Technical Report - IEEE 802.11 implementation issues/bugs in ns2

Ilango Purushothaman, Sumit Roy
{ilangop, sroy}@u.washington.edu

Department of EE, University of Washington,
Seattle, WA 98195-2500

The following are some bugs and issues, which were found in IEEE 802.11 implementation in ns-2.30.

Infrastructure mode :

Infrastructure mode support is very weak in ns2. Scanning, Association and Authentication functions have not been implemented. Beacon frames are also absent in the current implementation. While a beacon patch was developed last year by Matteo Rossi, which provided beacon frames and association, it has not been incorporated into ns2 yet.

While a node can be configured as an Access Point in tcl, nothing more can be done. NOAH routing agent has been suggested to force nodes to talk each other, only though the AP, thereby creating an apparent infrastructure scenario.

Direct Access Denial :

According to the IEEE 802.11 standard, section 9.2.5.1 states that,

In general, a STA may transmit a pending MPDU when it is operating under the DCF access method, either in the absence of a PC, or in the CP of the PCF access method, when the STA determines that the medium is **idle** for greater than or equal to a DIFS period

This means, that when a node can grab the channel (direct access), if the channel is found to idle initially, and still remained idle for DIFS. In other terms, backoff procedure should not be invoked.

But ns2's 802.11 code forces the STA to backoff and denies it direct access, even if the channel has been idle for more than DIFS. This is a deviation from the standard. Nevertheless, it might be a fair scenario in some cases, as this inadvertently makes sure that no STA grabs the channel finding it to be idle for >= DIFS, while another STA was actually deferring for DIFS + random time.

The relevant code is as follows.

```
if(mhBackoff_.busy() == 0) {
                if(is_idle()) {
                        if (mhDefer_.busy() == 0) {
                                /*
                                 * If we are already deferring, there is
                                 * no need to reset the Defer timer.
                                 */
                                if (bugFix_timer_) {
                                        mhBackoff_.start(cw_, is_idle(),
                                                        phymib_.getDIFS());
                                }
                                else {
                                        rTime = (Random::random() % cw_)
                                                * (phymib_.getSlotTime());
                                        mhDefer_.start(phymib_.getDIFS() +
                                                        rTime);
                                }
                }
```

Random Backoff time :

According to the IEEE 802.11 standard section 9.2.4, the random backoff time is calculated based on a random number in the interval [0, CW]. Therefore, the random number includes the numbers 0 and CW.

The IEEE 802.11 implementation of ns-2 selects the random number using,

```
Random::random() % cw_
```

where cw_ is the mentioned CW. Thus, the modulus % operation generates random numbers between 0 to CW - 1 and not CW.

Hence, the code should be

```
        Random::random() % (cw_ + 1)
```

Capture model :

In ns2, Physical layer capture (PLC) is implemented as follows.

When a STA is currently receiving a data packet (P1) and during the reception of P1, another packet (P2) reaches the STA, the model allows STA to be able to continue decoding successfully the first packet, if its power P1 is stronger than the power of the second packet (P2), by at least a factor called capture threshold (CPthr),

pow(P1) >= CPthr * pow(P2).
However in ns2, the stronger packet will be successfully received, only if it's the earlier packet.
Ns2's PLC doesn't allow the capture of the stronger 2nd packet (stronger by CPthr atleast). This is marked as a collision in ns2.

However in reality, even if the decode chain is synchronized to the 1st packet and a stronger 2nd packet arrives, the STA is able to resynchronize and decode this 2nd packet, provided the 2nd packet does not reach the receiver between 4µs and 10µs after the detection of 1st packet (within the physical layer preamble of 1st packet). The IEEE 802.11 standard allows for such a capture model.

Hence, the capture function must be changed in mac-802_11.cc, to allow for reception of stronger packets, irrespective of the order in which they are received.

The relevant code is as follows.

```
else {
                /*
                 *  If the power of the incoming packet is smaller than the
                 *  power of the packet currently being received by at least
           *  the capture threshold, then we ignore the new packet.
                 */
                if(pktRx_->txinfo_.RxPr / p->txinfo_.RxPr >= p->txinfo_.CPThresh) {
                        capture(p);
                } else {
                        collision(p);
                }
        }

Void Mac802_11::capture(Packet *p)
{
        /*
         * Update the NAV so that this does not screw
         * up carrier sense.
         */
        set_nav(usec(phymib_.getEIFS() + txtime(p)));
        Packet::free(p);
}
```

It should be noted that, Qualnet has recently updated its capture model to include the stronger 2nd packet reception too.

Infrastructure Mode Implementation:

STAGE 1: Packet filtering – Automatic Association

- Implemented automatic association, ie, STAs ask to be associated from the tcl script and AP adds the STA id to its association table (using a linked list). I bypassed association, authentication and beacon frames in stage 1.
- Packet filtering is hence facilitated and the AP rejects packets from non-associated STAs.

As of now, association is initiated in tcl as

```
set AP_ADDR [$mac_(AP_index) id]
for {set i 0} {$i – all STAs except AP } {incr i} {
   $mac_(AP_index) associate [$mac_($i) id]
}
```

```
# Setting bss_id to all associated STAs including AP
for {set i 0} {$i – all associated STAs } {incr i} {
  $mac_($i) bss_id $AP_ADDR
}
```

Files Affected: ./mac/mac-802_11.cc, ./mac/mac-802_11.h

STAGE 2: Beacons

- Beacon frames (Transmission and Reception) were implemented. (Beacons have not been implemented in ns-2.30).

Beacon Transmission:

- A new timer, BeaconTimer, was added to facilitate periodic transmission of beacons (set for BeaconInterval). The BeaconTimer on expiry calls the beacon transmission function and sets the BeaconTimer again.
- The beacon packet is built and transmitted (depending on channel status). According to the IEEE standard, AP should send the beacon at every expiry of BeaconInterval. If the medium is found to be busy, AP should defer until the channel becomes free again and should grab the channel after DIFS and transmit the beacon (No backoff).
- Sometimes, beacon transmission might be delayed (channel busy). Still, subsequent beacons should be sent at nominal beacon intervals. The mac-802_11.cc code was modified accordingly.
- Currently, Beacon frames just have address and bssid information. I will be adding Timestamp, supported data rates and other fields in Stage 3.

Beacon Reception:

- On receiving the beacon, each STA’s beacon reception function is called, which basically stores the address info and bssid.

Issues noted:

- For packets like Beacon and ACK, Send timer timer and IF timer expiry times are equal (Since Beacon and ACK dont expect a reply and so there is no timeout). But I found out that for Beacon and ACK, sometimes, the Send timer expires a miniscule amount of time before IF timer expires (probably due to floating point approximation) and this caused a problem in Beacon Transmission, which was eventually solved.

- The Subtype value for Beacon in 802.11(according to standard) is 0x08 while it is defined as 0x0f for 802.15.4 in ns2. Hence, 802.11’s “#define MAC_Subtype_Beacon 0x08” clashes with 802.15.4’s MAC_Subtype_Beacon, which led to incorrect labels in the trace file (see cmu-trace.cc). “Make” processes 802.11 first, before moving onto 802.15.4 and hence MAC_Subtype_Beacon is redefined as 802.15.4’s value. Hence, I had to change the #define to MAC_Subtype_11_Beacon. Suggestions to work around this small problem will be welcome.

Files affected: ./mac/mac-802_11.cc, ./mac/mac-802_11.h, ./mac/mac-timers.cc, ./mac/mac-timers.h, ./tcl/lib/ns-default.tcl, ./trace/cmu-trace.cc

STAGE 3: Association

- I am currently in the process of implementing Passive scanning using Association Request/Response frames. The code should be ready within two weeks.