VISIBLE LIGHT NETWORKING WITH QR CODES
A PROJECT REPORT FOR CSE 561

RYDER ZIOLA AND WILLIAM WEBB

1. Motivation and Design Considerations

Telecommunications networks and the internet have enabled many different forms of communication - email, phone, instant messaging, etc. For most users, the fact that all of these are mediated by external networks is not important. However, some users, for instance, human rights activists operating in certain countries, may wish transfer digital data in a fashion that is anonymous, leaving no identifying traces on either participating machine nor any electronic record of the communication in the network. Physical storage media, such as CDs and portable hard drives are possible candidates if mail or an in-person handoff is allowed, but such media can also carry physical evidence, such as hair or fingerprints, and also serial numbers or metadata attached to the stored files. Any of these things could potentially compromise collaborators. Another possibility is to instead execute file transfers between portable machines in a way that uses no physical media and does not rely on any telecom network or exchange of addresses. Smart phones are natural candidates, being portable and increasingly ubiquitous, so for this project, we attempted to leverage common built-in functionality to create a secure channel with the desired properties. We note that many smart phones can communicate via IR ports, but there is always the possibility the that IR transmissions may be intercepted since they are one dimensional and monochromatic. A remaining feasible possibility is to use camera to screen communication, which is the approach we adopt.

2. Design Considerations and Challenges

A few other design requirements present themselves immediately. Namely,

- Synchronizing two devices in a precise fashion using a visual channel is difficult to do, so simplicity and functionality virtually require an asynchronous protocol.
- Most smartphone have their cameras and screens facing in opposite directions, only one device can communicate at a time, meaning that we wish to avoid using back channels if at all possible.
- Low screen and camera resolution amplifies the deleterious effects of device movement, making errors and lost packets a foregone conclusion, so error correction and detection are absolutely required.

- Built in autoexposure can change camera camera performance in response to changing light levels, complicating the task, especially since the underlying operating system may not permit manual control of autoexposure functions.
- To get a good view, devices must be close to each other, which requires the camera to be precisely focused on the screen. This is further complicated by the physical jitter experienced during handheld operation.

3. Background

3.1. **QR Codes.** Initial experiments with simple binary screen-to-camera transmission showed an extreme amount of noise and synchronization problems. We thus switched to QR codes, which are two dimensional analogues of the well-known bar code. QR codes have many advantages over simple binary transmission, including higher information density per displayed screen, low sensitivity to varying lighting conditions and angles, built-in error correction and detection, and the fact that there are many open source implementations of QR code readers and generators. Additionally, the average brightness of different QR codes is very similar, which minimizes changes in the autoexposure mechanism and allows greater consistency.

3.2. **Forward Error Correction.** The high level of noise on our channel and the absence of a back channel means that we needed a very robust way to ensure all packets arrived. The general idea of Forward Error Correction (especially the Digital Fountain [1], which inspired part of our protocol) corresponds perfectly to this scenario. Loosely speaking, forward error correction is the process of taking k packets and transforming them into n packets (where $n > k$), each one of which contain enough redundant information so that the original packets can be reconstructed efficiently from any k of the n redundancy-reinforced packets. This obviates the need for getting the packets in any particular order or for getting any particular packet.

4. Implementation

We built our system in a Nexus One smartphone running the Android operating system. The Nexus One is equipped with and 800X480 display and a 5 megapixel camera. As per the demands of Android, we wrote the system in Java, borrowing a QR encoding/decoding library [3] and porting the FEC library described in [2] from C to Java. The FEC library we used is limited to $n = 255$, so we must design our protocol to do FEC encoding on only smalls sets of packets, which we call chunks. At a high level, the algorithm takes a file to be transferred, breaks it into packets, then divides the packets into chunks, k packets per chunk, then applies FEC to each chunk, turning it into a set of n packets, each of which is then appended with a label that tells how many chunks there are total, which chunk the packets is from, and its index within the packet. Each packet is then encoded as a QR code. All QR codes are assembled into a list, then the program cycles through them, displaying each QR code for a set amount of time. The exact sequence of displayed QR codes is determined by the display protocol in use, something we discuss below. The program continues cycling until the operator halts it.

On the receiving side, when activated, the algorithm scans the data stream from the camera. When it identifies and successfully decodes the first code, it builds a nested hash table that includes a sub-table for each chunk, then checks the chunk that the packet came from, depositing the packet it the corresponding sub table. It thereafter continues this process on each packet successfully decoded. Each subtable will keep accepting new packets (i.e. those it has not already received) until it has received k unique packets from the same chunk, at which points it stops accepting and registers as complete. When each subtable is complete, the algorithm reconstructs each chunk using the FEC library, then assembles all the reconstructed chunks into a file. It then informs the user that the process is complete. Physical jitter means the camera loses focus from time to time, so the operator must occaisionally activate the autofocus mechanism by touching the screen. This requirement for user intervention necessitates immediate feedback on successful packet capture, so we programmed the phone to beep every time a code is read and successfully decoded.

5. Performance

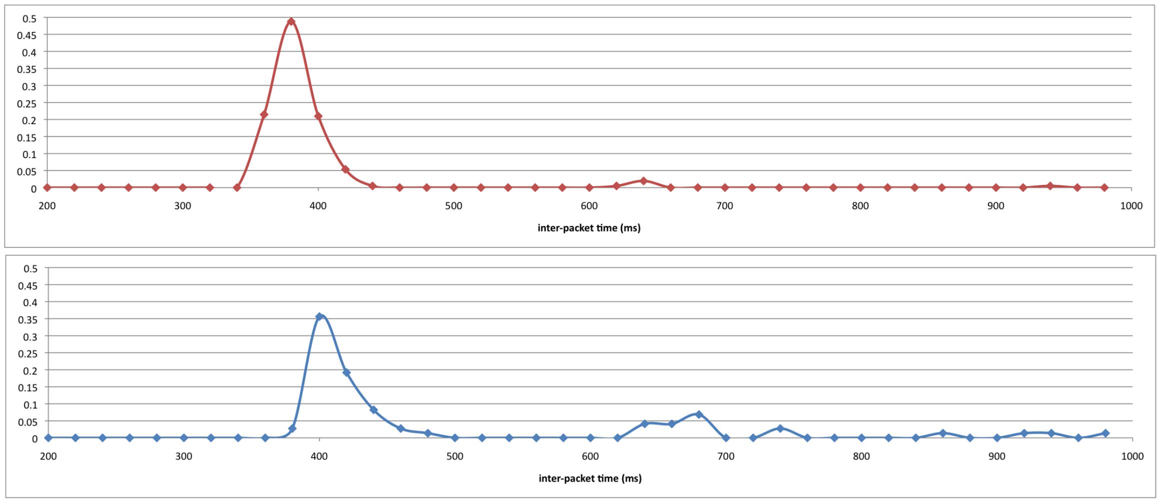

Figure 1. Burst characteristics of channel. The vertical axis is unitless probability, so the graph encodes the probability that the time between the successful decoding of two consecutive packets is a particular number of milliseconds. The top graph is for mounted (stabilized) units, the bottom is for handheld.

5.1. Parameters.

The main parameters that need to be fixed are packet size, chunk size k, encoded chunk size n, and frame cycling rate. For simplicity we consider these parameters largely orthogonal to each other in terms of their effect on performance, with the obvious exception of n and k. After roughly optimizing each parameter as we describe below, our system (while mounted) is able to transfer at roughly 2944 bits per second (368 bytes per second).

5.2. *n* **and** *k***.** $k = 1$ is just an inefficient repitition code, so we pick some $k > 1$. The ratio k/n is the main determinant of efficiency. A larger k/n implies more efficient use of the channel, while a smaller k/n results in more flexibility in which packets are received. Ideally, k would be chosen to contain the entire file, and n would be near infinite, requiring the receipt of any k packets to reconstruct the file. The practical limits of n=255 and the cost of generating each QR code means that this must be balanced against the conflicting demand of maximizing the probability that any cycle through the chunk will transmit enough packets to reconstruct the data represented by the chunk (which is made higher with lower k/n). Experiment confirmed our intuition that a k/n of about 1/3 is a good balance.

5.3. **Frame cycling rate.** The frame cycling rate is largely determined by the image capture speed of the camera and the QR-code detection time. Experimentally, we found the most common delta between received frames to be 380ms (See Figure 1). To avoid receiving duplicate frames, we fixed our transmission rate at 3fps.

5.4. **Packet size.** As each packet maps to a single QR code, the packet size determines the density of information displayed on the screen. The optimal value for this parameter is directly affected by the combined resolution of the screen, camera, and QR decoding algorithm, as well as movement introduced by the user. A larger packet size allows for the transmission of more information, but is less resilient to noise, particularly human movement. Figure 2 contrasts the throughput achieved at different packet sizes with both mounted and handheld phones. The roughly linear relationship between packet size and throughput over the considered range of values does not hold in the handheld case nearly as well as in the mounted case, presumably due to the additional noise introduced by physical jitter. Similarly, the channel becomes more bursty (Figure 1) in the handheld condition. Packets arrive less regularly, though still at multiples of the camera lag + processing time.

5.5. **Display Protocols.** We tried three different protocols governing the sequence of QR codes displayed. The naive protocol cycles through packets in order (by chunk). We also tried interleaving by chunk, in which we display from consecutive chunks, one packet at a time, until all packets have been shown, and then repeat. The other obvious protocol is one that displays packets uniformly at random. Surprisingly, the naive protocol behaved best, having the least divergence from the ideal case (in which all packets are received in one cycle with no redundant packets) and tying with per-chunk interleaving for the minimum total number of packets needed to reconstruct the file. The random protocol performed worst. All tests were performed with mounted units, and we expect that the naive protocol will increasingly poorly with higher levels of jitter, while the random protocol will remain largely unaffected.

6. Conclusion

We have shown that an anonymous, traceless visible light channel can be implemented in a normal smart phone. However, to be practical such a system would need many additional optimizations. Possible improvements include optimizing QR code decoding

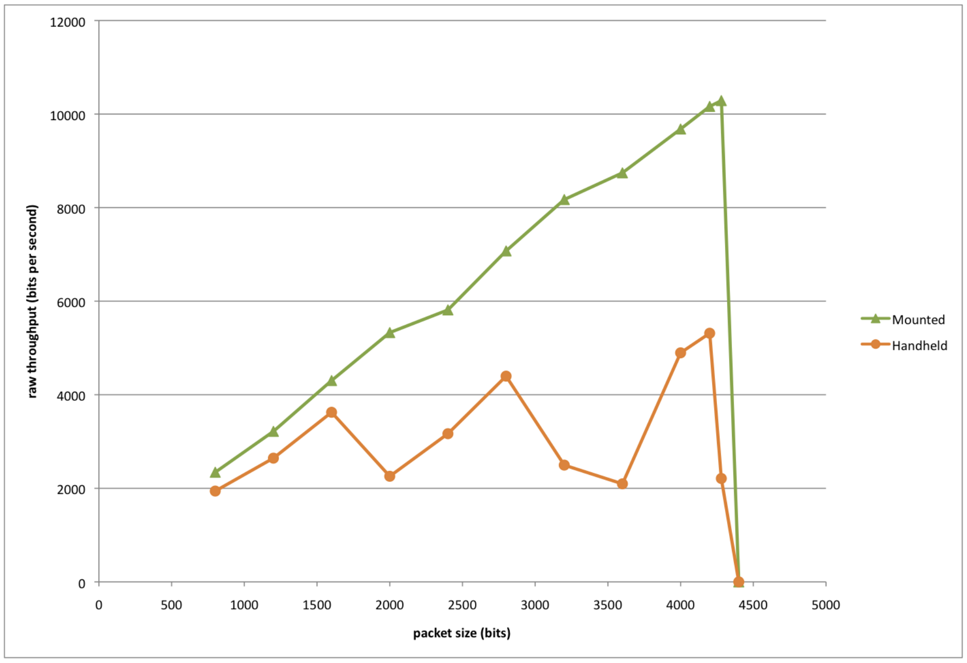

Figure 2. Throughput vs. packet size. Unsurprisingly, mounted units achieve better throughput and more predictable behavior. We conjecture that the rapid decrease in throughput near 4500 bits is primarily due to the the QR codes for packets of this size becoming so intricate as to go above the combined resolution of the screen/camera system.

and generation for added speed, display-as-generated functionality for the protocol, and provision for changing parameters (namely packet size, n, and k) to eliminate the need for file padding. Also, it would be interesting to develop an interface that can give feedback on the rate of incoming packets (for adjusting the position of the camera) and the overall completion simultaneously.

References

[1] J. Byers, M. Luby, M. Mitzenmacher, A Rege, "A Digital Fountain Approach to Reliable Distribution of Bulk Data". SIGCOMM 1998
[2] L. Rizzo, "Effective erasure codes for reliable computer communication protocols." Computer Communication Review 1997.
[3] ZXing (open source QR code library) - http://code.google.com/p/zxing.

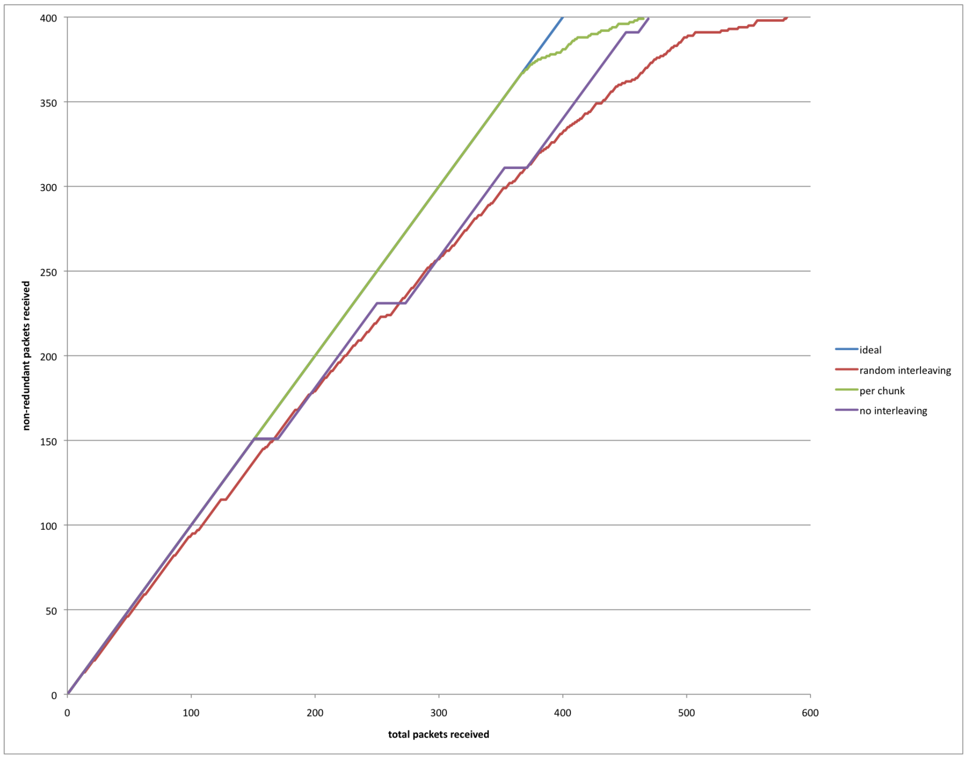

Figure 3. A comparison of the display protocols, showing that the naive protocol and interleaving by chunk tie for overall efficiency, with random performing the worst.