This document will continuously be updated.

1. Overview

The A-DCF module (device driver) is based on ath9k. We extend the ath9k module to implement A-DCF and O-DCF. For details on ath9k, refer to http://wireless.kernel.org/en/users/Drivers/ath9k.

2. Building the driver

First, your kernel version should be 2.6.32.

Decompress adcf-driver-0.1.tar.bz2 by typing the following:

tar xvjf adcf-driver-0.1.tar.bz2

Change into the directory by typing the following:

cd compat-wireless-2.6.32.16-adcf-0.1

Build by typing the following:

./scripts/driver-select ath9k
make && make install && make unload

To revert, type the following:

make uninstall

3. Loading the driver

Change into the directory by typing the following:

cd adcf_scripts
Run the script by typing the following:

```
./setup_adhoc.sh
```

To unload the driver, type the following:

```
rmmod ath9k
```

4. Script and module details

You can change the module configuration by changing values for appropriate variables in `setup_adhoc.sh` script. To use 802.11 DCF, set `MAC` to 0. To use O-DCF, set `MAC` to 1. To use A-DCF, set `MAC` to 2.

You can get more information on module parameters by typing the following:

```
modinfo ath9k
```

5. Implementation details

We implement A-DCF and O-DCF by extending the `ath9k` module. You will find source codes for A-DCF and O-DCF in `driver/net/wireless/ath/ath9k/` directory. Main procedures are written in `adcf-*.*` files.

In the original `ath9k` module, the main procedures to transmit frames are as follows:
In our modified module, the above procedures are split into two flows as follows:

```
ath9k_tx
  ↓
ath_tx_start
  ↓
ath_start_dma
  ↓
ath_tx_txqaddbuf
```

In the first flow, frames are just inserted into their corresponding CQs. Actual transmissions are started by the second flow, which is ignited when TX interrupts occur or a frame is injected into an empty MAQ.

```
adcftx
  ↓
adcfcinsertcq
```

```
adcftx_tasklet
  ↓
adcftx dcf or
adcftx btb
  ↓
adcftx txq
  ↓
adcftx start
```

Source rate control is performed by the following function call:

```
adcfinjecttimer
  ↓
adcfcremovecq and
adcfcinsertmaq
```
A. Major data structures

Here, we describe main fields of major data structures and their related functions.

```c
struct adcf_data{
    struct sk_buff *mpdu;
    ...
    struct timeval time;
}
```

802.11 MAC frame arrived at A-DCF module is inserted into CQ. The frame is wrapped in `struct adcf_data`, which has `struct timeval time` field: time to be inserted CQ.

```c
struct adcf_cq_maq{
    struct adcf_data *cq;
    u16 cq_head;
    u16 cq_tail;
    atomic_t cq_depth;
    u16 cq_capacity;

    struct adcf_data *maq;
    u16 maq_head;
    u16 maq_tail;
    atomic_t maq_depth;
    atomic_t maq_byte;
    u16 maq_capacity;
    ...
    struct timer_list inject_timer;
    ...
    u8 dst_addr[ETH_LEN];
    ...
```
struct adcfx *adcf;
}

struct adcf_cq_maq is composed of three parts: CQ, MAQ, and additional information. CQ and MAQ are realized by using the circular queue. The circular queue is constructed from array. The functions to operate CQ and MAQ are as follows.

- void adcf_insert_cq(struct adcf_cq_maq *adcf_queue, struct ieee80211_hw *hw, struct sk_buff *skb)
- struct adcf_data *adcf_remove_cq(struct adcf_cq_maq *adcf_queue)
- void adcf_insert_maq(struct adcf_cq_maq *adcf_queue, struct ieee80211_hw *hw, struct sk_buff *skb)
- struct adcf_data *adcf_remove_maq(struct adcf_cq_maq *adcf_queue)
- struct adcf_data *adcf_peek_maq(struct adcf_cq_maq *adcf_queue)

These functions internally update cq_head, cq_tail, cq_depth, maq_head, maq_tail, and maq_depth. struct adcf_cq_maq is initialized in int ath_init_softc(u16 devid, struct ath_softc *sc, u16 subsysid) function (lines 1502-1536 in main.c) where memory with the amount of cq_capacity and maq_capacity is allocated to each cq and maq, respectively.

struct timer_list inject_timer is the timer for source rate control. When this timer is expired, void adcf_inject_timer(struct adcf_cq_maq *adcf_queue) is called. This function moves multiple struct adcf_data instances (MAC frames with time tags) from struct adcf_data *cq into struct adcf_data *maq. The amount of instances is calculated by A-DCF's source rate control algorithm (or DRR).

u8 dst_addr[ETH_LEN] is the MAC address of the neighbors for this link. struct adcfx *adcf is the back pointer to the structure which manages A-DCF.
struct adcf_ta_time *adcf_rts_overheard;
    u32 adcf_v;
    struct timer_list normal_v_timer;
}

struct adcf_tx is mainly composed of three parts: queue, data for TXOP, and data for adaptive DRR. struct adcf_cq_maq *adcf_queue is the array for containing multiple struct adcf_cq_maq (link queue). In int ath_init_softc(u16 devid, struct ath_softc *sc, u16 subsysid) function (line 1502-1537 in main.c), adcf_cq_maq's with the number of ADCF_LINK are allocated, and the address of the memory is assigned to adcf_queue. The current prototyping version has the fixed number of link queue (i.e. ADCF_LINK), which is determined at compile time. We will update the fixed allocation into dynamic link queue.

When back-to-back transmissions are required, we use TXOP, which is enabled by setting appropriate field in Tx control descriptor. For this end, bool vmf and u16 burst_dur are used. vmf is set to true if the next MAC frame will be transmitted back-to-back. burst_dur is set to the duration of SIFS, MAC frame, DIFS, and a slot to reserve the channel for protection. These variables are configured in void adcf_tx_dcf(struct ath_softc *sc, struct adcf_cq_maq *adcf_queue) and void adcf_tx_btb(struct ath_softc *sc, struct adcf_cq_maq *adcf_queue). The former function is called for normal access of the HOL frame at MAQ and the latter function for back-to-back transmissions.

For adaptive DRR, struct adcf_ta_time *adcf_rts_overheard, u32 adcf_v, and struct timer_list normal_v_timer are used. These variables are used for detection of hidden nodes, current V, and timer for reverting to normal mode.

B. Major functions

Here, we describe main functions to implement O-DCF and A-DCF.

int adcf_tx(struct ieee80211_hw *hw, struct sk_buff *skb)

The interface between the upper layer and MAC is struct ieee80211_ops, which has int(*tx)(struct ieee80211_hw *hw, struct sk_buff *skb) function pointer. To operate as A-DCF or O-DCF, tx is set to adcf_tx in ath_init_softc(u16 devid, struct ath_softc *sc, u16 subsysid)
function (lines 1371-1376, main.c), MAC frame transmissions are started when the upper calls
adcf_tx(struct ieee80211_hw *hw, struct sk_buff *skb). Note that by default 802.11 DCF sets
\textit{tx} to \textit{ath9k_tx} in line 2951, main.c.

In \texttt{adcf\_tx(struct ieee80211\_hw *hw, struct sk\_buff *skb)} function, \textit{skb} is inserted to its
corresponding CQ, which is obtained by \texttt{struct adcf\_cq\_maq *adcf\_intra\_sched(struct ath\_softc
*sc)} function, by default. When its corresponding MAQ is waiting for insertion, \textit{skb} is directly
inserted into the MAQ. Overall, \texttt{adcf\_tx(struct ieee80211\_hw *hw, struct sk\_buff *skb)} function
is responsible for queueing into \textit{skb} its CQ or MAQ.

\begin{verbatim}
void adcf\_inject\_timer(struct adcf\_cq\_maq *adcf\_queue)

As explained before, the pair of CQ and MAQ is wrapped in \texttt{struct adcf\_cq\_maq}, which has
\texttt{struct timer\_list inject\_timer} variable. Each pair of CQ and MAQ controls source rate control
by using this variable. When this timer is expired, \texttt{void adcf\_inject\_timer(struct adcf\_cq\_maq
*adcf\_queue)} is called. This function moves multiple \texttt{struct adcf\_data} instances (MAC frames with
time tags) from \texttt{struct adcf\_data *cq} into \texttt{struct adcf\_data *maq}. The amount of instances is
calculated by A-DCF's source rate control algorithm (or DRR).

void adcf\_tx\_tasklet(unsigned long context)

This function is called when a MAC frame is inserted into empty MAQ or TX interrupt is occurred.
In this function, a link is selected by \texttt{struct adcf\_cq\_maq *adcf\_intra\_sched(struct adcf\_tx
*adcf)} function call (line 567, adcf\_xmit.c). From the selected link, MAC frame transmissions are
prepared by DCF (\texttt{void adcf\_tx\_dcf(struct ath\_softc *sc, struct adcf\_cq\_maq *adcf\_queue)}
function call, line 574, adcf\_xmit.c) or back-to-back (\texttt{void adcf\_tx\_dcf(struct ath\_softc *sc,
struct adcf\_cq\_maq *adcf\_queue)} function call, line 591, adcf\_xmit.c). The preparation is
performed multiple times until txq is full or remaining transmission length is insufficient.

void adcf\_tx\_dcf(struct ath\_softc *sc, struct adcf\_cq\_maq *adcf\_queue)

This function calculates DCF parameters and transmission length according to transmission
aggressiveness and calls \texttt{int adcf\_tx\_txq(struct ieee80211\_hw *hw, struct sk\_buff *skb, u32
dcf\_parameters)} function, where DCF parameters is applied in the chipset and \textit{skb} is inserted TXQ.
void adcf_tx_btb(struct ath_softc *sc, struct adcf_cq_maq *adcf_queue)

This function configures burst_dur and vmf field in struct adcf_tx for back-to-back transmissions. The values configured are used in void adcf_setburst(struct ath_softc *sc, struct ath_buf *bf) function, which is called in void ath_tx_qaddbuf(struct ath_softc *sc, struct ath_txq *txq, struct list_head *head) function. adcf_setburst(struct ath_softc *sc, struct ath_buf *bf) function sets some fields in Tx Control Descriptor for obtaining TXOP. Like adcf_tx_dcf(struct ath_softc *sc, struct adcf_cq_maq *adcf_queue) function, this function calls adcf_tx_txq(struct ieee80211_hw *hw, struct sk_buff *skb, u32 dcf_parameters) function.

int adcf_tx_txq(struct ieee80211_hw *hw, struct sk_buff *skb, u32 dcf_parameters)

This function corresponds to int ath9k_tx(struct ieee80211_hw *hw, struct sk_buff *skb) in 802.11 DCF. The difference is as follows. ath9k_tx(struct ieee80211_hw *hw, struct sk_buff *skb) function checks whether TXQ is full or not while this operation is omitted in adcf_tx_txq(struct ieee80211_hw *hw, struct sk_buff *skb, u32 dcf_parameters) function since the check is already done in adcf_tx_tasklet(unsigned long context). adcf_tx_txq(struct ieee80211_hw *hw, struct sk_buff *skb, u32 dcf_parameters) function configures DCF parameters from the parameter dcf_parameters while ath9k_tx(struct ieee80211_hw *hw, struct sk_buff *skb) function uses the fixed DCF parameters.

Finally, adcf_tx_txq(struct ieee80211_hw *hw, struct sk_buff *skb, u32 dcf_parameters) calls int ath_tx_start(struct ieee80211_hw *hw, struct sk_buff *skb) function, which processes the remaining operations for transmissions.