DEMO: Demonstrating Reactive Smartphone-Based Jamming

Matthias Schulz
Secure Mobile Networking Lab
TU Darmstadt, Germany
mschulz@seemoo.de

Efstathios Deligeorgopoulos
Secure Mobile Networking Lab
TU Darmstadt, Germany
edeligeorgopoulos@seemoo.de

Matthias Hollick
Secure Mobile Networking Lab
TU Darmstadt, Germany
mhollick@seemoo.de

Francesco Gringoli
CNIT / Dept. of Information Engineering
University of Brescia, Italy
francesco.gringoli@unibs.it

ABSTRACT

Reactive Wi-Fi jammers on off-the-shelf hardware that may facilitate mobile friendly jamming applications have only been shown recently. Until now, no demonstrators existed to reproduce the results obtained with these systems, hence, inhibiting re-use for further research or educational applications. In this work, we present an Android app that allows to create advanced jamming scenarios with four Nexus 5 smartphones. We use two of them to inject Wi-Fi frames with UDP payload, one to receive frames and analyze if they were corrupted and one that acts as a reactive jammer that selectively jams according to a UDP port. The user can choose between a simple reactive jammer and an acknowledging jammer. All jammers are implemented as Wi-Fi firmware patches by using the Nexmon framework. During the demonstration, users may adjust parameters of transmitted frames and observe the throughputs of correct and corrupted frames as bar graphs at the receiver. At the jamming node, users may design an arbitrary jamming signal in the frequency domain and adjust the jamming power, the target UDP port and the jammer type. The MAC addresses used during the experiments are hard coded to hinder users from simply abusing the app in other setups. Overall, the demonstration proofs that highly sophisticated Wi-Fi jammers can run on smartphones.

ACM Reference format:
Matthias Schulz, Efstathios Deligeorgopoulos, Matthias Hollick, and Francesco Gringoli. 2017. DEMO: Demonstrating Reactive Smartphone-Based Jamming. In *Proceedings of WiSec '17, Boston, MA, USA, July 18-20, 2017,* 3 pages.
DOI: 10.1145/3098243.3106022

1 INTRODUCTION AND RELATED WORK

Practical reactive jammers already existed for multiple years. For their implementation, researchers need to build systems that quickly react to incoming Wi-Fi frames and send out a jamming signal that overlaps with the target frame. To meet these strict timing requirements, researchers either used software-defined radios (SDRs) [2] or modified the Wi-Fi firmware of low-cost off-the-shelf routers [1] or, very recently, smartphones [3]. In this work, we focus on extending the latter by offering an Android app to easily reproduce the experiments of our prior work, which is based on the Nexmon framework [4, 5], that allows to modify Wi-Fi firmwares running in Broadcom FullMAC chips. Their jammer not only supports single-stream 802.11ac frames, but also allows to design arbitrary jamming waveforms by writing I/Q samples into a buffer that can be played back whenever a jamming condition matches. This combines the flexibility of an SDR with the ubiquitous availability of low-cost Wi-Fi chips. Additionally, the authors presented a new jamming attack that sends fake acknowledgements to the transmitter of a target frame. This avoids blocking the transmission of other non-targeted frames at the transmitter and is particularly useful for "jamming for good" scenarios.

In this work, we present an Android app that demonstrates and controls the jammers presented in [3]. It also implements a frame transmitter that injects frames directly from the Wi-Fi firmware, as well as a receiver that analyzes the throughput. To ease reuse of our app, we published the source code and an instruction video on our website[1]. In the following, we first present our app in Section 2 followed by a description on how users can interact with our app during the demonstration in Section 3.

2 PRESENTING THE APP

Our Android app has three operating modes: Jammer, Transmitter and Receiver. Users may select them using the menu button on the left. In the following subsections, we explain the three modes.

2.1 Jammer mode

The Jammer is the main operating mode (see Figure 1). It consists of a menu bar, a power control slider and a start button at the top. Below, user-interface (UI) elements can be loaded to either design a jamming waveform in the frequency domain or to analyze it in the time or frequency domain. The reason for designing jamming signals in the frequency domain stems from the design of the jammer. To send waveforms instead of Wi-Fi frames as jamming signals, we use an I/Q-sample buffer originally intended to hold signals for internal calibrations. The size of this buffer is limited to 512 samples that can be played back multiple times in a loop. As writing samples to the buffer takes much longer than playing them back,

WiSec '17, Boston, MA, USA
 978-1-4503-5084-6/17/07…$15.00
DOI: 10.1145/3098243.3106022

[1]Download the source code of our app from https://nexmon.org/jamming-app, the one of the firmware from https://nexmon.org/jamming-app-firmware and watch the instruction video on https://nexmon.org/jamming-app-video.

the buffer cannot be refilled in real-time to create a continuously arbitrary signal transmission. Hence, we need to limit ourselves to cyclically repeating signals to avoid glitches between each two playback repetitions. Those signals are sine and cosine functions, where integer multiples of their periods exactly fit into the selected buffer size. The inverse discrete Fourier transform (IDFT) creates exactly those signals. By dividing the sampling rate by the IDFT size, one defines the subcarrier spacing (e.g., 40 MSps / 128 Samples = 312.5 kHz, which equals Wi-Fi's subcarrier spacing). As the Wi-Fi chip oversamples its digital signals by a factor of two, the sampling rate is always twice the signal bandwidth (e.g., 20 MHz bandwidth is sampled at 40 MSps). As out-of-band subcarriers are removed by a low-pass filter in the analog domain, we only consider subcarriers up to the band edges.

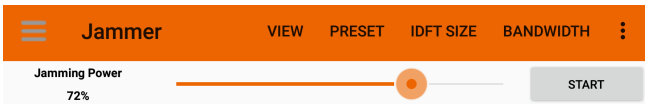
To manipulate the transmitted waveform, the user may change amplitudes and phases using the UI-elements illustrated in Figure 1b and Figure 1c, the resulting complex baseband signal is illustrated in Figure 1d. In general, the signal power used for jamming is split over all the activated subcarriers. If only one subcarrier is active, the power is focused on this subcarrier. Setting many subcarriers increases the peak-to-average-power ratio (PAPR, see Figure 1d) which reduces the power on each subcarrier that can be transmitted. To efficiently use the available transmit power, the user should design signals with low PAPR. To get started more quickly, the user may select presets from the menu bar to, for example, activate all pilot subcarriers on a 20 MHz channel. To configure additional settings such as the jammer type or the UDP port to target, the user may enter the options dialog from the menu. There are also dialogs to select the Wi-Fi channel, the bandwidth as well as a help dialog containing instructions on how to use the app.

2.2 Transmitter mode

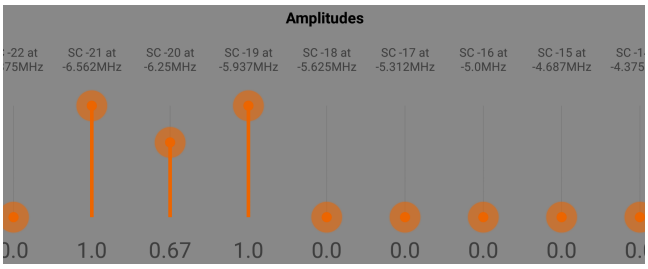
In the Transmitter mode, the user may create multiple UDP streams by clicking the plus-button in the lower right corner. In the upcoming dialog (see Figure 2a), the user may set a transmit power, a UDP port as well as modulation settings. After creation, the stream settings are listed in the UI-element (see Figure 2b) but are inactive by default. To start a stream, the user may press the play button on the left of each stream entry. This makes the app send an ioctl to the Wi-Fi firmware containing these settings. The firmware then starts a timer task to periodically inject Wi-Fi frames on the configured Wi-Fi channel. Those frames have their destination and source MAC addresses set to "NEXMON" and "JAMMER" which are the addresses that activate the reactive jamming operation in the firmware.

2.3 Receiver mode

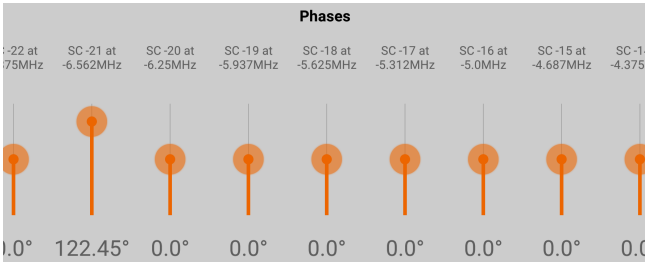
In the Receiver mode, our app shows a bar graph indicating the current throughputs (see Figure 3) per observed UDP stream (defined by the transmitting node id, UDP port and modulation settings). As the reactive jammer only starts jamming after analyzing the UDP header, this part is normally undamaged and jammed frames can be differentiated from unjammed frames by checking the frame check sequence (FCS). To simplify the app, we perform the FCS checks and filter frames in the firmware and send the results using new UDP frames on port 5500 to our app that creates the graphs.



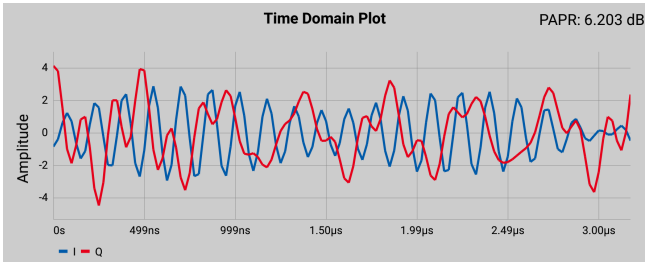
(a) Menu bar of the Jammer mode. Under VIEW, the user selects one (landscape orientation) or two (portrait orientation) of the UI-elements shown below.



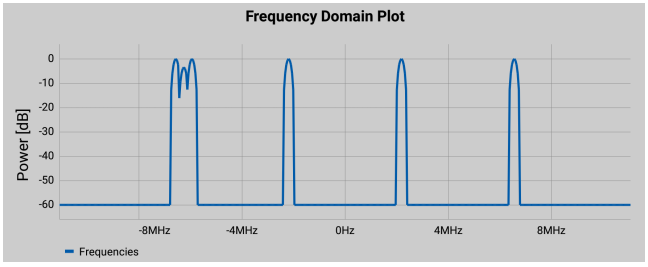
(b) In the *Amplitude* UI-element, the user sets the amplitudes per subcarrier of the signal used for jamming.



(c) In the *Phases* UI-element, the user sets the phases per subcarrier of the signal used for jamming.



(d) In the *Time Domain Plot* UI-element, we display the time domain waveform resulting from the amplitude and phase settings. We also display the PAPR.



(e) In the *Frequency Domain Plot* UI-element, we display the power distribution of the jamming signal in the selected signal bandwidth.

Figure 1: The user interface (UI) of the Jammer mode consists of a menu bar and four UI-elements that can be loaded below to either design a jamming signal in the frequency domain or to analyse the resulting waveform.

3 USING THE APP

During the demonstration, the user gets four rooted Nexus 5 smartphones (with stock firmware M4B30Z). We illustrate the setup in Figure 4. Each phone has our Jammer app preinstalled. In a **first experiment**, the user starts one phone in Receiver mode and a second in Transmitter mode and sets up a UDP transmission on port 3939 with 1000 802.11ac frames per second with MCS 6. An active transmission is indicated by the phone's notification LED blinking in cyan. The Receiver should show a bar graph for a new stream on port 3939 indicating only correct frame check sequences. Then the user starts another phone in Jammer mode, selects the "20 MHz pilots"-preset, targets port 3939 and activates the Simple Reactive Jammer whose activity is indicated by the phone's LED blinking in red. The graph at the receiver should now indicate a high amount of corrupted frames at a reduced frame reception rate due to the retransmission delay. In a **second experiment**, the user starts another transmitter on the forth phone on port 4040. This transmission should not be jammed. Hence, the Receiver should show a new bar graph indicating only correct frame check sequences, while frames with port 3939 are still corrupted. This experiment shows that the jammer differentiates between UDP ports. In a **third experiment**, the user starts another UDP stream on port 3030 on the second phone that also transmits on port 3939. As the transmitter is blocked by retransmitting jammed frames, only a few frames on port 3030 may be seen at the receiver. To unblock the transmit queue, the user may switch the jammer type to "Acknowledging Jammer" in the Options dialog. This lets the jammer transmit acknowledgments after jamming a frame to avoid retransmissions. As a result, the number of correctly received frames on port 3030 increase, but also the number of corrupted frames on port 3939. Overall, this experiment presents the effects of an acknowledging jammer. Last but not least, the user may experiment with different modulation and power settings, design arbitrary jamming signals in the frequency domain and change the distance between the smartphones.

4 ACKNOWLEDGMENTS

This work has been funded by the DFG SFB 1053 MAKI, by LOEWE NICER, LOEWE CASED and BMBF/HMWK CRISP and was supported by the EC framework H2020-ICT-2014-1 project WiSHFUL (Grant agreement no. 645274).

REFERENCES

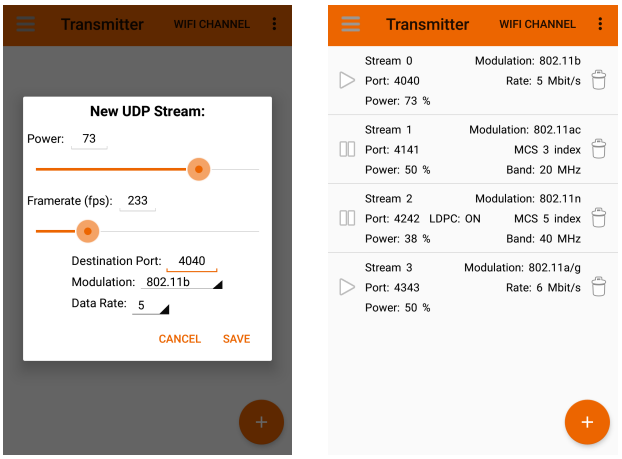
[1] Daniel S. Berger, Francesco Gringoli, Nicolò Facchi, Ivan Martinovic, and Jens B. Schmitt. 2016. Friendly Jamming on Access Points: Analysis and Real-World Measurements. *IEEE Transactions on Wireless Communications* 15, 9 (2016), 6189–6202.

[2] Danh Nguyen, Cem Sahin, Boris Shishkin, Nagarajan Kandasamy, and Kapil R. Dandekar. 2014. A real-time and protocol-aware reactive jamming framework built on software-defined radios. In *ACM workshop on Software radio implementation forum (SRIF) 2014.*

[3] Matthias Schulz, Francesco Gringoli, Daniel Steinmetzer, Michael Koch, and Matthias Hollick. 2017. Massive Reactive Smartphone-Based Jamming using Arbitrary Waveforms and Adaptive Power Control. In *ACM Conference on Security and Privacy in Wireless & Mobile Networks (WiSec) 2017.* Boston, USA.

[4] Matthias Schulz, Daniel Wegemer, and Matthias Hollick. 2016. DEMO: Using NexMon, the C-based WiFi firmware modification framework. In *ACM Conference on Security and Privacy in Wireless and Mobile Networks (WiSec) 2016.* ACM, Darmstadt, Germany, 213–215.

[5] Matthias Schulz, Daniel Wegemer, and Matthias Hollick. 2017. Nexmon: The C-based Firmware Patching Framework. (2017). https://nexmon.org



(a) Dialog for setting up new UDP streams. **(b) List of streams that can be started by clicking the play button.**

Figure 2: In the Transmitter mode, the user may setup UDP streams that the Wi-Fi firmware injects.

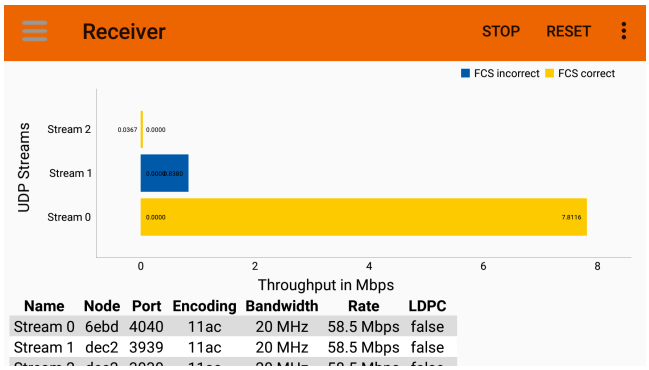


Figure 3: The Receiver mode displays bar graphs indicating the current throughput per stream. Jammed frames have an incorrect frame check sequence, while others are correctly received.

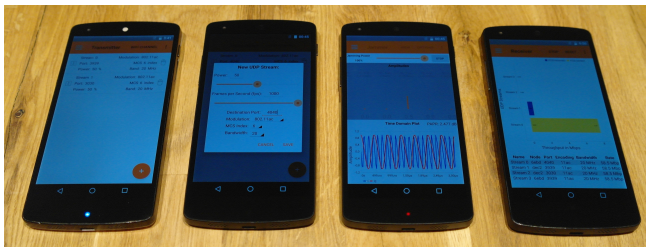


Figure 4: We provide these four Nexus 5 smartphones during the demonstration. The first one on the left is a transmitter sending at UDP ports 3939 and 3030. The second one sends at 4040. The third one is the jammer targeting frames on port 3939 and the fourth one is the receiver.