# General Overview

The SAVI-WARP Project is a key step in creating a protocol-based wireless virtualization test bed. Currently, there are very few means of obtaining both the software and hardware control necessary to achieve true protocol-based virtualization. Work on wireless virtualization has mostly centered around a flow-based approach (e.g. Aurora) due of the open-source nature of many software defined networks (SDN). The SAVI-WARP project seeks to expand on the flow-based approach by combining certain SDN applications, such as the UNIX-based hostapd daemon, with an open-access hardware solution called the Wireless Open Access Research Platform (WARP) board. By replacing a proprietary, closed-source, network interface card with a fully open radio board, true protocol-based virtualization can be achieved.

# Approaches

In order to combine existing flow-based virtualization functionality with the WARP board, it is first necessary to connect the WARP board with a Linux box in such a way that the WARP board can be seen as a regular NIC. There are two main approaches to do this. The first approach involves the creation a Linux network driver. This new network driver would need to have similar functionality to existing open-source Linux wlan drivers (such as ath5k). The main difference would be the method of communication. Regular wlan drivers communicate via either a PCI slot or USB port, allowing direct manipulation of physical device parameters. Since the main method of communicating from Linux to the WARP board is through the Ethernet port, the task of developing a wlan driver similar to existing ones becomes much more challenging.

The second approach involves using a previously existing hardware simulation driver (called the mac80211\_hwsim driver) as a middleman for communicating between Linux and the WARP board. From the software side, hostapd would operate normally, performing its functions on the virtual interface created by the simulation driver. A packet crafter tool is then used on the virtual interfaces in order to intercept packets generated by hostapd. These packets are then forwarded over Ethernet directly to the WARP board. Packets received over Ethernet from the WARP board are processed similarly. Using the packet crafting tool, all Ethernet packets originating from the WARP board are intercepted, re-crafted to meet hostapd's requirements, and then forwarded back to the virtual interface created by the simulation driver. From the hardware side, the WARP board operates as usual, sending and receiving packets on both Ethernet and Radio interfaces. The main tools required for this approach are a user-space daemon for wireless access points (such as hostapd or netsh), a packet crafting tool capable of packet sniffing (such as Scapy or Libtins), as well as the WARP development board. Below is a general overview of how the different components interact:

# *Figure 1 - SAVI-WARP Project Components and Setup*

# User-Space Daemon

Depending on the operating system, there are several user-space daemons that can be used to facilitate the creation of a wireless access point using a wireless NIC. For the SAVI-WARP project, we will be focusing on the hostapd user-space daemon for the Linux environment. Hostapd is an extremely powerful daemon that operates on supplied parameters from a configuration file. While there are configuration parameters available for nearly all aspects of an access point, only a small subset of these commands are required for testing purposes during development. Below is a sample configuration file used for hostapd:

interface=wlan0

driver=nl80211

ssid=SaviNet-1

hw\_mode=g

channel=6

macaddr\_acl=0

ignore\_broadcast\_ssid=0

Of interest to note are the driver and channel parameters. The driver being used is the Linux nl802.11 header. This header uses netlink to communicate between user-space and kernel-space, facilitating communication between a hardware (or simulated hardware in our case) NIC and hostapd (a user-space program). The channel is relevant for development and testing purposes, as the simulated hardware driver will only respond to packets sent over the same channel.

Hostapd may be started in daemon mode with the following command:

hostapd -B <path to file>

It is also possible to print debug messages with hostapd, resulting in all actions to be printed to the screen (e.g. when an association request is received or an association response is sent). The command to start hostapd in debug mode is as follows:

hostapd -dd <path to file>

When hostapd is started, the simulation driver creates a second interface for every SSID defined in hostapd. This interface serves as a virtual monitor interface (able to see management frames) and is, by default, named mon.wlan<interface number>.

# Outgoing Communications (Linux to WARP)

Outgoing communications from Linux to the WARP board include the following:

- All beacon frames (multiple SSIDs will result in a periodic beacon frame for each SSID)

- All other relevant management frames (Probe Response, Association Response, etc.)

- All data frames from Linux to be send to a client (through the WARP radio)

Note that due to the strict timing requirements of control frames, it is not possible to handle control frames from Linux. These frames are therefore handled on the WARP board directly.

The general algorithm for communications from Linux to the WARP board is shown below, note that this algorithm is independent of which tool or library is used. Currently, this algorithm is being implemented with Scapy.



***Figure 2 - Linux to WARP Algorithm Flow Chart***

All packets that are generated by hostapd have a RadioTap header. This header, which sits just above the Dot11 frames, provides additional information about each packet being sent (e.g. the Frame Check Sequence). Since the 802.11 reference design on the WARP board does not handle RadioTap headers, it is necessary to strip this information from each packet. The stripped packet is then wrapped with both a WARPControl header (more information in WARP section) as well as an Ethernet frame before sending.

A potential issue may arise with the speed of sniffing, crafting, and sending. By default, hostapd generates 10 beacon frames per second for every SSID, in addition to any other packets that need to be sent to the WARP board. If the rate of packets being generated exceeds the maximum rate the packets can be sent out, the sending queue will overflow, resulting in packets being lost of dropped.

# Incoming Communications (WARP to Linux)

Incoming communications from the WARP board to Linux include the following:

- All management frames sent from a device connected to the WARP radio (e.g. Probe Request)

- All data frames sent from a device connected to the WARP radio

The general algorithm for communications from the WARP board to Linux is shown below, note that this algorithm is independent of which tool or library is used. Currently, this algorithm is being implemented with Scapy.



***Figure 3 - WARP to Linux Algorithm Flow Chart***

The first step after receiving a packet over Ethernet from the WARP board is to strip the Ethernet frame. It is then necessary to cast the raw data to a WARPControl header, since the WARPControl header is a not a standardized protocol. If the casting is successful and the packet destination is valid (i.e. not the WARP board), then the WARPControl header is stripped and a RadioTap header is added. Finally, the newly crafted packet is sent to the monitor interface corresponding to the virtual interface, where hostapd will process the packet and resend relevant information to the virtual interface.

Currently, there is a slight issue with the crafting of RadioTap headers. Hostpad will only process packets that have the correct RadioTap header information and this information seems to differ depending on which type of management frame is being sent.

# WARP Board

The development code on the WARP board is based on the open source 802.11 reference design from Mango Communications. The reference design provides source code in both C and HDL for a basic implementation of wireless access points. Currently, modification work on the WARP board is confined to the mac\_high module. This work involves rewriting the logic of packet processing and transmission originating from both the Ethernet port and the radio. Since the majority of the high level processing features are performed on a Linux Machine with hostapd, most functions in the reference design are rendered obsolete and will need to be bypassed.

One of the key features of the mac\_high module is the ability to define a WARPControl header. This header, composed of one or many user defined fields, will be instrumental in limiting the development on the WARP board to the mac\_high module (e.g. in order to support multiple MAC addresses for multiple SSID support, we can refrain from modifying lower level modules by defining an SSID field in the WARPControl header, essentially "tricking" the WARP board to differentiate between devices connected on multiple SSIDs). Currently, the header used by the reference design is shown below:



# Other Resources

Hostapd Documentation:

http://wireless.kernel.org/en/users/Documentation/hostapd

MAC 802.11 HWSIM driver Documentation:

http://wireless.kernel.org/en/users/Drivers/mac80211\_hwsim

Scapy Documentation:

http://www.secdev.org/projects/scapy/doc/

Libtins Link (Downloads + Tutorials + API):

http://libtins.github.io/

WARP 802.11 Reference Design:

http://warpproject.org/trac/wiki/802.11

SAVI-WARP Repository:

Request GitHub private repository access from Kevin Han (username: iamtheone188)