Report Gruppo 46 – Homework 2

Membri:

- Lorenzo Pannacci - 1948926 (leader)
- Matteo Pannacci - 1948942
- Liwen Zheng - 1915940
- Matteo Terrinoni - 1883939

Assunzioni:

1. Esplicitiamo alcune specifiche non presenti nel testo ma dette a lezione oppure nella mailing list:

   - I programmi sono eseguiti passando un ulteriore parametro, *'--RngRun = sommaMatricole'*, che serve fissare gli eventi casuali ed è fissato ad un valore pari alla somma dei numeri di matricola dei membri del gruppo. Nel nostro caso:

     $$\boldsymbol{sommaMatricole = 1948926 + 1948942 + 1915940 + 1883939 = 7697747}$$

   - Le posizioni iniziali dei nodi seguono i seguenti parametri:
     'Tipo di posizionamento: griglia con layout che alloca prima le righe. La griglia inizia dalle coordinate (0,0), posiziona i nodi con un intervallo tra i nodi di (5,10). Per ogni linea vanno allocati al massimo 3 nodi.'
     (Si assume inoltre che il nodo AP sia di seguito ai nodi STA)

   - Quando '*useRtsCts = true'* impostiamo *'RtsCtsThreshold'* al valore *'100'*.

   - Includiamo nel file *.zip* da consegnare anche i file *.xml* prodotti dalle simulazioni.

2. Assumiamo che le domande facciano riferimento a simulazioni dove '*useRtsCts = false'*, salvo diversa indicazione nel testo.

3. Quando dobbiamo numerare i byte di un frame contiamo partendo da 0 come fa Wireshark.

4. Non siamo in grado di identificare le collisioni tra frame *RTS* e *CTS* tramite Wireshark con i metodi che usiamo per gli altri frame (non hanno il sequence number e non impostano la flag *retry*). Non ci riusciamo neanche analizzando il file *.xml* tramite NetAnim. Leggendo direttamente il file *.xml* invece siamo in grado di individuare l'invio e la mancata ricezione di questi frame. Dove necessario verificare la presenza di collisioni che coinvolgono frame *RTS* e *CTS* useremo questo metodo.

Task 1:

1) Tutti i frame ricevono l'acknowledgement? Spiegare perché.

La simulazione utilizza il protocollo CSMA/CA come implementato nello standard IEEE 802.11; questo prevede l'invio di un frame di acknowledgement per segnalare al mittente l'avvenuta ricezione di frame di dati.

Analizzando il file *'task1-on-2.pcap'* si osserva che il primo byte di ogni frame ne specifica il tipo. Si individuano:

- *Data frame*, di cui fanno parte i frame contenenti segmenti *UDP*, *ARP request* e *ARP reply*
- *Control frame*, di cui fanno parte *ACK*, *RTS* e *CTS*

I primi portano informazioni e generalmente richiedono l'acknowledgement, i secondi servono a gestire la comunicazione e non lo usano.

Le *ARP request* non richiedono acknowledgement nonostante siano *Data frame* in quanto sono broadcast.

2) Vi sono delle collisioni nella rete? Spiegare perché. Come sei arrivato a questa conclusione?

Non ci sono collisioni nella rete. Per i frame che ricevono *ACK* si nota la presenza di collisioni tramite Wireshark controllando la flag *'retry'* che troviamo nel secondo byte dell'header. Le *ARP request* non si aspettano un *ACK* e non impostano la flag alla ritrasmissione, possiamo però controllare il *'sequence number'* presente nei byte 22-23.

L'unico momento critico è intorno a *t = 2s* quando entrambi i client vogliono inviare un pacchetto.

Mentre il nodo 4 presenta già nella sua *ARP table* il MAC del server, questo non è vero per il nodo 3. Il ritardo introdotto dall'utilizzo del protocollo *ARP* è tale da permettere al nodo 3 di accorgersi che il canale è stato occupato dal nodo 4 e mettersi in attesa.

3) Come si può forzare i nodi ad utilizzare la procedura di handshake RTS/CTS vista in classe? Quale'è il ragionamento dietro questa procedura?

La procedura *RTS/CTS* permette di prenotare il canale. Il mittente notifica l'intenzione di comunicare inviando un *RTS* e se il destinatario è libero risponde con un *CTS*. Alla ricezione del *CTS* il mittente sa che il canale è prenotato.

Gli altri nodi che ricevono questi frame sono tutti coloro che risultano abbastanza vicini da poter causare collisioni; vengono perciò messi in attesa affinché il canale sia libero tramite il meccanismo descritto nella risposta *1.4.3*.

Questa procedura causa overhead e perciò è usata solo per messaggi di dimensione elevata; per forzarla possiamo diminuire la soglia sopra la quale si inizia ad usare *RTS/CTS*, come fatto nella simulazione *NS-3* quando si pone *'useRtsCts = true'*.

4) Forzare l'uso di RTS/CTS nella rete utilizzando il parametro useRtsCts:

- #### *4.1) Ci sono delle collisioni adesso?*

Non sono possibili collisioni tra unità informative, lo sono invece tra frame *RTS*, *CTS*, *ARP* e *ACK* ma qui non sono presenti. Osserviamo in *'wireless-task1-rts-on.xml'* che nel momento critico *t = 2s* non vi sono ritrasmissioni.

- #### *4.2) Quali sono i benefici di RTS/CTS?*

RTS/CTS risolve il problema dell'*hidden terminal* ed impedisce le collisioni tra frame di dati. Possono ancora collidere *RTS*, *CTS* e frame sotto il threshold, ma essendo piccoli è poco probabile.

- *4.3) Dove si può trovare ed analizzare le informazioni relative al Network Allocation Vector?*

Il *NAV* è un timer interno al nodo che indica per quanto tempo questo non può comunicare in un canale prenotato da un altro nodo; prende informazioni dal campo *duration* dei frame (byte 2-3) che il nodo riceve.

Il valore di d*uration* diminuisce in ogni messaggio successivo dello stesso scambio (figura 1).

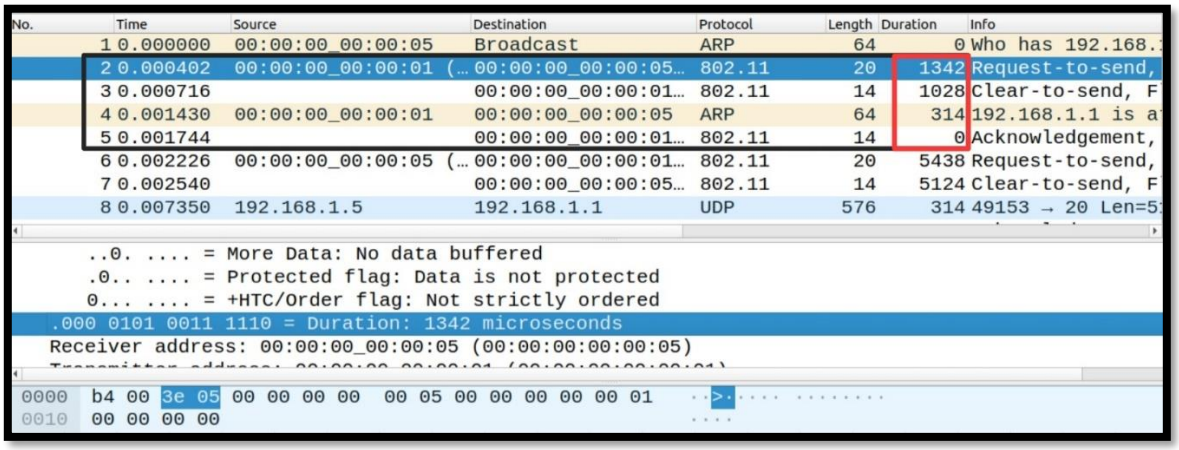

Figura 1: campo duration dei frame visti attraverso Wireshark

5) Calcolare il throughput medio complessivo delle applicazioni.

Tramite il logging (*figura 2*) vediamo che il primo frame viene inviato a *t1 = 1s* e l'ultimo viene ricevuto a *t2 = 4,01001s*. L'intervallo di tempo per il quale calcolare il throughput sarà:

$$\Delta t = t2 - t1 = 4,01001 - 1 = 3,01001s$$

Abbiamo due client che mandano due messaggi ciascuno verso un server Echo, che risponde inviando messaggi con lo stesso contenuto informativo, per un totale di *n = 8 messaggi*.
Ogni messaggio è contenuto all'interno di un frame che è composto da 576 byte, di cui *p = 512* sono il contenuto informativo e 64 l'header. Ci interessa calcolare il throughput delle applicazioni, perciò consideriamo solo i byte che compongono il messaggio applicativo.

$$throughput = \frac{dimensione\ messaggio * numero\ messaggi}{intervallo\ di\ tempo}$$

$$T = \frac{p * n}{\Delta t} = \frac{512\ byte/messaggio * 8\ messaggi}{3,01001\ s} = 1.360,7928\ byte/s = 10.886,3424\ bit/s$$

```
At time +1s client sent 512 bytes to 192.168.1.1 port 20
At time +1.01575s server received 512 bytes from 192.168.1.5 port 49153
At time +1.01575s server sent 512 bytes to 192.168.1.5 port 49153
At time +1.03058s client received 512 bytes from 192.168.1.1 port 20
At time +2s client sent 512 bytes to 192.168.1.1 port 20
At time +2s client sent 512 bytes to 192.168.1.1 port 20
At time +2.00485s server received 512 bytes from 192.168.1.5 port 49153
At time +2.00485s server sent 512 bytes to 192.168.1.5 port 49153
At time +2.01001s client received 512 bytes from 192.168.1.1 port 20
At time +2.0173s server received 512 bytes from 192.168.1.4 port 49153
At time +2.0173s server sent 512 bytes to 192.168.1.4 port 49153
At time +2.02809s client received 512 bytes from 192.168.1.1 port 20
At time +4s client sent 512 bytes to 192.168.1.1 port 20
At time +4.00485s server received 512 bytes from 192.168.1.4 port 49153
At time +4.00485s server sent 512 bytes to 192.168.1.4 port 49153
At time +4.01001s client received 512 bytes from 192.168.1.1 port 20
```

Figura 2: logging dell'esecuzione di task1 con 'useRtsCts=false'

Task 2:

1) Spiegare il comportamento dell'AP. Cosa succede fin dal primo momento dell'inizio della simulazione?

Fin dall'inizio della simulazione l'access point manda ad intervalli regolari di *0,1024s* un *'beacon frame'* in broadcast, che fornisce informazioni di base sulla rete e segnala l'esistenza dell'AP alle station in modo che queste possano richiedere l'associazione.

Vediamo, tramite Wireshark sul file *"task2-off-5.pcap",* che a seguito dei primi *beacon frame* arrivano dalle station delle *'Association request'* dirette all'access point che, prima risponde a queste con un *ACK*, poi con una *'Association response'* e infine riceve un *ACK* da parte della station.

Tramite il filtro *"wlan.fc.subtype == 0x0001"* sul file precedente, possiamo verificare che tutte le station hanno effettuato questa operazione.

2) Analizzare il beacon frame. Quali sono le sue parti più rilevanti? Specificare il filtro Wireshark ed il file utilizzati per l'analisi.

Prendiamo in esame il file *'task2-off-5.pcap'* e isoliamo i *beacon frame* utilizzando il filtro *'wlan.fc = 0x8000'*. Un *beacon frame* è di 71 byte, elenchiamo quelli che secondo noi sono i campi più significativi:

- *Byte 0*: tipo (*management frame*), sottotipo (*beacon*) e versione (*0*) del frame (fig. 3)
- *Bytes 4-9 e 10-15*: MAC address di destinazione (*broadcast*) e quello di sorgente (*AP*) (fig. 3)
- *Bytes 32-33*: intervallo temporale tra un beacon frame e il successivo, pari a *0,1024s* (fig. 4)
- *Bytes 38-44*: SSID della rete, come da consegna è *'7697747'* (fig. 4)
- *Bytes 45-54 e 61-66*: liste che contengono i data-rate supportati. Il massimo è 54 Mbit/s, coerente con le specifiche del protocollo 802.11g utilizzato (fig. 4)

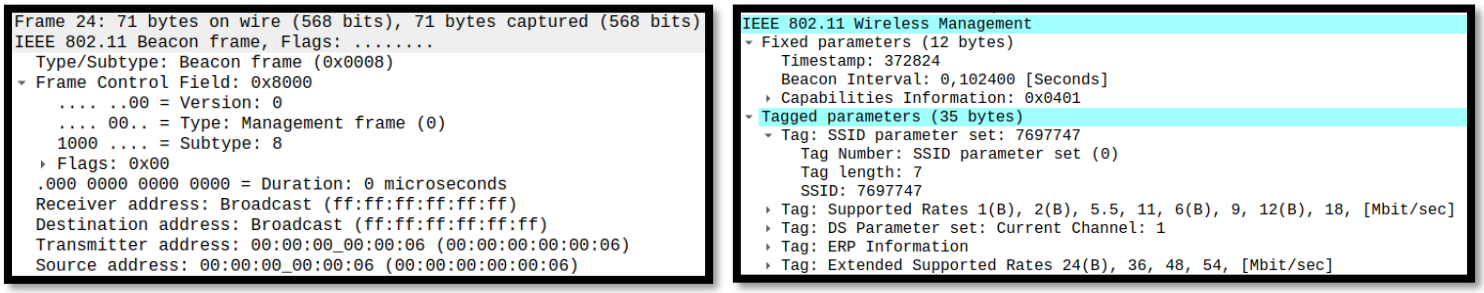

Figure 3-4: contenuto di un beacon frame letto attraverso Wireshark

3) Come per il Task 1, forzare l'uso di RTS/CTS nella rete utilizzando il parametro "useRtsCts":

- *Ci sono delle collisioni adesso? Spiegare il perché.*

Forzando *RTS/CTS* abbiamo le collisioni tra le *Association Request* presenti anche nel caso senza *RTS/CTS* (visibili con il filtro *'wlan.fc.retry == 1'*), dato che la loro dimensione resta sotto il threshold. Non c'è più collisione tra frame informativi, ma c'è tra *RTS*.

```
1189 <pr uId="132" fId="4" fbTx="4.000028" meta-info="ns3::WifiMacHeader(CTL_RTS Duration/ID=5438us,
      RA=00:00:00:00:00:06, TA=00:00:00:00:00:05) ns3::WifiMacTrailer ()" />
1190 <pr uId="133" fId="3" fbTx="4.000028" meta-info="ns3::WifiMacHeader (CTL_RTS Duration/ID=5438us,
      RA=00:00:00:00:00:06, TA=00:00:00:00:00:04) ns3::WifiMacTrailer ()" />
1191 <wpr uId="132" tId="5" fbRx="4.000032009" lbRx="0" />
1192 <wpr uId="132" tId="1" fbRx="4.000032029" lbRx="0" />
1193 <wpr uId="132" tId="2" fbRx="4.00003203" lbRx="0" />
1194 <pr uId="134" fId="5" fbTx="4.000390009" meta-info="ns3::WifiMacHeader (CTL_CTS Duration/ID=5124us,
      RA=00:00:00:00:00:05) ns3::WifiMacTrailer ()" />
...
1260 <pr uId="145" fId="3" fbTx="4.017500412" meta-info="ns3::WifiMacHeader (CTL_RTS Duration/ID=5438us,
      RA=00:00:00:00:00:06, TA=00:00:00:00:00:04) ns3::WifiMacTrailer ()" />
...
1266 <pr uId="146" fId="5" fbTx="4.017862455" meta-info="ns3::WifiMacHeader (CTL_CTS Duration/ID=5124us,
      RA=00:00:00:00:00:04) ns3::WifiMacTrailer ()" />
```

Figura 5: estratto del file 'wireless-task2-rts-on.xml'

Analizziamo il file *.xml* (fig. 5):

- [1189, 1190]: n4 e n3 inviano *RTS* a *t = 4,000028s*
- [1191, 1192, 1193]: n1, n2 e n5 (vicini a n4) ricevono l'*RTS* di n4 mentre n0 (circa equidistante da n3 e n4) non riceve niente, coerentemente con le figure 6, 7, 8 e 9
- [1194]: n5 risponde inviando il *CTS*
- [1260, 1266]: dopo l'attesa n3 ritrasmette l'*RTS* e n5 risponde con *CTS*

Vi è collisione perché i 2 client hanno già nelle *ARP table* il server ed inviano allo stesso istante l'*RTS*.

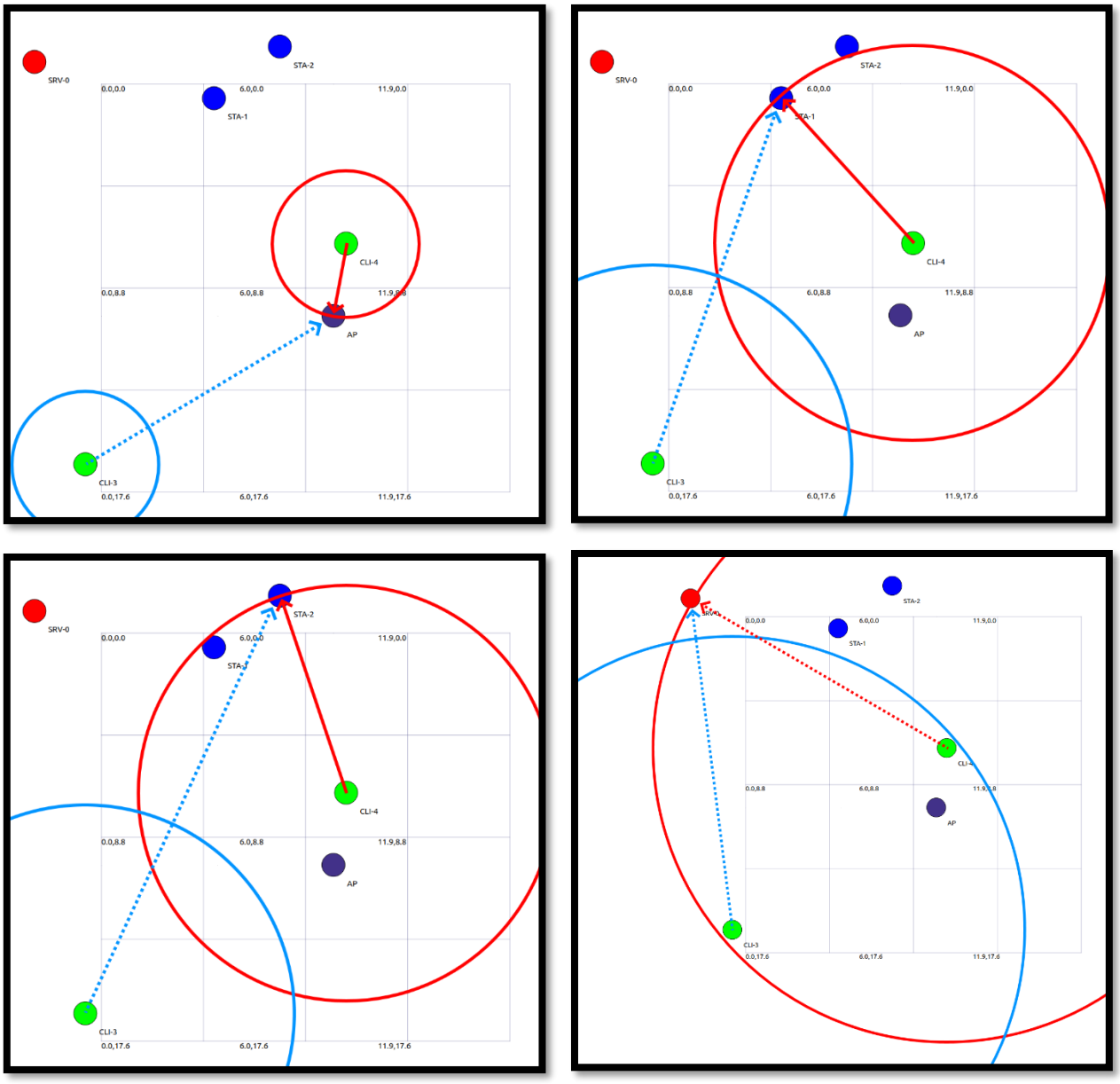

Figure 6, 7, 8, 9: rappresentazioni del propagarsi dei frame RTS di n3 e n4 nel tempo usando la visualizzazione fornita da NetAnim. Il frame di n4 raggiunge n5, n1 e n2 molto prima del frame di n3, mentre raggiunge n0 solo poco prima.