TransFi User Guide

Prerequisites:

Hardware:

1. AR9580 WiFi chipset: TransFi is implemented on the Atheros AR9580 WiFi chipset. In particular, we used The ETLEY Wireless N 5GHz 450Mbps 3*3MIMO WIFI CARD (https://us.amazon.com/NETELY-Wireless-Stations-PCIE-Card-Qualcomm-NET-N450B/dp/B07D8MRXRZ) in our implementation. Other WiFi card models have not been tested, and there may be some hardware configuration differences between models from different manufacturers.

2. WARP v3 SDR Boards: http://mangocomm.com/products/kits/warp-v3-kit/ with FMC-RF-2X245 Dual-Radio FMC Module (http://mangocomm.com/products/modules/fmc-rf-2x245/). As stated in the paper, in our implementation we use the WARP SDR as the custom WiFi receiver to collect data for evaluation purposes.

Software:

1. Atheros_CSI_tool: https://github.com/xieyaxiongfly/Atheros_CSI_tool_OpenWRT_src/wiki. This is a custom driver that allows direct packet load injection to the AR9580 WiFi chipset.

2. Lorcon: https://github.com/kismetwireless/lorcon. This program is used to inject the custom data payload into the Atheros wireless card. Since Lorcon is an old tool, there may be some compatibility issues. So please make sure it works after installation.

3. Ubuntu 14.04 with ATH9K driver (Other versions will not work)

4. Matlab R2017b

System Preparation:

1. (Optional) Modify the IEEE80211_MAX_FRAME_LEN & IEEE80211_MAX_DATA_LEN in ieee80211.h within Atheros_CSI_tool: https://github.com/xieyaxiongfly/Atheros-CSI-Tool/include/linux/ieee80211.h (Currently support up to 7000 Bytes)

2. Then Install the Atheros_CSI_tool following guide from here: https://github.com/xieyaxiongfly/Atheros_CSI_tool_OpenWRT_src/wiki

3. Install Lorcon via: https://github.com/kismetwireless/lorcon, follow the guide here http://blog.opensecurityresearch.com/2012/09/getting-started-with-lorcon.html to install.

4. Make sure all the installation are correct. Check the custom kernel version number.

5. Check the WiFi interface ID via ifconfig. Record the interface ID for further wireless card configuration

6. Use Emulation/Injection/Monitor_Channel_Set in command line to configure the AR9580 chip to be operating in monitor mode and set the wireless channel. The operating channel setup is at line 13 with channel index 40 (5200MHz), a 40Mhz channel that is set with HT40-. For example, in the current file, the channel is set with 40 HT40-, which is the 40MHz channel at channel 38 in this list: https://en.wikipedia.org/wiki/List_of_WLAN_channels. To specific the device to change the channel, replace wls4/wlp8s0 with your WiFi interface ID in line 13.

7. Set scrambler seed: Set npackets =1000 at line 46 in Emulation/Injection/scrambler_ini.c. Compile and run Emulation/Injection/scrambler_ini, and at the same time, run Emulation/Scrambler_calibration.m.

Scrambler_calibration.m outputs the number of additional frames need to be transmitted to set the seed to 1 with difference variable at line 255.

Change the npackets = difference in scrambler_ini.c. Recompile and run scrambler_ini to set the next scrambler seed at 1. (This should be done **ONLY ONCE** per restart of the PC.)
Note: I have prepared the predefined payload for scrambler initialization in Emulation/scrambler folder. You can inject those files for initialization with any emulation.

Use TransFi on the transmitter side:

1. Go to Emulation/Find_packet.m. Generate the target signal you want before line 280. And give the target signal to the target_signal variable at line 280.
2. Then run Emulation/Find_packet.m, it will compute the MAC payload and output to the injection program. The output MAC payload is in MAC_PayloadXXX.txt. The XXX is number from 001 to 127. I have pre-made MAC payloads in these files, but you can always use Find_packet.m to generate any other custom MAC payloads as you want.
3. The MAC payloads will be injected to the WiFi card via Emulation/Injection/injection_40M. You can configure the data rate with mcs_iter at line 180 of injection_40M.c (2Tx MIMO 64QAM with 5/6 coderate is 15. Full table is here: https://en.wikipedia.org/wiki/IEEE_802.11n-2009.). The total number of transmitted frames is set at ntimes at line 8, which is ntimes*127.
4. Configure the device you want to inject in via line 43 in Emulation/Injection/injection_40M.c with *interface = "Your Interface ID".
5. Compile the program with gcc -o Injection_40M Injection_40M.c -lorcon2.
6. Run the script with sudo ./Injection_40M -i **** (**** = WiFi card interface name) for MAC payload injection. (Please use sudo)

Receiver configuration:

1. In order to receive and see the transmitted emulation signal, please configure the WARP receiver with the corresponding target signal. The program used in the paper is in: Emulation/MISO_CustomReceiver.m, with respect to the MAC payloads being included in this code repository. Alternatively, you can write your own custom receiver program based on your target signal designs.

2. The Emulation/MISO_CustomReceiver.m configures the WARP to detect the emulated signal with custom signal. The program will detect the emulated signals transmitted from the AR9580 WiFi card. Configure your custom signal before line 517 and give your custom signal to custom_preamble_time variable. The detection threshold is set at LTS_CORR_THRESH. The detection results are shown in plotted figures. The general result is computed by counting the number of emulated frames that is being detected among all the figures. This program will also output SNR and bit sequence from the emulated signal.

Other files in the folder:
1. Emulation\wlanNonHTData_local.m is used to generate a target signal with 802.11g configuration.
2. Emulation\weight_bit.m is a subfunction used to generate the segmented trellis diagram.
3. Emulation\depuncture.m is a subfunction to remove X' bits.