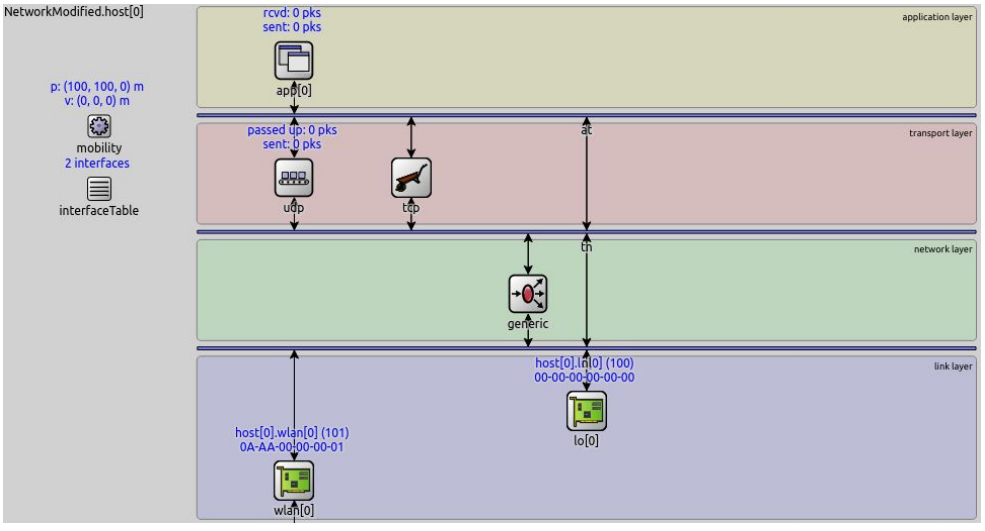
Progetto RAI Caruso e Fallica
Elaborato G 2019-2020

Introduzione

Con questo progetto si vuole simulare uno scenario di una rete wireless ad hoc IEEE 802.11 in cui dei nodi mobili scambiano messaggi.
Nella nostra rete, sono presenti ben 20 nodi di classe “AdHocHost”, la cui struttura semplificata è mostrata mediante la seguente figura:



Gli attori, ovvero i nodi della rete, sono tali da specializzarsi in tre ruoli ben precisi:

- **Sender:** mediante un livello applicativo di tipo “UdpBasicApp” viene generato un messaggio generico, il quale viene trasmesso

mediante la porta 7654 facendo uso di UPD come protocollo a livello di trasporto.
Il livello di rete di tipo “NextHopNetworkLayer” consente di incapsulare il messaggio proveniente dal livello applicativo, in modo tale da poterlo trasmettere al forwarder definito mediante il valore “NextHop”.
A livello di rete viene aggiunto un chunk al messaggio in modo tale da contenere i campi richiesti dalla consegna: tale chunk contiene informazioni su sender, receiver e sequence number, e viene “incollato” in fondo al “datagram” generato dalle operazioni a livello di rete.
Successivamente il pacchetto viene trasmesso mediante un link layer basato su un’interfaccia MAC.
In quanto si suppone uno scenario wireless, si trasmette in broadcast.

- **Forwarder:** questi attori hanno il compito di ricevere dei pacchetti inviati da un sender o da un altro forwarding node, e di inoltrarlo al successivo “nextHop” definito mediante il file di configurazione.
  Ovviamente, in quanto scenario wireless, la trasmissione avviene in broadcast, ma l’effettivo destinatario rimane il “nextHop”, quindi se un forwarding node riceve un pacchetto non destinato a lui stesso, provvederà a scartarlo.
- **Receiver:** questo attore ha il compito di ricevere dei pacchetti da uno specifico forwarding node, e una volta riconosciuto che tale pacchetto è effettivamente destinato a lui stesso, procede con l’invio ai livelli superiori dei dati acquisiti.

Si ricordi che in quanto si fa utilizzo di “UPDBasicApp”, se viene ricevuto correttamente un pacchetto, il ricevitore provvederà ad inviare un WLAN ACK, ai fini di garantire al sender la corretta ricezione del pacchetto.

Scenario

Per lo sviluppo del seguente elaborato, sono state ideate due tipologie di reti differenti:

- **Topologia “matriciale”:** in questa topologia si sono disposti i nodi della rete ai fini di comporre una matrice 4*5, ove la prima colonna (da sinistra) contiene i sender, la seconda, la terza e la quarta colonna contengono i forwarder, e la quinta colonna contiene i receiver.

In tale topologia si è supposto di distanziare i nodi di ben 50 metri in linea orizzontale, e di 100 metri in linea verticale: lo scopo di tale disposizione è di osservare il comportamento della rete nel caso in cui vengono applicate delle condizioni “ottimali” per il canale, confrontandolo con il comportamento nel caso in cui le condizioni del canale siano pessime.

- **Topologia a “fitta aggregazione”:** in questa topologia si sono disposti i nodi in modo tale da ottenere la disposizione che si osserva nella seguente immagine.

Tale disposizione viene ripetuta due volte in modo tale da presentare ben 20 nodi nella rete.
In particolare, per questa topologia si osserva quanto segue:

- Host 0 a 3: sender
- Host 4 a 15: forwarder
- Host 16 a 19: receiver

Lo spazio previsto tra ogni nodo in questo caso è di 10 metri in linea orizzontale, e 10 metri in linea verticale: questa topologia difatti verrà utilizzata per osservare il comportamento della rete in caso di condizioni pessime di canale.

Per entrambe le topologie si suppone che i nodi della rete siano in grado di trasmettere pacchetti in broadcast con una distanza massima di 250 metri: tale valore, così come si osserverà in seguito, al peggiorare delle condizioni di canale subisce una drastica riduzione.
Inoltre, è stata prevista una versione statica e una versione dinamica per entrambi gli scenari mostrati in precedenza: anche in tal caso si osserveranno dei comportamenti ben diversi a seconda delle condizioni del canale.

Scelte implementative

Nel progetto in questione abbiamo fatto uso del livello network “NextHopNetworkLayer”, il quale è stato opportunamente modificato per soddisfare tutti i punti dell’elaborato.
Ai fini di far utilizzare ai nodi di tipo “AdHocHost” della nostra rete tale livello network, nel file .ini sono state scritte le seguenti linee di codice:

```
**.host[*].hasIpv4 = false #Required in order to use NextHopNetworkLayer
**.host[*].hasIpv6 = false
**.host[*].hasGn = true
**.host[*].generic.typename = "NextHopNetworkLayer" #Required
#**.networkConfigurator.typename = "NextHopNetworkConfigurator" #Add Next Hop Configurator
**.host[*].generic.nextHop.hopLimit = 4  #Required
```

Ovvero, è stato necessario disabilitare la comunicazione mediante l’uso di IPv4 e IPv6, ed è stato necessario abilitare l’utilizzo di “generic” per la rete, in quanto il nostro livello network rientra in tale categoria di indirizzamento.
Nel codice mostrato in precedenza è stato impostato il limite degli hop che un pacchetto è in grado di effettuare attraverso la rete.

In quanto si è impostato un livello network che non fa uso di IPv4, è stato necessario utilizzare un “Configurator” ad hoc per la nostra situazione: la soluzione è ricaduta nella scelta di “networkConfigurator”, il quale mediante l’utilizzo del file .xml denominato “lessnodes.xml” ci ha permesso di inizializzare correttamente le tabelle di routing di tutti i nodi della rete.

```
<config>
    <interface hosts='*' address='10.x.x.x' netmask='255.255.255.x'/>
</config>
```

L’immagine posta sopra, mostra il contenuto del file .xml, il quale ci ha permesso di assegnare un indirizzo di tipo “L3Address” valido per ciascun nodo della rete, e di assegnare una netmask adeguata: questi valori sono stati utilizzati per riempire le tabelle di routing dei nodi della rete, in modo tale che questi siano in grado di conoscere la rotta di destinazione del “nextHop” e di qualsiasi altro nodo previsto dallo scenario.

Sempre mediante il file .ini abbiamo implementato degli effetti grafici ai fini di mostrare l'effettivo range di comunicazione di ciascun nodo, mediante una circonferenza di colore blu.

```
**.networkConfigurator.typename = "NextHopNetworkConfigurator"
#Add some fancy effects to our wireless network
*.host[*].wlan[2].radio.displayCommunicationRange = true
*.visualizer.sceneVisualizer.descriptionFigure = "title"
*.visualizer.mediumVisualizer.displaySignals = true
*.visualizer.physicalLinkVisualizer.displayLinks = true

#Some communication range
*.host[*].wlan[0].radio.transmitter.communicationRange = 200m
*.host[*].wlan[0].radio.displayCommunicationRange = true
```

Ai fini di mostrare questi effetti, è stata utilizzata la proprietà "displayCommunicationRange" del modulo compound "radio" contenuto a livello data link di ciascun "AdHocHost".

Successivamente a queste operazioni di "configurazione" del file .ini, siamo passati alla modifica del livello network "NextHopForwarding" di INET.
Il livello di rete in questione prevede procedure differenti in base al "tipo di nodo" della rete:

- **Sender**: questo nodo genera dati da inviare mediante un layer superiore al livello network.
  Quest'ultimo, una volta ricevuti i dati da inviare dal layer superiore, chiama una funzione "encapsulate": tale funzione consente di generare un pacchetto e di aggiungere ad esso un header "NextHopForwardingHeader", il quale presenta i seguenti valori:
  - **srcAddr:** effettivo indirizzo del nodo che ha generato i dati, ovvero è l'indirizzo di un nodo sender.
  - **dstAddr:** effettivo indirizzo del nodo che deve ricevere i dati e portarli ai layer superiori, ovvero è l'indirizzo di un nodo receiver.
  - **hopLimit:** massimo numero di hop che tale pacchetto incapsulato può supportare.

  Dopo aver concluso la funzione precedente, mediante il parametro "nextHop" viene recuperato un indirizzo "nextHopMAC", ovvero il mac address del nextHop: tale informazione viene trasportata con

il pacchetto al livello inferiore aggiungendo un tag di tipo “MacAddressReq”.

- **Forwarder**: questo nodo riceve un pacchetto da un sender, e ne controlla l’effettivo indirizzo di destinazione a livello data link.
  Tale operazione viene eseguita in quanto tale nodo riceve in broadcast da tutti i sender o altri forwarder, e quindi devono essere scartate tutte le frame con destinazione differente.
  Se il pacchetto è effettivamente destinato al nodo in questione, viene eseguita una lettura del campo “dstAddr” contenuto in “NextHopForwardingHeader”, e comprendendo che non si tratta di un “self packet” si riesegue una funzione di encapsulating come descritta per il sender, destinando il pacchetto così generato al successivo “nextHop”.
- **Receiver**: tale nodo, ricevuto un pacchetto, controlla l’indirizzo di destinazione a livello data link allo stesso modo di un nodo forwarder.
  Se la precedente operazione va a buon fine, si procede con la lettura del campo “dstAddr” contenuto in “NextHopForwardingHeader”: ivi il nodo comprende che si tratta di un “self packet”, motivo per cui il nodo procede con un’operazione di decapsulamento e al successivo invio dei dati al livello superiore.

Da come si può dedurre da quanto esposto sopra, è fondamentale il valore “nextHop” per il corretto funzionamento del livello di rete, motivo per cui la prima problematica da noi affrontata è stata l’inizializzazione di tale valore “nextHop” il quale si ricordi è un valore di tipo “L3Address”. Tale inizializzazione è stata svolta aggiungendo un valore parametrizzabile di tipo string “nextHopParam”, al modulo compound “NextHopForwarding”: valore che, una volta inizializzato nel file .ini, è stato richiamato ove opportuno all’interno del file “NextHopForwarding.cc” mediante un casting a “L3Address”.

```
// send out datagram to NIC, with control info attached
MacAddress nextHopMAC = arp->resolveL3Address((L3Address)par("nextHopParam"), ie);
```

L’immagine sopra mostra un esempio ove è stato richiamato il parametro “nextHopParam” all’interno della classe “NextHopForwarding”.

Per quanto riguarda l’aggiunta dei campi previsti dal testo dell’elaborato, la scelta implementativa è ricaduta nell di un file .msg denominato “customBottom.msg” contenente un’estensione della classe FieldsChunk.

```
class customBottom extends FieldsChunk{
    chunkLength = B(8); //Lunghezza totale
    B lengthField = B(2); //Lunghezza campo singolo
    L3Address src; //Source
    L3Address dst; //Destination
    int seq = 0; //Sequence number
}
```

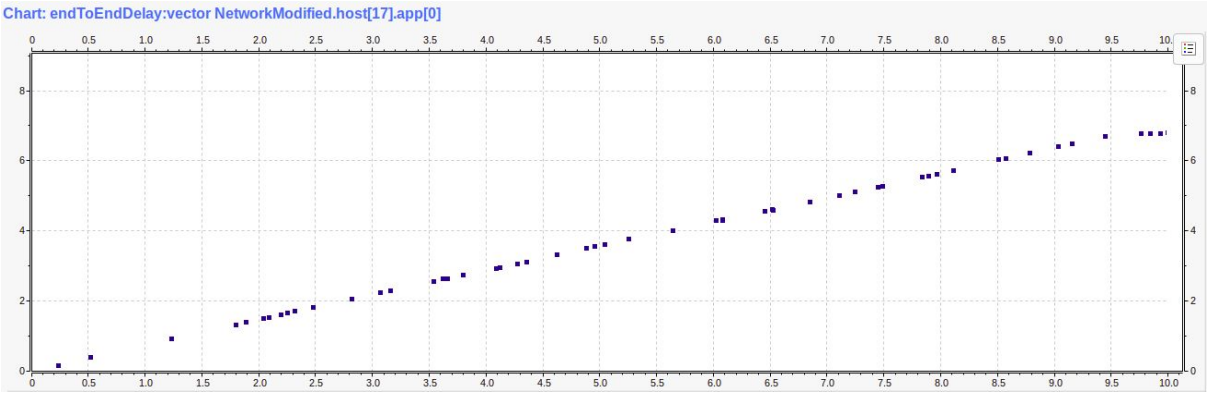
Mediante questa scelta implementativa, siamo stati in grado di scegliere la dimensione esatta di ciascun campo del chunk, senza compromettere l’header già utilizzato dal livello di rete.

```
//Creating info for customBottom chunk
counter += 1;
const auto& emptyBottom = makeShared <customBottom>(); //Create empty bottom chunk
emptyBottom->setSrc(header->getSourceAddress());
emptyBottom->setDst(destAddr);
emptyBottom->setSeq(counter);
datagram->insertAtBack(emptyBottom); //Add chunk
```

Come si osserva dall’immagine posta sopra, all’interno della funzione “routePacket” della classe “NextHopForwarding”, abbiamo inizializzato un chunk vuoto, per poi riempirne i campi con i valori richiesti.
Una volta riempiti i campi del chunk, questi è stato “incollato” in fondo al datagramma generato dal layer di rete, mediante la funzione “insertAtBack”.

Statistiche e risultati

Si supponga di eseguire la prima delle topologie osservate nel capitolo “Scenario”, mediante l’uso di un canale con condizioni non critiche, ovvero con dei valori di **alpha = 3** e **sigma = 7**.
Con un**’**esecuzione della durata di 10 secondi, un periodo di invio dei pacchetti di 20ms per ciascun sender, e in condizioni di completa staticità si osserva quanto segue per uno dei ricevitori:



Ovvero, con il passare del tempo, si osserva un incremento quasi lineare dell’end-to-end delay.
Tale andamento, può avere le seguenti motivazioni:

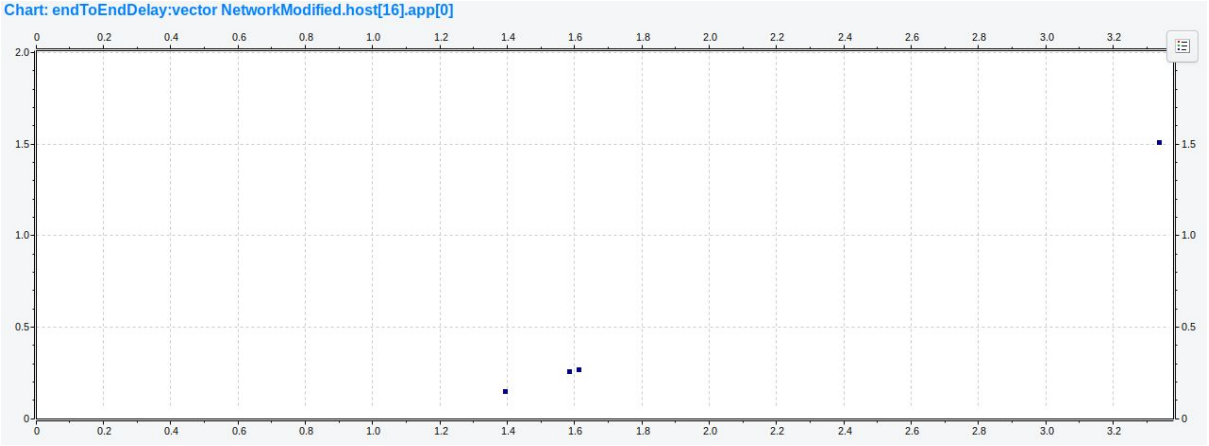
- Riempimento delle code di ricezione del ricevitore: in quanto ciascun sender e ciascun forwarder è in grado di inviare in broadcast le frame prodotte, il ricevitore riceve correttamente molte frame, e con il passare del tempo il carico delle code di ricezione del receiver aumenta notevolmente.
- Buone condizioni di canale: le condizioni del canale non impattano troppo sul range di trasmissione dei nodi di rete e sull’integrità dei pacchetti, motivo per cui le probabilità che un ricevitore riceva delle frame è più elevata.

Il risultato ottenuto sembra rispettare le nostre aspettative, motivo per cui si osservi cosa accade con le condizioni di canale previste dal testo dell’elaborato, ovvero con dei valori di **alpha = 4.03** e **sigma = 4.98**, restando sempre in condizioni statiche.

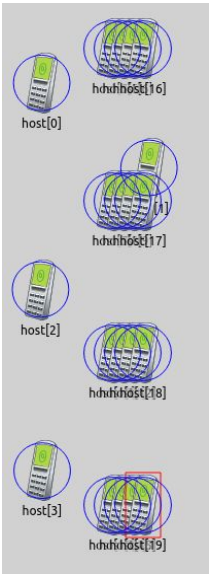
```
DEBUG:Computing whether listening is possible: maximum power = 0.01 pW, energy detection = 3.16228 pW -> listening is impossible
```

Con questi valori pessimi di canale, il range di invio effettivo (o meglio il range di invio che garantirebbe alte probabilità di corretto invio del pacchetto) è notevolmente ridotto: in quanto i nodi sono posti a una distanza di circa 50m in linea orizzontale, i sender provano all'infinito a inviare copie dei loro messaggi senza alcun successo.
La conclusione è che la "topologia Matriciale" con tali distanze e valori di canale non è in grado di far funzionare la rete.

In virtù di questo esito pessimo, si è deciso di "avvicinare" ciascuna colonna della topologia Matriciale (da 50m a 10m in linea orizzontale), e di introdurre la mobilità per i nodi sender della rete: anche per tale caso si è deciso di eseguire la simulazione per 10s, in modo tale da apprezzare il movimento (lento) dei senders.

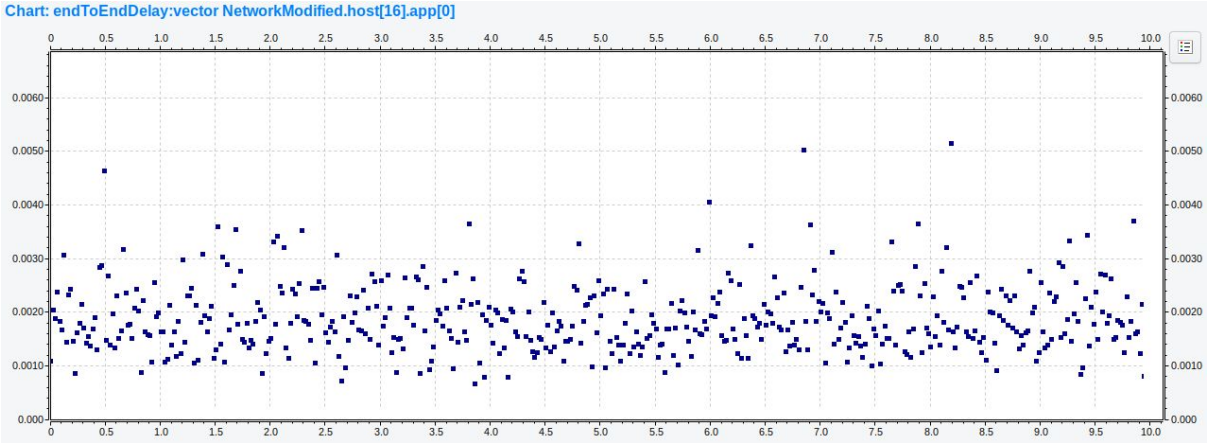


Dal grafico dell'end to end delay di un generico nodo ricevitore si osserva come inizialmente si ha una situazione similare alla prima delle simulazioni effettuate, tuttavia le condizioni del canale restano pessime e il numero dei pacchetti ricevuti dai nodi ricevitori è calato drasticamente.
Inoltre i senders muovendosi possono uscire dal range di ricezione dei forwarding nodes, causando una perdita massiccia di pacchetti.
L'immagine posta a lato mostra il secondo 3 della simulazione, ove 3 dei senders si sono completamente



allontanati dalla zona di ricezione dei forwarding nodes, portando ad un failure della rete.

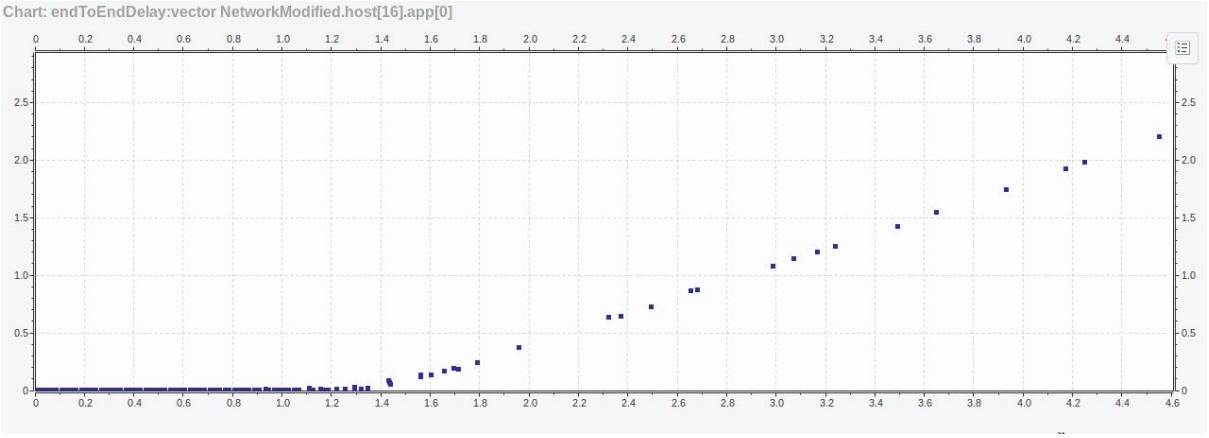
Si passi alla topologia “a fitta aggregazione”: ivi i nodi di rete sono disposti in due blocchi, con una distanza di circa 10m in linea orizzontale e 10m in linea verticale l’un l’altro.
Si osservi cosa accade con le condizioni di canale previste dal testo dell’elaborato, ovvero con dei valori di **alpha = 4.03** e **sigma = 4.98**, restando in condizioni di staticità e simulando per 10s.



Dal grafico ottenuto si osserva che grazie al fitto aggregamento tra i nodi, i ricevitori sono in grado di ricevere correttamente tutti i pacchetti. Inoltre, l’estrema vicinanza dei nodi riduce notevolmente il tempo di ricezione, il che comporta questi valori molto ridotti di end-to-end delay. Eventuali picchi di delay sono probabilmente causati da una condizione di carico delle code di ricezione dei ricevitori.

Introducendo la mobilità per i senders in tale topologia, ci si aspetta un comportamento similare alla simulazione precedente nel caso in cui i sender siano vicini ai forwarders, e una condizione di massiccio packet loss nel caso in cui i senders si allontanino troppo.

Ecco i risultati ottenuti osservando un generico nodo ricevitore:



Durante i primi secondi di simulazione si osserva come nessun pacchetto sia riuscito a raggiungere la destinazione: il movimento del sender è stato tale da comportare un packet loss.
Riportandosi in zona di ricezione, i receiver riescono a ricevere correttamente i pacchetti: tuttavia con l'allontanarsi del sender dalla zona di ricezione il delay aumenta notevolmente, fino a quando al secondo 4.6 il sender relativo al ricevitore in questione si è allontanato troppo e non è più tornato in zona di ricezione, causando un failure della ricezione dei pacchetti.