



---

# Cross-technology Channel Estimation Tool User Guide

---

*Shuai Wang*      *Song Min Kim*  
shine.hitcs@gmail.com    songmin@kaist.ac.kr

Smart and Mobile Systems (Smile) Lab, KAIST, South Korea  
<https://sites.google.com/view/smilelab/>

**Verion 1.0**

March 12, 2021

# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
<b>2</b>	<b>Tool Installation</b>	<b>2</b>
2.1	Prerequisite . . . . .	2
2.2	Installation at WiFi Helper . . . . .	2
2.3	Installation at WiFi Receiver . . . . .	3
2.4	Details about This Implementation . . . . .	3
2.5	Setting at ZigBee Device . . . . .	3
<b>3</b>	<b>WiFi Fragments Injection</b>	<b>4</b>
3.1	Some Useful Settings . . . . .	6
3.1.1	Find Scrambler Seed . . . . .	6
3.1.2	Set Transmission Power . . . . .	6
3.1.3	Set Fragmentation Threshold . . . . .	7
<b>4</b>	<b>Cross-technology Channel Estimation</b>	<b>8</b>
<b>5</b>	<b>Cite This Tool</b>	<b>10</b>

# Chapter 1

## Introduction

This tool implements the design in our paper [1]. This tool consists of two parts: (i), fragments injection, which injects the customized WiFi fragments with precise timing control. (ii), channel estimation, which estimates the cross-technology channel from the channel state information (CSI) collected from the injected fragments. As the design describes, two WiFi fragments are transmitted from a WiFi helper (a WiFi device) to X-MIMO (a WiFi receiver). To achieve this, this tool implements the WiFi helper, which injects WiFi fragments with precise timing control, on a TP-link wireless router running OpenWrt system. The X-MIMO simply reuses the existing Atheros CSI tool [2] to extract the CSIs from two WiFi fragments. Then, the cross-technology channel is estimated from the CSIs by applying the second part of this tool, i.e. channel estimation. Since X-MIMO only needs to extract CSIs from received packets, X-MIMO could be a wireless router or an Atheros WNIC as long as they support Atheros CSI tool. Hence, as illustrated in Figure 1.1, there are two types of WiFi receivers supported by this tool.

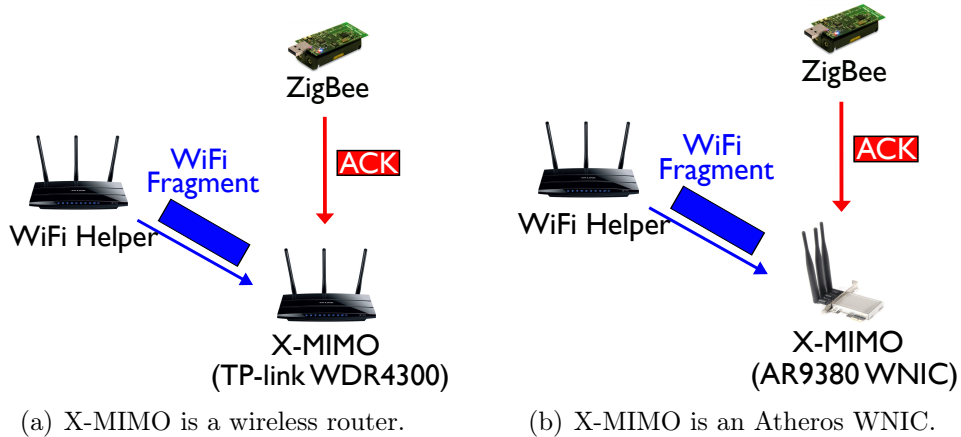


Figure 1.1: Two types of WiFi receiver supported by this tool.

# Chapter 2

## Tool Installation

### 2.1 Prerequisite

**Hardware:** This tool supports the TP-link WDR 4300 wireless router as the WiFi helper and the WiFi receiver. The Atheros WNIC, which supports the Atheros CSI tool, can also become the WiFi receiver. The ZigBee device used in this tool is TelosB mote.

**Software:** The Atheros CSI tool should be installed before using this tool. TinyOS is also needed to enable the Hardware ACK at TelosB mote.

### 2.2 Installation at WiFi Helper

The WiFi helper supported by this tool should be a TP-link WDR 4300 wireless router. The installation at the WiFi helper is basically installing our customized OpenWrt firmware (xmimo.bin).

**Step 1**, upload the xmimo.bin to the TP-link WDR 4300 wireless router via command scp:

---

```
scp xmimo.bin root@192.168.1.1:/tmp/
```

---

**Step 2**, log in the TP-link WDR 4300 wireless router (OpenWrt):

---

```
ssh root@192.168.1.1
```

---

**Step 3**, restore the firmware xmimo.bin:

---

```
mtd -r write /tmp/xmimo.bin firmware
```

---

Then, you would see the system is unlocking the firmware and the installation would be done within a few seconds:

```
root@OpenWrt:/tmp# mtd -r write ./xmimo.bin firmware
Unlocking firmware ...

Writing from ./xmimo.bin to firmware ... [w]
```

## 2.3 Installation at WiFi Receiver

If a TP-link WDR 4300 wireless router is chosen to be the WiFi receiver, please follow the same steps in Section 2.2 to install the firmware. However, if an Atheros WNIC (e.g., AR9380 WNIC) is chosen to be the WiFi receiver, please replace the `Atheros-CSI-Tool-master\drivers\net\wireless\ath\ath9k\ar9003_phy.c` with the `ar9003_phy.c` provided in this repository. After compiling and installing the Atheros CSI tool again according to the instructions [2], the installation is done.

## 2.4 Details about This Implementation

The development of this tool is based on the Atheros CSI tool [2]. In addition to the functionalities of the Atheros CSI tool, this tool

- supports large MTU ( $\leq 7000$  Bytes). This is achieved by adding a new patch to modify the `IEEE80211_MAX_DATA_LEN` and `IEEE80211_MAX_FRAME_LEN` in `Atheros_CSI_tool_OpenWRT_src\build_dir\target-mips_24kc_musl\linux-ar71xx_generic\linux-4.9.111\include\linux\ieee80211.h`.
- tracks scrambler seed. This is needed for emulating ZigBee packets successfully in each WiFi fragment 1 (details could be found in [1]).
- sets the transmission rate to be 26 Mbps (MCS 3).
- modifies the center frequency of WiFi channel 11 to be 2.461GHz by setting `freq` in `Atheros-CSI-Tool-master\drivers\net\wireless\ath\ath9k\ar9003_phy.c`. We assume that the ZigBee device is working on ZigBee channel 23 (2.465GHz). To make the design transparent to the ZigBee device, the center frequency of WiFi channel 11 to be 2.461GHz such that we can estimate the channel from the ZigBee devices on ZigBee channel 23 to the WiFi receiver.

## 2.5 Setting at ZigBee Device

The design in [1] requires the hardware ACK to be enabled at the ZigBee device. This is achieved by adding the following code in the `MakeFile` of your TinyOS code:

---

```
CFLAGS += -CC2420_HW_ACKNOWLEDGEMENTS
```

---

In addition, channel of the ZigBee device should be channel 23 (2.465GHz).

# Chapter 3

## WiFi Fragments Injection

After installing the tool, we need to inject the customized WiFi fragments at the WiFi helper. Before showing the details of injection, we recorded two videos to demonstrate how the tool is operated if the WiFi receiver is a TP-link wireless router (<https://youtu.be/zoNW761Damo>) and an Atheros WNIC (<https://youtu.be/HPsjK79-gDY>).

The WiFi fragments injection is performed at the WiFi helper. After logging into the helper, we would see the following setting:

```
root@OpenWrt:~# iwconfig
eth0.2      no wireless extensions.

br-lan      no wireless extensions.

lo          no wireless extensions.

wlan0       IEEE 802.11  Mode:Master  Tx-Power=17 dBm
            RTS thr:off   Fragment thr=1896 B
            Power Management:off

eth0.1      no wireless extensions.

eth0        no wireless extensions.

wlan1       IEEE 802.11  ESSID:off/any
            Mode:Managed  Access Point: Not-Associated   Tx-Power=19 dBm
            RTS thr:off   Fragment thr:off
            Encryption key:off
            Power Management:off
```

wlan0 is the WiFi card working on 2.4GHz. The fragment threshold is set to be 1896 Bytes, which is specified according to the precise timing control in [1].

**Step 1**, run the following command to create a new interface wlan0mon:

---

```
injector -i wlan0 -c 11HT20 -l 1910 -t 0 -e 63 -r 5
```

---

This command sets the channel to be WiFi channel 11 (2.461GHz in this tool), ‘-l’ specifies the length of the WiFi packet to be 1910 Bytes. ‘-e’ specifies the number of packets in each

round. ‘-r’ specifies how many rounds for the injection. Now, a new interface wlan0mon is created:

```
root@OpenWrt:~# iwconfig
eth0.2    no wireless extensions.

br-lan    no wireless extensions.

wlan0mon  IEEE 802.11  Mode:Monitor  Tx-Power=17 dBm
          RTS thr:off   Fragment thr=1896 B
          Power Management:off

lo        no wireless extensions.

wlan0     IEEE 802.11  Mode:Master  Tx-Power=17 dBm
          RTS thr:off   Fragment thr=1896 B
          Power Management:off

eth0.1    no wireless extensions.

eth0      no wireless extensions.

wlan1     IEEE 802.11  ESSID:off/any
          Mode:Managed Access Point: Not-Associated   Tx-Power=19 dBm
          RTS thr:off   Fragment thr:off
          Encryption key:off
          Power Management:off
```

However, the injection cannot be executed because the default MTU (1500 Bytes) does not allow us to inject 1910 Bytes packets. Then, we would change the MTU in step 2.

**Step 2**, set a large MTU:

---

```
ifconfig wlan0mon mtu 5000
```

---

Now, we check whether the MTU is set correctly or not via ifconfig:

```
wlan0mon  Link encap:UNSPEC  HWaddr 14-CC-20-B9-42-9B-00-00-00-00-00-00
-00
          UP BROADCAST RUNNING PROMISC MULTICAST MTU:5000 Metric:1
          RX packets:460 errors:0 dropped:0 overruns:0 frame:0
          TX packets:0 errors:0 dropped:0 overruns:0 carrier:0
          collisions:0 txqueuelen:1000
          RX bytes:126783 (123.8 KiB) TX bytes:0 (0.0 B)
```

**Step 3**, set the receiver’s MAC address in helper.py. The line 167 in helper.py is for specifying the receiver’s MAC address: The first 4 elements specify the type of the packet

```
167 packets[:10]=[128,0,0,0,160,243,193,255,50,89]
```

(e.g. Beacon). The subsequent 6 elements represent the MAC address of the receiver (X-MIMO device). For example, the following MAC address is A0:F3:B1:FF:42:59. Then, the CSI in the injected WiFi fragments would only be extracted by the specified WiFi receiver (A0:F3:B1:FF:42:59). You need to change the last 6 elements in this array to be your receiver’s address and run:

---

```
python helper.py
```

---

**Step 4**, upload all the generated files in `gen_frag_packets` to the wireless router:

---

```
scp -r ./gen_frag_packets root@192.168.1.1:~
```

---

**Step 5**, inject the fragments:

---

```
injector -i wlan0 -c 11HT20 -l 1910 -t 0 -e 63 -r 5
```

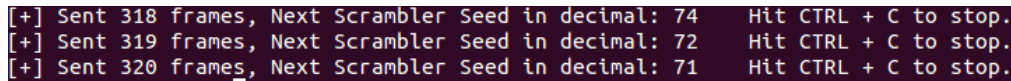
---

Meanwhile, the WiFi receiver should execute `recvCSI` to store the CSIs extracted from two WiFi fragments. After executing this command, there would be  $63 \times 5 = 315$  packets injected and these packets would trigger 315 ZigBee ACKs consequently.

## 3.1 Some Useful Settings

### 3.1.1 Find Scrambler Seed

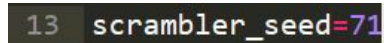
The scrambler seed is a critical parameter to emulate ZigBee signal. [3] demonstrates that there are many commodity WiFi devices that use incremental scrambler seed, which increments by one per WiFi packet. We discover that the initial scrambler seed on the TP-link WDR 4300 router is always 0x47. Thus, tracking the scrambler seed is possible and implemented in our tool. To make tracking seed easier, the scrambler seed is printed when a WiFi packet is injected:



```
[+] Sent 318 frames, Next Scrambler Seed in decimal: 74    Hit CTRL + C to stop.  
[+] Sent 319 frames, Next Scrambler Seed in decimal: 72    Hit CTRL + C to stop.  
[+] Sent 320 frames, Next Scrambler Seed in decimal: 71    Hit CTRL + C to stop.
```

Figure 3.1: The scrambler seed is printed as the WiFi packets are injected.

As Figure 3.1 illustrates, the scrambler seed is printed in decimal. Then, we just need to set `scrambler_seed` to be the scrambler seed in `helper.py`:



```
13 scrambler_seed=71
```

After we run `helper.py`, the correct WiFi payloads for emulating ZigBee signals are generated.

### 3.1.2 Set Transmission Power

---

```
iwconfig wlan0 txpower 13
```

---

This command sets transmission power to be 13 dBm:



```

root@OpenWrt:~# iwconfig
eth0.2    no wireless extensions.

br-lan    no wireless extensions.

wlan0mon  IEEE 802.11  Mode:Monitor  Tx-Power=17 dBm
          RTS thr:off   Fragment thr=1896 B
          Power Management:off

lo        no wireless extensions.

wlan0     IEEE 802.11  Mode:Master  Tx-Power=17 dBm
          RTS thr:off   Fragment thr=1896 B
          Power Management:off

eth0.1    no wireless extensions.

eth0      no wireless extensions.

wlan1     IEEE 802.11  ESSID:off/any
          Mode:Managed  Access Point: Not-Associated   Tx-Power=19 dBm
          RTS thr:off   Fragment thr:off
          Encryption key:off
          Power Management:off

root@OpenWrt:~# iwconfig wlan0 txpower 13
root@OpenWrt:~# iwconfig
eth0.2    no wireless extensions.

br-lan    no wireless extensions.

wlan0mon  IEEE 802.11  Mode:Monitor  Tx-Power=13 dBm
          RTS thr:off   Fragment thr=1896 B
          Power Management:off

lo        no wireless extensions.

wlan0     IEEE 802.11  Mode:Master  Tx-Power=13 dBm
          RTS thr:off   Fragment thr=1896 B
          Power Management:off

eth0.1    no wireless extensions.

eth0      no wireless extensions.

wlan1     IEEE 802.11  ESSID:off/any
          Mode:Managed  Access Point: Not-Associated   Tx-Power=19 dBm
          RTS thr:off   Fragment thr:off
          Encryption key:off
          Power Management:off

```

Figure 3.2: A comparison of transmission power before and after executing the above command.

### 3.1.3 Set Fragmentation Threshold

---

```
iw phy phy0 set frag XXXX
```

---

XXXX is the new fragmentation threshold in Bytes.

## Chapter 4

# Cross-technology Channel Estimation

Assuming the CSIs are stored in a file ‘data\_csi.txt’, we run the Matlab code ‘XMIMO\_CSI\_read.m’ to convert the CSI data into imaginary and real parts, which would be further processed by the python code to calculate the cross-technology channel. Here is an example of CSI extracted from WiFi fragments (overlapping with ZigBee ACK):

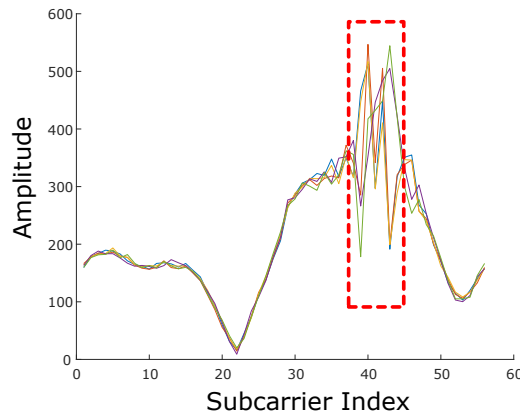


Figure 4.1: The amplitude of CSI extracted from the WiFi fragment. The red box highlights the subcarriers overlapped the ZigBee ACK.

Finally, we run the following command under the same folder containing the CSI files:

---

```
python zigbee_channel_calculation.py
```

---

Then, we would see the reconstructed overlapped ZigBee signal and the corresponding cross-technology channel:

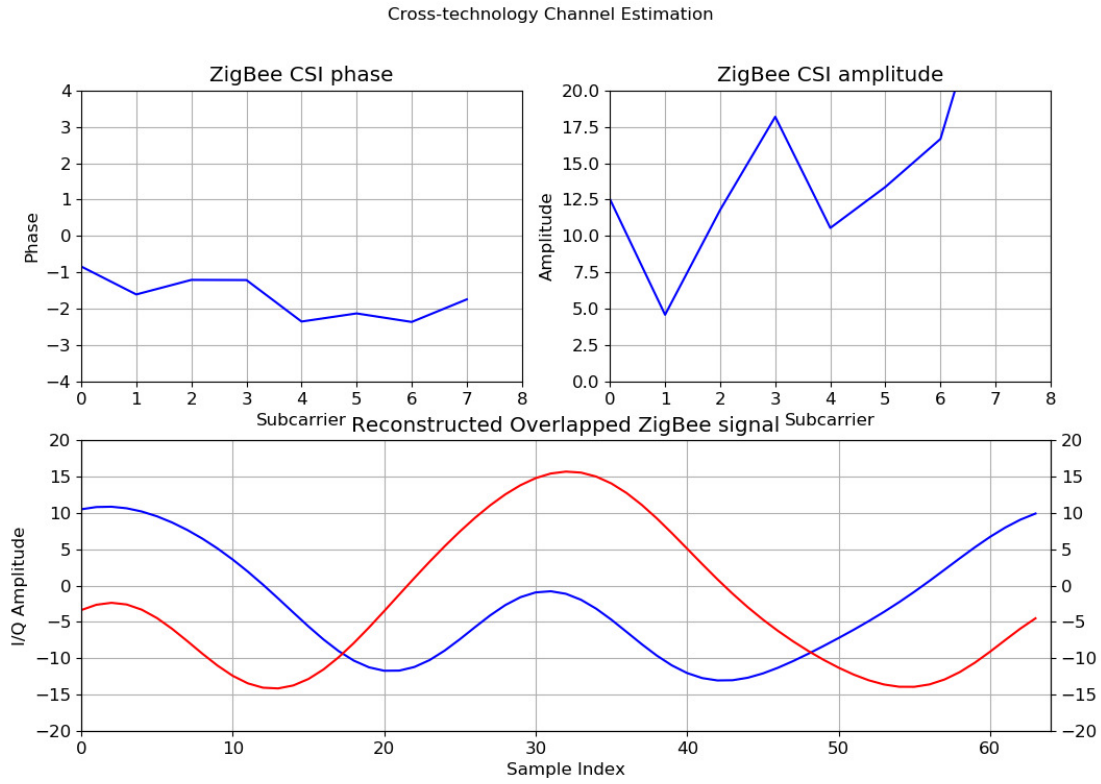


Figure 4.2: Illustration of the result of cross-technology channel estimation. The bottom figure shows the reconstructed overlapped ZigBee signal ( $3.2\mu s$ ). The top figures show the phase and amplitude of the estimated cross-technology channel.

# Chapter 5

## Cite This Tool

---

```
@inproceedings{wang2020x,  
  title={X-MIMO: cross-technology multi-user MIMO},  
  author={Wang, Shuai and Jeong, Woojae and Jung, Jinhwan and Kim, Song Min},  
  booktitle={Proceedings of the 18th Conference on Embedded Networked Sensor  
    Systems},  
  pages={218--231},  
  year={2020}  
}
```

---

- [1] Shuai Wang, Woojae Jeong, Jinhwan Jung **and** Song Min Kim. “X-MIMO: cross-technology multi-user MIMO”. **in:** *Proceedings of the 18th Conference on Embedded Networked Sensor Systems*. 2020, **pages** 218–231 (pages 1, 3, 4).
- [2] Yaxiong Xie, Zhenjiang Li **and** Mo Li. “Precise Power Delay Profiling with Commodity WiFi”. **in:** *Proceedings of the 21st Annual International Conference on Mobile Computing and Networking*. MobiCom ’15. Paris, France: ACM, 2015, **pages** 53–64. ISBN: 978-1-4503-3619-2. DOI: [10.1145/2789168.2790124](https://doi.org/10.1145/2789168.2790124). URL: <http://doi.acm.org/10.1145/2789168.2790124> (pages 1, 3).
- [3] Vikram Iyer, Vamsi Talla, Bryce Kellogg, Shyamnath Gollakota **and** Joshua Smith. “Inter-technology backscatter: Towards internet connectivity for implanted devices”. **in:** *Proceedings of the 2016 ACM SIGCOMM Conference*. 2016, **pages** 356–369 (page 6).