Enhancing Reliability to Boost the Throughput over Screen-Camera Links

Anran Wang[1] Shuai Ma[1] Chunming Hu[1] Jinpeng Huai[1] Chunyi Peng[2] Guobin Shen[3]
[1]SKLSDE Lab, Beihang University, China [2]The Ohio State University, USA [3]Microsoft Research, China
{wangar@act., mashuai@, hucm@, huaijp@}buaa.edu.cn chunyi@cse.ohio-state.edu jackysh@microsoft.com

ABSTRACT

With the rapid proliferation of camera-equipped smart devices (*e.g.,* smartphones, pads, tablets), visible light communication (VLC) over screen-camera links emerges as a novel form of near-field communication. Such communication via smart devices is highly competitive for its user-friendliness, security, and infrastructure-less (*i.e.,* no dependency on WiFi or cellular infrastructure). However, existing approaches mostly focus on improving the transmission speed and ignore the transmission reliability. Considering the interplay between the transmission speed and reliability towards effective end-to-end communication, in this paper, we aim to boost the throughput over screen-camera links by enhancing the transmission reliability. To this end, we propose RDCode, a robust dynamic barcode which enables a novel packet-frame-block structure. Based on the layered structure, we design different error correction schemes at three levels: intra-blocks, inter-blocks and inter-frames, in order to verify and recover the lost blocks and frames. Finally, we implement RDCode and experimentally show that RDCode reaches a high level of transmission reliability (*e.g.,* reducing the error rate to 10%) and yields at least two-fold improvement of transmission rate, compared with the existing state-of-the-art approach COBRA.

Categories and Subject Descriptors

C.2.1 [**Network Architecture and Design**]: Wireless communication

Keywords

Dynamic barcodes, transmission, reliability, smartphones

1. INTRODUCTION

Visible light communication (VLC) over screen-camera links has become an attractive short-range wireless communication solution due to the high availability of camera-equipped smartphones over the past years [12, 20, 25, 27, 33]. When data is encoded as a stream of images and displayed in screens, a smartphone can receive the data by capturing the images with its camera and decoding the image stream. Compared with the radio frequency (RF) techniques such as Bluetooth and Wi-Fi, VLC enables direct and secure communications, *e.g.,* by controlling the direction and distance of screen-cameras such that the visible range of a phone's screen content is limited to a few inches. Hence VLC simplifies the complicated authentication process for setting up link connections [12]. In addition, VLC is user-friendly and does not count on the Internet infrastructure. These make VLC over screen-camera links highly competitive for short-range wireless communications, *e.g.,* for applications such as file transfer between smartphones when either no wireless connections are available or security is much concerned. As pointed out by [33], VLC is well suited for one-time transfer as it incurs no charge and no overhead in link setup and management, compared with traditional RF techniques.

QR–Code [17] is commonly adopted for advertisement in the market for retail and tourism for easier information acquisition via smartphones with built-in barcode scanners [33]. By Mary Meeker's latest predictions on mobile Internet trends, four times more QR-Codes were scanned in March 2013 than a year before [1]. However, the capacity limitation makes such static 2D barcodes not appropriate for high-speed VLC.

While [25] and [3] achieve high-speed communications between large LCD monitors and high-speed digital cameras, they are not designed for smartphone platforms with limited capabilities such as small screens and low-quality cameras [12]. Recently, COBRA [12] and LightSync [33] are proposed to solve this problem.

However, data transmission over screen-camera links requires continuous image capturing, and a slight tremble may lead to parts of a frame unrecognizable or loss of a sequence of frames. Different smartphones also have different camera performance, which leads to different error rates by existing works. Further, the asymmetry nature of one-way transmission makes no feedbacks available. Reliable transimission is rather difficult to be obtained.

The absence of reliability also limits the throughput. For example, to keep the bit-error-rate below 10%, we have to restrict the frame of COBRA with less than 8000 symbols in our tests, much smaller than its maximum frame capacity, because distortions and trembles make the error rate increase seriously with the increase of the frame capacity. Another example is that even when we introduce an error correction code (Reed-Solomon [34]) to COBRA, it still has to transmit 2.03 times of the encoded data on average when

MobiCom'14, September 7 - 11, 2014, Maui, Hawaii, USA.

http://dx.doi.org/10.1145/2639108.2639135.

receiving a 128KB file intactly in our tests. This essentially reduces the transmission speed to 3.0KB per second.

Transmission reliability. Reliability is an important concern in short-range wireless communications, including VLC over screen-camera links, for ensuring dependable and effective data transmissions. Different applications have different requirements, and have different definitions of reliability [10]. Here we focus on one-time file transmissions that need fast and/or secure data delivery, and hence, we define *reliability* in terms of three aspects as follows:

(1) *Correctness* ensures that a receiver is able to check whether a received packet is exactly the one sent by the sender. Only correct packets are accepted.

(2) *Integrity* measures the ability that a receiver could obtain and assemble the data sent by the sender, which indicates the percentage of correctly delivered data.

(3) *Ordering* indicates the ability that a receiver is able to know the ordering of packets.

We argue that to achieve high-speed data transmissions, the *transmission rate* is only one side of the coin. The *transmission reliability*, the other side of the coin, is equally important since failed packets in general need to be retransmitted. This motivates us to develop new techniques towards reliable transmissions for applications such as file transfer.

Limitations and uncertainty. To achieve fast and reliable transimissions over screen-camera links, we face substantial challenges, primarily caused by:

Smartphone limitations. It is common that (a) senders with different types of screens may display colors differently and in different luminance levels, and (b) receivers with different lens and sensors may have different color processing mechanisms. This could lead to a common scenario such that the image captured by a receiver may be highly different from the original one from a sender. We summarize these problems as follows.

(1) *Lens distortions*[1]. Commonly found lens distortions of smartphone cameras are radial distortions that straight lines in a captured image become distorted.

(2) *Low border performance*. Due to the physical structure of lens, most digital cameras essentially have poor border performance, *e.g.,* low luminance, more noises and worse sharpness in the border area.

(3) *Color inaccuracies*. Different luminance brightness distribution within a screen and different view angles of a camera often cause colors displayed differently in different areas of shooting ranges.

(4) *Rolling shutter*[2]. Most smartphone cameras use CMOS sensors that adopt roll shutter for image acquisition, in which each frame is recorded not as a single snapshot, but rather by scanning across the frame either vertically or horizontally. As a result, different parts of a captured frame may belong to distinct frames when capturing image streams.

User behavior uncertainty. Different users may behave differently when shooting pictures. The user behavior uncertainty includes but not limited to:

(1) *Different positions*. It is common that users may capture only a part of the barcode on the screen because of the restriction of shooting positions.

(2) *Unexpected behaviors*, including mostly trembles of the hands, could interrupt the transmission for a short period.

The two challenges typically introduce two issues: (a) the *locality problem* that for an image with uniform features, the close places in its captured image have similar features, while places far from each other present different features, and (b) the *partial unavailability problem* that leads to either unrecognizable parts of a frame caused by *e.g.,* unsuitable shooting positions or temporally sequential frames loss caused by *e.g.,* trembles.

Other restrictions include the limited computing ability such that standard image processing techniques are not appropriate for continuous image decoding with smartphones. These make it even challenging to build high-speed reliable VLC over screen-camera links using smartphones, as only light-weight techniques could be employed to address the above mentioned problems.

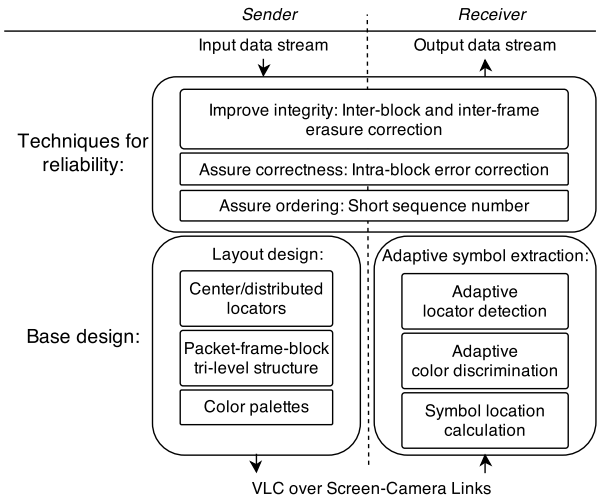


Figure 1: RDCode architecture

Contributions&Roadmap. We propose a robust dynamic barcode (RDCode) by encoding and decoding a stream of barcode images, which boosts the throughput over screen-camera links by improving the reliability. RDCode reaches a high level of transmission reliability and doubles the transmission rate compared with the existing state-of-the-art approach COBRA, by addressing the aforementioned limitations and uncertainty. To our knowledge, this is the first call for high-speed VLC over screen-camera links using smartphones by enhancing the transmission reliability.

(1) We design a novel packet-frame-block barcode layout (Section 2), such that a packet comprises a sequence of frames, each of which further consists of a set of independent blocks. This design not only enhances the transmission reliability by addressing the smartphone limitations and user behavior uncertainty, but also improves the transmission rate.

(2) We develop a tri-level data correction and ordering approach to enhancing reliability (Section 3), based on the observation that error distributions follow a certain spatial and temporal regularity. Error corrections at intra-blocks, inter-blocks and inter-frames are for the verification of blocks, recovery of lost blocks and recovery of lost frames, respectively. A short sequence number is also attached with each block for

[1]http://en.wikipedia.org/wiki/Lens_Distortion
[2]http://en.wikipedia.org/wiki/Rolling_shutter

Figure 2: A center locator pattern with 7×7 pixels

synchronization. These together enhance the transmission reliability of RDCode with a reasonable overhead.

(3) We implement RDCode on top of an Android platform for file transmissions (Section 4), and its architecture is shown in Fig. 1. We also conduct an extensive experiment study (Section 5), which shows that RDCode reaches a high level of transmission reliability (*e.g.,* reducing the error rate to 10%) and yields at least two-fold improvement of transmission rate, compared with the existing state-of-the-art approach COBRA.

2. BASIC DESIGN

In this section, we introduce the basic design of RDCode, including layout design and adaptive symbol extraction that lay down the base towards high-speed reliable VLC over screen-camera links via smartphones.

2.1 Design Principles

We have done extensive tests, and found that failures of recognizing barcodes are typically due to locality and partial unavailability problems. For example, when capturing a barcode with four corner locators such as COBRA [12], the failed detection of one or two corner locators leads to the complete failure of the entire barcode. Another example is the light reflection or bad border performance that makes a certain region of the captured barcode unreadable. This situation is even serious for dynamic barcodes since continuous locating and decoding frames needs more robustness. This implies that to obtain high transmission reliability, a barcode layout should address the locality and partial unavailability problems (Section 1).

In order to address them, the ideal method is to make each frame and each symbol independent and recoverable, *e.g.,* each frame or symbol has independent locators, sequence number and decoding methods. But it is impossible because massive overhead will be introduced, and a rather high computational complexity is required. A practical solution is to build intermediate layers above the symbols and frames.

This leads to a new design of dynamic barcodes: (a) a packet-frame-block tri-level structure to address the two aforementioned problems, in which each frame is associated with a single center locator and multiple distributed locators, and (b) a sequence of frames are aggregated into a packet as a logical transmission unit.

2.2 Layout Design

Symbols, blocks, frames, packets, center locators, distributed locators and color palettes form the basic structure of RDCode, and we first introduce them in detail.

Symbols. A symbol is a $p \times p$ square of pixels with the same color. Symbols are the basic units for data transmission.

Data symbols. A data symbol is a special class of symbols that is not in a black or white color.

There are in total $c \geq 4$ colors for data symbols, and hence a data symbol indeed encodes a $log_2 c$-bit data. Here black and white colors are reserved for the center and distributed locators to be introduced immediately.

Blocks. A block is a $h \times h$ square of symbols.

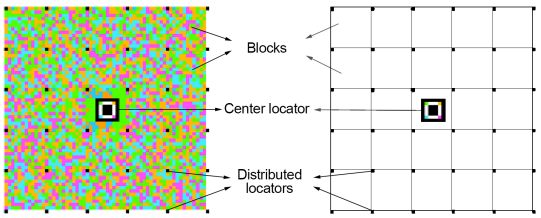


Figure 3: RDCode frame layout

Frames. A frame is a $m \times n$ rectangle of blocks, in which both m and n are odd.

Packets. A packet is a sequence of a fixed number of continuous frames.

Moreover, each block within a frame has a unique index, and each frame has a unique index (i.e., its order in the sequence) within a packet.

We next introduce the center and distributed locators on which symbol extraction heavily relies.

Center locators. There is a unique symbol pattern, referred to as a *center locator*, embedded in the center block of an RDCode frame. The detailed design of center locator is orthogonal to our work. As illustrated in Fig. 2, a center locator of RDCode is a square of symbols, in which (a) the center is a square of black symbols, (b) the borders are all black symbols, (c) there is a symbol inside each border corner with a distinct color of the c colors for data symbols; and finally (d) the rest are filled with white symbols.

Center locators allow fast detection of frames, and the four colorful symbols are used to estimate the shooting angle and image rotation.

Distributed locators. For the ease of explanation, we append a frame with $m \times n$ blocks to $(m+1) \times (n+1)$ blocks with dummy blocks. The top left most *black* symbol inside a block is referred to as a *distributed locator* of the original frame. Here (a) a frame with $m \times n$ blocks has $(m+1) \times (n+1)$ distributed locators, (b) there are four distributed locators around a block, which are used to locate the data symbols of the block, and (c) a frame is further attached with $m+n+1$ extra symbols after distributed locators are introduced, as illustrated by the example frame in Fig. 3.

Distributed locators allow simple and efficient locating and tracking of RDCode, as will be seen immediately.

Color palettes. As most data transmission systems do, a receiver needs to turn the raw channel signals into discrete bits. For barcode based systems, raw channel signals are the pixel colors. To design effective methods to distinguish different colors, we embed another pattern into each block at a fixed position, referred to as a *color palette* which is a square of c symbols, each in a distinct color of the c colors for data symbols.

As a block often has more than 100 symbols, the color palettes and distributed locators in a frame typically incur a reasonable overhead, the price that we have to pay for enhancing transmission reliability. For example, for a RDCode frame containing 11*7 blocks, each of which has 12*12 symbols, only 563 out of all 14,275 of its symbols are dedicated to its center locator, distributed locators and color palettes. In contrast, for a COBRA frame with $132 * 108 = 14,256$ symbols (roughly the same number of symbols as the previous RDCode frame), 1,404 out of its symbols are dedicated to those frame markers, around 2.5 times of RDCode.

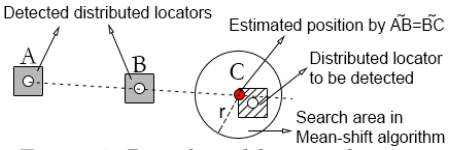


Figure 4: Distributed locator detection

2.3 Adaptive Symbol Extraction

We will next introduce the techniques to achieve adaptive symbol extraction when receiving frames. *Adaptive* means a receiver could adjust its parameters with the changing of its captured frames. This is essential to transmission reliability since the transmission performance should not be affected by different situations. Moreover, in order to address the partially unavailability problem, blocks and locators are treated independently in RDCode .

(1) Adaptive locator detection. We first introduce the adaptive method to detect locators within a frame such that even if some locators fail to be located, those blocks whose locators are correctly detected remain readable.

A receiver will first locate the center locator at the first captured frame, and we adopt the Mean-shift algorithm [6] to detect it in our implementation. Then the receiver analyzes the four colorful symbols in the center locator to detect the rough direction and distance of these symbols, and the perspective distortion of the frame. The receiver could also know the rough side length of symbols using the black symbols inside the center locator.

After that, the receiver recursively detects the distributed locators starting from the four distributed locators around the center block. (a) The positions of the four distributed locators around the center block can be easily computed based on the position of the center locator, the direction and the symbols' side length. The precise positions of the locators are obtained by the Mean-Shift algorithm or fast adjusting algorithm. (b) Then, as long as three out of the four distributed locators around the center block are successfully located, the receiver can recursively locate the rest distributed locators. We define locator A is *adjacent* to locator B if and only if they are around the same block. Then, for any three distinct locators A, B and C in a straight line in the sender frame where A is adjacent to B and B is adjacent to C, if the receiver knows the precise positions of any two of them in the captured frame, it can estimate the position of the third one based on their geometry relationship. For example, if the precise positions of locators A and B are known, the position of locator C can be estimated by assuming vectors $\tilde{\mathbf{AB}} = \tilde{\mathbf{BC}}$, as shown in Fig.4. The Mean-Shift algorithm or fast adjusting algorithm is then processed to get its precise position. This above process repeats until no more distributed locators can be successfully located.

Fast adjusting algorithm. We next introduce the main idea of the fast adjusting algorithm. In our work, Mean-Shift algorithm [5] is used to get the center position of the black area in a center or distributed locator, given an estimated initial search position. The essential step of Mean-Shift algorithm is to calculate the mean position of all the black pixels in a circular search area with radius r in each iteration, which needs to identify the colors of $O(r^2)$ pixels.

Observe that in a captured distributed locator, the black pixels are clustered as a rectangle, and when locating a distributed locator, its precise position is very close to the initial search position that is well estimated by the positions of other distributed locators or the corresponding locator's precise position in the last captured frame. Hence we can use linear search instead of quadratic search (with respect to the search radius r), when the initial search position is already in the distributed locator to be located. We describe this optimization in detail as follows.

Given the pixel at the initial search position, if it is black, it has a high probability that it is right in the distributed locator. As a result, we use *fast adjusting algorithm*: instead of searching the large circle area, it simply searches the left most, right most, top most and bottom most continuous black pixels from the initial search position. Let these four black pixels' positions be p_l, p_r, p_t and p_b, respectively. We denote the precise position of the distributed locator as $((p_l + p_r)/2, (p_t + p_b)/2)$ if the color of the pixel at this position is black. Otherwise, the Mean-shift algorithm is used. Gaussian filter [31] is adopted to reduce the interferences from the noises in the process. The above method works well in our implementation.

Note that (a) once a locator is located, it is independently tracked while dealing with the successive frames, and the failed detection of one distributed locator has no effects on other distributed locators. The locators are autonomous, but also co-related. Failed distributed locators still have a chance to be relocated later when two of its adjacent distributed locators are detected. Even if it fails again, the receiver could still use the estimated position instead. (b) Because locators split one frame to several small-sized blocks, this method also addresses radial distortions and increases the accuracy of decoding. (c) Except for detecting the center locator, our approach doesn't need to search the entire frame, and the center locator only needs to be located for the first captured frame. In most cases, the receiver only needs to visit several thousand pixels to locate one frame.

(2) Adaptive color discrimination. We then introduce the adaptive methods to discriminate different colors.

To decode data symbols, a receiver needs to correctly identify their colors. The color palette in a block is firstly identified as its position is fixed. Then the c colors in the color palette are detected using Gaussian filter [31]. Then, for each located data symbol, the Euclidean distances of its color and the c colors in the color palette are computed, and the color with the closest distance is chosen as the color of the data symbol.

The color palette must be identified accurately, or the entire block could be affected. Temporally changing environments and location errors may lead to the failure of the color palette's correct detection. In order to solve this, we adopt a smoothing method. Assume that we have the old color palette of a block represented as a vector $Color_{old}$ and the block at the same position of the next coming frame with a new color palette P. If the block is decoded successfully, we let $Color_{new} = (1 - \mu)Color_{old} + \mu P$ where μ is a parameter set to 0.3 in practice, and keep $Color_{new} = Color_{old}$, otherwise. To prevent incorrectly recognized color palette from affecting subsequent blocks, if the following error correction method cannot correct the block, the color palette of its subsequent block will not adopt the smoothing method.

Black and non-black colors also need to be discriminated when locating distributed locators. The process is similar. The initial color behavior may be known by analyzing the pixels in the central part of a frame. A histogram of their

brightness can be built, where the x-axis is the brightness of pixels, and the y-axis is the number of pixels that fall into a brightness bin. Typically, there exist a maximal bin, a minimal bin and another maximal bin in order. The first bin is for black pixels, the third one is for colorful pixels, and hence the mid brightness of the second bin can be used as a brightness threshold, referred to as m, to distinguish black from non-black colors.

Each distributed locator has a different threshold. When detecting a locator, all visited pixels recognized as black or non-black are used to revise the threshold for that locator, and they are split into two groups: (a) one with brightness less than the old brightness threshold m_{old}, and (b) another one with brightness equal or greater than m_{old}, where b and n are the average brightness of the two groups, respectively. Then we adjust the brightness threshold as follows: $m_{new} = (1-\mu)m_{old} + \mu(b+n)/2$. After several frames, each locator has its own adaptive brightness threshold.

(3) Symbol location calculation. After all distributed locators around a block are located, it is easier to locate the symbols of the block. Previous barcode designs such as COBRA and DataMatrix [16] use *timing symbols* at the borders of a frame as references to locate all other symbols, and this incurs high overhead. For RDCode, a block contains much less symbols than a frame, and hence the perspective distortion is not serious. As a result, the location accuracy is high, and there is no need to introduce timing symbols around each block. We simply compute the position P_{xy} of the symbol at coordinate (x, y) as follows.

$$P_t = \begin{pmatrix} \frac{h-x}{h} & \frac{x}{h} \end{pmatrix} \begin{pmatrix} P_1 \\ P_2 \end{pmatrix}$$
$$P_b = \begin{pmatrix} \frac{h-x}{h} & \frac{x}{h} \end{pmatrix} \begin{pmatrix} P_3 \\ P_4 \end{pmatrix}$$
$$P_{xy} = \begin{pmatrix} \frac{h-y}{h} & \frac{y}{h} \end{pmatrix} \begin{pmatrix} P_t \\ P_b \end{pmatrix},$$

where P_1, P_2, P_3 and P_4 denote the four positions of the left top most, right top most, left bottom most and right bottom most distributed locators around a block with $h \times h$ symbols, respectively.

Remarks. RDCode is in nature to address the locality and partial unavailability problems caused by smartphone limitations and user behavior uncertainty.

(1) Blocks and distributed locators in a frame are autonomous. For instance, distributed locators are tracked independently, each of which has its own black color threshold and own color palette. Blocks could also be decoded asynchronously by a receiver, so the rolling shutter limitation could be alleviated. Block autonomy also enables the flexibility to design effective and efficient reliability techniques (to be seen in Section 3).

(2) The adaptive techniques for symbol extraction improve the locating and decoding accuracy, by using successive frames in a dynamic barcode system.

(3) Compared with the design of four corner locators in a frame like COBRA and LightSync, our design of center locators has less failure possibility and less locating time. The incorporation of the center and distributed locators further improves the locating success rate, as distributed locators help the locating of their adjacent distributed locators. As a result, the block locating success rate is significantly improved as shown by our experiments.

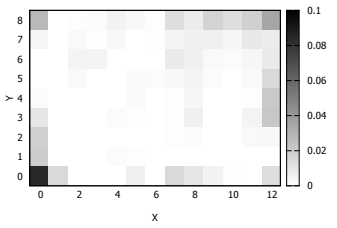


Figure 5: The average bit-error rate in the blocks displayed on an Asus Nexus 7 tablet and captured by smartphones LG Nexus 4 and Samsung Galaxy S3.

(4) It is worth mentioning that there exist more advanced algorithms to improve the success rate of locating frame corners, such as Hough transform [4] and local binarization [35]. However, in a barcode stream decoding scenario using smartphones, the computational overhead should be small in the first place. It is this reason why COBRA adopts a simple heuristic localization algorithm, but not those advanced algorithms. Our localization algorithm is effective and incurs only a small overhead, and it is specifically designed for RDCode, which cannot be directly adopted by COBRA.

3. TECHNIQUES FOR RELIABILITY

In this section, we first analyze the characteristics of error distributions. We then introduce the error correction technique for data correctness and data integrity, followed by the data ordering technique.

3.1 Analyses of Data Errors

We utilize a tablet as the sender and two smartphones with different cameras as receivers to experimentally analyze the error characteristics of the RDCode frames.

Uneven error distribution. We give an analysis of the RDCode frames with 9×13 blocks displayed on a tablet, captured by two different smartphones, and find that in barcode systems, errors are not randomly distributed. We count the average bit-error rates of blocks, *i.e.,* the ratio of error bits among all bits in a block. As shown by Fig. 5, errors are unevenly distributed in a frame and have certain regularity: (a) blocks close to the border area have higher error rates, due to the limitations of smartphones, and (b) other blocks may have a rare amount of errors on average.

Wide range of error rates. Basically the data errors of a dynamic barcode system can be classified into three types with different range of error rates:

(1) *Caused by noises.* Noises are inevitable and the border area has more noises due to the low border performance and the physical structure of image sensors. Typically the error rate is less than 10% due to the data errors caused by noises, depending on the environments and devices.

(2) *Caused by block decoding failures.* When (a) errors occur at some locators due to noises or at data symbols due to partial unavailability, or (b) dirty marks that cause their shelter symbols unreadable, the corresponding blocks often fail to be correctly decoded. This often results in an error rate around 10% to 100% for those affected blocks. Also, border areas has high possibility of block decoding failures since unsuitable shooting positions could easily make border areas out of the capture scope.

(3) *Caused by frame decoding failures.* When trembles happen, the affected frames are likely to be streaking due to motion blur. Almost an entire frame are unable to be decoded. Another reason to cause frame decoding failures is

the fast refresh rate with low brightness, so that receivers could capture overlapped frames. The colors of data symbols within adjacent frames may overlap with each other to make them unreadable. Although LightSync [33] focuses on the overlapped frames, it is for data symbols with black and white colors only, and cannot be used in our work for free.

Limitations of error correction techniques. In one-way communication, the standard approach to protecting data correctness is to use Forward Error Correction (FEC) to correct errors by adding redundant data [32]. When burst errors happen frequently, FEC often cuts no ice. Interleaved codes ameliorate FEC by shuffling the data symbols to several messages to create a uniform error distribution [32]. However, FEC and interleaved FEC cannot be easily incorporated into RDCode, as analyzed below.

(1) FEC is designed to correct random errors, but when the error distribution has regularity, it is not very efficient. A small amount of redundant codes cannot correct all errors, but a large amount of redundant codes lead to small throughput. Hence, messages at different areas should have different error correction ability.

(2) The error rate fluctuates widely and distributes unevenly. To design a good interleaved FEC is extremely difficult, since the intense burst errors may have impacts on other good areas with low-error rates. As data symbols in good areas may not be correctly decoded due to the interleaving, which in turn involves more data to be re-transmitted.

3.2 Tri-level Error Correction

Due to the above error characteristics and the restrictions of traditional error correction techinques, we develop an error correction method of RDCode for: (a) nonuniformity that different areas within a frame should have different error correction abilities, (b) autonomy that partial unavailability should not spread to other locality, and (c) effectiveness that only reasonable overhead should be attached.

The error correction method involves with three levels for the errors within blocks, frames and packets, respectively, among which the first is for data correctness and the last two are for data integrity.

Block level. *Intra-block error correction* is for blocks. Inside the blocks we focus on the errors caused by noises. It's more important to detect errors rather than correct slight errors in this level, since the transmission is not reliable if erroneous data are detected as correct. In our implementation, we use the Reed-Solomon (RS) code [34] for its strong error correction and detection ability. Based on a finite field with 256 elements (1 element represents 1 byte), an $RS(n,k)$ code has the ability to correct up to $\lfloor (n-k)/2 \rfloor$ error bytes and to detect any combinations of up to $n-k$ error bytes, and all the encoded bytes in a block are called an RS message. In order to improve the error detection ability, before encoding into an RS code, the last byte of the original message is set to the XOR of all other bytes in it. A receiver calculates their XORs to verify the correctness after the RS code has been successfully decoded.

Blocks in different areas are encoded with different parameters. For example, blocks in the border and corner areas have more correction ability than the other areas.

Frame level. *Inter-block erasure correction* is for frames. If a block fails to be located, or the intra-block error correction method can detect but cannot correct the errors, it is marked as a missing block. Since we know their indexes, we can apply any erasure code across the blocks. As the error distribution has regularity, RDCode uses a simple parity-check code to recover erased blocks because of its computational efficiency. Other erasure codes (*e.g.,* Reed-Solomon erasure code) could also be used for stronger correction guarantee, but with more computation.

We treat each block as a code element. Some blocks are parity-check blocks, which are also protected by intra-block error correction, but their contents are generated by other blocks. Suppose we have N blocks with indexes $1...N$ in a frame. The statistical missing probabilities of each block [3] are expressed by $P(1..N)$. We sort this array in decreasing order and generate another array $P_{sorted}(1..N)$ in which $P_{sorted}(i)$ is the index of the block with the ith largest missing probability. There are in total p parity-check blocks where p is a parameter set manually. The indexes of those blocks are expressed by $Check(1..p)$, and are determined by:

$$Check(i) = P_{sorted}(\lceil N/2^{p-i} \rceil)$$

Suppose that $Block(i)$ denotes the original message of the block with index i, and for each $i \in [1,p]$, $start = i, end = Check(i) - 1$, then

$$Block(Check(i)) = \bigoplus_{j=start}^{end} Block(P_{sorted}(j)),$$

where $\bigoplus$ is the direct sum operator. When decoding, the parity-check blocks are used to recover the missing blocks. Each parity-check block can recover one missing block that it covers. Since a block with a higher rank in P_{sorted} is involved to a parity-check block that covers fewer blocks, blocks with a high missing possibility are more possible to be recovered. A situation may happen that two blocks may have different original message lengthes due to intra-block error correction. When computing XOR, we let the result have the longest length of all the involved original messages, so that all the content of these messages can be covered.

The code design makes encoding and decoding procedures take linear time regardless of p. This code could recover at most p blocks by using p parity-check blocks.

Packet level. *Inter-frame erasure correction* is for packets. The blocks with the same block index i in a packet are denote as $subpacket(i)$, and $subpacket(i)(j)$ denotes the content of jth block in $subpacket(i)$.

Entire frame loss happens mostly sequentially since trembles last a period of time, typically 0.1 to 0.5 second. Our idea is that it is guaranteed to tolerate q consecutive frames loss within a packet containing n frames by using the last q frames as parity-check frames. We set $start = 1$, $end = \lceil (n-j)/q \rceil - 1$ $(j \in [0,q))$, and for each $i \in [1, block\_number\_per\_frame]$,

$$subpacket(i)(n-j) = \bigoplus_{k=start}^{end} subpacket(i)(n-j-kq)$$

While receiving a packet, after the first two corrections, each valid or missing block is marked in a table. When the next packet is arriving, the receiver will check if there exist some missing but recoverable blocks within each subpacket, and then recover them.

[3]In our implementation, corner blocks have higher missing probability than border blocks, and border blocks have higher missing probability than the remaining blocks.

3.3 Data Ordering

Due to the rolling shutter problem, blocks are asynchronously decoded, *i.e.,* blocks belonging to the same frame of a sender may be captured in different frames by a receiver. A straightforward method to keep the ordering of blocks is to add a full sequence number (including the frame index and packet sequence number) to each block, but this brings too much redundancy. Observe that at most two partial frames from a sender can overlap in a captured frame of a receiver, so the difference of any two blocks' sequence number within a captured frame is equal or less than 1.

The block containing the center locator is different from other blocks since it is always being first located and recognized in RDCode. We simply put the full sequence number of a frame in the center block. Note that this method remains protecting all the data in a block. The data within a block, including the sequence number, has few chances to be wrong if it is correctly decoded by the intra-block error correction method.

Inside each block, there exists a short relative sequence number containing the last 2 bits of the frame index. When dealing with a corrected block, RDCode compares its 2-bit frame index with the last 2 bits of the frame index in the center block, and checks the following.

(1) If they are equal, the full sequence number of this block is the full sequence number in the center block.

(2) If its 2-bit frame index is one smaller/bigger than the last 2 bits of the frame index in the center block (3 is one smaller than 0, and 0 is one bigger than 3), the full sequence number of this block is one smaller/bigger than the full sequence number in the center block.

(3) The case that the 2-bit frame index is two smaller/bigger than the last 2 bits of the frame index in the center block happens rarely. If it happens, the block is simply dropped.

With these, the accurate sequence number of each block is obtained with only 2-bit overhead in each non-center block that contains more than 200 bits data.

Remarks. (1) The novelty of our techniques to enhance the reliability lies in that instead of simply applying a single Reed-Solomon code like most barcode systems, we analyze the characteristics of data errors, utilize our tri-level barcode layout and design a tri-level error correction method to decode the decodable parts of data as much as possible to increase the data integrity. (a) The block level is designed for detecting errors and correcting slight errors caused by noises. (b) The frame level is designed for recovering undecodable blocks caused by partial unavailability problems such as dirty marks or unsuitable shooting positions. And (c) the packet level is designed for tolerating typical trembles by temporally recovering undecodable frames. We adopt Reed-Solomon codes at the block level because of its strong error detection and correction ability, and we also design own parity check codes, a light-weight solution, at the frame and packet levels for the high efficiency.

(2) Although the tri-level scheme used in RDCode works well in our implementation and evaluation, our scheme is not optimal given that many FEC codes exist. However, for communications over screen-camera links, we are the first to analyze some characteristics of the data errors, and explore a multi-level scheme which works much better than using Reed-Solomon codes alone in our tests.

(3) Based on the multi-layer design of RDCode, a more suitable error correction scheme for VLC over screen-camera links is worth further study. For example, we expect to associate each symbol with the confidence of color discrimination, then LDPC codes [7] are feasible for RDCode. In addition, our tri-level scheme belongs to block codes, which enables almost any transmission application including file transmission and data stream transmission. If only targeted on file transmissions with fixed lengths, suboptimal rateless (fountain) codes such as Raptor codes [29] are capable to replace the erasure correction codes in RDCode. It should be pointed out that the visual cryptograhy [22,23] proposed some interesting ideas, *e.g.,* by dividing a pixel to subpixels, to increase the transmission robustness and security, which provides more insights for a further study on coding.

4. IMPLEMENTATION

We have implemented RDCode on an Android 4.2 platform using Scala with the Android SDK and Scala compiler. Our experiments involve with three devices: one tablet (Asus Nexus 7) and two smartphones (LG Nexus 4 and Samsung Galaxy S3). During all the tests, our system runs on all these smart devices smoothly.

We have also developed an online encoder and an online decoder for file transmissions. Since Fountain codes [29] require high time complexity on solving linear systems as pointed out by [33], we make the file repeating in the data stream plus a simple coding scheme. Note that the effectiveness of RDCode design significantly reduces the chances of packet retransmissions. In order to reduce the time used for retransmission waiting, for even loops, we add the first half packets to the second half packets using XOR operations, and for odd loops, we add the second half packets to the first half packets. Using such a coding technique, it significantly reduces the waiting time up to 0.5 loop since if a packet in the second half is missing, receivers can recover it by decoding the corresponding packet in the first half with a constant computation complexity.

How to choose the c colors of symbols is important and complementary to the design of RDCode, which itself deserves a further study. In our tests, the error rate of using 2 colors are almost the same of using 4 colors, while using 8 colors introduces tripled error rate on average. Since 8 colors can present 3 bits data which is only 3/2 times of what 4 colors can present, we choose to use 4 colors. Since the raw data from the camera of Android phones are in the YUV format, we choose to use the most distinct four colors, orange, green, pink and blue in YUV color space for efficiency.

The sender of RDCode has an option to let a user initiate a file transmission, after which the file is encoded to RDCode packets. When forming frames, the sender changes the symbol size to fill in different number of symbols on the screen. Once a frame is produced, it is drawn on the screen.

The drawing procedure itself takes much more time than the encoding procedure since thousands of color squares need to be rendered. To address this problem for the real-time encoding, we use four additional threads, which is equal to the number of CPU cores of Nexus 4 and Galaxy S3. They're all for rendering the squares to a bitmap buffer. Each thread takes charge of rendering one-fourth distinct symbols of a frame. The four threads and the encoder thread form a pipeline that produces and draws frames effi-

ciently. When all packets of a file are transmitted, the sender repeats the above process and resends the packets until the entire file is correctly received.

When a frame contains about 10,000 symbols, our online encoder can produce ten RDCode frames per second, and display the RDCode frames at a 10FPS refresh rate on the screen. The rendered frames are also saved to the flash storage of smartphones to further speed up the process offline and reach a faster refresh rate.

We also use the multi-thread technique to reduce the processing time of a receiver. Since the blocks and distributed locators are autonomous, it is easy to make the decoding procedure in parallel. We further insert a flag to each of the blocks that indicates the frame parity. If a block to be decoded has a flag that is the same as the block in the same place of the last processed frame, and the block is successfully decoded, then there is no need to re-decode the block again since it is identical to the last decoded block. This optimization improves the efficiency of real-time decoding. When the bit rate reaches 300Kbps, it is still able to decode RDCode in real time using our tested smartphones. We have also implemented an offline decoder by further adding a layer to save or load the raw captured data to or from a file stored in an SD card.

The Android platform also provides a few parameters to customize a camera's performance. In our implementation, the camera's ISO is set to 400, its *RecordingHint* is set to true, and the others are using the default. The captured frame rate of our smartphones is kept above 24FPS with this setting in our tests.

5. EVALUATION

In this section, we report the experimental results of RDCode on the various factors of transmission reliability and throughput, compared with the existing state-of-the-art approach COBRA [12]. We test RDCode from four aspects. (a) Base design: to evaluate the RDCode layout and symbol extraction methods to see how it improves the decoding accuracy as well as the reliability under different environments. (b) Error correction: to know how the parameters of our error correction method would affect the result of data correctness and integrity; (c) Benchmark: to measure and compare the system performance of our work and previous works by transmitting files; (d) Decoding efficiency: to know whether our implementation could support real-time decoding on off-the-shelf smartphones.

Experimental settings. Our experimental receiver devices are LG Nexus 4 and Samsung Galaxy S3. They are the off-the-shelf normal Android smartphones equipped with standard screens or cameras, and they both have a camera of 8M pixel resolution and 1280*720@30FPS video capturing rate. Besides the two smartphones, we additionally use an Asus Nexus 7 tablet as the sender device.

Unless otherwise stated, our experiments are tested under an indoor environment with normal light condition (1 to 100 *lx*). (a) Except those tests for trembles and capture positions, the sender and receiver devices are kept still without any relative movement, and the capture scope of the receiver just fits the complete screen of the sender[4]. (b) Except those tests about display brightness and ambient luminance, the brightness of the screen is set to 50% of its maximum brightness. The workload is evaluated by a binary file containing integers produced by the Scala Class Random. In the tests, we use the offline encoder and offline decoder of RDCode, and the source code of COBRA comes from its authors, including an online encoder and an offline decoder written in Java and Matlab, respectively.

[4]It is tested that when the captured image just fits for the entire screen of the sender, their distance is about 21cm.

5.1 Base Design Comparison

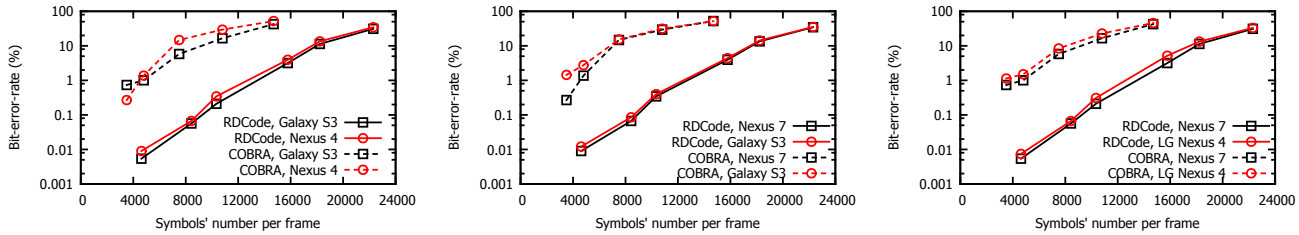
We first evaluate whether our base design could lay down the possibility to reliability. As a most recent approach to establishing screen-camera links between smartphones, the major contribution of COBRA [12] is its base design for fast and adaptive transmissions. The following LightSync [33] adopts COBRA's design so we only compare our work with COBRA. The comparison focuses on two aspects: code extraction accuracy and frame capacity. We disable the error correction method of RDCode in this test.

Accuracy of extracted symbols using different receivers and senders. We measure the bit-error-rate of RDCode and COBRA under different receivers and senders. Note that in COBRA [12] the metric *decoding rate* denotes the percentage of correctly decoded data in the total amount of data contained in a barcode. Since COBRA does not utilize error correction or detection codes, this metric is equal to one minus the *bit-error-rate* if COBRA takes *bits* as the transmission unit. Our experimental results coincide with the results in COBRA.

Since the frame capacity directly determines the performance of barcodes. For RDCode, we set blocks with $12 * 12$ symbols per block and change the number of blocks per frame. For COBRA, we vary the symbol size from 5 to 15, *i.e.,* $5 * 5$ to $15 * 15$ pixels. We only take the successfully located frames into account. We first use Asus Nexus 7 as the sender and the two smartphones as receivers, and report the result in Fig. 6(a). Because of the distortion caused location error and inaccuracy of color matching, the error rates of COBRA differ obviously and increase quickly with the increment of data capacity. Moreover, the original COBRA does not have error detection and correction ability so that COBRA cannot support much reliability. In contrast, the error rate of RDCode keeps low because of its adaptive methods. RDCode improves the maximum available data capacity to about 30000 bits per frame, more than two times of COBRA, with only less than 10% bit errors using two smartphones as receivers. Since the two results of RDCode are very similar, we will only show the average result of the two receivers in the following experiments for simplicity.

Since different senders have different color accuracies and luminance, which may affect the noises on the barcodes, we keep one smartphone as the receiver and use different senders to do the same experiments as above. Results reported in Fig. 6(b) and Fig. 6(c) show that both COBRA and RDCode are not sensitive to senders, while RDCode has a slightly better performance on the uniformity.

Accuracy of extracted symbols under different display brightness and ambient light intensity. Two key parameters of screen to camera communication are the brightness of the display and the ambient light intensity of the capturing environment. The brightness of the display affects the captured noises and color accuracy, and the ambient light intensity affects the visibility of the display. We



(a) Nexus 7 as the sender (b) Nexus 4 as the receiver (c) Galaxy S3 as the receiver

Figure 6: The bit-error-rates of RDCode and COBRA under different senders and receivers.

Settings	Illuminance (lx)	Brightness of display (%)	Located block ratio (%)	Bit error rate (%)
outdoor (sunlight)	6000-9000	50 100	0 39	- 15
outdoor (shadow)	100-1000	50 100	76 98	9 5
indoor (normal)	2-100	50 100	99 100	1 1
indoor (dark)	0	50 100	100 100	1 1

Table 1: The located blocks ratio and bit-error-rate under different display brightness and ambient illuminance.

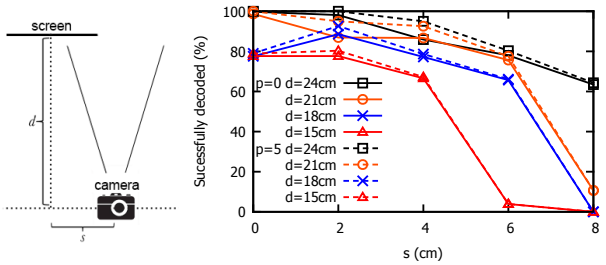


Figure 7: Left figure illustrated the settings of testing partial unavailability. Right figure shows the successfully decoded block ratio *w.r.t.* different distance and shifting.

test two different display brightness, 50% and 100%, under different environments including outdoor under sunlight, outdoor in a shadow, indoor with normal light condition, totally dark indoor, by recording the ambient illuminance using the front ambient light sensor equipped in the sender. In this test we use LG Nexus 4 as the sender and Galaxy S3 as the receiver, respectively. We fix a RDCode frame to contain 11*9 blocks each of which contains 12*12 symbols in the tests.

The result is shown in Table 1 by measuring two metrics: (a) the *located blocks ratio*, which is defined as the ratio of the successfully located blocks by the receiver in all the displayed blocks by the sender, and (b) the *bit-error-rate*, the ratio of bit errors in successfully located blocks. Results show that RDCode does not work well under intense sunlight. This is a common problem for VLC over screen-camera links due to the limitations of current LCDs, as the luminance of LCDs is much smaller than the strong sunlight. However, with the emerging of new technologies such as e-ink display[5], this problem could be alleviated.

5.2 Data Correction

In this experiment, we fix in a frame the block number to 13*9 and each block contains 12*12 symbols, so that a frame contains 156*108 symbols (exclude the leftmost and bottommost distributed locators).

Comparison with standalone error correction codes. We first compare the decoding performance of our tri-level error correction scheme with a standalone error correction code applied to our design. We have tested two candidates: (1) RS code over GF(256) and each block contains one message. (2) Interleaved RS code over GF(256), those bytes of the same index within each block form a message. We change the code rate by varying the parameter $n-k$. For our tri-level error correction scheme, we change the code rate by varying the parameters $n-k$ and p . As Fig. 10 shows, our method is more efficient than the two standalone error correction codes on the screen-camera transmission channel, by obtaining more corrected blocks with the same code rate. Although there exists a rather small possibility that a wrong message is determined right by RS code in theory, during all of our experiments, no such cases happen.

Data integrity: successfully decoded block ratio within a frame. By combining intra-block error correction with inter-block erasure correction, we may change their parameters p and $n-k$, capture and decode one static barcode frame, and evaluate the average *successfully decoded block ratio* (Fig. 8) that denotes the percentage of successfully decoded blocks in all sent blocks. Note that border blocks have worse error rates so we double the $n-k$ parameter of those blocks. From the figures we know that the setting of $n-k=6$ and $p=4$ is enough for the balance of per-frame data integrity and data capacity.

When partial unavailability problems happen, some blocks may be unavailable. We do a simple test that measures the data integrity under this case. Fig. 7 (left) illustrates that the receiver is placed at different distance d and different shifting s. It is clear that the more s or less d is, the smaller part of a frame can be captured. The right chart of Fig. 7 shows RDCode has the ability to decode available blocks as much as possible. Oppositely, as we tested, previous works such as COBRA have no such ability and a whole frame is undecodable as soon as a corner tracker is lost tracking, which reduces the data integrity seriously.

Data integrity: successfully decoded block ratio among multiple frames. When displaying barcode sequence, the aim of inter-frame erasure correction is to tolerate temporal partial unavailability problems mainly caused by trembles. Figure 11 plots the decoding ratio and transmission speed with regards to the changing acceleration caused by trembles. In this experiment, we put the sender at a fixed position, and held the receiver with a hand by a real

[5]http://en.wikipedia.org/wiki/E_Ink

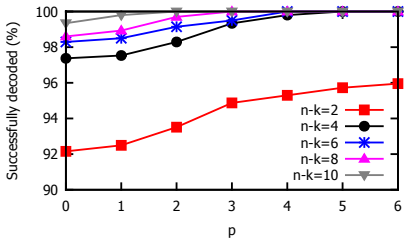


Figure 8: The successfully decoded blocks ratio within a single frame *w.r.t.* error correction parameters.

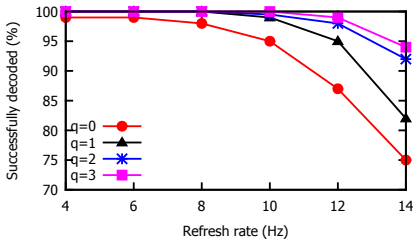


Figure 9: The successfully decoded blocks ratio *w.r.t.* sender's refresh rate and inter-frame erasure correction parameters.

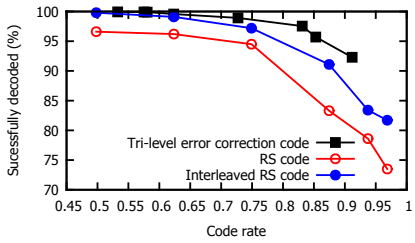


Figure 10: The correcting ability of tri-level error correction code, Reed-Solomon (RS) code and interleaved RS code.

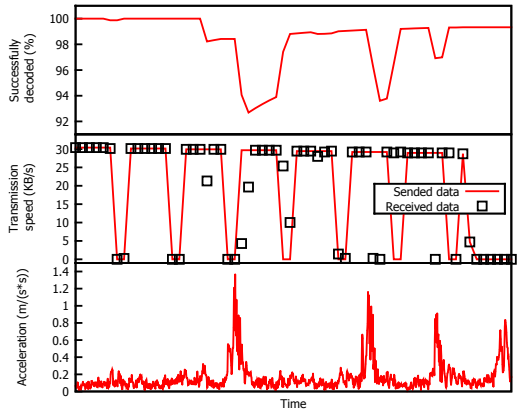


Figure 11: The relationship among successfully decoded blocks ratio, instant transmission speed and changing acceleration caused by trembles.

user to capture RDCode. We fix the sender's refresh rate to 8FPS and $q = 2$. We then capture an RDCode containing 8 packets by hand and record the changing acceleration of the receiver, and in the meanwhile we record the instant data sent rate, received rate and successfully decoded block ratio. As shown in Fig. 11, the inter-frame erasure correction can recover most blocks in lost frames caused by unexpected trembles. As a result, more than 99% data is obtained. Although not all data are received, inter-frame erasure correction plays an important role by recovering most lost blocks.

When no trembles happen, since inter-frame erasure correction is based on block operations, it could also significantly improve the packet integrity. For example, in order to solve *aliasing*, the refresh rate of senders should be at most half of the capture rate of receivers, which is at least 24FPS. As Fig. 9 shows, when $q = 0$, the successfully decoded block ratio drops suddenly when the refresh rate is beyond 10FPS. But when $q = 2$, inter-frame error correction increases the maximum tolerable refresh rate to 12FPS by recovering overlapped frames.

5.3 Benchmarks

We now set $n - k = 6$, $p = 4$ and $q = 2$ as the error correction parameters, and focus on the whole system to find out the best trade-off between transmission speed and reliability. We do experiments under three common block numbers (11*7, 13*9, 15*9), and each block contains 12*12 symbols. We also set different refresh rates for senders.

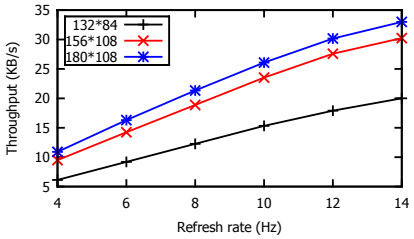
Throughput and packet error rates. *Throughput* means successfully decoded bytes per unit time excluding the redundant error correction codes. Packet is the transmission unit of the system. If a packet is not completely received, we say that it is a packet error since the packet cannot be recovered unless re-transmission. The packet error rate is the metric for packet-level data integrity. We measure the throughput and packet error rate by processing 100 packets under different settings. The result is reported in Fig. 12, which shows that the packet error rate is tolerable, about 5%, under 10FPS refresh rate and 156*108 symbols per frame. In this case, the maximum throughput is about 21.8KB/s.

Comparison on goodput. The final measurement of transmission performance is to transmit a real file and test the goodput. *Goodput* measures the application layer performance. In RDCode, goodput means the transmitted file size divided by total transmission time. We use a 128,000 byte file containing pseudo-random data. Figure 13 shows the maximum, minimum and average goodput. The entire transmission process finished within 1.5 loops in most cases.

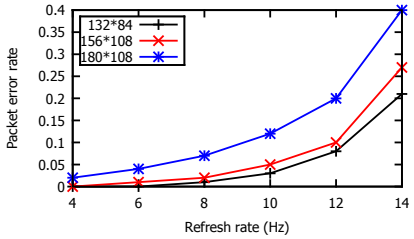
We also want to show how COBRA performs at the application layer. Since COBRA does not focus on reliability, we first encode the original data by Reed-Solomon code over $GF(256)$, and then add a header to indicate the full sequence number to each frame. The online encoder of COBRA cannot be used under refresh rates larger than 5FPS due to its low drawing performance. Hence, we use the generated image files to make a video to play offline.

The experiments of COBRA cannot be successfully done when the symbol number per frame is more than about 110*70 since location errors happen frequently. Also, when the refresh rate reaches 10, *i.e.,* the same as RDCode, a number of frames are lost due to the rolling shutter problem. As a result, we set 100*60 symbols per frame, and test two refresh rate: 10FPS and 7FPS.

We test the average loops and goodput by transmitting a single 128KB file. Meanwhile, we also record the bit-error-rate and successfully decoded block ratio of both COBRA and RDCode. We use *bandwidth* to measure the raw bit rate sent by the receiver. Table 2 shows that RDCode provides a big advantage for both transmission speed and reliability. First, keeping error rates at a low level, the base design of RDCode supports a higher bandwidth by increasing the data capacity within each frame. Second, our goodput is even much better than the throughput of COBRA, because our error correction methods increase the data integrity and reduce the retransmission loops. The big difference between the throughput and goodput of COBRA indicates COBRA is not designed for reliable transmissions and frequent retransmissions are typically needed.



(a) Throughput



(b) Packet loss rate

Figure 12: The relationship between the throughput and packet error rate under different refresh rates and symbol counts.

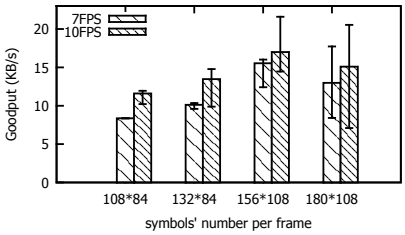


Figure 13: The maximum, minimum and average transmission speed when receiving a 128KB file under different barcode sizes and refresh rates.

Setting		Base design		After error correction		File transmission	
		Bandwidth	Bit-error-rate	Maximum Throughput	Successfully decoded blocks	Goodput	Average loops
RDCode	156*108@10FPS, $p = 4, n - k = 6, q = 2$	$324kbps$	1.1%	21.8KB/s	99.2%	17.0KB/s	1.28
COBRA	100*60@10FPS, $RS(255, 191)$	$120kbps$	4.5%	8.8KB/s	89.4%	2.1KB/s	4.13
	100*60@7FPS, $RS(255, 191)$	$84kbps$	2.1%	6.1KB/s	94.6%	3.0KB/s	2.03
	100*60@7FPS, $RS(255, 127)$			4.1KB/s	98.8%	2.6KB/s	1.60

Table 2: Comparison of the goodput of file transmit application between RDCode and COBRA.

We did not compare RDCode with LightSync. One reason is that LightSync is based on the design of COBRA, and limits its symbol to only represent one bit as LightSync only uses two colors. Hence its available data capacity per frame is the half of COBRA. Its inter-frame erasure code also halves the throughput. Another reason is that we could not get its source code and its binary executable file cannot run properly under Android 4.x. However, LightSync [33] stated its maximum throughput is about 11KB/s, much less than the one of RDCode.

5.4 Decoding Efficiency

We finally show that the decode processing time is the requirement of online real-time decoding. In order to decode the frames in real time with smartphones, in our implementation, the processing time of decoding each frame at smartphone must be smaller than the frame interval of the sender. If the sender's refresh rate is 10FPS, then the processing time must be less than 100ms. We keep $12 * 12$ symbols per block, vary the frame size, and use LG Nexus 4 to capture and decode about 5 seconds, and then we test the average processing time per frame. Error correction parameters are set to $p = 4, n - k = 6, q = 2$. The center locator is only needed to be located at the first captured frame, and the decoder tracks the distributed locators during the following frames. As shown in Table 3, the experiments demonstrate that locating does not take much time. The main drawback is that the pixel sampling depends on the operating system. Overall, the decoder can process the received data in real time under the limited computing ability of smartphones.

Symbols count	84*60		156*108	
Thread count	1	4	1	4
Center locator	63ms		68ms	
Distributed locator	13ms		14ms	
Data extraction	44ms	13ms	150ms	43ms
Error correction	7ms	4ms	27ms	9ms
Total	64ms	30ms	191ms	66ms

Table 3: The average processing time per frame by decoder.

6. RELATED WORK

Visible Light Communication. Visible Light Communication(VLC) [13] is a kind of data communications that uses the visible light as a medium. Many works [2, 8] are dedicated to gain a fast transmission channel using visible light. However, extra equipments are needed, which are not available by ordinary people.

Barcodes. Various barcodes are usable nowadays including one dimensional barcodes [15] and two dimensional barcodes [16, 17]. Some new barcode designs such as High Capacity Color Barcode (HCCB) [24] use multiple colors to multiply the data capacity. With the help of various decoding techniques to improve the decoding quality [19], popular barcodes have a high first-read rate (FRR) [18] during the scanning process.

Screen-camera links. Due to the high availability of smartphones, VLC over screen-camera links are widely studied recently. Approaches like [25] modulate data using OFDM technology and achieve a high throughput using high definition LCDs and high quality cameras. 4D Barcode [21] is the first attempt to decode sequential barcode streams using phones to the best of our knowledge. It uses sequential Data Matrix barcodes [16] to multiply the throughput, and uses different color channels with redundant data to reduce errors. Because its design is based on traditional barcodes, decoding takes lots of time and the throughput is low. COBRA [12] contributes a novel barcode design for real-time phone to phone transmission, and optimizations for alleviating decoding errors caused by motion blur. COBRA makes the link more adaptive to smartphones, but the locality and partial availability problems discussed in Section 1 are not in their consideration.

These works use at most half of the capture frame rate as the sender's refresh rate since the continuous frames may overlap. LightSync [33] contributes to solve the frame synchronization problem based on COBRA. However, it limits

the colors to black and white which reduces the throughput. Further, although both LightSync and RDCode address the rolling shutter problem, they are orthogonal to each other.

Data transmissions. In duplex unicast transmission, many network protocols such as TCP [26] support reliable transmissions. However, it is difficult to apply their techniques for one-way communications.

[11] discussed some schemes to obtain a reliable multicast channel, and their methods use feedbacks to ensure the data integrity [30]. If no feedbacks are available, FEC codes [32], such as Reed-Solomon [34] and LDPC [7], are used to keep the correctness [28], and to control the errors on one-way channels, which has been widely studied to obtain better correction ability and more efficient encoding and decoding algorithms. Most standard barcodes such as QR-Code use Reed-Solomon code as their FEC codes. If error positions are known, *erasure codes* can be used to fix errors with less redundancy [9]. Fountain codes such as Raptor codes [29] are a class of erasure codes that do not have a fixed code rate. Different codes can be combined to improve the correction ability for various channels [14].

7. CONCLUSION

We have proposed RDCode, a robust dynamic barcode, which boosts the throughput over screen-camera links via smartphones by enhancing transmission reliability. (a) We design a packet-frame-block barcode layout and symbol extraction techniques to address the locality and partial unavailability problems, caused by smartphone limitations and user behavior uncertainty. (b) We develop a tri-level error correction approach to enhancing reliability. (c) We implement RDCode on top of an Android platform for file transmissions, and our experimental results show that RDCode both improves the transmission reliability and speed, compared with the existing state-of-the-art approach COBRA.

Several topics are targeted for future work. First, our design brings the possibility to customize colors, but how to choose colors effectively and efficiently worths a further study. Second, we are to investigate the impacts of different FEC codes on RDCode. In addition, we are exploring other possibilities to boost reliability over screen-camera links. Finally, customized frame shapes may be incorporated to improve the aesthetical effect.

Acknowledgments. This work is supported in part by 973 program (No. 2014CB340304), NSFC (No. 61322207) and SKLSDE grant (No. SKLSDE-2014ZX-20).

8. REFERENCES

[1] Mary Meeker's internet trends report, 2013.
[2] M. Z. Afgani, H. Haas, H. Elgala, and D. Knipp. Visible light communication using OFDM. In *TRIDENTCOM*, 2006.
[3] A. Ashok, M. Gruteser, N. B. Mandayam, J. Silva, M. Varga, and K. J. Dana. Challenge: mobile optical networks through visual mimo. In *MOBICOM*, 2010.
[4] D. H. Ballard. Generalizing the hough transform to detect arbitrary shapes. *Pattern recognition*, 13(2):111–122, 1981.
[5] Y. Cheng. Mean shift, mode seeking, and clustering. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 17(8):790–799, 1995.
[6] D. Comaniciu, V. Ramesh, and P. Meer. Real-time tracking of non-rigid objects using mean shift. In *CVPR*, 2000.
[7] M. C. Davey and D. J. MacKay. Low density parity check codes over GF (q). In *Information Theory Workshop*, 1998.
[8] H. Elgala, R. Mesleh, and H. Haas. Indoor broadcasting via white LEDs and OFDM. *TCE*, 55(3):1127–1134, 2009.
[9] G. Forney Jr. On decoding bch codes. *IEEE Transactions on Information Theory*, 11(4):549–557, 1965.
[10] I. Guvenc, S. Gezici, Z. Sahinoglu, and U. C. Kozat. *Reliable Communications for Short-Range Wireless Systems*. Cambridge University Press, 2011.
[11] M. Handley, S. Floyd, B. Whetten, R. Kermode, L. Vicisano, and M. Luby. rfc 2887. *IETF Request For Comments*, 2000.
[12] T. Hao, R. Zhou, and G. Xing. Cobra: color barcode streaming for smartphone systems. In *MobiSys*, 2012.
[13] S. Haruyama. Visible light communication. *Journal of JSME*, 107(1030):710–711, 2004.
[14] H. Imai and S. Hirakawa. A new multilevel coding method using error-correcting codes. *TIT*, 23(3):371–377, 1977.
[15] I15420:2009. Information Technology - Automatic identification and data capture techniques - EAN/UPC bar code symbology specification.
[16] I16022:2006. Automatic identification and data capture techniques - Data Matrix bar code symbology specification.
[17] I18004:2000. Automatic identification and data capture techniques - Bar code symbology - QR Code.
[18] H. Kato and K. T. Tan. First read rate analysis of 2d-barcodes for camera phone applications as a ubiquitous computing tool. In *TENCON*, 2007.
[19] H. Kato, K. T. Tan, and D. Chai. Development of a novel finder pattern for effective color 2d-barcode detection. In *ISPA*, 2008.
[20] R. Kraemer and M. Katz. *Short-Range Wireless Communications: Emerging Technologies and Applications*. Wiley, 2009.
[21] T. Langlotz and O. Bimber. Unsynchronized 4d barcodes. In *Advances in Visual Computing*, pages 363–374. Springer, 2007.
[22] M. Naor and A. Shamir. Visual cryptography. In *Advances in Cryptology–EUROCRYPT'94*, pages 1–12. Springer, 1995.
[23] M. Naor and A. Shamir. Visual cryptography ii: Improving the contrast via the cover base. In *Security Protocols*, pages 197–202. Springer, 1997.
[24] D. Parikh and G. Jancke. Localization and segmentation of a 2d high capacity color barcode. In *WACV*, 2008.
[25] S. D. Perli, N. Ahmed, and D. Katabi. Pixnet: interference-free wireless links using lcd-camera pairs. In *MobiCom*, 2010.
[26] J. Postel. Transmission control protocol. 1981.
[27] Q. Pu and W. Hu. Smooth transmission over unsychronized vlc links. In *CoNEXT student workshop*, 2012.
[28] L. Rizzo. Effective erasure codes for reliable computer communication protocols. *Computer Communication Review (ACM SIGCOMM)*, 27(2):24–36, 1997.
[29] A. Shokrollahi. Raptor codes. *IEEE Transactions on Information Theory*, 52(6):2551–2567, 2006.
[30] P. R. T. P. Specification. Rfc 3208, 2001.
[31] S. E. Umbaugh. *Computer Vision and Image Processing: A Practical Approach Using Cviptools with Cdrom*. Prentice Hall PTR, 1997.
[32] C. Wang, D. Sklar, and D. Johnson. Forward error-correction coding. *Crosslink*, 3(1):26–29, 2001.
[33] Q. P. Wenjun Hu, Hao Gu. Lightsync: Unsynchronized visual communication over screen-camera links. In *MobiCom*, 2013.
[34] S. B. Wicker and V. K. Bhargava. *Reed-Solomon Codes and Their Applications*. Wiley, 1999.
[35] H. Yang, A. C. Kot, and X. Jiang. Binarization of low-quality barcode images captured by mobile phones using local window of adaptive location and size. *TIP*, 21(1):418–425, 2012.