = c_{2j} - c_{1j} < T$, where $j \in \{1, 2\}$. Our problem of designing a code with good distance properties amounts to selecting two constellation points $\delta_{1j} \in [0, T]$. If we select $\delta^{\epsilon}_{11} = \epsilon$ and $\delta^{\epsilon}_{12} = T - \epsilon$, then we have two constellation points with separation $d^*_{\mathcal{C}} = T - 2\epsilon$. This corresponds to the arrival code $\mathcal{C} = \{(\alpha, \alpha+\epsilon), (\beta, \beta+T-\epsilon)\}$, where α and β are arbitrary and satisfy the appropriate time constraints. Tuning ϵ to be smaller increases the codeword separation and will thereby increase decoding reliability. It is straightforward to extend this simple example to cases where $k = 2$ and m is arbitrary. For example, the following $\mathcal{C} = \{(\alpha, \alpha + \epsilon), (\alpha, \alpha + 2\epsilon), (\alpha, \alpha + 3\epsilon), (\alpha, \alpha + 4\epsilon)\}$, where $\alpha + 4\epsilon < T$ corresponds to a $k = 2$, $m = 4$ code. Later, in Section 4.2, we shall present a code construction for $k = 3$.

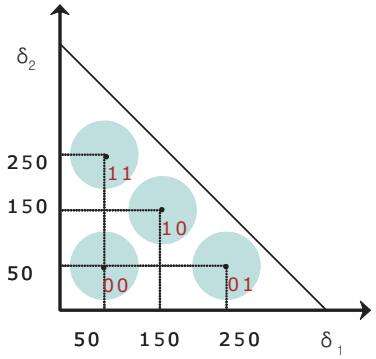


Figure 4: Triangular simplex for an inter-arrival code with $k = 3$ and $m = 4$.

Encoding: The encoding process at the sender S is straight forward. To transmit a symbol s_j, which corresponds to the inter-arrival codeword $\delta_j = \{\delta_{1j}, \delta_{2j}, \cdots, \delta_{k-1,j}\}$, the sender S sends out packets at times $(\alpha, \alpha + \delta_{1j}, \alpha + \delta_{2j}, \cdots, \alpha + \delta_{k-1,j})$, where α is an arbitrary starting time for transmitting the symbol s_j.

Decoding: A receiver R records the arrival time of a stream of k packets, $\mathbf{c}' = (c'_1, c'_2, \cdots, c'_k)$ and calculates the observed inter-arrival time sequence $\delta'_r = (\delta'_1, \delta'_2, \cdots, \delta'_{k-1})$. Decoding involves finding the codeword $\delta_r \in \Delta_C$ that has the smallest distance from δ'_r out of all codewords from Δ_C, subject to a constraint that

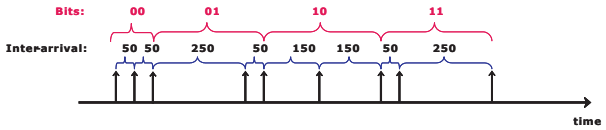
$$d_r = \min_{\delta_j \in \Delta_{\mathcal{C}}} \|\delta_j - \delta'_r\|_2, d_r < D,$$

where D is a threshold set to prevent decoding errors. If the smallest distance $d_r > D$, then the received inter-arrival code was far from all legitimate codes and likely an error has occurred (such as a failure to witness a failed packet arrival or jitter in transmission/reception timing operations).

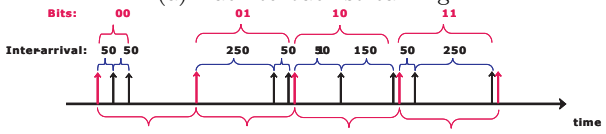
4.2 System Implementation

The above discussion provided theoretical guidelines for communicating via a timing channel. When applying theory to a real system, many new challenges arise and thus we validate our timing channel in a real system implementation using MICA2 motes.

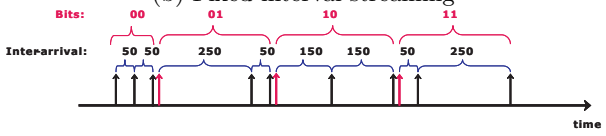
Code Setup: In order to illustrate the systems issues that must be addressed, we shall focus our discussion on a code involving $k = 3$ packets. The techniques that we have developed apply naturally to other values of k. For this case, our arrival codeword $\mathbf{c}_j = (c_{1j}, c_{2j}, c_{3j})$ satisfies $0 < c_{1j} < c_{2j} < c_{3j} < T$. The corresponding inter-arrival code $\delta_j = (\delta_{1j}, \delta_{2j})$ satisfies the property $\delta_{1j} + \delta_{2j} \leq T$, and $0 <$



(a) Back-to-back streaming



(b) Fixed interval streaming



(c) Marker streaming

Figure 5: Streaming strategies.

$\delta_{ij} < T$. In $\delta_1 \times \delta_2$ space, this maps out a triangular simplex $\Delta = \{(\delta_1, \delta_2) | \delta_1 + \delta_2 < T, \delta_1 > 0, \delta_2 > 0\}$, as depicted in Figure 4. We note that other values of k lead to similar simplexes in higher-dimensional spaces.

In terms of our code design, we seek to put m disks of radius D within Δ so that these disks do not overlap. Each disk corresponds to one of the symbols' decoding region. D serves as the maximum allowed distance a received codeword can have from legitimate codewords in order to be declared legitimate, otherwise we declare a decoding error. We note that a generalized nearest neighbor decoding involving Voronoi tesselation of the triangular simplex is also possible. However, our choice to leave a gap between codeword disks was intentionally done to provide a means to identify potential errors that might arise from our timing channel failing to detect a packet arrival.

Intuitively, the decoding process involves mapping the observed inter-arrival time δ_r' into the $\delta_1 \times \delta_2$ space, and finding which disk the measured δ_r' belongs to. We then declare that the sender sent the symbol s corresponding to that disk's codeword. If δ_r' does not fall within any of the disks, then a decoding error occurred.

For our validation effort, we shall focus on a $k = 3$, $m = 4$ inter-arrival code. To ensure that we have the ability to uniquely decode and utilize the timing-channel's bandwidth effectively, we place four disks centered at four constellation points: $\delta_1^\epsilon = (\epsilon, \epsilon)$, $\delta_2^\epsilon = (5\epsilon, \epsilon)$, $\delta_3^\epsilon = (3\epsilon, 3\epsilon)$, and $\delta_4^\epsilon = (\epsilon, 5\epsilon)$, where ϵ is a parameter that governs placement of the constellation.

Selection of ϵ: We now turn to the issue of selecting ϵ (and, consequently, T), which determines the number of symbols that can be delivered in a given period of time. Since a major design objective is to maximize the amount of symbols that can be transmitted per unit time, smaller ϵ is preferred to achieve such a goal. However, in a real system implementation, the minimum value of ϵ is limited by many factors. First of all, it must be larger than the amount of time required to complete one packet transmission. Additionally, latency uncertainties, which accumulate as the packet reception event traverses across various layers and buffers, can cause jitter of the packet arrival times. Those unwanted variations introduce errors in the inter-arrival times. Therefore, larger MDD is needed in order to correctly decode messages when there are arrival time inaccuracies.

For our implementation, we used a reactive jammer. When nodes detect jamming and initiate the timing channel, they use a 13 byte packet to construct the timing channel. Since it takes approximately 12 ticks for a MICA2 mote to transmit a 13 byte packet[3], we have chosen $\epsilon = 50$ ticks in order to leave a safe margin for measurement error, giving an inter-arrival code $\Delta_\mathcal{C} = \{(50, 50), (250, 50), (150, 150), (50, 250)\}$ for the symbol set $\{00, 01, 10, 11\}$. For this code, $T = 310$ ticks, which leads to an ideal timing channel capacity of 6.61 bps, and to increase the timing channel capacity we may decrease ϵ. This value of T corresponds to the theoretical (fixed-interval) value and in practice it is necessary to employ delimiting methods to concatenate successive symbols over the timing channel.

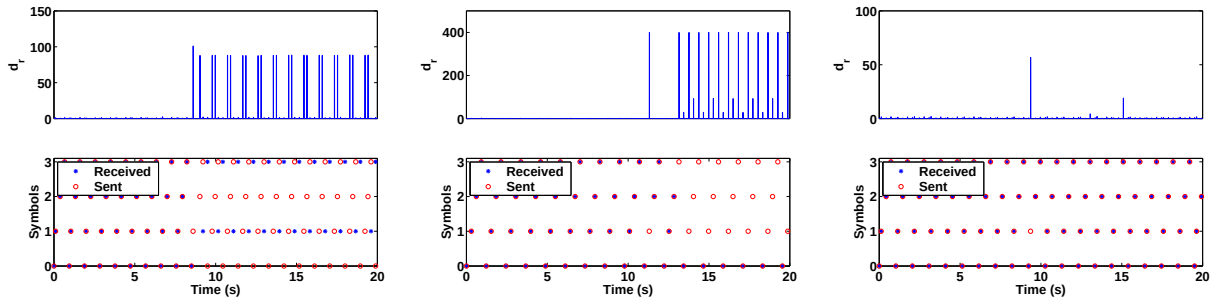
Symbol Streaming Through the Timing Channel: Following the establishment of basic elements for the timing channel, i.e. symbols and codewords, we next discuss how we can achieve efficient symbol streaming through the timing channel. In order to facilitate streaming, both the sender and the receiver have to agree on how to mark/identify the beginning of the transmission of a symbol. In this study, we have designed and built the following streaming methods:

- *Back-to-back Streaming:* The most efficient way of transmission is to transmit subsequent symbols back to back, using the last packet of a symbol as the starting of the next symbol. We illustrate this approach pictorially in Figure 5 (a). In this example, the sender intends to transmit two symbols, 00 and 01, and it achieves this by sending 5 packets at ticks $(0, 50, 100, 350, 400)$ respectively, where the packet sent at tick 100 serves as the last packet for 00 and the first packet for 01. This approach is the most efficient in terms of energy and throughput. The downside of this approach, however, lies in its poor reliability: a miss of a single packet can lead to errors in decoding all subsequent symbols.

- *Fixed-interval Streaming:* At the other extreme of the spectrum is the method that transmits at fixed intervals (illustrated in Figure 5 (b)), which has the least efficiency. In this scheme, we break the time into small intervals (310 ticks in our case) and transmit a symbol at the beginning of each interval. Therefore, a packet miss in an interval will not affect the reception of the next symbol. Using this protocol with intervals of 310 ticks, we need 620 ticks to transmit 2 symbols, while Back-to-back Streaming only needs 400 ticks. In the absence of clock drift, this method has high reliability. However, in reality, time re-synchronization between the sender and receiver is needed to compensate for clock drift effects.

- *Marker Streaming:* The third method tries to balance communication efficiency and reliability by sending symbols faster than Fixed-interval Streaming yet having less dependence between adjacent symbols than Back-to-back Streaming. The idea in this method is to employ a (temporal) "marker" at the boundary of two consecutive symbols by separating their transmissions

[3]In this paper, we note that the term tick refers to the unit of measurements reported by the software timer interface `Timer`, as opposed to the micro-controller time units.



(a) Back-to-back Streaming (b) Fixed-interval Streaming (c) Marker Streaming

Figure 6: The results of the single-sender timing channel experiment: the minimum Euclidean distance to the codewords, and the comparison between decoded symbols and the symbols sent.

by a predefined, short interval, the duration of which (denoted by ξ) is much smaller than the value of ϵ. We note that once the value of ξ is chosen, it will not be used by the inter-arrival codewords. For example, suppose we want to send 01 following 00. We first send the packets at ticks $(0, 50, 100)$ to represent symbol 00, then pause for $\xi = 20$ ticks, and send three more packets at ticks $(120, 370, 420)$ to represent symbol 01. In this example, the gap between packets transmitted at ticks 100 and 120 is the marker. Using this method, upon the reception of the last packet of a symbol, the receiver expects the start of the next symbol in ξ ticks.

The performance gain of Marker Streaming over Fixed-interval Streaming is obvious, and we will thus focus our discussion on its improved reliability compared to Back-to-back Streaming. In the latter, any single miss leads to decoding errors of all the subsequent symbols. Now let us look at the reliability of Marker Streaming. In the example mentioned above, we need to transmit six packets at ticks $(0, 50, 100, 120, 370, 420)$ for symbols 00 and 01. Suppose only one packet is missing. First, if the missing packet's transmission is neither at tick 100 nor at 120, then the marker will clearly indicate the boundary between two symbols. Suppose, for example, that packet 100 is missing, then the receiver will receive a sequence of packets at ticks $(0, 50, 120, 370, 420)$, and another packet at tick 440 if there are more symbols to send. From this sequence, the receiver can infer that two symbols were sent between ticks 0 and 440, and further infer that one of the marker packets is missing. To compensate for the loss, it will assume a transmission sequence at ticks $(0, 50, 120, 120, 370, 420)$, which is likely to decode correctly. Finally, in an adverse situation where multiple packets are missing for a symbol, the receiver won't be able to correctly decode, but the decoding error can be limited whenever a preset ξ is observed between two packets. Thus, Marker Streaming provides better reliability than Back-to-back Streaming.

Experimental Results: We have implemented the above three streaming protocols using MICA2 Motes. Before we could build the proposed timing channel on Berkeley Motes, we had to modify the existing TinyOS code to address a number of implementation-related issues. First, we created a deterministic packet transmission path to control the packet transmission times. Second, we modified the reception path to timestamp each packet arrival at the MAC layer to suppress receiver delay jitters.

In our experiments, we had a sender S, a receiver R and a reactive jammer J. The distance between S and R was 4 feet, and distance between J and the R was 3 feet. The transmission power was $-10dBm$. The sender S repeatedly cycled through the symbols $\{00, 01, 10, 11\}$, for an inter-arrival code $\Delta_{\mathcal{C}} = \{(50, 50), (250, 50), (150, 150), (50, 250)\}$ with $k = 3$ and $m = 4$. The receiver R timestamped each received packet, formed tuples of interarrival times (δ_1, δ_2) (since $k = 3$) from the packet stream following the streaming protocol, and decoded them based on their nearest distance to each codeword. In the experiments, a tick takes approximately 1 ms. For all three streaming protocols, we introduced one transmission error roughly at time $10s$.

The decoding results for each streaming protocol are shown in Figure 6. The upper figures plot d_r, the minimum distance between the observed interarrival tuples with all the codewords. A d_r value smaller than the threshold of $D = 40$ means the receiver can associate the tuple with a symbol, though the decoded symbol may be erroneous, while a d_r value larger than the threshold means the receiver cannot associate the tuple with any symbol. The bottom figures of Figure 6 depict both the decoded symbols and the original symbols from the sender. At any time, a miss match of these two corresponds to a decoding error.

Back-to-back Streaming (Figure 6 (a)), though yielding the highest raw send rate among the three (8.52bps), demonstrates a poor reliability: missing one packet can throw off the rest of the decoding process. On the other hand, the reliability of Fixed-interval Streaming was not as good as we anticipated (Figure 6 (b)), due to the clock drift between the sender and receiver. We also note that this scheme has the lowest bandwidth $5.36bps$. Finally, Figure 6 (c) shows that Marker Streaming demonstrates a good reliability. A summary of the results is provided in Table 2. In the experiments, the sender sent out 200 symbols in total, and

Methods	errors	Time (s)	goodput (bps)	raw bps
Back-to-back	162	47.00	1.62	8.52
Fixed-interval	119	61.12	2.17	5.36
Marker	1	51.31	7.76	7.80

Table 2: The statistics for transmitting 200 symbols, including the total times spent, the bandwidth. Each symbol contains 2 bits. In each case, the receiver failed to detect the presence of one packet.

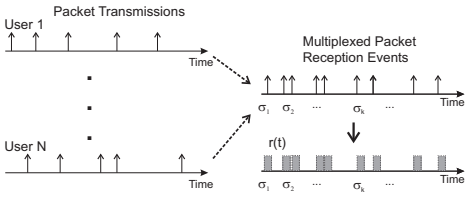


Figure 7: Packets from unsynchronized transmitters lead to a sequence of packet reception events at the receiver with time stamps $(\sigma_1, \sigma_2, \cdots)$, which is further transformed into a binary signal $r(t)$.

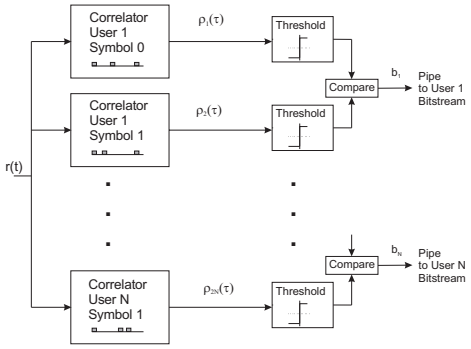


Figure 8: Decoding involves a bank of parallel correlators, followed by a threshold module, and a comparator. Resulting bits are appended to the corresponding user's timing channel output bit pipe.

the transmission error was introduced at the second packet of the 38th symbol. Since Marker Streaming carefully balances the efficiency and reliability, it achieved the highest goodput, 7.76bps (bits per second, each symbol conveys 2 bits of information), which is a factor of 3.6 improvement compared to the other two strategies.

5. MULTIPLE SENDERS

Generally we expect to have more than one sender and thus we next examine constructions for multiple senders. In this paper, we shall restrict our focus on star-topology scenarios (e.g. WLAN-type situations) involving multiple senders trying to communicate with a single receiver.

5.1 Coding Constructions

Constructing a multi-sender timing channel is challenging. First, packets from different senders are interleaved with each other. Second, these senders do not share a common, global clock as synchronizing such a clock would itself be a target of jamming. Thus, the challenge here is to extract individual sender's communications from the mixed packet arrivals at the receiver.

In this section, we start our discussion from the decoding strategy and then work our way back to a coding construction as the decoding requirements motivate the choice of the coding schemes. For the sake of simplicity, we limit our discussion to binary symbols. We assume there are a total of N users, and consequently $2N$ (asynchronous) codewords. Each user will transmit a sequence of packets for each symbol, and the receiver will detect packet transmissions and timestamp these events, giving a time sequence $(\sigma_1, \sigma_2, \cdots)$, as shown in Figure 7. To decode this sequence, we first map the arrival times into a binary signal $r(t)$ by $r(t) = \sum_{j=1}^{M} \chi(t - \sigma_j)$ where $\chi(t)$ is a square pulse of width B in ticks, and M is the number of packets detected. The signal $r(t)$ is a 0/1 signal with square pulses at times σ_j, as illustrated in Figure 7.

Similarly, we construct binary signals for each codeword, where the square pulses correspond to the intended packet transmission times (both starting from tick 0). The resulting $2N$ binary signals from the codewords, $c_i(t)$, will be used for correlation processing. Specifically, the receiver employs a bank of parallel correlators that operate on $r(t)$ in real-time, as shown in Figure 8. At any instance τ each correlator's output $\rho(\tau) = \int c(t)r(t+\tau)dt$ is compared against a detection threshold Γ to decide if the corresponding user might have transmitted that symbol. Outputs from both symbol-0 and symbol-1 threshold modules for a specific user are compared and, since a sender would only have sent one symbol per symbol period T, the comparator chooses the symbol that has resulted in a higher output. The resulting bit is then appended to that user's output bit pipe.

Clearly, the performance of our multi-user timing channel decoder depends on a particular codeword's auto-correlation, $\int c_i(t)c_i(t+\tau)dt$, and cross-correlations, $\int c_i(t)c_j(t+\tau)dt$, between codeword i and j. It is desirable to have a low cross-correlation between two codeword signals to ensure clear distinction between symbols. Meanwhile, it is desirable to have small non-zero lags of a codeword's autocorrelation function to allow for accurate, non-ambiguous determination of when a symbol was sent. For this choice of decoding scheme, the problem of building efficient multi-user timing channel codewords is thus equivalent to constructing binary signals with good auto-correlation and cross-correlation properties.

In general, this problem is quite challenging as the formulation involves an optimization over arbitrary transmission times and pulse widths B. However, a reasonable solution can be obtained if we let both the sender and receiver operate in a time-slotted manner. We now explain the implementation of our time-slotted system. To start, each sender partitions its local time into ticks (e.g. by using the TinyOS component `TimerC`), and we collect S successive ticks to form time slots. Further, a fixed number of successive time slots define an epoch. Asynchronously, each sender will transmit one symbol during each epoch by transmitting short (jammed) packets at a select set of time slots. In the context of this system, a multi-user timing channel codeword for a particular user and symbol is a description of time slots when that user should transmit a packet and when it should not transmit packets in order to convey that symbol.

Our time-slotting allows us to borrow codes from optical CDMA systems, which have an inherent time-slotting requirement. An optical orthogonal code (OOC) [11] is a family of (0,1) sequences with good auto correlation and cross correlation properties. A (n, w, λ) OOC C is a family of $(0, 1)$ sequences of length n with weight w, non-zero lag auto-correlation and cross-correlation less than λ. As an example, $C = \{1100100000000, 1010000100000\}$ is a $(13, 3, 1)$-OOC with two codewords. A shorthand notation for this code is $C = \{\{0, 1, 4\}, \{0, 2, 7\}\} \pmod{13}$. Now, each code-

word can be mapped into a corresponding binary signal

$$c(t) = \sum_{j=1}^{n} b_j \chi(t - jS)$$

where n is the number of bits in a codeword, b_j is the j-th bit value of the binary sequence representation for that OOC and S is the slot duration in ticks. In the above $(13, 3, 1)$ code, $c(t) = \chi(t) + \chi(t - S) + \chi(t - 4S)$, would correspond to u_1, symbol 0.

5.2 System Validation

Although the basic multi-sender scheme described above seems simple, many complex challenges arise.

Timeline: The first issue involves the selection of slot duration and epoch duration. A tick is the smallest unit of time that a node provides, and packet transmission/reception requires several ticks to transpire. As noted earlier, in our MICA2 prototype, a tick corresponds to $0.986ms$, and a 13 byte packet transmission corresponds to 12 ticks. Although it might seem natural to define a slot to have the same duration as a packet transmission, it is desirable to adopt a slot width longer than the packet duration to increase the likelihood that the entire packet falls within a slot. Further, since synchronization across senders is impossible, increasing the slot duration assists in decoding events where multiple senders transmit packets during the same slot, but yet these packets do not overlap in time. Finally, wider slots allow one to increase S, and thereby improve decoding gain against timing jitters in transmission/reception scheduling. Taking these three factors into consideration, in our MICA2 prototype, we have set our time slots to have a duration of 100 ticks for a 13 byte packet.

Encoding: Each sender partitions its local time into slots and epochs. In order to convey a bit of information to the receiver, a user maps the OOC into a number of specific slots and transmits a short packet during these time slots. In order to validate our multi-user timing channel, we have implemented a prototype involving 3 senders communicating with one receiver. For this example, we use a $(37, 3, 1)$ OOC, where we have the following 6 codewords.

User	Symbol 0	Symbol 1
u_1	$\{0, 1, 11\}$	$\{0, 2, 9\}$
u_2	$\{0, 3, 17\}$	$\{0, 4, 12\}$
u_3	$\{0, 5, 18\}$	$\{0, 6, 12\}$

A user will transmit a symbol during an epoch by transmitting at the corresponding time slots listed for that symbol. In this example, an epoch consists of 37 time slots. For example, if user u_1 wishes to transmit a '1' he will transmit a short packet during time slots 0, 2 and 9. We note that, although all codewords involve slot 0, the fact that the users are not synchronized implies that their 0 slot transmissions will not necessarily overlap.

Decoding: The receiver records the arrival times of each packet, $(\sigma_1, \sigma_2, \cdots, \sigma_M)$ (M being the number of packets). We preconstruct the binary waveforms $c_i(t)$ for each codeword, and construct the binary signal $r(t)$ for the packet arrival time sequence as described earlier. We constructed our waveforms at a rate of 1 point per tick. Since our signals are binary, the correlation processing may be efficiently done through bit masking, shifting and accumulation. The bank of parallel correlators operate on $r(t)$ in real-time and, at any instance τ, each correlator's output $\rho_j(\tau) = \int c_j(t) r(t+\tau) dt$ is compared against a detection threshold Γ. In our implementation, for a box width of $B = 10$ points, we used a threshold of $\Gamma = 16$, and compared the output against any output from that user's other symbol decoder.

5.3 Experimental Results

Our experimental setup involved five MICA2 motes: three senders (u_1, u_2 and u_3), one receiver, and one reactive jammer. The senders and the receiver form a star topology. We used the OOC code $(37, 3, 1)$ with each slot spanning 100 ticks. A sender sends out a packet at the beginning of the designated slots. The experimental results are summarized in Table 3.

In the first set of experiments, u_2 and u_3 sent out symbols while u_1 remained silent. We repeated the experiment three times and both u_2 and u_3 each sent a total of 630 symbols. For all three instances, the decoder correctly inferred that no symbol from u_1 was present. Out of the 630 symbols, the decoder only failed to detect 3 or 4 symbols. Specifically, the symbol error ratio for the link from u_2 to the receiver was 0.64%, while the symbol error ratio for the link from u_3 to receiver was 0.48%.

In the second set of experiments, all three users sent symbols. Again, we repeated the experiment several times, yielding a total of 650 symbols transmitted by each user. As reported in Table 3 the resulting symbol error ratios for each link were smaller than 0.5%.

We have shown in our experiments that we can restore the availability of the communication link in the presence of jamming, thereby creating a virtual physical layer. Since our timing channel is still constructed via a shared medium, it is possible that an adversary may try to subvert the integrity and reliability of the timing channel. To explore this, we implemented a reactive jammer and introduced a second adversarial entity that injected false packet events onto the communication channel as a periodic, constant rate source while the three senders continuously sent out their own symbols. The receiver recorded the interarrival time sequence and tried to decode the symbols sent by each legitimate sender. As before, we repeated the experiment three times to ensure repeatability and, in total, each user sent around 800 symbols. The resulting symbol error rates are plotted in Figure 9 versus the interval between the adversary's false packet events. We observe that when the jammer injects at a rate of one packet every 500 ticks, the aggregated symbol error rate was less than 1% (i.e. 99% of the time, the receiver decoded each user's symbols). When the adversary decreases its interval between fake packet events, the decoding error rate increases. Ultimately, when the jammer injects packets every 100 ticks, i.e. one false packet per slot, the decoding rate became close to 50%. Compared with normal users sending 3 packets per epoch, this is quite a severe operating condition. More complex cases, where the adversary knows the OOC and uses this to inject false bits into the

Scenarios	u_1	u_2	u_3
(u_2, u_3)	0%	0.64%	0.48%
(u_1, u_2, u_3)	0.15%	0.31%	0.46%

Table 3: The symbol error rate

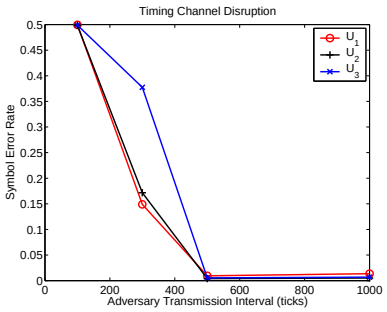


Figure 9: The aggregated symbol decoding error rates for three users using the $(37, 3, 1)$ OOC timing channel in the presence of a reactive jammer and an adversary that periodically injects false packet events at different rates.

timing channel must be addressed through authentication methods.

6. THE FOUR OUNCE OVERLAY

The timing channel created by failed packet reception events provides an overlay "physical" layer. The network layer, however, needs a reliable packet service beneath it and, hence, an overlay data link layer must bridge the gap between the timing channel's bitpipe and the network layer. To provide a packet-based interface, we need a framing strategy to break the bit stream into discrete frames, an error detection/correction scheme to deal with errors, and finally, an authentication mechanism to handle cases where an adversary corrupts the timing channel bit stream. In this paper, we shall focus on providing an unacknowledged connectionless service, but note that the overlay data link layer can be extended to an acknowledged connectionless service easily.

Framing: Framing within the 4-Ounce Overlay involves using conventional preamble detection to mark the start of a frame, while our end-of-frame delimitation is accomplished through length framing. A typical frame in our 4-Ounce overlay is depicted in Figure 10. Each frame begins with the preamble of 01010101. When the receiver sees 01010101, it starts to receive the frame. We have chosen to keep our framing simple since we have a limited data rate, and thus we employ a fixed frame length of 6bytes. In order to cope with the potential that the payload is accidentally decoded as preamble, we use bit stuffing. We note that we choose a small overlay frame size to achieve effective error recovery. Larger network layer packets are fragmented into 4-Ounce payloads with size 3 bytes, and an extra 1 byte sequencing field is added to the payload to facilitate reassembly. Following the payload, we append a 1 byte authentication field and a 1byte CRC field.

Error Detection and Correction: To enhance our overlay link reliability, we use small frames with error detection/correction. By breaking data into smaller packets, we can easily assert which data is correctly received. In our current implementation, we use CRC8 with polynomial $x^8 + x^2 + x^1 + 1$ for overlay frame error detection. Although our authentication field can serve the role of error detection, we have left the CRC field for forward evolution to error correction coding. We note that the overlay link reliability and

Figure 10: The 4-Ounce Overlay frame format.

efficiency can be enhanced if we replace error detection with error correction coding. We are implementing such enhancements into a future version of the 4-Ounce Overlay.

Authentication: Error checking can handle minor errors in the overlay. However, as noted at the end of Section 5, a clever adversary can seek to corrupt the timing channel by injecting false packet events. Such an attack is not an attack on the *availability* of the link, but is instead an attack seeking to corrupt the *integrity* of the link. We note that any physical layer is susceptible to such integrity attacks. Although the focus of this paper is primarily on *availability*, we have incorporated a simple authentication mechanism in our overlay link to cope with minor integrity threats. Specifically, we employ a standard message authentication code (MAC) to ensure that an adversary has not injected false bits or frames in our overlay. To construct the MAC, we assume each sender has a key K that it shares with the receiver, and a clock that provides a time t that has low granularity synchronization with the receiver (e.g. a few minute accuracy). For a particular payload P, the MAC is $MAC(t||P||K)$. One challenge is keeping the data frame size small and, although we recognize the limitations of short MACs, we have chosen to employ a 8-bit MAC based on an CBC-MAC, via a construction similar to the 4-byte CBC-MAC in [12]. To prevent simple replay attacks, the time t is included in the MAC, but we note that t need not be in the payload due to the loose synchronization we assume. Finally, we note that more powerful authentication can be achieved through higher-layer services.

Validation Results: We validated the 4-Ounce overlay by implementing a simple prototype where framing, error correction, and authentication were implemented as described above. Our overlay payload was 4 bytes per frame, we employed a 1 byte authentication field, and 1 byte of CRC. Our first experiment had a single sender using the $(37, 3, 1)$-OOC timing channel, and transmitted 60 overlay frames. In this experiment, we successfully received and decoded 56 frames, yielding a frame delivery rate of 93.33%. In our second experiment, we introduced an adversary and assumed the adversary knew the OOC that we used but not the authentication key K. We then had a legitimate sender send out 39 packets interleaved with the adversary sending 13 false packets and, as expected, we were able to successfully reject all forged packets that passed CRC checking.

7. RELATED WORK

Coping with jamming and interference is usually a topic that is addressed through conventional PHY-layer communication techniques. In these systems, spreading techniques (e.g. frequency hopping) are commonly used to provide resilience to interference [13, 14]. Although such PHY-layer techniques can address the challenges of an RF interferer, they require advanced transceivers.

The issue of detecting and mapping jamming for sensor networks was studied by Wood and Stankovic in [8]. The problem of jamming detection was further studied by Xu

et al. in [3], where the authors presented several jamming models and explored the need for more advanced form of detection algorithms to identify jamming. Additional jamming strategies were studied by Law et al. [4], and the efficiency of these methods was quantified in terms of the amount of resources needed to conduct an attack. Further work on jamming has studied MAC-layer jamming attacks on reservation-based medium access control schems [15].

Countermeasures for coping with jammed regions in wireless networks have been studied in [2, 7, 16, 17]. In [16], the use of error correcting codes is proposed to cope with jamming. In [2], channel surfing and spatial retreats are proposed. Spatial retreats was studied in more detail in [7], while a channel surfing system was built and studied in [5]. A variation of channel surfing was studied in [6], where the channel assignments are constantly adapted to emulate link-layer frequency hopping. Channel surfing strategies are not a suitable defense against broadband jamming. In [18], a protocol suite is proposed to cope with jamming that reduces the impact of jamming. Wormhole-based antijamming techniques have been proposed as a means to allow the delivery of important alarm messages [17].

Our work takes a different strategy to communicating in the presence of interference. We create a timing channel that serves as the basis for a new link-layer overlay, that can allow communication of important messages in spite of the presence of broadband interference. Timing channels have traditionally been studied in the context of covert communication, e.g. [19, 20]. Techniques to mitigate the effectiveness of covert channels have been studied in the context of network pumps [21]. More generally, the fundamental information-bearing capacity of timing and queuing channels has been studied in [22, 23]. In the jamming context, we assume the adversary knows the presence of communication, and our work differs from covert timing channels in that we seek availability instead of covertness.

8. CONCLUDING REMARKS

We have proposed the creation of a jamming-resistant timing channel to restore the availability of communication links in the presence of interference. The timing channel is built as a low-rate physical layer overlay on top of the traditional physical/link-layers. Our timing channel uses the detection and timing of failed packet receptions at the receiver, which we have shown is possible by time stamping CRC failures or by monitoring the signal strength. We then proposed interarrival codes for building a single-sender, single-receiver timing channel, and validated three different packet sequencing methods on the MICA2 platform. To cope with multiple sender, single-receiver cases, we presented an asynchronous code construction and a decoding procedure that employs a bank of parallel correlators. We evaluated our multi-sender construction in several cases and found that we could achieve an overlay link reliability better than 99%. Following the construction of the timing channel bit pipe, we presented our construction of an overlay data link layer, which provides framing, error detection/correction, and authentication. Our timing channel can resume communications and, although the data rate may be low, this accomplishment is significant when compared to the alternative of the jammer causing there to be no communication capabilities. Finally, we note that the timing channel construction is intended primarily to restore the availability of communication functions and that additional mechanisms, such as authentication, may be used in the overlay or at higher layers to assure the new channel is secure.

9. REFERENCES

[1] J. Bellardo and S. Savage, "802.11 denial-of-service attacks: Real vulnerabilities and practical solutions," in *Proceedings of the USENIX Security Symposium*, 2003, pp. 15–28.

[2] W. Xu, T. Wood, W. Trappe, and Y. Zhang, "Channel surfing and spatial retreats: defenses against wireless denial of service," in *Proceedings of the 2004 ACM workshop on Wireless security*, 2004, pp. 80 – 89.

[3] W. Xu, W. Trappe, Y. Zhang, and T. Wood, "The feasibility of launching and detecting jamming attacks in wireless networks," in *MobiHoc '05: Proceedings of the 6th ACM international symposium on Mobile ad hoc networking and computing*, 2005, pp. 46–57.

[4] Y. Law, P. Hartel, J. den Hartog, and P. Havinga, "Link-layer jamming attacks on S-MAC," in *Proceedings of the 2nd European Workshop on Wireless Sensor Networks (EWSN 2005)*, 2005, pp. 217 – 225.

[5] W. Xu, W. Trappe, and Y. Zhang, "Channel surfing: defending wireless sensor networks from interference," in *IPSN '07: Proceedings of the 6th international conference on Information processing in sensor networks*, 2007, pp. 499–508.

[6] V. Navda, A. Bohra, S. Ganguly, R. Izmailov, and D. Rubenstein, "Using channel hopping to increase 802.11 resilience to jamming attacks," in *IEEE Infocom Minisymposium*, May 2007, pp. 2526 – 2530.

[7] K. Ma, Y. Zhang, and W. Trappe, "Mobile network management and robust spatial retreats via network dynamics," in *Proceedings of the The 1st International Workshop on Resource Provisioning and Management in Sensor Networks (RPMSN05)*, 2005.

[8] A. Wood, J. Stankovic, and S. Son, "JAM: A jammed-area mapping service for sensor networks," in *24th IEEE Real-Time Systems Symposium*, 2003, pp. 286 – 297.

[9] "Tinyos homepage," http://webs.cs.berkeley.edu/tos/.

[10] Chipcon, "Chipcon cc1000 radio's datasheet," http://www.chipcon.com/files/CC1000_Data_Sheet_2_1.pdf.

[11] F. Chung, J. Salehi, and V. Wei, "Optical orthogonal codes: design, analysis and applications," *IEEE Trans. on Information Theory*, vol. 35, no. 3, pp. 595 – 604, 1989.

[12] C. Karlof, N. Sastry, and D. Wagner, "Tinysec: a link layer security architecture for wireless sensor networks," in *SenSys '04: Proceedings of the 2nd international conference on Embedded networked sensor systems*, 2004, pp. 162–175.

[13] J. G. Proakis, *Digital Communications*, McGraw-Hill, 4th edition, 2000.

[14] C. Schleher, *Electronic Warfare in the Information Age*, MArtech House, 1999.

[15] A. Rajeswaran and R. Negi, "DoS analysis of reservation based MAC protocols," in *Proceedings of the IEEE International Conference on Communications*, 2005.

[16] G. Noubir and G. Lin, "Low-power DoS attacks in data wireless lans and countermeasures," *SIGMOBILE Mob. Comput. Commun. Rev.*, vol. 7, no. 3, pp. 29–30, 2003.

[17] M. Cagalj, S. Capkun, and J.P. Hubaux, "Wormhole-Based Anti-Jamming Techniques in Sensor Networks," *IEEE Transactions on Mobile Computing*, pp. 100 – 114, January 2007.

[18] A. Wood, J. Stankovic, and G. Zhou, "Deejam: Defeating energy-efficient jamming in ieee 802.15.4-based wireless networks," in *Proceedings of the IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON)*, 2007, pp. 60–69.

[19] S. Cabuk, C. Brodley, and C. Shields, "IP covert timing channels: Design and detection," in *Proceedings of ACM CCS 2004*, 2004, pp. 178–187.

[20] I. Moskowitz and M. Kang, "Covert channels-here to stay," in *Proceedings of the 1994 Annual Conf. on Computre Assurance*, 1994, pp. 235 – 243.

[21] M. Kang, I. Moskowitz, and S. Chinchek, "The pump: a decade of covert fun," in *Proceedings of IEEE Computer Security Applications Conference*, 2005.

[22] V. Anantharam and S. Verdu, "Bits through queues," *IEEE Trans. on Info. Theory*, vol. 42, no. 1, pp. 4–18, 1996.

[23] I. Moskowitz and A. Miller, "The channel capacity of a certain noisy timing channel," *IEEE Trans. on Info. Theory*, vol. 38, pp. 1339–1344, 1992.