Reactive Jamming

Lab Report

Daniel May
Simon Schmitt
Technische Universtität Darmstadt
daniel_nicolas.may@stud.tu-darmstadt.de
simon_johannes.schmitt@stud.tu-darmstadt.de

ABSTRACT

This lab report presents the creation of an reactive jammer. We will describe how the frame handling works on the WARP and how it can be used to suppress individual targeted devices or communications respectively. At the end of this report we evaluate the performance of our jammer and discuss possible improvements.

1 INTRODUCTION

Wireless signals, as they are used in most of today's analog or digital communications, are very sensitive and affectable by the environment. Signals with the same frequency can interfere and suppress each other. This effect is typically used by jammers to prevent a certain receiver from decoding a signal. While jamming is typically associated with malicious behaviour or within military conflicts to hinder an opposing party from exchanging information, there also exist other jamming schemes, so called friendly jamming. Friendly jamming can be used to protect vulnerable systems from adversarial actions, e.g., pacemakers that can be wirelessly reprogrammed. More recent work also demonstrated that secrete key-exchanges can be realized at the physical layer utilizing a jammer.

The objective of this lab was to create a reactive WiFi jammer using the Wireless Open-Access Research Platform (WARP). WARP is a programmable Software-Defined Radio (SDR) which provides a basic implementation of the 802.11g WiFi standard. The architecture of the WARP allows to transmit frames while still receiving a signal. Thus WiFi transmissions containing a certain Medium Access Control (MAC) address can be analyzed and jammed if they are matching a target address.

In comparison with existing jammers this approach is more precise as it only suppresses the signals of a certain target, while still allowing the communication of other devices. This also results in much lower power-consumptions, due to the smaller amount of frames that have to be jammed.

2 BACKGROUND ON THE WARP

In this Section we describe how the frame handling works on the Wireless Open-Access Research Platform (WARP) and why there are multiple receive and transmit buffers.

The WARP is a transceiver, which means it can be used to send and receive frames respectively. This is realized with two independent paths of circuits that are connected to a single antenna (1). The antenna is followed by a switch (2) to either connect to the transmitter or receiver path. At the receiving path the incoming

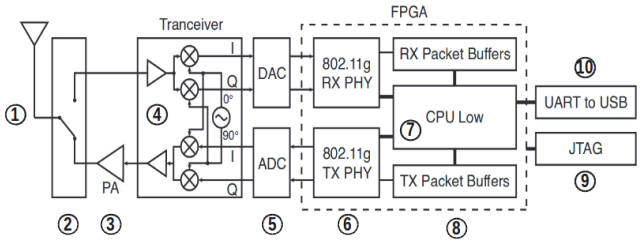

Figure 1: Block diagram of the 802.11 WARP

frames are directly forwarded to the transceiver module (4), while the signals leaving this module are amplified to a fixed gain (3). The transceiver module controls the conversion between the complex baseband signal and the Radio Frequency (RF) using a quadrature modulator. The next layer (5) contains the Digital-to-Analog and Analog-to-Digital Converter, which are connected to the chips that implement the 802.11 physical layer (6). Incoming frames are written into a RX Packet Buffer, while outgoing frames are read from the TX Packet Buffer. Both buffers represent shared memory that is also accessible by the MicroBlaze processor (7), which handles the MAC layer of the network interface. Whats special about the WARP, is the fact that the processor allows to start processing while incoming frames are still being received. This allows us to implement a reactive jammer. It is only necessary to prepare the frames used for the jamming signal. Those frames are stored in the TX Packet Buffer and can be send as soon as a condition matches to the incoming frame.

The JTAG port (9) is used to flash the firmware of the processor and to upload the implementation of the reactive jammer. Any debugging messages that are written to the standard output can be observed in a terminal that is connected to the UART to USB port (10).

3 IMPLEMENTATION

In this Section we describe how a reactive jammer can be implemented using the previous described Wireless Open-Access Research Platform (WARP). As a development environment we used the Eclipse-based Xilinx Software Development Kit and Putty as a terminal for the standard output. The logic is implemented in C and executed by the CPU low of the WARP.

SEEMOO PhySec (Lab RDS) 2017/18, February 2018, Darmstadt, Germany

As a first step we setup the WiFi 802.11 standard and the callback-function that are executed if a new frame is received or needs to transmitted.

```
wlan_mac_low_init(WARPNET_TYPE_80211_LOW);

// ...

wlan_mac_low_set_frame_rx_callback((void*)frame_receive);
wlan_mac_low_set_frame_tx_callback((void*)frame_transmit);
```

In order to check for incoming frames, the `wlan_mac_low_poll_frame_rx` function needs to be executed in a loop.

```
while(1) {
    wlan_mac_low_poll_frame_rx();
}
```

If the RX PHY received a new frame this function will call the frame_received function to handle the reception. Since we are about to implement a reactive jammer, it is of importance that the jamming signal is sent as soon as the currently received frame can be assigned to a certain transmitter. Therefore, we add the check for the jamming condition (MAC address of the target system) to the frame_received function. If the condition is "true" we create the jamming signal and transmit it using the frame_transmit function. Notice that the transmission is started, while still receiving the targeted signal. As a result the signal gets destroyed for other receivers. To ensure this happens we have to implement the transmission of our jamming signal right before the call to the blocking function `wlan_mac_dcf_hw_rx_finish`, since this function blocks the execution of the further code until the complete frame is received.

If we receive a frame that we want to jam, we need a valid WiFi frame that can be transmitted to jam the original signal. Since the frames we copied to the TX Packet Buffer of the WARP were apparently cleared automatically after their transmission, we copy a new one to the Buffer if the above mentioned condition applies. To create the MAC header we use the predefined mac_header_80211 structure and set the fields to the following values, where the `mac` variable contains our MAC address.

```
mac_header_80211 header;
header.frame_control_1 = MAC_FRAME_CTRL1_SUBTYPE_DATA;
header.frame_control_2 = MAC_FRAME_CTRL2_FLAG_FROM_DS;
header.duration_id = 0;
memcpy(header.address_1, mac, 6);
memcpy(header.address_2, mac, 6);
memcpy(header.address_3, mac, 6);
header.sequence_control = 0;
```

Beyond this, the WARP needs some further information for each frame with details about its transmission that has to be stored in the Packet Buffer. For one, there is a predefined structure `tx_frame_info` which we do not use for our jammer but have to consider when calculating the size of the final block to write into the buffer. Then follows a header containing information that is used to give the chip on the WARP details about handling the frame at the physical layer. This contains the transmit power and the antenna interface which should be used.

In order to know to which address we have to store our MAC header we first need to know the order in which the parts described above are expected to be in the buffer. First the chip looks for the `tx_frame_info`, then the physical and finally for the MAC header. Thus we need to sum up the size of the `tx_frame_info` structure and the physical layer header and add them to the address of the current packet buffer. To this address we can finally write the MAC header we created above.

```
memcpy((TX_PKT_BUF_TO_ADDR(pkt_buf) +
    PHY_TX_PKT_BUF_PHY_HDR_SIZE +
    sizeof(tx_frame_info)),
    &header,
    sizeof(header));
```

With this header finished, we still need to write the header for the TX PHY core. Since we want our jammer to be as effective as possible, we set the transmission power to the maximum and choose the only attached antenna at port A.

```
set_tx_power(pkt_buf, TX_POWER_MAX_DBM);
set_tx_ant_mode(pkt_buf, TX_ANTMODE_SISO_ANTA);
```

Being done with the preparation for the transmission of our jamming frame, we can finally tell the CPU to transmit our frame.

```
frame_transmit(pkt_buf,WLAN_PHY_RATE_BPSK12,
    sizeof(header), NULL);
```

The one thing still missing from our implementation is the analysis of the incoming frame with the check if we need to jam it. This has obviously to be done before the transmission of the jamming frame. To lose as little time as possible, we want our receiving function to wait just until the first target MAC address has been received. Since we know that the thirteenth byte is the beginning of the first MAC address and the addresses are six byte long, we tell our receiver to wait until the reception of the 19-th byte. In the lab we tried to do this using an empty `while` loop, which did not work in practice. We assume that the compiler removed the empty loop in an optimization. When we added an console output of an empty string, it worked as expected.

```
while(wlan_mac_get_last_byte_index() <= 19) {
    xil_printf("");
}
```

As soon as these first bytes are received we copy the contents of the RX packet buffer to a `mac_header_80211` structure variable. The information after the first MAC address is still not valid, but this does not pose a problem since we are only interested in this MAC address. For reading from the receive buffer we can proceed exactly like before with the transmit buffer because they are structured identically.

```
memcpy(&header, ((void *)(RX_PKT_BUF_TO_ADDR(rx_pkt_buf))
    + PHY_RX_PKT_BUF_PHY_HDR_SIZE
    + sizeof(rx_frame_info)), sizeof(header));
```

The only missing part is the actual comparison of the included MAC address and the one we want to jam. We can access the received address using our `header` structure variable and compare it to the `jam_me_mac` address, that contains the address we want to jam.

```
if (wlan_addr_eq(header.address_1, jam_me_mac)) {
```

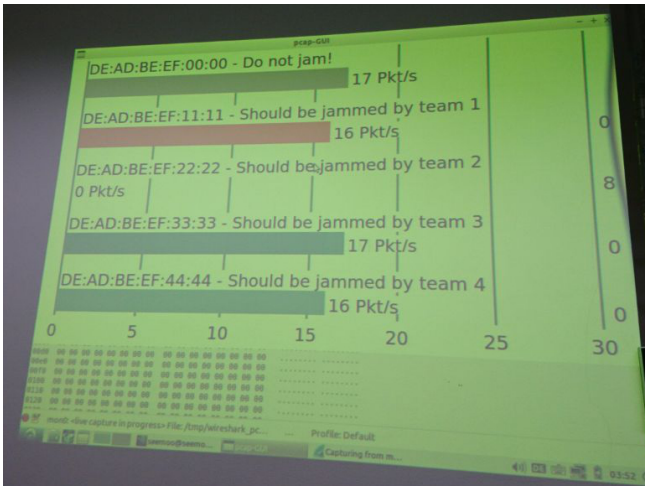

Figure 2: Jamming results using our jammer

In this if block follows the code we wrote to transmit our jamming signal described above.

4 EVALUATION

The address we wanted to jam in this lab was in our case `DE:AD:BE:EF:22:22`. Figure 2 shows the result when our jammer was turned on while none of the other groups were active. The bars show how many valid packets containing the corresponding MAC address were received in packets per second at the evaluation netbook. The bars for the other addresses stayed always close to 16-17 packets independent of the activity of our jammer. Just for our target the packet rate dropped to 0 packets per second while our jammer was active and regained the same value as the other receivers when we switched it off. This shows that our reactive jammer indeed works in practice, where he can jam all frames to a target MAC address while not noticeable affecting one of the other ones.

5 CONCLUSION AND TAKE-AWAY

In this lab we implemented our own reactive jammer. We started by looking at the Wireless Open-Access Research Platform, how it works, how we can use it to implement our jammer and which peculiarities we need to consider. Then we implemented the building of a WiFi frame with a valid MAC header and the additional informations the WARP requires. Finally we added an analysis of received frames that checks for certain MAC addresses in the headers and included the transmission of our prepared frame in the case of a match with the address we want to jam. Finally we tried our implementation using a WARP and could evaluate the success of the reactive jammer.

6 FUTURE WORK

For future work, this implementation of our reactive jammer might be expanded by additional features. For example it would be possible to offer different options for the frames the jammer should jam. Instead of jamming all frames to one certain receiver, one might jam only those using certain services or protocols, or combinations of these and the MAC addresses. To avoid emerging problems with this approach it would also be possible to extend this implementation to an acknowledging jammer, that fakes acknowledgement messages to the original sender. If a lower power consumption is important, another option is to look for improvements that improve the jammer in a way so that it does not have to transmit with full power all of the time without losing too much efficiency. One approach might be to simply try out lower power levels and then measure if the jamming is still successful, which could be done dynamically while jamming.