MadWifi Wireless Network Driver

Jin Xie
OBJECTIVES

- Improve the throughput of wireless network by using partial packets
  - Current 802.11 networks operate in an all-or-nothing mode
  - It wastes the air time to retransmit the entire packet if there are just several bytes errors in it
DEVICES

- CISCO AIRONET 802.11a/b/g wireless adapter which use the Atheros 5212 Chipset
**MadWifi Driver**

- MadWifi is one of the most advanced WLAN drivers available for Linux today. The driver itself is open source but depends on the proprietary Hardware Abstraction Layer (HAL) that is available in binary form only.

- The driver is for Atheros-based wireless LAN devices, the Atheros driver depends on a device-independent implementation of the 802.11 protocols that originated in the BSD community.
**MadWiFi Architecture**

- The driver is broken into multiple modules

- **wlan**
  - More generic code contains 802.11 state machine, protocol support, etc

- **ath_pci**
  - Callbacks for the wlan module and contains Atheros_specific device support, access HW through ath_hal

- **ath_rate**
  - Apply the bit-rate selection algorithm

- **ath_hal**
  - Contains the hardware access layer, manages much of the chip-specific operation of the driver
### Modes in MadWiFi

- Access Point
- Station
- Ad hoc
- Ahdemo
- Monitor

This project will run on the monitor mode, which will pass all packets to the driver regardless of whether they pass the crc test.
Packet Processing

- Packet Receiving

 ath_intr -> ath_rx_tasklet -> ieee80211_input -> ieee80211_deliver_data -> netif_rx

 dev_queue_xmit
**Packet Processing**

- Packet Transmitting

After the HAL successfully transmitted a packet:

- **ieee80211_hardstart** → **ath_hardstart** → **ath_tx_start** → **ath_tx_txqaddbuf** → **ath_hal_txstart**

- **ath_intr** → **ath_tx_tasklet** → **ath_tx_processq** → **ath_tx_capture** → **ieee80211_input_monitor**
IMPLEMENTATION

Upper Layer Buffer

Our Buffer

ath_buf

The new protocol
**The Protocol**

- When packets are received correctly, just like the original 802.11 protocol use acknowledgments to synchronizing sender and receiver.
- When packets are received with errors, send asynchronous feedback in batches:
  - Waits until either it receives 8 more packets, or a timeout is reached.
  - Feedback contains information about multiple recent erasures and partial packets.
- The sender will send parity checking bytes either combined with next packet after receiving the feedback.
CURRENT PROGRESS

- Implemented the new protocol in the extra layer, tested it. (still has lower throughput than the original driver in some cases)
Tasks to Do

- Improve the current protocol
- Adding a new bit-selection algorithm which is based on the current sample-rate