LTE-Wifi Controller – Station Management

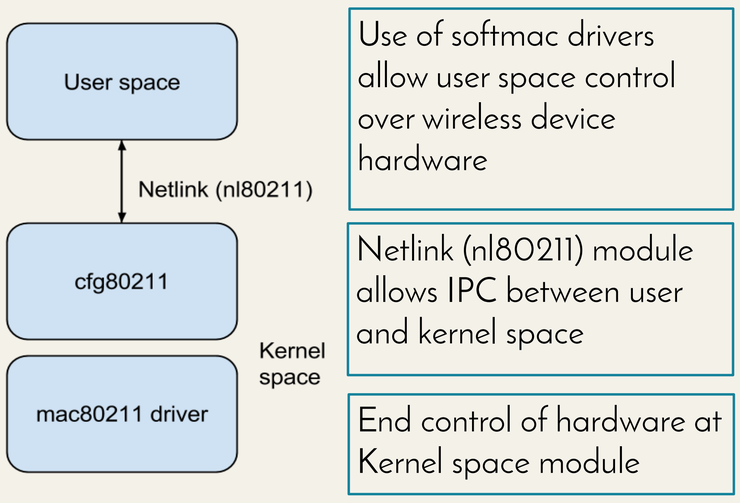
Guide: Prof. Abhay Karandikar

Pravesh Kochar
Electrical Engineering - Indian Institute of Technology, Bombay

Introduction

The Access Controller (AC) requires information about the connected stations to manage new requests and balance the overall load on a single Access Point (AP). We wish to fetch information of connected stations/clients from a Wireless Termination Point (WTP), and deliver it to a centralized management server for monitoring, control and management of the network. Depending on the requirement of the CAPWAP operations one might need to either view all the stations connected to the particular AP, view details of only one particular station connected to the AP identifying the station by its MAC address or delete the connection between a particular station and AP again identifying it by its MAC address.

Framework



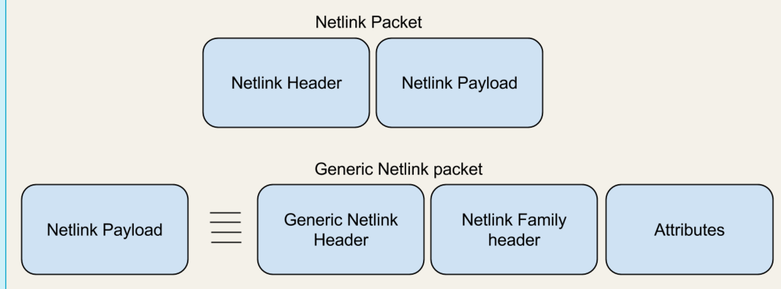
Demonstration Parameters

Environment: AP that is sending out beacons continuously (test bench on laptop). This AP is connected to Stations. The stations are using the internet via AP and it is managing all these stations.

Inactive time	452 ms
RX bytes	56953
RX packets	368
TX bytes	80120
TX packets	237
TX retries	4
TX failed	0
Signal	22.2 dBm
Signal average	22.2 dBm
TX bit-rate	72.2 MBit/s MCS 7
RX bit-rate	12.0 MBit/s
Authorized	Yes
Authenticated	Yes
Preamble	short
WMM/WME	yes
MFP	no
TDLS peer	no

Packet Structure

Netlink family has many sub-types depending on the kind of IPC that is being acquired. In our case the specific sub-type we use is 'genl' netlink. The sub-type has its own packet structure that is put in the payload of the original netlink packet. The attributes are concatenated one after the other.



Conclusion

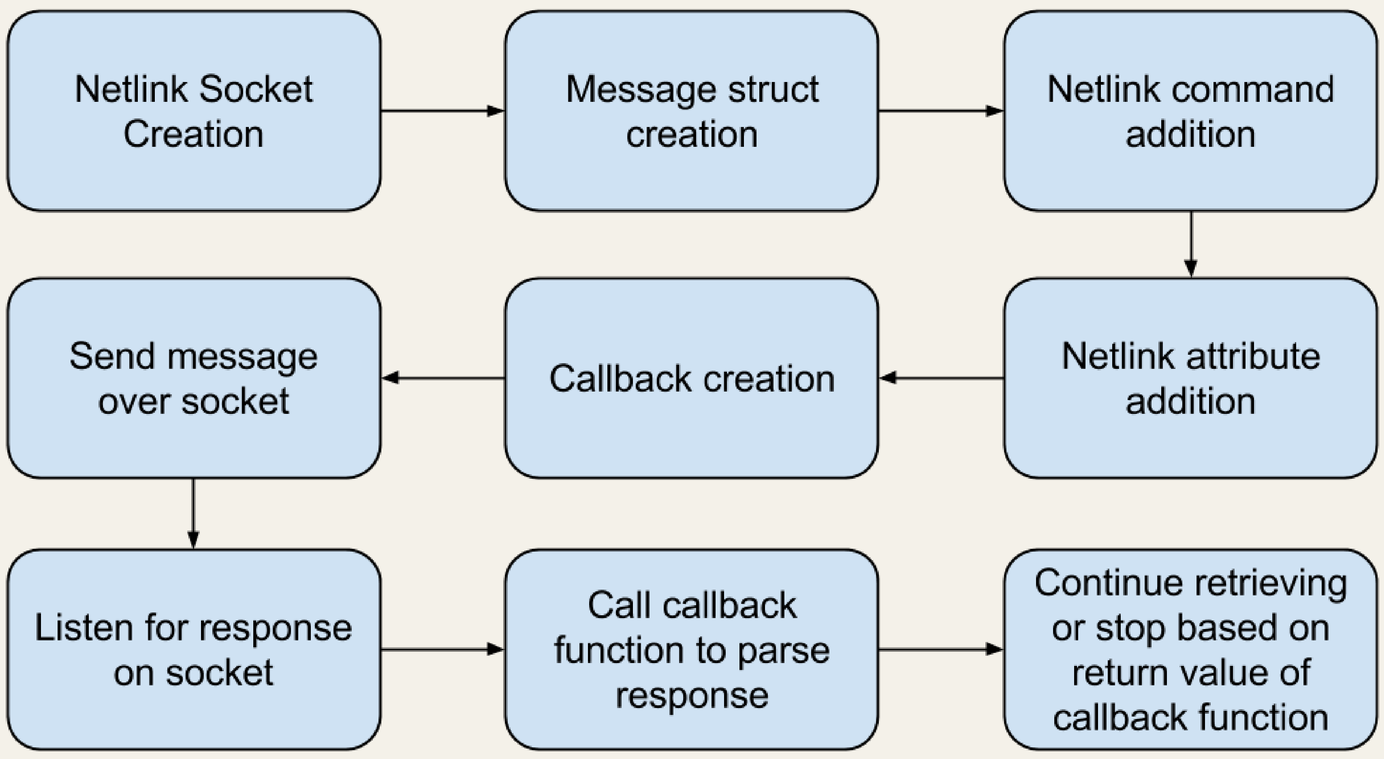
•**Inter Process Communication (IPC):** Simple code snippets available on the internet connecting a server and a client were examined. Familiarity with PIDs, sockets and port terminologies utliized to understand the advanced implementations focusing on kernel userspace communication.

•**Driver operations in Linux:** A broad insight into the kernel space programs was achieved. Differences in functioning of softmac and hardmac was exploited

•**Netlink Sockets:** With the ioctl paradigm being deprecated in all modern linux wireless driver interaction utilities, a learning of the netlink sockets was a must. A deep dive into the primary netlink functions from the libnl and lib-genl libraries was undertaken. This would be utilized in furthering many other wireless module related projects since it is the current development paradigm.

•**Object programming:** The work involved a lot of nested function calls making use of pointers to structs mandatory. Memory management and custom requirements of the project made it essential to create and use structures. A fair amount of work was in building structures from scratch, allocating memory space, modifying their variable values, creating arrays of structs by pointer manipulations and eventually cleaning them up.

Process Flow Diagram



Future Scope

•Retrieval mechanism to be implemented for getting the chain level signal strength in a PPDU.

•The memory management for the current case is static. The idea about dynamic memory would be to provide a framework where in APs with few 10s of stations and those with few 100s of them could both work on a single platform without worrying about memory optimization.

•Currently there is no way to directly access the packets that are coming in from the stations to the WTP. If we could achieve this, it would be a matter of just forwarding these packets to the AC enabling the ACs direct control over applications being run over the stations.

•The WTP side requires a mechanism to control its frequency and power parameters which are not controlled through netlink parameters. Since we are using a hostapd framework to create AP mode on the wireless device, changes in frequency and power require a restart of the entire hostapd system. Need to create a hostapd independent system

Netlink Internals

•Socket family in the Linux kernel interface that is used for inter-process communication (IPC)

•Utilizing the netlink socket to communicate with mac80211 based wireless drivers in the kernel space from functions written in the user space

•Like any regular inter process socket, we create the netlink socket initialized with its AF_NETLINK socket family

•The initialization process ensures that we have set the family id to the one corresponding to nl80211 so that the commands are recognized properly

Callbacks

•Callbacks are the single most important functions while reading the response from the driver in the kernel space

•Callbacks are of different types (ex: ack_handler, finish_handler etc) depending on functionality

•Inside the callback, netlink response messages are parsed and stored in custom structures created in the user space to utilize them further

•Current callback that is undertaking station management, parses the response to retrieve all station attributes in an array

Roll: 110070007 Email: praveshkochar@iitb.ac.in