A Flexible Software Radio Transceiver for UHF RFID Experimentation: UW TR: UW-CSE-09-10-02

Michael Buettner*
*University of Washington
buettner@cs.washington.edu

David Wetherall†*
†Intel Research Seattle
david.wetherall@intel.com

ABSTRACT

We present the design and evaluation of a flexible UHF RFID reader suited for experimentation. Our reader is built using the USRP software radio platform in conjunction with software we developed in the GNU Radio framework. We believe this is the first inexpensive tool that readily enables changes to the physical and MAC layer of RFID systems. Our reader further has sufficient performance to interoperate with commercial EPC Gen-2 RFID tags and achieves roughly 75% of the range of a commercial reader with comparable power. To support communication with commodity RFID tags, we aggressively reduced system latency from the 10s of milliseconds typical of USRP applications to consistently under 500 μs.

1. Introduction

Radio Frequency IDentification (RFID) is an emerging wireless technology that allows small, inexpensive computer chips to be remotely powered and interrogated for identifiers and other information. While there are many kinds of RFID, e.g., HF RFID in credit cards, recent advances in RFID have focused on passive UHF RFID as standardized by the EPC Class-1 Generation-2 (Gen 2) specification in 2004 [6].

UHF RFID was originally developed as a replacement for barcode identification systems. It provides key advantages such as a read range of up to 10 meters, non-line-of-sight operation, high inventory rates, and rewritable product IDs. It has seen widespread deployment for supply chain pallet tracking and is rapidly being expanded to new applications. These include large scale, item-level tracking of consumer goods such as apparel[2] and books[1], and even the tracking of people, e.g., multiple pilot studies in which US school children are tracked as they enter school buses and buildings. Earlier this year, the US Passport Card and the New York and Washington state enhanced drivers licenses were introduced which have integrated Gen 2 RFID tags; this is intended to reduce delays at border crossings. Additionally, as the capabilities of RFID devices advance to include storage and sensing [21], new uses that stretch the application space even more will emerge.

Given this rapidly growing application space, understanding UHF RFID operation in practice and experimentating with realistic UHF RFID systems is of significant interest. For instance, privacy and security issues are central in any application that tracks people directly or indirectly. Yet almost all work on RFID security via lightweight cryptographic schemes has been done via paper designs and analysis rather than experimentation; we are aware of only one exception [5]. The reliability and performance of RFID readers is also of importance, especially in dense, fine-grained settings (such as item-level tracking) and for demanding applications (such as searching over the states of objects in a ubicomp application). However, there is almost no published information on low-level RFID performance in these settings. Our prior work on this topic concludes that current reader systems suffer various performance degradations and have ample opportunity for improvement [4]. And looking forward, sensor-equipped RFID tags will pose a new set of problems for researchers. This is because the Gen 2 protocol was designed with object identification in mind rather than gathering sensor data.

This dearth of low-level experimentation with RFID systems is a direct consequence of the current lack of tools available to researchers. Existing RFID readers are generally black box systems which provide only limited configuration and return high-level results that simply indicate identifiers of tags that are in range. These systems do not provide the low-level flexibility to observe or modify the MAC or PHY layers, which makes the study of existing protocols difficult; let alone experimentation with alternative designs. Some high-end RFID test equipment is available in the form of protocol analyzers, but it is expensive (>100K) and of limited use in evaluating changes to the protocol as opposed to monitoring operation.

In this paper, we present what we believe is the first inexpensive, open-source platform for low-level UHF RFID experimentation that gives users control of the PHY and MAC layers. It consists of the Universal Software Radio Peripheral (USRP) hardware and software that we have developed using the GNU Radio framework that implements a EPC Class-1 Generation-2 reader. The use of this framework provides a high-level of flexibility allowing MAC and PHY functionality to be changed by simply re-writing

user-level software.

To the best of our knowledge, this is the first implementation of an interactive, real-world networking protocol implemented using the USRP. There has been significant prior work that uses the flexibility of the USRP to implement wireless protocols. However, moving functionality away from the NIC is at odds with performance, particularly with respect to the strict timing requirements of essentially all network protocols. As a result, most experimentation has been receive only or not done in real-time. Our RFID reader, in contrast, operates with commodity RFID tags.

Our contributions are twofold. The first is our flexible RFID reader platform. In the body of the paper, we benchmark our reader and report the results to demonstrate that it provides a useful level of performance even when compared to commercially available RFID readers. We further describe potential applications of our reader to highlight how it enables RFID experimentation, and present two example studies.

The second contribution is the set of techniques that we use to meet the timing requirements of the Gen 2 protocol. We reduce latency by identifying bottlenecks in the GNU Radio architecture and using techniques to eliminate or tune internal buffers, and schedule signal processing intelligently. The cumulative effect of these techniques is to reduce our system latency by almost two orders of magnitude from the typical 10s of milliseconds to reliably under 500 μs.

The remainder of this paper is organized as follows. In Section 2 we present the goals of this study and discusses the challenges. Section 3 and 4 provide targeted introductions to the two technologies that are key to understanding our work, namely Gen 2 RFID, and the USRP and GNU Radio. We then present the architecture of our reader, and the techniques we use to reduce system latency in Sections 5 and 6. In Section 7 we evaluate our reader performance, and in Section 8 we discuss applications. We then describe related work and conclude.

2. Goals and Challenges

Our goal is to develop a Gen 2 RFID transceiver that can communicate with commercial tags while giving researchers complete control over the physical and MAC layers of the protocol. Such tranceiver flexibility would enable the detailed study of current RFID systems, and provide a platform for experimentally validating proposed protocols; both of which are difficult using current platforms.

To provide a low barrier of entry for experimentation, our transceiver should be based on common off the shelf components, and must not use specialized or custom built hardware. Additionally, it should not require that users have FPGA expertise, nor a deep background in signal processing.

To meet these goals, our system is built using the Universal Software Radio Peripheral (USRP) and the GNU Radio signal processing toolkit. The USRP is a low-cost, general purpose RF front-end for software radio development. This device interfaces with a standard PC via USB, with essentially all signal processing being performed on the host using GNU Radio.

With this, the Gen 2 protocol can be implemented completely in software giving unprecedented control over the behavior of the transceiver. Additionally, the architecture of GNU Radio allows for a highly modular transceiver design. This enables us to localize the PHY and MAC layers of the Gen 2 protocol so that researchers can focus only on those aspects of the protocol in which they are interested.

Using the USRP and GNU Radio for our transceiver is ideal in terms of flexibility. However, the platform has limitations with respect to capability. Previous work has highlighted two major limitations when using the platform to implement real world prototocols.

First, the maximum signal bandwidth that can be supported is approximately 8 MHz. This precludes the implementation of protocols such as 802.11 and WiMAX which use much larger channels. Fortunately, the bandwidth requirements of the Gen 2 protocol are minimal and are well within the capabilities of the USRP.

The second major limitation is transceiver latency. Performing signal processing in software at the host greatly increases system latency compared to convential hardware tranceivers. Specifically, the platform incurs the latency cost of the low rate USB interface, a series of buffers in the receive and transmit chains, and the fact that the GNU Radio software is running on a general purpose computer on top of an OS. Previous work using the USRP and GNU Radio has shown transceiver latency on the order of tens of milliseconds, far too high for implementing most wireless MAC protocols.

Because the Gen 2 protocol is designed for use with very low cost, low capability devices, i.e. simple RFID tags that cost only a few cents, the timing requirements of the MAC are relaxed compared to most protocols. Depending on the system configuration, the maximum time in which an *ACK* must be sent can be as high as 500 μs. While this is much greater than is seen with other wireless protocols, it is two orders of magnitude lower than prior implementations have achieved. Consequently, reliably meet-

ing the timing requirements of commercial tags was the major challenge to implementing our transceiver.

3. Class-1 Generation-2 RFID Primer

In this section we describe the essential features of Gen 2 RFID to highlight the functionality that our reader must implement, paying special attention to timing considerations.

The Gen 2 standard defines communication between RFID readers and passive RFID tags in the 900 MHz band. A reader transmits information to a tag by modulating an RF signal, and the tag receives both down-link information and the entirety of its operating energy from this RF signal. For up-link communication, the reader transmits a continuous RF wave (CW) which assures that the tag remains powered, and the tag communicates by modulating the reflection coefficient of its antenna. By detecting the variation in the reflected CW, a reader is able to decode the tag response. This is referred to as "backscattering".

3.1 Physical Layer

The Gen 2 down-link uses Amplitude Shift Keying (ASK), where bits are indicated by brief periods of low amplitude, and Pulse-Interval Encoding (PIE), where the time between low amplitude periods differentiates a zero or a one. The reader can choose pulse durations that result in data rates ranging from 26.7 kbps to 128 kbps.

Through the use of a structured down-link preamble the tag determines the pulse lengths being used by the reader, and also what data rate should be used by the tag for up-link communication. These settings allow for an ASK up-link with a frequency ranging from 40 kHz to 640 kHz. Along with setting the up-link frequency, the reader also sets one of four up-link encodings, namely FM0, Miller-2, Miller-4, or Miller-8. The Miller encodings are more robust to error but have a lower data rate as there are more subcarrier cycles per bit. The up-link frequency along with the up-link encodingdetermines the data rate, which can range from 8 kbps to 640 kbps.

3.2 MAC Layer

The Gen 2 MAC protocol is based on Framed Slotted Aloha [19]. Each frame, or *Query Round*, has a number of slots and tags reply in one randomly selected slot per frame. Figure 1 shows the reader and tag transmissions that make up a *Query Round*. First, the reader can optionally transmit a *Select* command which limits the number of active tags in the round. A *Query* command is then transmitted which determines the up-link data encoding and specifies the number of slots in the *Query Round*.

When a tag receives a *Query* command it chooses a random slot in which to reply, and if it chooses slot 0 it responds immediately with a random 16-bit number (*RN16*). After receiving the *RN16* the reader sends an *ACK* command which includes the *RN16*, and the tag backscatters its *ID* (referred to as EPC in the figure). The randomized slot selection and three way hand-shake arbitrates channel access between multiple tags, with the 128 bit *ID* being sent only after the channel has been reserved.

After a tag transmits its ID, a subsequent *QueryRepeat* command causes the tag to be inactive during subsequent *Query Rounds*. Additionally, a *QueryRepeat* signals the end of the slot. The remaining tags decrement their slot counter and transmit the *RN16* if their slot counter reaches 0. After iterating through all of the slots the reader begins another *Query Round*, possibly changing the number of slots to best accomodate the remaining tags. A series of *Query Rounds* is performed until no tags reply as this indicates that all tags have been read.

3.3 Timing

The Gen 2 protocol defines strict timing requirements for reader to tag communication as shown in Figure 1. In particular, the timing between dependent transmissions, such as the *RN16* and *ACK*, are precisely specified. For our purposes, *T1* and *T2* are of particular interest. These timing requirements are specified in terms of the up-link frequency and are independent of the up-link encoding. Hence, when using lower up-link frequencies the timing requirements are relaxed.

T1 specifies the time at which a tag must begin its response to a reader command, measured from the last rising edge of the reader command to the first rising edge of the tag response. *T1* is defined as 10 times the period of a single up-link cycle with an accuracy of approximately +/- 2 μs. For instance, *T1* is approximately 250 μs when using an up-link frequency of 40 kHz and 15.6 μs when the up-link is set to 640 kHz.

T2 specifies the maximum allowable time in which a reader must respond to a tag response. If a reader fails to respond within this period, the reader command will be ignored by the tag. For instance, if a reader fails to send an *ACK* within *T2* the tag will not transmit its identifier. The maximum *T2* is specified to be 20 times the period of an up-link cycle, which results in *T2* being 500 μs and 31.25 μs when using up-link frequencies of 40 and 640 kHz respectively.

4. The USRP and GNU Radio

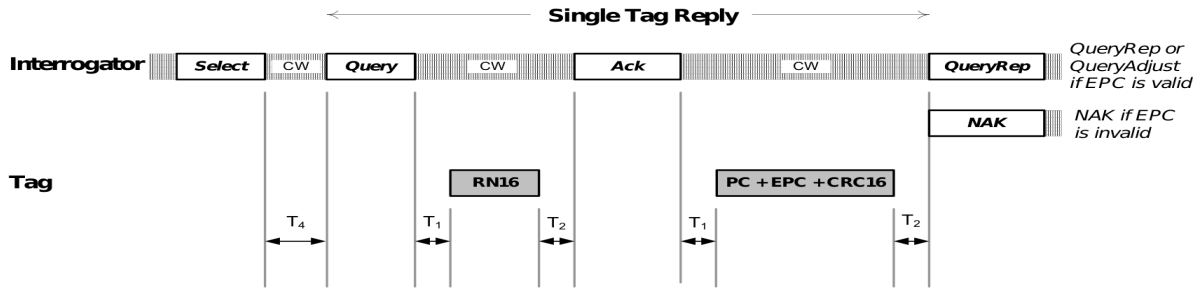

Figure 1: Gen 2 Protocol (Courtesy of EPCglobal)

The USRP is an open-source, general purpose platform for software radio development. When used in conjunction with the GNU Radio toolkit, it enables rapid prototyping of radio systems and low-level wireless experimentation. The flexibility, low cost, and vibrant user community of the USRP and GNU Radio make it an attractive architecture for our RFID transceiver.

4.1 USRP

The USRP provides an interface between four 64 Msps analog to digital converters (ADCs), four 128 Msps digital to analog converters (DACs), and a USB 2.0 interface for communication with a host computer. Daughterboards are available for the USRP that convert RF signals to and from an intermediate frequency (IF) that is within range of the ADCs and DACs. These daughterboards enable operation at a range of frequencies including the 900 MHz ISM band at which Gen 2 RFID operates. The USRP can be equipped with two daughterboards that function independently, which enables simultaneous transmit and receive.

While the USRP sampling rate is 64 Msps, the USB interface acts as a bottleneck and the signal must be decimated at the USRP. This results in a maximum effective sampling rate of 8 Msps, and approximately an 8 MHz band being received at the host. Additionally, transceivers must allocate USB bandwidth to both the receive and transmit streams. As the sampling rate of the ADC and DAC are constant, this is achieved by controlling the decimation and interpolation rates of the USRP.

4.2 GNU Radio

The GNU Radio toolkit is a software library and runtime system designed as a counterpart to the USRP. It provides signal processing blocks, such as filters, and infrastructure for composing blocks into signal processing flowgraphs. Flowgraphs are implemented as user level Python applications which configure the USRP and specify the connections between

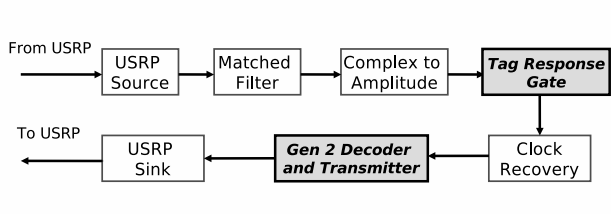

Figure 2: Block Diagram of our Reader Architecture

the C++ based signal processing blocks. These flowgraphs then execute using the GNU Radio runtime system, which connects the blocks using FIFOs and provides a scheduler that controls the flow of samples through the graph. By implementing custom signal processing blocks, GNU Radio can be used to realize a wide range of wireless protocols.

4.3 Hardware

5. Reader Software Architecture

Using the USRP and GNU Radio, we implemented a flexible Gen 2 reader. Our GNU Radio based software architecture uses standard blocks provided by the toolkit, along with custom blocks that implement the Gen 2 specific functionality. We first give an overview of the architecture and describe how it is configured. We then describe in detail the functionality of our custom blocks.

5.1 Overview

Figure 2 shows the block diagram of our reader architecture with our custom blocks highlighted. The first GNU Radio block in the receive chain is the source which pulls received samples in from the kernel and feeds the flowgraph. The samples are passed first to a matched filter which is configured to maximize the signal-to-noise ratio (SNR) of tag transmissions. As the Gen 2 up-link uses amplitude shift keying (ASK), the output of the filter is transformed

from a stream of complex I and Q values to a stream of amplitude values.

The resultant signal is then sent to the *Tag Response Gate* which acts as a signal gate, only ungating the incoming signal when a tag response must be decoded. When ungated, the signal is passed through a clock recovery block which resamples the tag response and outputs one sample per subcarrier cycle. These samples then enter the *Gen 2 Decoder and Transmitter* which implements the protocol specific behavior and generates the ASK modulated reader commands for transmission. These commands are then sent to the sink where they are passed to the kernel for transmission to the USRP.

5.2 Configuration

The configuration parameters for our architecture can be broken into three categories; USRP configuration, Gen 2 protocol parameters, and GNU Radio block parameters. USRP configuration for our transceiver consists solely of setting the frequency in the range of 902-928 MHz. For reasons discussed in a later section, the decimation and interpolation rates are fixed.

All Gen 2 MAC parameters can be configured, such as the number of slots in the frame and the up-link encoding. The down-link and up-link rates in Gen 2 are determined by the pulse widths used during the preamble that precedes the *Query* command; these pulse widths are exposed as configuration options. Based on the up-link PHY parameters the pulse width for the matched filter is set, along with the filter decimation in order to provide two samples per subcarrier cycle as required by the clock recovery block.

5.3 Tag Response Gate

The Gen 2 protocol uses a reader talk first communication paradigm, which means that a tag transmission can only occur immediately following a reader command. Thus, the *Tag Response Gate* only passes the received signal to the downstream block after a reader command is detected. After the tag response has been decoded a signal is sent to the *Tag Response Gate* which gates the signal until the next reader command is received. By gating the signal, computation is reduced as clock recovery and tag decoding are executed only as needed.

The *Tag Response Gate* also increases the modularity of our implementation. The block simply detects the last pulse of a reader command and passes through the tag response. Consequently, tag response signal processing and MAC implementation is localized and separated into one or more downstream blocks. This allows for the "dropping in" of different signal processing algorithms and application specific protocol behavior.

5.4 Gen 2 Decoder and Transmitter

The *Gen 2 Decoder and Transmitter* block decodes the symbols of the tag transmissions, implements the Gen 2 MAC protocol, and generates the amplitude modulated reader commands. The input to the block is a stream of samples with one sample per subcarrier cycle. This allows us to detect the preamble of the tag response via correlation, and the subsequent bits are then decoded also using a a correlator. By correlating for individual bits we take advantage of the processing gain inherent in the Miller up-link encodings.

We provide functions that generate ASK modulated Gen 2 commands. For example, to generate a *Query* command the *gen_query()* function uses the configured PHY and MAC layer parameters to construct the bit level command, calculate the CRC, and transforms these bits to the ASK modulated signal.

The complete Gen 2 MAC protocol is implemented. Tag *IDs* include a CRC, and if the *ID* passes the checksum a *QueryRepeat* is sent and a *NAK* is sent otherwise. The number of tags is configurable, and the reader determines the correct number of slots for each *Query Round*. As tags are read, the reader recalculates the appropriate number of slots and modifies this for the next round. When all tags have been read, the reader powers down for 1 ms. The number of times that the readers powers up and reads tags is configurable.

For analysis, our reader writes data to a log file. Each entry is labeled with the command or tag response name, and consists of the time the event happened in microseconds and the decoded bits and SNR values when appropriate. To reduce the overhead of logging data our implementation stores the complete trace in memory, only writing to disk when the reader powers down.

6. Reducing System Latency

A key challenge for our reader is that the flexibility of the USRP and GNU Radio comes at the cost of high transceiver latency. The acceptable system latency for a Gen 2 reader is determined by the up-link rate and is 20 times the period of a single up-link cycle. This means that our reader must have a latency of no more than 500 μs to operate with commodity tags using a 40 kHz up-link. To achieve this, our reader implementation had to be highly optimized in order to communicate with commodity RFID tags. Aside from optimizing our software, we also attempted to increase the performance of our

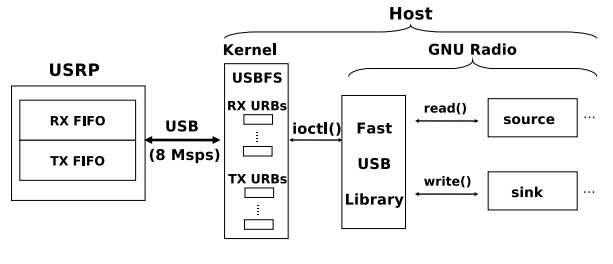

Figure 3: System Diagram

host computer. Specifically, we use a host machine with a quad-core 3.2 GHz Xeon processor running the Linux 2.6.20-16-lowlatency kernel, and the GNU Radio runtime is given the highest priority.

In this section we first discuss the sources of latency when using the USRP and GNU Radio, particularly for low bandwidth applications. We then describe the techniques we use to reduce the impact of each of these sources, showing experimental results that pinpoint the effect these have on system latency.

6.1 Sources of Latency

Latency in our receive chain can be attributed to four factors:

1. The period of time from when a signal hits the antenna until it is digitized, processed, and placed in the receive FIFO on the USRP.

2. The time from when a sample is placed in the USRP receive FIFO until it is transmitted across the USB bus and received by the host.

3. The time a sample spends in the USB receive buffer on the host before it reaches the first GNU Radio signal processing block.

4. The time it takes for GNU Radio to process a given signal

Analogs of the first three of these are experienced again, in the reverse direction, for the transmit chain. The latency cost of (1) is a constant 16 μs, and cannot be reduced.

Figure 3 shows the subsystems composing the path to and from the USRP and GNU Radio. As an example, we will explain its functioning for the receive chain. The received RF signal is digitized and the samples are placed in the 10 kB RX FIFO on the USRP. The rate at which the FIFO fills, and the duration represented by each sample, is determined by the USRP decimation rate set by the GNU Radio application. USB packets have a 512 byte payload and hence contain 128, 4 byte samples. These packets are only sent when full and only if the host has sufficient buffer space to receive them. The maximum sustainable data rate across the USB is 8 Msps, and the bandwidth must be shared between the receive and transmit signals. Selecting the appropriate decimation rate has a large impact on (2).

The Linux USB subsystem on the host maintains a set of USB Request Blocks (URBs), which are essentially buffers that store samples until GNU Radio is ready to process them. The "Fast USB" subsystem, which is part of GNU Radio, is responsible for draining these URBs and resubmitting them to be refilled by the kernel. Whenever the GNU Radio *source* block requests more samples, all full URBs are drained. However, if all receive URBs are full the USRP will begin to drop samples, and if no URBs are full when the *source* requests samples the "Fast USB" subsystem will block. The size and number of URBs is set by the application and largely impacts (3).

Finally, (4) depends on the complexity of the signal processing and the number of samples that are processed for a given signal.

6.2 Eliminating the Transmit Buffer

An RFID reader must send a continuous RF wave (CW) to power and communicate with tags. However, there is a 32 kB buffer in the *sink* block of the GNU Radio flowgraph, some number of transmit URBs in the kernel, and another 10 kB FIFO on the USRP. When transmitting at the maximal 8 Msps these transmit buffers introduce over 1 ms of system latency as the buffers are always kept full and reader commands are enqueued at the tail. Additionally, transmitting at 8 Msps leaves no USB bandwidth for receiving samples. As we must receive as well as transmit, the sample rate of the CW could be reduced as much as possible leaving the remainder of the USB bandwidth for receive. However, this results in a 250 ksps transmit sample rate incurring 40 ms of latency caused by the transmit buffers. Even if the size of the host transmit buffer is reduced, the 10 kB buffer on the USRP introduces significant, and unnecesary, latency.

To eliminate this latency, we modified the FPGA of the USRP to send the CW independently. With this modification, the USRP continually sends a sine wave at the highest transmit power. Our reader only transmits reader commands, and when a packet arrives from the host the USRP begins transmitting the samples and when the transmission is complete, the USRP returns to sending the CW. This assures an empty transmission buffer on the host and effectively bypasses the transmit buffer completely. Because the

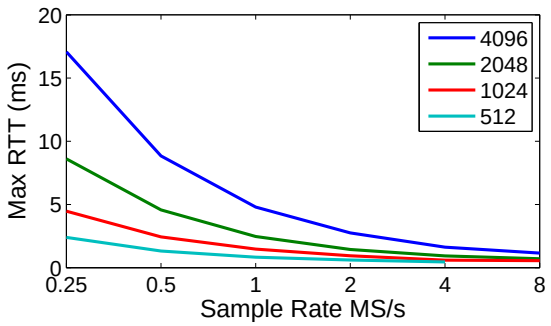

Figure 4: RTT across GNU Radio *read()* request sizes

CW is a fundamental but simple aspect of the Gen 2 protocol, moving this functionality to the FPGA significantly reduces latency without greatly reducing the flexibility of our system.

6.3 Rate Matching GNU Radio *read()*s

Previous work using the USRP and GNU Radio has shown system latencies on the order of 10s of milliseconds [22], with latency increasing as the sample rate decreases. To determine the root cause of this behavior we replicated their experiment, including the use of eight 2048 byte (512 samples) USB Request blocks. In our experiment, we used two host machines *HOST1* and *HOST2*, each equipped with a USRP with two attached 900 MHz daughterboards. *HOST1* emits a pulse once per second. *HOST2* uses a simple pulse detector to detect the falling edge of the pulse and immediately transmits a shorter pulse in response. Both the original pulse and the response pulse are received at *HOST1* and the time between the pulses is measured. It should be emphasized that the time between the two pulses does not depend on *HOST1*, and the interval between the two pulses is an accurate measure of the total system latency of *HOST2*. We measure the time from the falling edge of the first pulse to the rising edge of the response pulse.

The prior work, and the GNU Radio documentation, states that latency in the receive chain depends largely on the choice of USB Request Block size. However, we found the effects of this design choice are superseded by the behavior of the GNU Radio scheduler and *source* block. The scheduler only schedules the *source* block when all previous samples have been processed by the flowgraph, and it tells the *source* to produce enough samples to completely fill the input buffer of the downstream block. This buffer is hardcoded to be 16 kB, or 4096 samples.

Figure 4 shows the maximum round trip time (RTT) of our simple edge detector as we vary the request size of the *read()* function call; *read()* is a blocking function called by the *source* that blocks until the request can be satisfied. When requests are for 4096 samples, the default for GNU Radio, the latency of our system closely matches the microbenchmarks presented in [22]. This latency is largely due to GNU Radio blocking until more samples arrive at the host, particularly at low sample rates. At 8 Msps 4096 samples represents 512 μs worth of signal while at 250 ksps the same number of samples represents over 16 *ms* worth of signal. Because the signal of interest, the last edge of a bit for instance, will be randomly located in a given block of 4096 samples it will incur 256 μs of latency on average (in the 8 Msps case). We refer to this as the *fundamental latency*, as it is an unavoidable result of packetizing a continuous signal.

To reduce the *fundamental latency*, we reduced the *read()* request size to 2048, 1024, and 512 samples, and the RTT decreased monotonically as one would expect. However, with a request size of 512 samples the overhead was such that our application was unable to support 8 Msps. With a more computationally intensive signal processing graph, or if the graph has a highly variable workload, selecting a static request size that minimizes latency while not introducing excessive overhead is difficult.

Fortunately, the appropriate request size can be determined dynamically by simply requesting the number of samples that are already available at the host. This approach, which we refer to as *rate matching*, eliminates latency due to blocking and limits overhead by processing all available samples at once. To implement *rate matching* we modified the *read()* function to be non-blocking and to return all samples that have been received by the host. In our experiment, system latency with *rate matching* enabled saw the same performance as the 512 sample case, but it also supported 8 Msps with latency below the 1024 sample case. When *rate matching* is used the bottleneck for low bandwidth applications becomes the USB Request Block size.

6.4 Reducing USB Request Block Size

In the previous experiment we used 512 sample USB Request Blocks (URBs), which are the blocks that transfer samples from the kernel to user space. Consequently, a 512 sample *read()* request size equates to one URB; i.e. even *rate matching* will block if there are less than 512 samples available in the kernel. When *rate matching* is enabled, the user configurable URB size *does* affect system latency.

Figure 5 shows the maximum RTT as we reduce

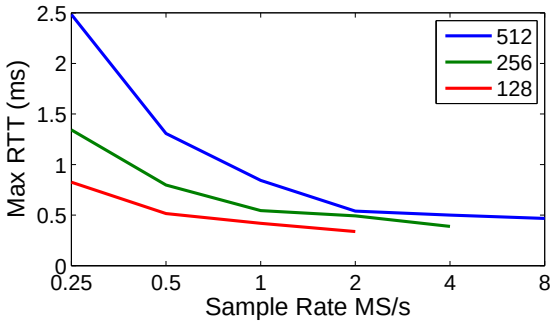

Figure 5: RTT across USB Request Block sizes (rate-matched *read()*)

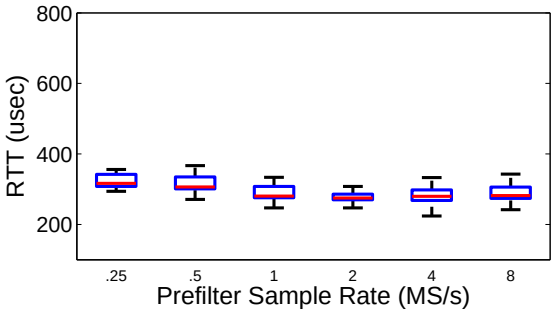

Figure 7: SDR reader latency with protocol aware scheduler. Post-filter sample rate fixed at 250 ksps

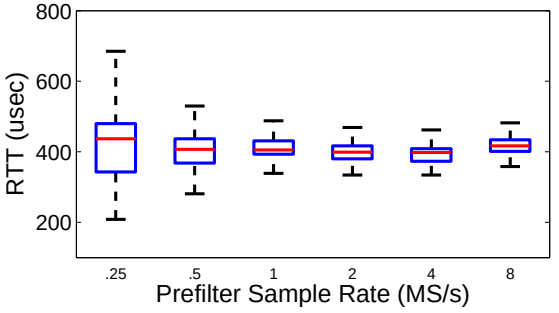

Figure 6: SDR reader latency with default scheduler. Post-filter sample rate fixed at 250 ksps

the URB size to 128 samples (512 bytes), the minimum size given that the USRP firmware transmits 128 sample packets across the USB. As mentioned previously, the 512 sample case can now support 8 Msps due to *rate matching* changing the request size dynamically. However, as we reduce the URB size the kernel to userspace overhead increases to a point where we are unable to support 8 Msps with 256 sample blocks, or 4 Msps with 128 sample blocks. However, for this simple signal processing graph, the maximum RTT when using *rate matching* with 128 sample URBS is 340 μs at 2 Msps, which is within the latency and bandwidth requirements for "Gen 2" RFID. Consequently, our reader implementation uses *rate matching* with 128 sample URBs.

6.5 Reducing Sample Rate at the Host

Along with reducing the *fundamental latency*, overall system latency can be lowered by reducing the computation of the signal processing. This can be achieved by optimizing algorithms, but also by reducing the number of samples that must be processed for a given signal. As "Gen 2" RFID has up-link rates in the 40-640 kHz range, the matched filter in our reader implementation additionally reduces the sample rate via decimation.

Figure 6 shows the RTT of our "Gen 2" reader when we fix the post filter sample rate to 250 ksps, sufficient to support a 125 kHz uplink, while varying the USRP sample rate (and the decimation rate of the matched filter, in order to maintain the 250 ksps output rate). The RTT is measured by reading a commercial RFID tag with our reader, and measuring the time from the last bit of the tag's *RN16* to the first bit of the reader *ACK*. The measurement is done using the infrastructure presented in [4].

With a prefilter sample rate of 250 ksps, where the USRP streams samples at 250 ksps and the filter performs no decimation, the max RTT is similar to the 250 ksps case seen in Figure 5 for the simple edge detector. However, the RTT is reduced when streaming samples from the USRP at a higher rate, and then reducing the sample rate with the filter on the host. This is because the *fundamental latency* is reduced by increasing the sample rate but the signal processing graph must process fewer samples, which reduces processing time. The benefits of this approach are minimal beyond 2 Msps, where the maximum RTT is 469 μs. This is sufficient to interoperate with commercial tags using a 40 kHz uplink.

6.6 Managing the GNU Radio Scheduler

Reducing the sample rate at the host sufficiently reduced latency to enable interoperation with commercial tags, but just barely. To increase our margin of error, we implement *protocol aware scheduling* to further reduce system latency. With *protocol aware scheduling*, once a tag response preamble is detected, the block requests exactly the number of samples that make up the remainder of the tag response. This is in contrast to the default behavior where the next

request would result in all available samples being passed to the flowgraph by the *source* block. This default behavior results in either, 1) the early stages of the flowgraph processing more samples than are necessary, increasing compute time, or 2) the flowgraph processing the sample in small chunks when, in this case, blocking is appropriate as we know exactly how many samples to wait for.

Figure 7 shows the RTT of our reader with *protocol aware scheduling* enabled, and shows that making use of protocol knowledge can significantly reduce latency. This is because the block makes requests sized to precisely encompass the tag reply. However, as the URBs still comprise 128 samples, the *time* granularity with which the requests can be satisfied depends on the prefilter sample rate. For example, if the block requests 12 μs worth of samples (3 samples at 250 ksps) in order to process the rest of the tag reply, a 250 ksps prefilter sample rate will return 512 μs worth of data whereas at 8 Msps only 16 μs will be returned. In the former, 500 μs of latency is introduced compared to only 4 μs in the latter case. This is why the gains are larger for lower prefilter sample rates. Beyond 2 Msps the benefits are outweighed by the overhead of high sample rates as described earlier. By using *protocol aware scheduling* the maximum RTT is approximately 300 μs at 2 Msps, well below the 500 μs required by the "Gen 2" protocol.

Previous results had shown minimum latencies for full duplex USRP based systems to be on the order of 10s of milliseconds. Furthermore, microbenchmarks suggest best case latencies on the order of 3 ms. In our implementation, we were able to achieve maximum latencies of around 300 μs by addressing each of the major bottlenecks in the system. Along with having the FPGA transmit the "Gen 2" *continous wave* (bypassing the cost of the transmit buffer) and using the smallest USB Request Block available, we presented two techniques of more general interest:

- *Rate matching*, which automatically processes the number of samples that matches the processing speed to the sample rate. This eliminates blocking and minimizes overhead, reducing average latency.

- *Protocol aware scheduling*, which uses knowledge of the protocol and the physical layer to optimize requests for samples. This minimizes the latency for a given operation.

7. Gen 2 Reader Evaluation

In this section we present an evaluation of our reader to show how well it performs, with a focus on the degree to which the transceiver can be used for further research. Additionally, we discuss the limitations of our reader and identify how it can be improved.

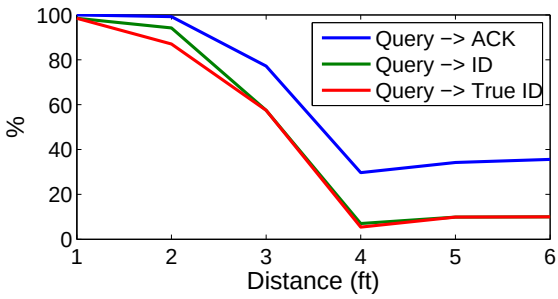

Figure 8: Performance of our Gen 2 Reader

7.1 Experimental Setup

For all experiments in this evaluation, we use Alien "Omni-Squiggle" tags attached to a sheet of poster board with the tags being approximately 4 feet off the ground. Our reader uses a 40 kHz up-link, and has an output power of 75 mW as measured by a power meter. Unless noted, we use a ThingMagic integrated bi-static antenna that has two independent antennas housed in a single enclosure. The experiments were conducted in a standard office setting, and we attempted to produce ideal conditions for the reader; i.e. line of sight with minimal objects in the area. As a comparison, we use the ThingMagic Mercury 5e reader and related experiments are conducted without moving the tags.

7.2 Reader Performance

For our reader to be useful it must be able to read tags reliably at a range of at least a few feet. We performed an experiment reading a single tag repeatedly at increasing distances while measuring the read success of each *Query*. At each distance we performed 5000 Query attempts. Figure 8 shows three metrics: 1) the percentage of *Query* commands where the *RN16* from the tag was decoded by the reader, thus causing an *ACK* to be sent, 2) the percent where the *ACK* elicited an *ID* which was decoded by the reader, and 3) the percent where the decoded *ID* passed the CRC check.

The first thing to notice is that the tag can be read from up to six feet, which meets our threshold for usability; beyond 6 feet the tag was never read. Second, even at six feet *ACKs* were sent nearly 40% of the time, while received *IDs* fell to below 10%. This is because there is no error detection in the *RN16*, so an *ACK* is sent whenever a preamble is detected

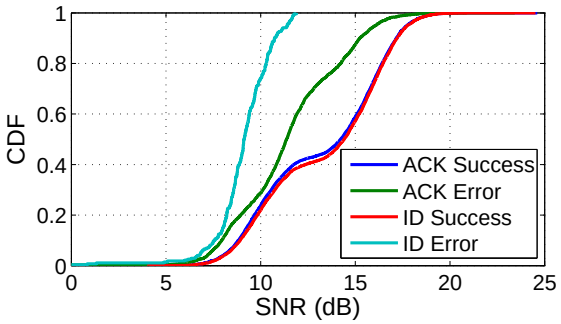

Figure 9: Signal to Noise Ratios

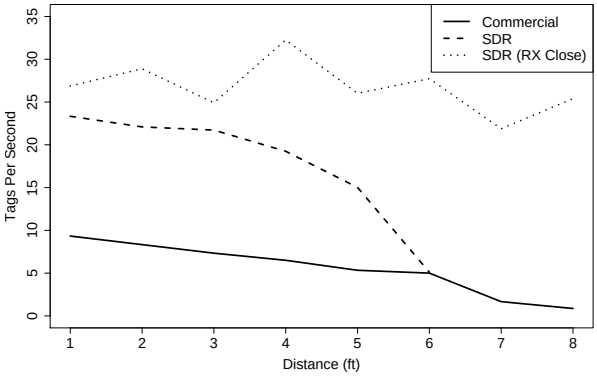

Figure 10: Reader Range

even if some of the bits in the *RN16* were decoded incorrectly. One interesting finding is that *IDs* rarely failed the CRC check; if the preamble was detected the *ID* was generally error-free.

Figure 9, which shows the SNR for *ACKs* and *IDs*, gives some insight into this behavior. All of the *IDs* that failed the CRC, and more than two thirds of failed *ACKs* (those which did not result in an *ID* being sent), were received with an SNR below 12 dB. In contrast, approximately 60% of successful *ACKs* and *IDs* were received with a better than 12 dB SNR. Hence, a successful *ACK* is a strong indicator that the subsequent *ID* will be decoded correctly. This is a benefit of the two stage handshake of the "Gen 2" protocol, with the first stage being a short 16 bit response that effectively filters out weak tags before the 128 bit *ID* is sent. It should also be noted that in [16] the authors show that a 13 dB SNR is necessary to achieve a BER of 10^{-3} in "Gen 2" systems. At least for *IDs*, our reader only saw bit errors below this threshold, and above this we see good success for both *ACKs* and *IDs*. This indicates that our demodulater implementation performs reasonably well.

7.3 Comparison to a Commercial Reader

As our reader generally requires a high SNR for reliable communication, we know that our receiver is not ideal. However, to determine how much range we could attain given a better implementation we performed a series of experiments.

First, using the same set up as the previous experiment we read the tag using the ThingMagic reader, with the tag and antenna remaining in the same position. To assure the same output power we used a power meter and decreased the output of the ThingMagic reader until it matched that of our reader; i.e. 75 mW.

Second, we used our reader and replaced the integrated bi-static antenna with two independent antennas. We positioned the receive antenna a few inches from the tag and placed the transmit antenna alongside the bi-static antenna used in the previous experiments.

When comparing RFID readers, gross tags read per second is a misleading metric as PHY layer behavior makes direct comparison difficult[4]. However, given that the ThingMagic reader only returns performance information in terms of gross tags per second, this is the metric we use for our comparison. While absolute comparisons cannot be made, the trends in performance are worth consideration.

Figure 10 shows the gross tags read per second when using the ThingMagic reader with the integrated anteanna, our reader using the same antenna, and our reader when the receive antenna is placed near the tag. Our reader succesfully reads the tag more times per second out to approximately 6 feet, beyond which it cannot read the tag at all. This higher read rate is a result of our implementation performing a single *Query* and powering down for 2 ms between *Queries*, where the ThingMagic reader performs a series of *Queries* and powers down for approximately 40 ms between rounds.

The ThingMagic reader can read the tag out to 8 feet while our reader has a range of only 6 feet. To determine if this was a hard limit, or indicated a limitation of the ThingMagic reader, we can compare to the results when our reader had its receive antenna close to the tag. In this case, our reader can successfully read the tag at 8 feet, with little degradation in performance compared to only 1 foot. Beyond 8 feet we were unable to read the tag even once, and upon examining our logs we found that no preambles had even been detected. This tells us that the tag does not receive enough power to operate at 9 feet, and regardless of the receiver performance the tag cannot be read. Thus, we see that the range of our reader

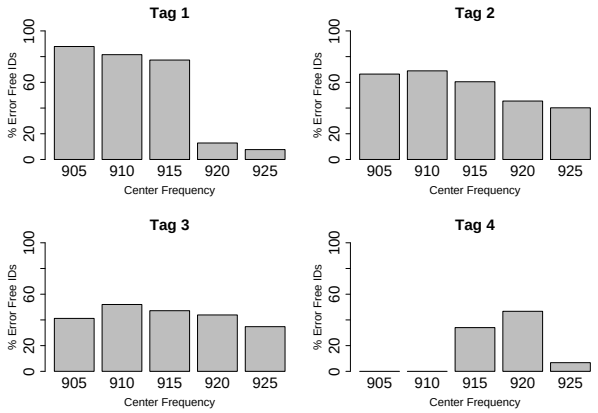

Figure 11: Error free IDs for tags at 3 ft

is approximately 75% of the optimal given its output power.

7.4 Reading Multiple Tags

When attempting to read more than one tag, we found it difficult to find an arrangement of tags where all tags could be read reliably. We experimented to determine why we had so much trouble reading a collection of tags.

Our reader implementation uses a statically configured frequency while commercial readers frequency hop across 50. To determine how this impacts our reader, we placed four tags at three feet and measured the percentage of *IDs* that passed the CRC check for each tag. For *IDs* that failed the checksum we look at the first nibble that was decoded, which is unique for each tag, and if it matched we consider it an error for that tag. While this introduces inaccuracy in our evaluation, we saw that errors were generally well distributed in the *ID*.

Figure 11 shows the success rate for each tag, and indicates that no single frequency reliably reads all tags. In this particular arrangement, tag four can be read reasonably well only at 920 MHz, but tag one performs poorly at this frequency. This is due to multipath effects which result in frequency selective fading. We explore this problem in a later section.

7.5 Discussion

Considering we use no specialized hardware in our implementation, we are pleased with the read range of our Gen 2 reader. In particular, we feel that a range of 6 feet is sufficient for RFID experimentation. Commercial readers generally use hardware such as directional couplers to limit the *CW* signal that bleeds into the receive chain, and sharp cut-off bandpass filters that suppress all but the tag response; both of these decrease noise in the system. Additionally, our reader has an output of only 75 mW, and the range may be increased by using a high-power amplifier in the transmit chain. However, the goal of our study is to provide a low-cost, low-complexity solution for researchers interested in RFID. As such, we leave evaluating the benefits of additional hardware to future work.

Based on our experiences, we found that we need to integrate frequency hopping capability into our implementation. Fortunately, the USRP should be capable of changing frequencies on the order of 100 μs, and the Gen 2 standard requires that the reader be powered down for at least 1 ms after changing channels. Consequently, integrating frequency hopping should be straightforward.

8. Applications

By providing flexibility at both the PHY and MAC layers and enabling detailed feedback from the receiver, our Gen 2 transceiver can be applied to a wide range of applications. First, it can be used as a tool to understand the underlying PHY and MAC layer behavior of RFID systems. This is difficult when using commercial platforms as they provide limited configurability and the underlying behavior must be largely inferred.

Second, it can be used with standard tags to explore ways to enhance the Gen 2 protocol. For instance, reader techniques to increase the performance of Gen 2 systems can be experimentally validated, and non-standard uses of the Gen 2 protocol can be developed.

We present two studies as examples. First, we use the PHY layer flexibility to determine the precise effects of multipath in UHF RFID systems. Second, we implement and prototype a technique to rapidly estimate the number of tags in a population

8.1 Understanding Multipath Effects

Fading due to multipath is a well known problem in UHF RFID systems. However, previous work has considered this problem from the viewpoint of the tag[15], or by examining overall reader performance as was done in our previous work[4]. While we could show that fading reduced reader performance, we could not determine exactly how it interacted with the reader to cause this effect. This was because commercial platforms must frequency hop due to FCC regulations and they do not give detailed feedback as to error rate or signal strength.

With our transceiver we can specify a single frequency for reading tags and we can gather detailed measurements to resolve questions we previously left unanswered. For our experiment, we read a single tag using five different center frequencies. We can then

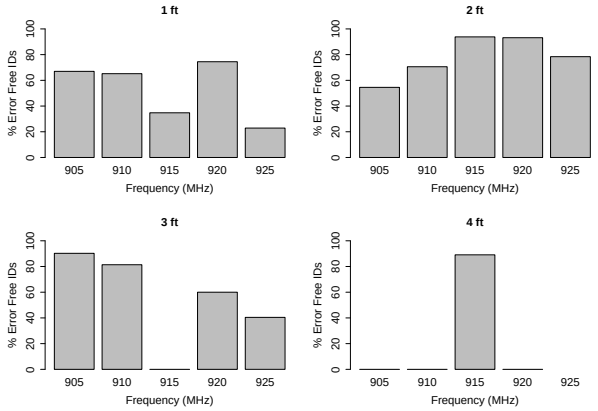

Figure 12: Fading at distance

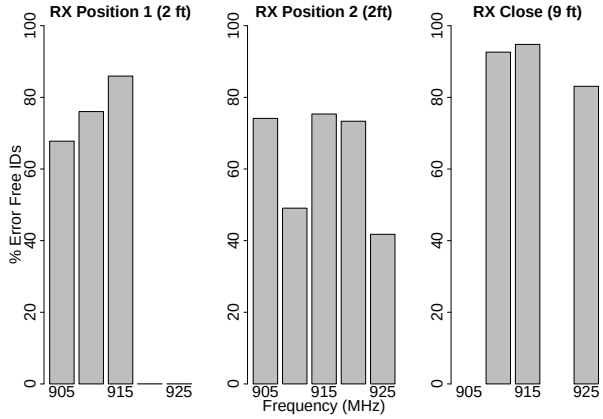

Figure 13: Fading when moving RX antenna

use the error rates of the *IDs* to infer the effects of fading.

Figure 12 shows the percentage of successfully received *IDs* for different frequencies as distance is increased from one to to fouer feet. We see that frequency has a significant effect on success rate even at short distances, and the effects of fading can be significant; at 3 feet the difference in success rate between 905 and 915 MHz is almost 90%. Additionally, we found that constructively interfering lobes were also evident, as is the case for 915 MHz at four feet.

Fading in RFID systems results in bit errors because it reduces the power of the back-scattered signal. However, fading can occur due to the transmit path, resulting in less signal being reflected from the tag, and also due to the receive path from the tag to the receive antenna. To consider the effect for each path independently we performed two sets of experiments.

First, we place the tag at two feet and use two independent antennas, one for transmit and one for receive. We then compare the success rate when the receive antenna is moved approximately 1 foot laterally from the original position while maintaining the same distance to the tag. Second, we place the tag just beyond 8 feet, and place the receive antenna inches away from the tag to determine the effect of fading on the forward path.

The results for both experiments are shown in Figure 13. Moving the receive antenna only a few inches results in drastically different performance for the different frequencies. In the first position nearly all *IDs* had bit errors at both 920 and 925 MHz, while in the second position 920 MHz sees a success rate of over 75%. This indicates that fading on the reverse channel significantly affects bit-error rate.

Looking at the case where the receive antenna is close to the tag, we see a second effect of fading. Here, three out of the five frequencies see very high success rates while the remaining two see no *IDs* at all. In this case, fading on the forward path results in the tag not receiving enough energy to power up.

These results show that fading degrades performance in two ways; bit-errors due to low signal strength and tags not harvesting enough energy to respond. As fading on the reverse channel plays a significant role in bit-errors, this effect could be mitigated by using multiple receive antennas where at least one would be likely to avoid fading for a given frequency.

8.2 Fast Tag Count Estimation

Techniques for rapidly estimating the number of tags in a population are desirable for a number of applications. First, if the size of the population may vary significantly an estimate of the number of tags would allow the reader to choose the correct number of slots before beginning to inventory the tags. Second, for many applications the reader may not need to know the *IDs* of all the tags but instead may be interested only in how many tags are present. For example, a supply chain applications may only need to verify that the correct number of items are on a pallet.

To count the number of tags using a commercial reader, all tag *IDs* must be read which incurs the overhead of transmitting the ACK and ID; this overhead increases significantly in the presence of errors. An alternative is to forgo sending *ACKs* and simply detect the number of *RN16s* transmitted during a round. By increasing the number of slots, the likelihood of a collision for a given number of tags can be reduced to acceptable levels and the number of tags can be accurately determined. Additionally, in the presence of errors such a technique could indicate

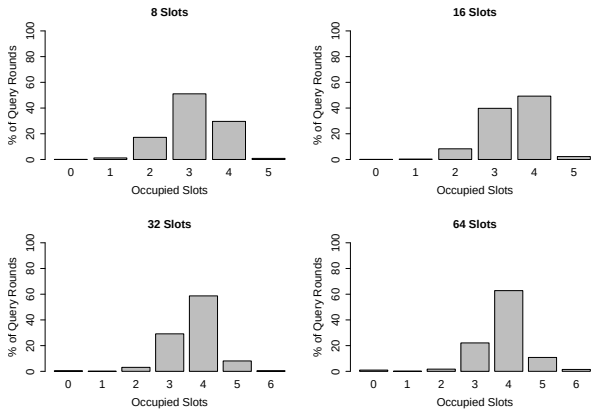

Figure 14: Number of Occupied Slots

how many tags were left unread.

The theoretical basis for a fast tag estimation technique was presented in[13] and was based on detecting slots with zero, one, or multiple tag replies. As an initial prototype of a such a system, we modified our reader so that it did not send *ACKs* but instead simply detected slots where an *RN16* was transmitted. To detect transmissions, we considered the SNR during the slot and used a cutoff of 6 dB, as this was just below the SNR for which we could reliably decode tag transmissions.

Figure 14 shows the results for our system when detecting tag responses for four tags using a varying number of slots. As the number of slots increased the number of *Query Rounds* that detected four tag responses increased because collisions are reduced. However, we found that our simple SNR based technique resulted in a significant number of false positives, indicated by rounds that detected more than 4 tags. Also, we saw many false negatives as indicated by the fact that with 64 slots and 4 tags, many rounds detected only three tags while the probability of a collision is low.

While the quantitative results for our tag estimation scheme are unsatisfying, they do show the value of a software defined MAC protocol for investigating proposed techniques. To conduct this study only a single line needed to be changed that ignored the result of the preamble detection.

9. Related Work

There is a variety of existing experimental work on RFID in the literature. [3, 14] provide a custom fabricated platform in which signal processing is implemented in an FPGA. The RFID Guardian[18] is another custom platform, but is not compatible with the Gen 2 standard and provides very limited flexibility at the physical and MAC layers. In contrast to these systems, our work is implemented using standard hardware and the complete Gen 2 protocol is implemented as a user process on the host.

The USRP and GNU Radio have also been used to study interactive communications protocols. [10, 8, 9, 7, 11] use the USRP to study the effects of cross layer enhancements on network performance for both hand-rolled physical layers [7, 11] and 802.15.4 [8, 9, 10]. [17, 20] implemented systems to monitor HF RFID building and subway access systems. Our previous work[4] implemented a Gen 2 monitoring system to measure commercial Gen 2 reader performance.

Note that in all of this work, software radio nodes communicated *only* as a transmitter or a receiver, but never both simultaneously. Katti et al [12] emulate an interactive protocol by slowing the MAC timing on a hand-rolled communication scheme several orders of magnitude to overcome the long latencies. In contrast to all of this related work, we present here the first implementation of a real-time, interactive transceiver for a commercial, standardized communications protocol using a commodity software radio.

10. Conclusion

We present a Gen 2 UHF RFID reader developed using the USRP and GNU Radio which can communicate with commodity RFID tags at up to 6 feet. As the complete Gen 2 protocol is implemented in software, our reader gives a high degree of flexibility with respect to both the MAC and PHY layers of the system. This flexibility enables low-level RFID research which is not possible using commercial platforms.

To operate with commodity tags, the system latency of our reader must be below 500 μs. We present techniques that reduce the latency of the USRP and GNU Radio so that we can reliably meet this timing constraint. We then evaluate our reader performance and compare this to a commercial reader with comparable power, and show that we achieve 75% of the read range while using no specialized hardware. We also discuss potential applications of our reader, and present two initial studies as examples. Our software defined Gen 2 reader provides a flexible and capable development platform that extends the application space of UHF RFID systems.

11. References

[1] Portuguese book retailer rolls out item-level rfid deployment. http://www.rfidnews.org.

[2] Why metro's item-level rfid deployment matters. http://www.rfidupdate.com.

[3] C. Angerer, M. Holzer, B. Knerr, and M. Rupp. A flexible dual frequency testbed for rfid. In *Proc. TridentCom*, 2008.

[4] M. Buettner and D. Wetherall. An empirical study of uhf rfid performance. In *Proc. Mobicom*, 2008.

[5] H. J. Chae, D. J. Yeager, J. R. Smith, and K. Fu. Maximalist cryptography and computation on the wisp uhf rfid tag. In *Proc. Conference on RFID Security*, 2007.

[6] EPCglobal. Epc radio-frequency identity protocols class-1 generation-2 uhf rfid protocol for communications at 860 mhz-960 mhz version 1.0.9. 2005.

[7] S. Gollakota and D. Katabi. Zigzag decoding: combating hidden terminals in wireless networks. *SIGCOMM Comput. Commun. Rev.*, 38(4):159–170, 2008.

[8] D. Halperin, J. Ammer, T. Anderson, and D. Wetherall. Interference cancellation: Better receivers for a new wireless mac. In *HotNets*, 2007.

[9] D. Halperin, T. Anderson, and D. Wetherall. Practical interference cancellation for wireless lans. In *Proc. of ACM MOBICOM*, 2008.

[10] K. Jamieson and H. Balakrishnan. PPR: Partial Packet Recovery for Wireless Networks. In *ACM SIGCOMM*, Kyoto, Japan, August 2007.

[11] S. Katti, S. Gollakota, and D. Katabi. Embracing wireless interference: analog network coding. In *Proc. SIGCOMM*, pages 397–408. ACM Press, 2007.

[12] S. Katti, D. Katabi, H. Balakrishnan, and M. Medard. Symbol-Level Network Coding for Wireless Mesh Networks. In *ACM SIGCOMM*, Seattle, WA, August 2008.

[13] M. Kodialam and T. Nandagopal. Fast and reliable estimation schemes in rfid systems. In *MobiCom*, pages 322–333, 2006.

[14] R. Langwieser, G. Lasser, C. Angerer, M. Rupp, and A. L. Scholtz. A modular uhf reader frontend for a flexible rfid testbed. In *2nd EURASIP RFID Workshop*, 2008.

[15] J. Mitsugi. Uhf band rfid readability and fading measurements in practical propagation environment. In *Auto-ID Labs White Paper Series Edition 1*, 2005.

[16] M. Mohaisen, H. Yoon, and K. Chang. Radio transmission performance of epcglobal gen-2 rfid system. In *Conference on Advanced Communication Technology*, 2008.

[17] H. Plotz. Rfid hacking. events.ccc.de/congress/2006/Fahrplan/attachments/1232-23C3-RFID_Hacking-3.pdf, 2005.

[18] M. R. Rieback, G. N. Gaydadjiev, B. Crispo, R. F. H. Hofman, and A. S. Tanenbaum. A platform for rfid security and privacy administration. In *Proceedings of ACM LISA*, 2006.

[19] L. G. Roberts. Aloha packet system with and without slots and capture. *SIGCOMM Comput. Commun. Rev.*, 5(2):28–42, 1975.

[20] R. Ryan, Z. Anderson, and A. Chiesa. Anatomy of a subway hack. tech.mit.edu/V128/N30/subway/Defcon_Presentation.pdf, 2008.

[21] A. P. Sample, D. J. Yeager, P. S. Powledge, and J. R. Smith. Design of an rfid-based battery-free programmable sensing platform. In *IEEE Transactions on Instrumentation and Measurement (accepted)*, 2008.

[22] T. Schmid, O. Sekkat, and M. B. Srivastava. An experimental study of network performance impact of increased latency in software defined radios. In *WinTECH*, pages 59–66, New York, NY, USA, 2007. ACM.