SquawkComm: Practical Cost-Effective Vehicular Communication via Smartphones and FM Signals

Adam C. Champion*, Austin Gilliam*, Qiang Zhai†, Linghe Kong†, and Dong Xuan*
*Department of Computer Science and Engineering, The Ohio State University, USA
†Department of Computer Science and Engineering, Shanghai Jiao Tong University, Shanghai, P.R. China
{champion,zhaiq,xuan}@cse.ohio-state.edu, gilliam.115@osu.edu, linghe.kong@sjtu.edu.cn

Abstract—Vehicular communication applications such as injury notifications and socialization are very appealing. However, it is challenging to realize them cost-effectively with low latency using today’s smartphone technology. In particular, technology such as ad hoc or mobile hotspot WiFi suffers limitations such as specialized knowledge requirements for “rooting” devices as well as high latency. To overcome these limitations, we propose SquawkComm, a novel vehicular communication system using smartphones and FM signals. SquawkComm leverages inexpensive automotive FM transmitters that connect to smartphones as well as vehicles’ built-in stereos. Stereos are installed in most vehicles and they can rapidly deliver audio without tedious connection establishment. SquawkComm encodes smartphone users’ textual data to audio, which FM transmitters send to nearby vehicles. SquawkComm uses two supporting technologies: SquawkCode and SquawkLink. SquawkCode encodes data as audio using OOK and chooses the lowest unused FM frequency. SquawkLink frames data and controls channel access via an algorithm that builds on CSMA/CA. We implement SquawkComm using commercial off-the-shelf smartphones and FM transmitters. Our experimental evaluation in laboratory settings demonstrates its promise for vehicular communication with low latency. Tests with real-world vehicles in both stationary and mobile settings are an important part of future work.

I. INTRODUCTION

Both academia and industry pay close attention to vehicular area networks (VANETs) [1]. However, VANETs are in the development phase and there is no single recognized standard [1]. Currently, diverse technologies try to solve communications problems that arise in different applications.

Wireless communication among nearby vehicles has several uses:

– Suppose someone has an injury and needs to be driven to the hospital. The vehicle driver wants to notify nearby vehicles of the situation in order to expedite medical care. NHTSA estimates that over 2.3 million people were injured on U.S. roads in 2013 alone [2].

– Occupants in nearby vehicles traveling to a common destination converse by sharing commonalities [3], which eases social interaction. A 2014 survey found that 97% of young people post on social media when traveling [4].

– When one driver lets another enter a lane of traffic, the second driver electronically thanks the first driver. Hence, the second driver does not need to wave to the first one or quickly blink emergency flashers, which are customary.

Certain communication requirements need to be met in order to enable such applications. First, latency must be minimal as opportunities for communication among proximate vehicles may be short (e.g., a few seconds). Second, people should not need to set up irritating manual connections, which may lead to missed communication opportunities. Third, communications need to be cost-effective in order to reach a majority of users with limited budgets. Fourth, communication should be convenient for vehicle occupants whose mobile devices accompany them everywhere.

Table I shows the limitations of existing work on vehicular communication regarding our requirements. We describe each type of work and its limitations:

<table>
<thead>
<tr>
<th>Existing Work</th>
<th>Latency</th>
<th>Automatic Communication?</th>
<th>Cost-Effective?</th>
<th>Convenient?</th>
</tr>
</thead>
<tbody>
<tr>
<td>Two-way radios</td>
<td>Low</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
<tr>
<td>Smartphone FM</td>
<td>Low</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Bluetooth</td>
<td>High</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Ad hoc Wi-Fi</td>
<td>Low</td>
<td>✗</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Wi-Fi Direct</td>
<td>High</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Mobile hotspot Wi-Fi</td>
<td>High</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Cellular</td>
<td>Moderate</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>DSRC broadcast</td>
<td>Low</td>
<td>✓</td>
<td>✓</td>
<td>✗</td>
</tr>
</tbody>
</table>

TABLE I: Limitations of existing work for our requirements.

– Two-way radios: Two-way radios (“walkie-talkies”) seem promising as they let vehicle occupants talk with low latency and no connection establishment. However, using two separate devices (two-way radios and mobile devices) is inconvenient.

– Smartphone FM: Researchers have used Nokia N900 smartphones’ built-in FM hardware for proximity social networking [5], [6], but these old devices have low market share. As some mobile devices do not support FM signals [7], this approach does not fit communication among nearby vehicles.

– Bluetooth: Bluetooth takes over 10 seconds to discover nearby devices and requires connection establishment, which is tedious and ill-suited for our purposes.

– Ad hoc Wi-Fi: Ad hoc Wi-Fi is unavailable on most COTS mobile devices [8] unless they are rooted or jailbroken. Hence, it is impractical.
We propose SquawkComm, a novel vehicular communications system using COTS mobile devices and FM signals that achieves low latency, cost efficiency, and convenience without connection establishment or external infrastructure. SquawkComm leverages vehicle stereos’ built-in FM reception and low-cost COTS FM transmitters. It is easy to use: one just tunes transmitters and stereos to the same FM frequencies. Many countries allow unlicensed FM communication within tens of meters [20], [21], [22], [23]. This paper focuses on low data rate applications only. Applications needing higher data rates can use other communications techniques.

SquawkComm sends data among vehicle occupants’ mobile devices as well as data from vehicles themselves (e.g., from Bluetooth dongles plugged into OBD-II ports). This is convenient for users. We identify senders and receivers via numeric vehicle IDs (VIDs). As each sender generates a new VID before sending, SquawkComm resists attempts to identify users via their devices’ MAC addresses [24], [25]. SquawkComm uses two supporting technologies:

- **SquawkCode**: At the physical layer, SquawkComm achieves low latency using SquawkCode, which leverages on-off keying (OOK) to encode data as audio signals. For all audio baseband frequencies, we encode 1s as sine waves with these frequencies and 0s as unchanged audio. In the frequency domain, mobile devices detect “spikes” in Fast Fourier Transform (FFT) amplitudes corresponding to frequencies for 1s (and 0s otherwise). FM transmitters send audio to proximate vehicles. Each device chooses the lowest unused carrier frequency to tune transmitters and stereos. Thus, nearby vehicles receive audio as rapidly as that from local radio stations.

- **SquawkLink**: At the link layer, SquawkComm operates without infrastructure via SquawkLink, our distributed channel access algorithm that adapts to the number of nearby vehicle senders. We send data in frames, each of which has source and destination VIDs and a CRC checksum. If there are duplicate VIDs, the receiver notifies the sender, which generates a new VID. SquawkLink sends frames using communication “rounds” with two parts: RTS/CTS exchange and transmission. During transmission, we divide time into slots. Each vehicle sorts its list of received VIDs. The position of the vehicle’s VID in the list determines its time slot.

  - **Typical Working Scenario**: Now we give a typical working scenario that shows SquawkComm in action. Suppose Alice and Bob drive vehicles, she is injured on the way to the hospital, and her vehicle is in front of hers. SquawkComm chooses the lowest unused FM carrier frequency (e.g., 88.3 MHz) to which both of them tune their vehicles’ stereos. Alice sends the message “Injured” via her device. SquawkComm encodes her message using as audio and sends it via her transmitter. Bob’s stereo plays the audio, which his device receives via its microphone, decodes the text, and displays it. Bob changes lanes, helping Alice arrive at the hospital sooner.

We implement SquawkComm on COTS mobile devices and FM transmitters. We evaluate its bit error rate and latency in a laboratory setting. We find that SquawkComm achieves low bit error rate and latency.

In summary, we make the following contributions:

- We propose SquawkComm, the first communications system for nearby vehicles that achieves low latency without infrastructure via pervasive mobile devices, automotive FM transmitters, and vehicle stereos;

- We design SquawkCode, an OOK-based physical layer scheme for encoding data as audio signals and choosing unused FM carrier frequencies;

- We design SquawkLink, a distributed link layer algorithm for channel access among communicating vehicles; and

- We implement SquawkComm on COTS mobile devices and evaluate its performance.
II. RELATED WORK

a) Vehicle Communications: Communications among vehicles have been studied for a long time and we refer the reader to [1] for surveys of this topic. Some work [26], [27], [28] studies sharing Internet connections among vehicles and cluster-based vehicular communication. RoadSpeak [12] lets drivers speak with each other via 3G cellular infrastructure. Vehicular testbeds such as CarTel [29] and DieselNet [30] have been developed for delay tolerant networking where vehicles serve as “data mules” for nearby data transfer. Dedicated Short Range Communications (DSRC) vehicular safety technology is under development; in DSRC, vehicles rapidly send messages to each other (at 1–10 Hz) regarding speed, braking, and turning [1]. Bai et al. measured DSRC’s performance on the road [31]. Some work [13], [14] has integrated DSRC with smartphones using cellular networks, WLANs, or custom silicon [32]. NHTSA intends to expedite DSRC development and installation in new vehicles [33], [34]. In this paper, our SquawkComm complements DSRC broadcast, but we use RTS/CTS with commercial off-the-shelf (COTS) FM transmitters, smartphones, and vehicle stereo to avoid broadcast storms to which DSRC is susceptible [17].

b) Participatory Sensing Regarding Vehicles: Researchers have developed participatory sensing systems using people’s smartphones in vehicles. (We refer the reader to surveys [35], [36] for more information.) Nericell [37] senses road and traffic conditions using smartphones’ accelerometers, GPS, and microphones. GreenGPS [38] uses smartphones’ GPS and vehicular OBD-II data for fuel-efficient routing in cities. SignalGuru [39] predicts traffic signal schedules using smartphones’ cameras and accelerometers. WreckWatch [40] uses smartphones’ GPS, microphones, and accelerometers to detect accidents and report them to first responders. VTrack [41] uses smartphones’ GPS and Wi-Fi to estimate traffic delays. Unlike this body of work, our SquawkComm enables cost-effective low-latency communication among nearby vehicle occupants using their smartphones, FM transmitters, and vehicular stereo.

c) Smartphone Communications: Recently, various systems have been developed for communication among smartphones. Better Approach To Ad Hoc Networking (B.A.T.M.A.N.) uses ad hoc Wi-Fi for smartphone communication without infrastructure [42]. FireChat uses Bluetooth and ad hoc Wi-Fi for similar purposes [43]. Smartphone applications such as Zello, Heytell, and Vower [44] enable voice chat over Wi-Fi or cellular infrastructure. GoTenna uses expensive VHF antennas ($199 per pair) with smartphones for communication over a few kilometers without infrastructure [45]. Hu et al. [11] minimize smartphones’ switching time between Wi-Fi “AP” and “client” modes in order to enable phone-to-phone communication among vehicles, but their latency (> 3 s) is too high for our purposes. Su et al. [46] used Wi-Fi for communication between smartphones in vehicles. However, B.A.T.M.A.N. requires rooting or jailbreaking smartphones, which entails specialized knowledge. Ad hoc Wi-Fi is typically unavailable on COTS smartphones [8]. Wi-Fi Direct requires manual connection establishment, which can take up to two minutes [10], [9]. Bluetooth requires at least 10 seconds for nearby device inquiry as well as manual connection establishment. In contrast, our SquawkComm uses vehicle occupants’ smartphones as well as vehicular FM transmitters and stereos for cost-effective low-latency communication among nearby occupants.

Researchers have explored FM technology for communications. Rahmati et al. [6] and Paolini et al. [47] developed systems using RDS for communication among smartphones. Yu et al. [5] proposed a smartphone communication system for nearby people using FM audio. Wang et al. [48] developed a hybrid system using smartphones and FM transmitter circuit boards to share music playlists among people in proximity. In addition, researchers have used RDS for synchronization in wireless sensor networks [49] and among Wi-Fi APs [50]. By contrast, our SquawkComm achieves communication among nearby vehicle occupants using their smartphones as well as automotive FM transmitters and stereos.

d) DSRC and Privacy: Vehicles’ rapid DSRC broadcast messages can reveal their MAC addresses to attackers, which is a serious privacy issue. Researchers have investigated various schemes for assigning vehicles non-identifying pseudonyms instead of MAC addresses and updating pseudonyms on the road [51]. In this paper, SquawkComm communicates using FM audio, not DSRC, and our messages do not contain geolocations. In SquawkComm, each sender randomly generates a new vehicle ID (VID) upon message transmission, which helps SquawkComm resist fingerprinting via MAC addresses [24], [25].

III. SYSTEM DESIGN

In this section, we present SquawkComm’s design rationale in Section III-A and discuss its system workflow in Section III-B. Sections III-C and III-D detail SquawkCode and SquawkLink, respectively.

A. Design Rationale

We face several challenges in designing such a system:

- Carrier frequency selection: Regulations specify that unlicensed FM transmitters have maximum ranges of tens of meters, which limits transmit power [20], [21], [22], [23]. Under such restrictions, vehicles cannot communicate on carrier frequencies used by licensed FM radio stations. In addition, if vehicles communicate on different unused carrier frequencies, they cannot hear each other;

- Data representation efficiency: Our system needs to encode data as audio and decode data from audio efficiently, since FM’s audio bandwidth is limited;

- Error detection: Transmitted data may be corrupted by errors at the receiver side. Our system needs to detect errors and handle them accordingly; and

- Channel collisions: If multiple vehicles send data at the same time, their transmissions will collide, yielding unintelligible results. Our system needs to control channel access in order to avoid collisions.
SquawkComm addresses these challenges as follows. At the physical layer, we design SquawkCode, which encodes data to audio using OOK and finds the lowest unused FM carrier frequency locally on mobile devices. We develop an algorithm for unused carrier frequency selection that all devices run locally. At the link layer, we design SquawkLink, which frames data for transmission and controls channel access to avoid collisions. Recall that we repurpose vehicle stereos for receiving data sent by SquawkComm. Most vehicles have built-in stereos that demodulate FM audio in hardware and play it via speakers. We piggyback on this functionality: mobile devices receive audio via their microphones and decode data accordingly.

In addition to user-entered text, SquawkComm can send data such as vehicular speed via the vehicle’s OBD-II port.

C. Physical Layer: SquawkCode

In this subsection, we first provide background information on FM. Next, we discuss SquawkCode, which has two parts: our on-off keying (OOK) mechanism that encodes data as audio and our algorithm that chooses the lowest unused FM carrier frequency for transmission. We discuss each in turn.

1) Background: FM: Fig. 2 shows the frequency spectrum of baseband FM signals. Monaural audio is transmitted between 30 Hz and 15 kHz. A pilot tone is transmitted at 19 kHz in order to receive stereo audio, which is transmitted as two 15 kHz wide signals from 23–38 kHz and 38–53 kHz, respectively. Stereo receivers reconstruct stereo audio at 38 kHz, the second harmonic of the pilot tone. A ±4 kHz guard band surrounds the pilot tone. The Radio Data System (RDS) (known as the Radio Data Broadcast System (RDBS) in the U.S.) transmits textual information at 57 kHz (±6 Hz). This information may include flags for types of audio (e.g., traffic, music genre), radio stations’ carrier frequencies and callsigns, and the name of the current song playing [53]. While RDS seems appealing for vehicular communication, not all vehicle stereos can receive RDS information.

FM transmitters modulate baseband signals to carrier frequencies (∼88–108 MHz) that vehicle stereos receive and demodulate. Countries including the U.S., Canada, European states, and Japan allow low-power unlicensed FM transmission within tens of meters [20], [21], [22], [23]. SquawkComm’s design follows U.S. FCC regulations.

2) Encoding Data to Audio: Since all vehicle stereos can receive audio, we design SquawkCode’s scheme for encoding data as audio signals. SquawkCode’s coding scheme achieves a maximum bitrate of 108.75–118.57 bits per second, which suits low data rate applications. We elaborate this scheme as follows.

One solution is to use the entire FM monaural bandwidth (30 Hz–15 kHz) to encode data. However, this solution ignores people’s varying perception of audio loudness at different frequencies. Fig. 3 plots ISO 226 equal loudness contours at various phons, or levels of perceived loudness [54]. 1 phon is defined as the sound pressure level (in dB) of a 1 kHz sound with equal loudness. As Fig. 3 shows, for any sound pressure level, frequencies above 10 kHz are perceived as louder than those below 10 kHz. In order to minimize irritation of vehicle
occupants, we set 10 kHz as the upper bound of SquawkCode’s audio bandwidth.

Road noise in vehicles can also interfere with audio coding. We conduct an experiment in two vehicles (a 1999 Buick Century and a 2003 Chevrolet Malibu) driving 35–65 mph (~15.6–29 m/s) on city roads and highways with the windows closed, radio off, and no one talking. We measure the frequency response over ~90 s using a Samsung Galaxy Nexus running Android 5.1 and the mobile application (app) Audalyzer [55]. We set the app’s sampling rate to 44.1 kHz and the block size to 4,096 using a Blackman-Harris window function. Fig. 4 shows the results; considerable road noise is evident for frequencies below 2 kHz. Typically, the average amplitude of ambient noise was between −15 dB and −12 dB, with maximum amplitudes ranging from −12 dB to −3 dB. Various data spikes occur due to uneven roads and audible noise such as turn signals and vehicles’ climate control systems. We could use lower frequencies down to 30 Hz in our system, but road noise could interfere with audio coding. As a tradeoff, we treat 1 kHz as the lower bound of SquawkCode’s audio bandwidth.

SquawkCode’s coding scheme uses OOK with audio frequencies between 1 kHz and 10 kHz. For each audio baseband frequency \( f \), we encode a 1 as a sine wave \( \sin(2\pi ft) \) and a 0 as unchanged audio at that frequency. We superpose the sine waves to generate the encoded audio signal. In the frequency domain, this yields “peaks” and “valleys” in FFT amplitudes corresponding to 1s and 0s, respectively. Mobile devices can easily detect these via the FFT, which decodes the data. Due to these properties, OOK suits SquawkComm well. We use an FFT with sampling rate 44.1 kHz and size 2,048. However, signal aliasing limits the granularity with which we can recover each frequency \( f \). To mitigate aliasing, we consider each sequence of five adjacent bins as a “single unit” and employ a peak detection algorithm for each sequence of bins. As a result, each frequency 1,098 Hz, 2,048 Hz, . . . , 10,034 Hz represents a single bit with 84 bits in total.

Now we give an example that shows SquawkCode in action. Sending the message “Hello!!” takes 56 bits using ASCII encoding. Our scheme places this message in a 76-bit or 83-bit frame (depending on the CRC checksum length as discussed in Section III-D). The receiver receives the message in 0.7 seconds. Thus, we can achieve a peak bitrate of 108.75–118.57 bits per second.

3) FM Carrier Frequency Selection: Successful system operation requires senders and receivers to use a common carrier frequency for FM transmission. Carrier frequency selection entails avoiding licensed FM radio stations in order to avoid interference. In addition, in order to maximize SquawkComm’s communication range, we need to use the lowest possible unused FM carrier frequency. This follows from the Friis transmission equation:

\[
\frac{P_r}{P_t} = G_t G_r \left(\frac{\lambda}{4\pi R}\right)^2.
\]

Higher carrier frequencies have smaller wavelengths than lower carrier frequencies. If antenna gains \( G_r \) and \( G_t \) as well as the transmitted power \( P_t \) are fixed, the received power \( P_r \) is larger with lower carrier frequencies.

The number of FM transmitters near vehicles influences SquawkCode’s number of possible unused carrier frequencies. Consider \( N \) (isotropic) transmitters on a two-dimensional plane where each transmitter \( i \) is at geolocation \( x_i \) (\( i = 1,\ldots,N \)). Each transmitter transmits at a frequency \( f_i \) with a certain power and gain. We have a received power threshold \( P_{th} \) that is needed to receive the transmission. There is a circle around each transmitter \( i \) with center \( x_i \) and radius \( d_{max} \) where received power equals \( P_{th} \). For any point \( x \) at time \( t \), we receive all FM frequencies \( \{f_i\} \) whose transmitters’ Euclidean distance to \( x \) is at most \( d_{max} \). However, certain frequencies may be so close together (e.g., 89.3 MHz and 89.5 MHz) that there is insufficient “wiggle room” to transmit on the average frequency between them (e.g., 89.4 MHz) without interference. Thus, we cluster frequencies \( \{f_i\} \) (including the minimum and maximum FM frequencies). Frequencies whose separation is below a threshold (e.g., 0.3 MHz) are placed in one cluster.

Let \( N_{unused}(t,x) \) denote the number of carrier frequencies, which is one less than the number of clusters. In addition, traffic patterns upper-bound the number of carrier frequencies. Let \( N(t,x) \), \( N_{max}(t,x) \), and \( \rho(t,x) = N(t,x)/N_{max}(t,x) \) denote the number of vehicles on the road, the maximum number of vehicles corresponding to stationary bumper-to-bumper traffic, and the density of vehicles near \( x \) at time \( t \), respectively [56]. Thus, the number of used carrier frequencies in SquawkCode is at most

\[
\min\{\lfloor N(t,x)/2 \rfloor, N_{unused}(t,x)\} = \min\{\rho(t,x)N_{max}(t,x), N_{unused}(t,x)\}.
\]

We design Algorithm 1 that finds the lowest unused carrier frequency. In Algorithm 1, candFmFreqs is the set of all unused FM carrier frequencies, usedFmFreqs is the set of all used frequencies, and fmCarFreq is the returned FM frequency. getCandFmFreq() takes one parameter, the smartphone’s estimated geolocation \( (\phi, \lambda) \), and calls getCandFmFreq() with this location. getUsedFmFreq() returns all unique FM transmitter frequencies within a bounding box with coordinates \( (\phi \pm \Delta \phi, \lambda \pm \Delta \lambda) \), where \( \Delta \phi \) and \( \Delta \lambda \) are respective latitude and longitude thresholds specifying the size of the box. In practice, we search databases of radio stations such as FMLIST [57]. findEquidistFreqs() finds frequencies “equidistant” to those in usedFmFreqs. (For instance, given the input \( \{87.5, 93.1, 101.1, 107.9\} \),
Algorithm 1: SquawkCode’s FM Carrier Frequency Selection

```
1: function getCandFmFreqs(φ, λ)
2:   allFmFreqs ← {87.5, ..., 107.9};
3:   usedFmFreqs ← getCandFreqs(φ, λ);
4:   usedFmFreqs ← sort(usedFmFreqs ∪ {87.5, 107.9})
5:   candFmFreqs ← getCandFreqs(usedFmFreqs)
6:   findEquidistFreqs(candFmFreqs)
7:   for freq ∈ candFmFreqs do
8:     if not isInUse(freq) then
9:       fmCarFreq ← freq
10:      break
11:   end if
12:  end for
13: return fmCarFreq
14: end function
15: function getCandFreqs(φ, λ)
16:   fmFreqs ← unique transmitter frequencies in bounding box (φ ± Δφ, λ ± Δλ)
17:   for freq ∈ fmFreqs do
18:     usedFmFreqs ← usedFmFreqs ∪ {freq}
19:   end for
20: return usedFmFreqs
21: end function
22: function findEquidistFreqs(usedFmFreqs)
23:   availFmFreqs ← ∅
24:   for each pair of consecutive frequencies freq_i, freq_j ∈ usedFmFreqs do
25:     newFreq ← (freq_i + freq_j)/2
26:     availFmFreqs ← availFmFreqs ∪ {newFreq}
27:   end for
28: return availFmFreqs
29: end function
```

findEquidistFreqs() returns \{90.3, 97.1, 104.5\}. It can be seen that the algorithm yields the lowest unused FM carrier frequency.

D. Link Layer: SquawkLink

In this subsection, we discuss SquawkLink, which has two parts: framing and channel access. We describe each in turn.

1) Framing: Before we describe SquawkLink’s frame structure, we estimate how many vehicles on the road need to be addressed at any point in time. Consider a straight length of roadway with \(ℓ\) lanes in each direction, each of width \(w\) (where \(\ell > 0\) and \(w < r\)). We model the FM communication range as a disc with radius \(r\) and center \(C\) where the roadway length is much greater than \(r\). Algorithm 5 shows our communication model; the dotted line divides traffic moving in different directions. We assume that \(ℓ\) ranges between 1 and \(r/w\). Angle \(θ\) extends between points \(A\) and \(B\) on the boundary of the disk about its center \(C\), and \(θ = \min\{\arctan(ℓw/r), π/2\}\.

We want to find the roadway area \(A\) inside the disk that comprises the areas of sectors \(BCI\) and \(CEG\) and triangles \(BDC\), \(DEC\), \(CGH\), and \(CHI\). It follows that

\[
A = 2r^2 \min\{\arctan(ℓw/r), π/2\} + \frac{(ℓwr) + \sqrt{(ℓwr)^2 + 1}}{2}
\]

In practical situations, while \(ℓ\) varies for different roadways, \(w \geq 2.4\ m\) [58]. For example, \(ℓ = 2\) on residential streets, but \(ℓ ≥ 4\) on highways. We consider \(w = 2.6\ m\) as this is the maximum width for commercial trucks, \(r = 60\ m\) for regulatory reasons, and \(ℓ = 4\) [59], [20]. We find \(A = 2,465.39\ m^2\), enough space for \(~ 256 \times 1.663\ m\) stationary Smart Fortwos, each of which has 1 m behind it and in front it
[60]. (Fortwos are the smallest vehicles we find with built-in FM stereos.) However, real-world roadways have irregularities such as median barriers and vehicles such as trucks are much larger than Fortwos. In addition, traffic patterns vary throughout the day, leading to congestion at “rush hours” and nearly empty roads at night. Consequently, practical roadways tend to have fewer vehicles in FM communication range at one time than in our stationary case. Thus, we design SquawkLink to address 64 vehicles.

Next, we consider how to address each vehicle in FM communication range. At minimum, a vehicle needs to address each vehicle adjacent to it on roadways. One approach uses the 17-character Vehicle Identification Number (VIN) that each vehicle on the road must have by law [61]. However, representing the VIN would take at least 136 bits (in ASCII), which does not fit SquawkComm. Another approach uses the vehicle license plate number. While license plate numbers are short (6–7 characters), administrative regions within countries typically issue license plates separately [62]. This could lead to two different vehicles having the same ID, which is unacceptable. For efficiency, we generate a random vehicle ID (VID) for each vehicle: \( VID \sim U(0, N - 2) \), where \( N = 64 \) is the maximum number of vehicles. We represent \( VID \) using \( \lceil \log_2(N) \rceil \) bits. If a sender and a receiver have duplicate VIDs, the receiver notifies the sender, which generates a new VID.

### Algorithm 2 Channel Access Protocol

1: Generate random vehicle ID \( VID \)
2: if vehicle needs to send data then
3: Generate new random \( VID \)
4: if frame has urgent status then
5: \( t_{wait} \sim U(0, t_{max}) \)
6: Send frame
7: if collision then
8: \( t_{max} \leftarrow \min\{t_{max}, 2 \cdot t_{max}\} \)
9: \( t_{max} \leftarrow \max\{t_{min}, t_{max}/2\} \)
10: end if
11: else
12: Send CTS, timestamp
13: if \( \{\text{vehicle receives frame with type } 11\} \) then
14: Generate new random \( VID \)
15: end if
16: Wait until time slot
17: Send frame for time slot duration
18: Wait until all time slots finish
19: end if
20: else
21: \( t_{wait} \sim U(0, t_{max}) \)
22: Send CTS, timestamp
23: Send frame; if collision then
24: Send CTS, timestamp
25: Wait until time slot
26: Send frame with priority
27: end if

2) Channel Access: SquawkComm faces an important challenge: preventing several vehicle occupants from sending data at the same time. Each vehicle occupant on the road that wants to send data may not see other senders nearby and transmit, leading to collisions. Since no sender can “hear” the data it is sending, collision avoidance is desirable. No one should be starved from sending data due to other senders constantly transmitting. However, urgent frames should be delivered before other frames.

We address these challenges via SquawkLink’s channel access protocol shown in Algorithm 2. Although Algorithm 2 builds atop CSMA/CA, we tailor it for communication among rapidly moving vehicles. Initially, each vehicle occupant’s mobile device chooses its own VID at random following the approach discussed previously. The remainder of the protocol only applies to senders. For privacy reasons, each sender generates a new temporal VID before sending (line 3). If a frame has high priority, we choose a random wait time \( t_{wait} \sim U(0, t_{max}) \) and wait, where \( t_{max} \) is the longest possible wait time. Next, we send the frame. We double or halve \( t_{max} \) based on whether there is a collision (subject to thresholds \( t_{min} \) and \( t_{max} \)). Otherwise, communication takes place in rounds (lines 13–22). At the start of a round, after waiting a short random period of time, each sender transmits an RTS frame while listening for other senders’ RTS frames. Each RTS frame includes the sender’s VID and a timestamp. When the RTS period ends, each sender has
a list of all other senders’ VIDs. Each sender sorts its list and determines the position of its VID in the list. If a sender determines its VID is the maximum among all VIDs and there are least two VIDs, the sender sends a CTS frame. Time is then divided into slots; each sender waits for its slot and transmits its data. The round ends once all senders finish transmitting. If any vehicle sender has more data to send, it needs to do so in the next round. If a sender and a receiver have duplicate temporal VIDs, the receiver notifies the sender via a frame with type 11 (see Section II). The sender then generates a new random temporal VID (lines 22–24). SquawkLink’s time slot allocation mechanism prevents any nearby vehicle occupant from sending data for more time than it is allotted.

Now we analyze this protocol’s collision probability. Suppose there are $n$ vehicles in proximity, each of which has an occupant that transmits independently with probability $p$. Let $t_{\text{RTS}}$ and $t_{\text{CTS}}$ denote the time periods where vehicles send RTSs and CTSs, respectively, and let $ts$ denote the length of a time slot. We consider two extreme cases:

- **Case 1**: All vehicle occupants transmit urgent frames; and
- **Case 2**: All vehicle occupants transmit frames using time slots.

We consider Case 1 first. Collisions occur when two or more vehicles transmit simultaneously (i.e., $Pr\{\text{collision}\} = \sum_{i=2}^{n} \binom{n}{i} p^i (1-p)^{n-i}$). It follows that $Pr\{\text{collision}\} = 1 - (1-p)^{n-1} + np$. However, in Case 2, collisions only occur before vehicle occupants determine their time slots. Thus, the collision probability only holds during RTS and CTS transmission: $(t_{\text{RTS}} + t_{\text{CTS}})/(t_{\text{RTS}} + t_{\text{CTS}} + n \cdot ts)$. As $n$ increases, the number of time slots increases and this fraction decreases.

IV. IMPLEMENTATION AND EVALUATION

In this section, we first discuss our implementation of SquawkComm followed by our experimental evaluation.

A. Implementation

We implement SquawkComm using COTS Samsung Galaxy Nexus smartphones running Android 5.1 as well as two GoGroove FlexSmart X2 FM transmitters for vehicles. Each FM transmitter plugs into a vehicle’s cigarette lighter. Each smartphone connects to an FM transmitter via 3.5 mm audio cable. Our mobile application (app) includes an offline database of FM transmitters for several countries [57]. We use these databases to find nearby transmitters (within 50 kilometers) in order to determine an unused FM carrier frequency. The app’s total size is 10.52 MB, of which 1.59 MB is used for the databases. Due to its size, our app can be installed on a wide variety of mobile devices.

We implement our app’s audio processing using FFTPack [63]. One thread in the app continuously listens for audio, computes the FFT, parses the frame of delivered data, and decodes the data. Another thread encodes user-specified textual data as audio, performs framing, and sends out the data via the smartphone’s 3.5 mm audio jack to the FM transmitter.

B. Evaluation

We evaluate our SquawkComm app in both laboratory and real-world vehicular environments. We describe each in turn.

1) **Laboratory Environment**: We evaluate SquawkComm in a laboratory environment with two Galaxy Nexus devices running Android 5.1: one device is the sender and the other is the receiver. The sender device connects to an FM transmitter whose carrier frequency is 88.3 MHz. (Our app indicates this is the lowest unused frequency.) We tune a C. Crane Pocket Radio [64] to this frequency. The receiver phone is placed a few centimeters from a Logitech Z80 computer speaker that connects to the radio via 3.5 mm audio cable. We use 1.25-second RTSs and data frames. We send the message “hello” in ten rounds using both urgent frames and “non-urgent” ones (i.e., our channel access algorithm). For each set of ten rounds, we record the reception latency and bit error rates (BERs) and average the results. Fig. 7 and Fig. 8 illustrate SquawkComm’s latency and BERs, respectively, for urgent and non-urgent frames.

![Fig. 7: SquawkComm overall latency.](image)

![Fig. 8: SquawkComm bit error rate.](image)

Fig. 7 shows that SquawkComm’s overall latencies are 0.357 seconds and 3.178 seconds for urgent and non-urgent frames, respectively. Similarly, Fig. 8 shows that SquawkComm’s BERs are 0.1776 (for urgent frames) and 0.1714 (for non-urgent frames). Fig. 9 shows the components of
SquawkComm’s overall latency for both types of frames. There are five components: the RTS frame duration, the RTS processing latency, the “inter-frame” latency, the data frame duration, and the frame processing latency. The first three components are absent for urgent frames. Fig. 9 illustrates the 1.25-second frame lengths (for both RTS and data frames). For non-urgent frames, there is a 0.394-second delay between processing RTS and data frames (the inter-frame latency). RTS and data frame processing latencies are 0.146 seconds and 0.138 seconds, respectively.

2) Vehicular Environment: We evaluate our app using two vehicles: a 2017 Subaru Legacy and a 2003 Mazda CX-5. Both vehicles are parked in a parking lot next to each other in December 2016. The Legacy sends messages using one Galaxy Nexus phone and the CX-5 receives them using another Galaxy Nexus phone; both phones run Android 5.1. We synchronize both phones’ clocks via NTPSync [65] over Wi-Fi beforehand. These experiments are similar to those in the laboratory environment except the CX-5 moves one parking space away after each round of ten messages. We observe that SquawkComm sends urgent frames with latency similar to that in the laboratory. Using our channel access algorithm, we observe SquawkComm sends RTSs and frames within a few seconds. Data analysis for these experiments are part of our ongoing work.

V. CONCLUSION

This chapter presented SquawkComm, the first system for cost-effective vehicle communication using mobile devices and FM signals. At the physical layer, we developed SquawkCode, which included a scheme for encoding data as audio and an algorithm for choosing unused FM carrier frequencies. At the link layer, we developed SquawkLink, which included a framing mechanism as an channel access algorithm. We implemented SquawkComm using off-the-shelf smartphones and FM transmitters and experimentally evaluated it in laboratory environments. The results showed SquawkComm’s promise for cost-effective communication among nearby vehicles with low latency and bit error rates.

Our SquawkComm experiments have studied communication in a laboratory environment. In future work, we will evaluate SquawkComm in real-world vehicular environments and analyze the resulting latencies and bit error rates. We will study the impact of distance among nearby vehicles in stationary and mobile settings.

REFERENCES


