Telepati

versione 1.5 – 12 febbraio 2005

Gilberto Decaro
AIS-Lab – ais-lab.dsi.unimi.it

Indice

1. Introduzione

1.1. Rete Wireless di Aibo

1.1.1. Hardware di Rete degli Aibo e sua configurazione

Il modello ERS7 degli Aibo della Sony è fornito di una scheda di rete wireless integrata (da quanto sembra della Orinoco) compatibile con il protocollo IEEE 802.11b. Le impostazioni della scheda di rete vengono definite nel file **WLANCONF.TXT** in /MS/OPEN-R/SYSTEM/CONF (per un esempio guardare Esempio di configurazione di Rete). Per informazioni sulla configurazione di questo file leggere il file installationguide_e.pdf nella documentazione di OPEN-R.

1.1.2. Caratteristiche protocollo IEEE 802.11b

Il protocollo **IEEE 802.11 pubblicato nel 1996** definisce lo standard IEEE per le comunicazione di rete Wireless.

Per il livello Data Link utilizza il **CSMA/CA**, *Carries Sense Multiple Access with Collision Avoidance*, che dovrebbe garantire l'assenza di collisioni di pacchetti durante la comunicazione; mentre come livello di accesso al mezzo fisico permette tre possibilità FHSS e DSSS, via radio, e Infrared.

Il protocollo IEEE 802.11 definisce due tipologie di utilizzo: la modalità **Ad-hoc** attraverso la quale le varie schede di rete comunicano tra di loro direttamente senza passare attraverso apparecchiature esterne o centralizzate e la modalità **Infrastructured** nella quale un Access Point funge da nodo centrale della rete, tutte le schede di rete wireless per comunicare tra di loro manderanno i pacchetti all'Access Point che li smisterà poi alle schede di destinazione; l'Access Point può essere visto come il rispettivo di un hub nelle reti wired con la differenza che un è tecnicamente molto più complesso. Un'altra importante funzione dell'Access Point è quella di fare da *bridge* con altre reti sia wired sia wireless, spesso infatti viene collegato alla rete ethernet per permettere alle postazioni mobili wireless di accedere alla rete fissa dell'edificio.

La sostanziale differenza tra le due tipologie e' che la modalità Ad-hoc impone dimensioni ristrette alla rete wireless e un minor controllo sul traffico trasitante sulla rete.

L'IEEE 802.11 fornisce poi la possibilità di cifrare i dati attraverso il protocollo **WEP**, Wired Equivalent Privacy e utilizza un protocollo **RTS/CTS** per evitare il problema del "*nodo fantasma*": supponiamo di avere tre host, A, B e C, A e B si vedono reciprocamente, come anche B e C, A, di conseguenza, non vedrà C; se A mandasse dei dai a B che sta comunicando con C interferirebbe con la comunicazione quindi se A deve comunicare con B gli manda prima un RTS e attende di risposta un CTS prima di iniziare a mandare i dati.

Sono state poi pubblicate altre versioni del protocollo, tra le quali l'IEEE 802.11b, il protocollo utilizzato dalla scheda di rete degli Aibo; tale protocollo è diventato "famoso" con il nome di Wi-Fi e sostanzialmente implementa l'802.11 con una velocità massima di trasmissione di 11Mbit/s.

1.1.3. Hardware necessario per strutturare una Rete di Aibo

Per poter sfruttare le potenzialità offerte dalla scheda di rete integrata negli Aibo, il requisito minimo è di disporre di una scheda di rete wireless (compatibile con il protocollo IEEE 802.11b) da installare su un PC e impostare una comunicazione **AdHoc** (voce *APMODE* di WLANCONF uguale a 0), cioè una comunicazione peer-to-peer tra una scheda Wireless ed un'altra, in questo

caso è necessario impostare su entrambe le schede, quella sul PC e quella sull'Aibo, lo stesso canale di comunicazione Wireless (voce *CHANNEL* nel file WLANCONF).

Per ridurre i problemi di dimensioni della rete e di diminuzione delle prestazioni indotte dall'AdHoc mode è necessario un AccessPoint (che supporti il protocollo IEEE 802.11b e il WEP nel caso si voglia cifrare i dati scambiati), che può essere considerato il corrispettivo di un Hub per le reti wireless, e impostare sia l'AccessPoint sia gli Aibo in modalità **Infrastructured** (voce *APMODE*=1 nel file WLANCONF.TXT). Per poter poi accedere alla rete anche attraverso un PC sarà necessaria un'ulteriore scheda di rete Ethernet tradizionale da collegare all'Access Point oppure una scheda di rete wireless che comunicherà con gli Aibo collegandosi via radio all'Access Point.

E' consigliabile poter accedere alla rete attraverso un PC sia per poter sfruttare la console Telnet offerta da OPEN-R, molto utile per fare il debug degli oggetti caricati sull'Aibo, sia in caso si intenda utilizzare il Remote-Processing (guarda TCPGatway).

1.1.4. Esempio di configurazione di Rete

Come esempio di rete wireless di Aibo viene riportata la rete che è stata strutturata nel AIS-Lab (Applyed Inteligent System Laboratory) del DSI.

Nel laboratorio è presente una rete Ethernet, mentre la rete Wireless è composta da:

- un Access Point IEEE 802.11b.
- un Server, Posidone, con due schede di rete, una verso la rete del laboratorio e una verso l'Access Point, e attraverso questo, verso la rete wireless di Aibo chiamata **Aibonet**.

Di seguito un diagramma della rete:

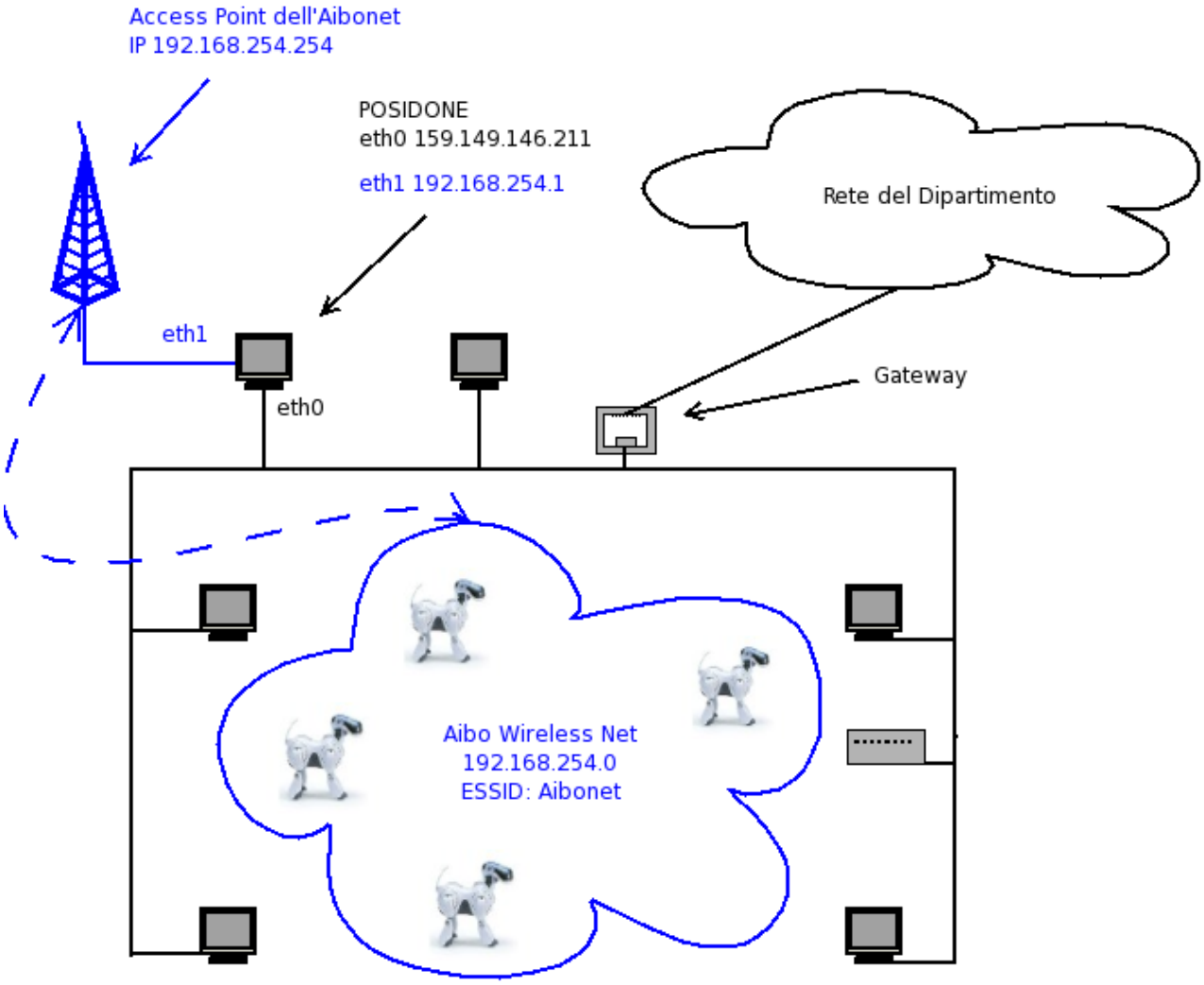


Figura 1 Diagramma della Rete dell'AIS-Lab

Configurazione:

Posidone: monta due schede di rete, una PCI verso la rete Ethernet di laboratorio (ETH0, IP 159.149.146.211) e una USB verso l'Access Point (ETH1, IP 192.168.254.1; netmask 255.255.255.0; gateway 159.149.146.211). Lavora in WIN2000 Professional con l'*IP Forwarding Attivato*[1], funzionalità che permette a Posidone di spedire un pacchetto proveniente dalla scheda di rete ETH0 verso la ETH1, quindi verso la rete a cui essa è collegata, e viceversa.

Access Point: è collegato alla scheda USB di Posidone. Il suo IP è 192.168.254.254 (netmask 255.255.255.0; gateway 192.168.254.1) è l'HUB per la rete wireless 192.168.254.0 (la rete degli Aibo) identificata (ESSID) dal nome Aibonet.

Aibo: Gli Aibo utilizzeranno l'intervallo di IP 192.168.254.2-253 (netmask 255.255.255.0; gateway 192.168.254.1), non è stata attivata la cifratura dei dati scambiati e sfrutteranno il server Posidone come gateway per accedere alla Rete del Laboratorio. Di seguito è riportato il file WLANCONF.TXT associato ad uno dei robot AIBO:

```
HOSTNAME=AIBO                    /*Nome dell'host*/
```

[1] Purtroppo per motivi non del tutto chiari al mondo informatico, per attivare l'IP Forwarding in Win2000 è necessario apportare delle modifiche al Registro.

```
ETHER_IP=192.168.254.2       /*Indirizzo IP dell'AIBO*/
ETHER_NETMASK=255.255.255.0  /*Netmask*/
IP_GATEWAY=192.168.254.1     /*IP del Gateway*/
ESSID=Aibonet                /*Nome della rete Wireless a cui collegarsi*/
WEPENABLE=0                  /*Cifratura Web disabilitata*/
WEPKEY=AIBO2                 /*Password per la cifratura dei dati*/
APMODE=1                     /*Modalità Infrastructured*/
CHANNEL=3                    /*Canale da usare nella modalità AD-HOC*/
```

Le due voci più importanti sono ETHER_IP che specifica l'IP dell'Aibo e ESSID che identifica il nome della rete Wireless.

AIBO diversi avranno lo stesso file WLANCONF.TXT ad eccezione che per la stringa HOSTNAME e ETHER_IP.

Altri PC: avendo attivato l'IP Forwarding sul server è possibile permettere agli altri PC di accedere ad Aibonet sfruttando Posidone come un router. Per farlo è necessario aggiungere una *route* che instradi i pacchetti destinati alla rete 192.168.254.0 verso la scheda di rete ETH1 di Posidone con il comando:

```
ROUTE ADD 192.168.254.0 MASK 255.255.255.0 192.168.254.1 (in Window)
route add -net 192.168.254.0 netmask 255.255.255.0 192.168.254.1 (in Linux)
```

Per impostare il comando ROUTE in Windows, si può utilizare la Dos Shell (CMD).

1.2 Servizi e API offerte da OPEN-R per la gestione di Rete

OPEN-R offre tutti i tool necessari per gestire lo stack del protocollo **Ipv4** (TCP-IP, versione 4, indirizzi a 32 bit), il quale è implementato attraverso l'OPEN-R Networking Toolkit **ANT**. ANT è una serie di librerie che devono essere incluse nel progetto.

Runtime lo stack è rappresentato **dall'Oggetto di sistema IPStack** che include l'ambiente di gestione di rete ANT. Gli Oggetti OPEN-R **possono usare i servizi di rete offerti dallo stack TCP/IP** sfruttando *inter-object comunication*, **mandando e ricevendo speciali messaggi all'oggetto IPStack.**

Il primo passo che un oggetto deve fare per poter sfruttare l'IPV4 è richiedere a IPStack di creare un **Endpoint**; un endpoint è uno speciale costrutto di ANT che è posizionato sulla cima dello stack del protocollo ed è responsabile della comunicazione tra l'oggetto e lo stack stesso; un Endpoint puo' essere vista come una socket di Unix.

ANT offre la possibilità di gestire buona parte dei protocolli offerti da IPV4 tra cui: il TCP, l'UDP, il DNS, il DHCP e l'IP. La documentazione su come gestire tali protocolli è contenuta nel file InternetProtocolVersion4_E.pdf presente nella documentazione di OPEN-R.

Oltre alla gestione dell'IPV4 OPEN-R offre alcune API per la gestione della scheda wireless contenute nella libreria ERA201D1 tra le quali: ERA201D1_GetMACAddress(), ERA201D1_GetWLANSetting() e ERA201D1_GetEtherStatics(). Per informazioni su questa libreria guardare il file Level2ReferenceGuide_E.pdf contenuto nella documentazione di OPEN-R ed è consigliabile guardare anche il *common sample ERA201D1Info*, contenuto negli esempi di OPEN-R, sia la documentazione sia i sample possono essere scaricati dal sito ufficiale di Open-r della Sony.

2. TCPGatway

Il **TCPGateway** è un oggetto OPEN-R implementato sostanzialmente per permettere il **Remote Processing**: la possibilità di fare eseguire parte dell'elaborazione richiesta da un Aibo ad un PC collegato alla rete wireless. Il TCPGatway permette uno scambio di messaggi tra gli oggetti

OPEN-R operanti sull'Aibo e gli oggetti OPEN-R operanti sul PC sfruttando i normali metodi dell'Inter-object comunication, facendoli però transitare sulla rete.

E' possibile anche sfruttare il TCPGateway per connettere più Aibo via rete ad un PC che funge da Proxy, generando così una rete a stella, in cui il centro è il PC. Il PC ha quindi il compito di ricevere i messaggi da un Aibo e inoltrarli all'Aibo di destinazione sfruttando il protocollo TCP. Per farlo un Aibo manderà il messaggio da spedire al suo oggetto **TCPGatway,** questo inoltrerà il messaggio ad un altro oggetto TCPGatway operante sul PC Proxy. Questo PC inoltrerà a sua volta il messagio all'oggetto TCPGatway operante sull'Aibo di destinazione che manderà il messaggio all'oggetto OPEN-R di destinazione (guarda Figura 2).

Prima di proseguire nella spiegazione di come utilizzare questo metodo bisogna considerare che i vari passaggi che un messaggio subisce dall'oggetto mittente all'oggetto destinatario causano una latenza che nel caso di messaggi di grosse dimensioni non è da sottovalutare (si sono osservati tempi di spedizione di secondi in reti disturbate), anche se l'ultima versione di OPEN-R (1.1.5-r3) ha ridotto in parte questa latenza.

Questa modalità di funzionamento apparenetemente funziona solamente sotto Linux, il tentativo di lanciare l'esecuzione del remote processing sotto Window genererà l'errore:

```
msgget: Function not implemented
```

2.1. Configurazione degli Aibo:

Per poter utilizzare il TCPGateway è necessario, prima di tutto, utilizzare la configurazione di OPEN-R **WCONSOLE->nomemprot**, questa è l'unica configurazione di Open-R in cui è presente l'oggetto TCPGateway (cf. Manuale Introduttivo).

Supponiamo di voler far comunicare due Aibo (Aibo 1 e Aibo 2) di IP 192.168.254.2 e .3. In ogni Aibo sarà in esecuzione un oggetto MainObject che dovrà spedire un messaggio, un double, all'altro Aibo sfruttando il TCPGatway.

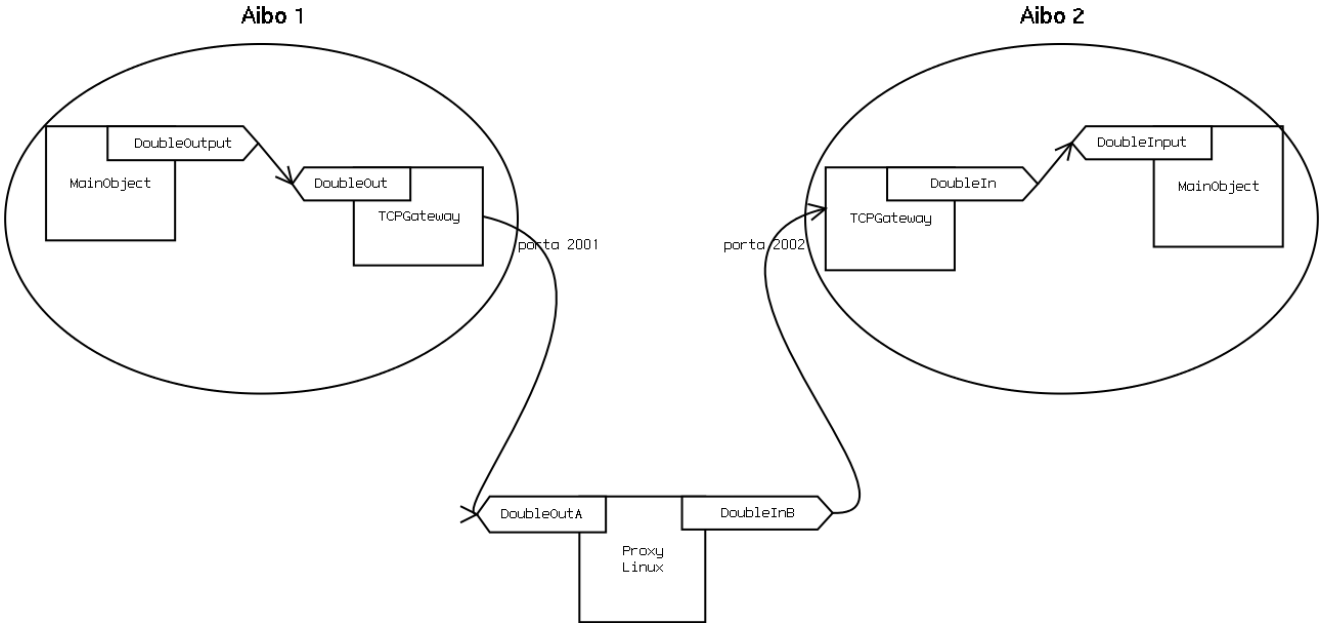


Figura 2 Comunicazione via TCPGateway. Aibo 1 spedisce un messaggio double a Aibo 2

In ogni Aibo bisognerà prima di tutto modificare il file /MS/OPEN-R/MW/CONF/OBJECT.CFG e aggiungere la riga **/MS/OPEN-R/SYSTEM/OBJS/TCPGW.BIN** che caricherà l'oggetto TCPGateway all'avvio dell'Aibo.

Successivamente verranno definiti i vari servizi offerti da MainObject nel file stub.cfg, **servizi che verranno poi collegati con l'oggetto TCPGateway** nel file /MS/OPEN-R/MW/CONF/CONNECT.CFG.

Se, per esempio, un oggetto presente su un AIBO che chiameremo, MainObject, offre i seguenti servizi:

```
MainObject.DoubleOutput.DOUBLE.S
MainObject.DoubleInput.DOUBLE.O
```

Questi verranno connessi nel file CONNECT.CFG come segue:

```
MainObject.DoubleOutput.DOUBLE.S TCPGateway.DoubleOut.DOUBLE.O
TCPGateway.DoubleIn.DOUBLE.S MainObject.DoubleInput.DOUBLE.O
```

E' da notare che essendo TCPGateway un oggetto di sistema non sarà possibile modificare il suo file stub.cfg, che risulta definito internamente nel sistema. Sarà quindi necessario un altro metodo per dichiarare i servizi messi a disposizione; questi saranno dichiarati in un file esterno: /MS/OPEN-R/MW/CONF/ROBOTGW.CFG che ha una sintassi particolare, ben diversa dalla sintassi del file stub.cfg. Viene riportato di seguito un esempio:

```
TCPGateway.Proxy.AperiosMessage.P 1080

TCPGateway.DoubleOut.DOUBLE.O 2001
TCPGateway.DoubleIn.DOUBLE.S 2002
```

Verranno dichiarati i servizi di nome DoubleOut e DoubleIn che saranno collegati rispettivamente con le porte TCP/IP 2001 e 2002; gli unici vincoli che si devono seguire per la scelta del numero da utilizzare sono: devono essere univoci, ogni servizio deve avere la sua porta, bisogna utilizzare numeri superiori a 1024 non si può utilizzare il numero 1080. La prima riga è necessaria per attivare la comunicazione attraverso un Proxy, la seconda riga può essere intesa come: "Inoltra tutto quello che arriva al servizio di TCPGateway DoubleOut nello stream TCP connesso alla porta 2001" e la terza come "Inoltra i dati provenienti dallo stream TCP di porta 2002 verso l'observer collegato al servizio TCPGateway DoubleIn".

L'oggetto MainObject per la comunicazione via rete si interfaccerà solo all'oggetto TCPGateway dell'Aibo stesso, di conseguenza il ricevimento da parte di MainObject di un AR da TCPGateway non significa che è stata instaurata una connessione con il Proxy o con l'altro Aibo, ma significa semplicemente che l'oggetto TCPGateway, o meglio i suoi servizi observer sono pronti a ricevere un messaggio; **come conseguenza bisognerà implementare un metodo per poter controllare se l'altro Aibo è in ascolto o meno**.

2.2. Configurazione del Proxy Linux:

Per configurare il Proxy sotto Linux è necessario creare sul PC Proxy, la struttura della directory normalmente presente in una memory stick. Nel nostro esempio ipotiziamo di lavorare nella directory /proxy, in questo caso sarà sostanzialmente necessario creare la directory /proxy/MS/OPEN-R/MW/CONF/ nella quale dovranno essere presenti i file HOSTGW.CFG e CONNECT.CFG.

Il file corrispettivo di ROBOTGW.CFG nel proxy Linux è HOSTGW.CFG che ha il compito di definire i servizi offerti da TCPGatway ed ha una sintassi come di esempio:

```
!ROBOT_PROXY 1080 192.168.254.2
!ROBOT_PROXY 1080 192.168.254.3
```

```
TCPGateway.DoubleOutA.DOUBLE.S 2001 192.168.254.2
TCPGateway.DoubleInA.DOUBLE.O 2002 192.168.254.2

TCPGateway.DoubleOutB.DOUBLE.S 2001 192.168.254.3
TCPGateway.DoubleInB.DOUBLE.O 2002 192.168.254.3
```

E' da notare che le porte utilizzate in questo file sono le stesse utilizzate nel file ROBOTGW.TXT. E' quindi necessario modificare il file CONNECT.CFG nel modo seguente:

```
TCPGateway.DoubleOutA.DOUBLE.S TCPGateway.DoubleInB.DOUBLE.O
TCPGateway.DoubleOutB.DOUBLE.S TCPGateway.DoubleInA.DOUBLE.O
```

Che unisce nella prima riga il servizio DoubleOutA con DoubleInB, quindi permetterà all'Aibo 1 di mandare un messaggio ad Aibo 2 e viceversa, la seconda, permetterà all'Aibo 2 di mandare un messaggio all'Aibo 1.

Come detto l'oggetto TCPGateway è stato implementato per permette il Remote Processing e di conseguenza per poterlo utilizzare è necessario attivarlo. Per farlo, nel caso non sia già stato attivato, è necessario eseguire il comando */usr/local/OPEN_R_SDK/RP_OPEN_R/bin/setup-rp-openr* che ha il compito compilare i file necessari per l'esecuzione del Remote Processing.

A questo punto i passi da seguire sono:

1 Accendere i vari Aibo.

2 Andare nella direcory /proxy del Proxy Linux.

3 Lanciare *start-rp-openr* (presente in /usr/local/OPEN_R_SDK/RP_OPEN_R/bin)

A questo punto la comunicazione è attiva e si potranno vedere i vari IP degli Aibo connessi dal gateway. Di seguito un esempio di schermata del Proxy:

```
[gilberto@zeus proxy]$ /usr/local/OPEN_R_SDK/RP_OPEN_R/bin/start-rp-openr
[pid:1511,msqid:1114114,oid:0x00110002] oserviceManager
[pid:1512,msqid:1146883,oid:0x00118003] tcpGateway
TCPGW connecting to 192.168.254.4 Port:2001
TCPGW connecting to 192.168.254.4 Port:2002
TCPGW connecting to 192.168.254.4 Port:2001
observer[1] new connection established!
subject[0] new connection established!
TCPGW connecting to 192.168.254.3 Port:2002
observer[0] new connection established!
subject[1] new connection established!
```

Per chiudere le connessioni è sufficiente, sul Proxy, eseguire i seguenti passi:

1 Terminare il processo con *CTRL-C*

2 Lanciare il programma */usr/local/OPEN_R_SDK/RP_OPEN_R/bin/rp-openr-ipcrm* che ha il compito di pulire le varie risorse occupate e non liberate dal Remote Processing.

2.3. Utilizzo

Una volta configurati i vari servizi resi disponibili dai vari oggetti TCPGateway e MainObject, Aibo1 potrà mandare un messaggio ad Aibo2 come segue:

```
double dummy=3.1415;
subject[sbjDoubleOutput]->SetData(&dummy, sizeof(dummy));
subject[sbjDoubleOutput]->NotifyObservers();
```

Che manderà il messaggio, il double dummy, a tutti i servizi connessi al subject DoubleOutput e di conseguenza, guardando il file CONNECT.CFG, manderà il messaggio al servizio

TCPGateway.DoubleOut.DOUBLE.O il quale lo spedirà alla porta 2001 del proxy Linux. Guardando il file HOSTGW.CFG e CONNECT.CFG nel proxy, i dati ricevuti sulla porta 2001 dall'Aibo 1 (TCPGateway.DoubleOutA.DOUBLE.S 2001 192.168.254.2) verranno mandati al servizio TCPGateway.DoubleInB.DOUBLE.O, quindi verranno mandati alla porta 2002 dell'Aibo 2, dove l'oggetto TCPGateway rispedirà il messaggio al MainObject.

Al ricevimento del messaggio sull'Aibo2 Aperios eseguirà il metodo associato al servizio *DoubleInput* passandogli come argomento, per esempio *OsystemEvent& event*, il messaggio stesso.

Anche utilizzando il TCPGateway sarà possibile utilizzare, come si vedrà in seguito, i TelepatiPkg come tipo di messaggi da utilizzare durante la comunicazione.

3. Telepati

3.1. Descrizione

Telepati è un oggetto OPEN-R che si occupa della comunicazione via rete Wireless tra Aibo, comunicazione **message-oriented** e orientata alla cooperazione, di conseguenza è ottimizzata per lo scambio di brevi messaggi del tipo: *"dove ti trovi?", "vai alla posizione x,y", "passami la palla"*, ma puo' essere utilizzato anche per lo streaming di dati.

La struttura di Telepati è basata su alcuni aspetti ritenuti importanti:

- La possibilità di essere facilmente **riutilizzabile** senza dover apportare modifiche al codice.
- La **semplicità** di utilizzo.
- Il **risparmio** delle risorse utilizzate.
- La **libertà** di comunicazione, quindi l'assenza di strutture esterne per la comunicazione, come la presenza di un Proxy o di un server centrale.
- La possibilità di **espandere** le sue funzionalità.

Non si basa sul paradigma delle reti Client/Server; in Telepati non esistono server, ogni Aibo può comunicare con gli altri autonomamente.

E' necessario distinguere due concetti fondamentali utilizzati in Telepati, i package ed i message:

- **package**: sono oggetti, rappresentano il messaggio che dev'essere mandato in rete; forniscono metodi utili alla loro gestione
- **message**: sono degli array e sono il messaggio che effettivamente verrà mandato nella rete e che verrà scambiato tra Telepati e l'oggetto OPEN-R a lui collegato.

I messaggi che vengono spediti in rete e che vengono scambiati tra gli oggetti sono tutti dei TelepatiPkg (ereditano tutti da questa classe). In base all'ambiente ed allo scopo in cui si intende utilizzare Telepati è possibile, anzi necessario**, costruire un pool di package** per implementare l'effettivo protocollo di comunicazione tra gli Aibo.

Sostanzialmente Telepati è una sorta di Framework che si occupa della spedizione di messaggi tra Aibo curando tutti gli aspetti della spedizione, ma senza prendere in considerazione il tipo di messaggio speditio sarà il progettista che, in base all'ambiente ed al tipo di compito che dovranno svolgere gli Aibo, implementerà i package necessari al suo scopo.

Sono state implementate due versioni di Telepati, la versione UDP chiamata TelepatiUdp (guarda 3.4.)e una versione Tcp chiamata TelepatiTcp (guarda 3.5.).

3.2. Tipi di package e loro header

Tutti i package utilizzati da Telepati ereditano dalla classe TelepatiPkg che rappresenta la struttura di base di un package e permette il *Polimorfismo*, questo per offrire la possibilità' di implementare messaggi di rete in maniera piu' semplice e senza dover modificare il codice di Telepati (guarda Aggiungere tipi di pacchetti per la creazione di package personalizzati).

Lo standard header, l'header che tutti i message conterranno e che contiene le informazioni di base per la spedizione del messaggio, è composto dai seguenti campi:

byte	*byte*	*Short*	*Byte*	*Short*	*Bytes...*
Pkg type	Aibo dest	Size of message	Need ack	Serial number	Pkg data...

- **Pkg type**: tipo di package (da 0 a 9 riservati a Telepati per i messaggi di controllo).
- **Aibo dest**: indice dell'Aibo di destinazione.
- **Size of message**: dimensione in byte del message da spedire (compresa la dimensione dello standard header).
- **Need ack**: 0 se non è richiesto l'Acknowledgment, >0 se richiesto.
- **Serial number**: numero di serie che rappresenta il message (16 bit), utilizzato nella versione UDP per la gestione degli Acknowledgment.
- **Pkg data**: i dati da spedire (sono possibili altri header specifici impostati dai package figli).

I package utilizzati da Telepati sono:

- **AiboInfoPkg**: contiene informazioni sugli Aibo presenti nella rete, numero di Aibo presenti e loro stato di attività (non circola sulla rete).
- **HiPkg**: messaggio di saluto necessario per saggiare lo stato di attività' degli Aibo.
- **HiReplyPkg**: messaggio di risposta al saluto, viene mandato in risposta al ricevimento di un HiPkg.
- **ByePkg**: messaggio di arrivederci, viene mandato da Telepati durante il suo spegnimento per informare gli altri Aibo del suo futuro stato di inattività.
- **AckPkg**: messaggio di Acknowledgment, viene spedito da Telepati, versione UDP, in risposta ad un messaggio che richiede conferma.
- **CounterPkg**: un messaggio che contiene informazioni sul numero di messaggi ricevuti e spediti (non circola sulla rete).
- **DummyPkg**: package dummy usato per scopi di debug e controllo (contiene la stringa "Dummy message").
- **DummyLenPkg**: package dummy usato per scopi di debug e controllo (accende i Led sul muso).

A parte l'AiboInfoPkg che viene utilizzato per dare informazioni all'observer sugli Aibo presenti in rete e sul loro stato di attività, CounterPkg che offre informazioni statistiche sulla comunicazione e i package Dummy (DummyPkg e DummyLedPkg) che vengono utilizzati solo per scopi di testing, gli altri package sono utilizzati internamente da Telepati, non saranno quindi visibili agli oggetti OPEN-R che intendono utilizzare Telepati.

Per maggiori informazioni riguardo la struttura e l'utilizzo dei TelepatiPkg guardare Aggiungere tipi di pacchetti e la documentazione della libreria Packages in /lib/Packages/Doc/html.

3.3. Configurazione della rete

Il primo passo è configurare correttamente l'accesso alla rete dell'Aibo attraverso il file /MS/OPEN-R/SYSTEM/CONF/WLANCONF.CFG[2].

E' quindi necessario modificare il file /MS/OPEN-R/MW/CONF/AIBONET.CFG; questo file rappresenta tutti gli Aibo presenti in rete con i quali si intende comunicare ed è composto da una lista degli indirizzi IP dei vari aibo con cui comunicare, uno per riga, compreso l'IP dell'Aibo stesso su cui gira Telepati, questo per permettere di utilizzare lo stesso file su tutti gli Aibo del gruppo.

3.4. Telepati Udp

3.4.1. Descrizione

TelepatiUdp si basa sul protocollo **UDP** scelta effettuata sia per risparmiare risorse, che nel caso di agenti mobili è meglio dedicare principalmente a funzioni di elaborazione video e planning, sia perchè meglio si adatta a modellare uno scambio di brevi messaggi tra agenti che cooperano, messaggi del tipo *richiesta – risposta*.

Come si vedrà più avanti si è osservato sperimentalmente che il numero di messaggi persi durante la trasmissione è relativamente alto, questa è la motivazione che ha spinto da un parte la realizzazione di una versione Tcp di Telepati e dall'altra l'aggiunta della **possibilità di richiedere un ack** attivabile alla creazione del messaggio da spedire.

E' da notare che basandosi sul protocollo UDP la spedizione di messaggi **senza la richesta di Ack**, non garantisce né l'**effettiva ricezione** del messaggio da parte del destinatario né il rispetto dell'**ordine di consegna**. Bisogna però considerare che il percorso svolto dal messaggio dall'Aibo mittente all'Aibo di destinazione è sostanzialmente breve e non deve passare attraverso altri server, di conseguenza si è osservato sperimentalmente che l'ordine di consegna viene sempre rispettato, stessa cosa non si puo' dire per l'effettiva ricezione, come già detto il numero di messaggi persi è molto alto.

L'utilizzo di messaggi **con richiesta di ack** risolve molti dei problemi dell'Udp; in questa modalita' vi è garanzia dell'effettiva ricezione del messaggio e vi è garanzia che l'ordine di consegna venga rispettato.

[*** aggiungi parte riguardante la Fuzzy]

2 Per informazioni sulla configurazione di questo file leggere il file installationguide_e.pdf nella documentazione di OPEN-R.

3.4.2. Come funziona

All'avvio TelepatiUdp apre tre connessioni, una in ricezione, ascoltando sulla porta 3030[3], e due in spedizione, una per la spedizione dei messaggi di **controllo** (tipo messaggi di Ack, messaggi di saluto...) e una per la spedizione dei messaggi di **dati**, questo a prescindere dal numero di Aibo presenti in rete. Leggendo il file di configurazione AIBONET.CFG ricava gli indirizzi IP degli Aibo con cui può comunicare quindi manda ad ognuno di questi un **Hi message**, un messaggio di saluto che serve sia ad informare l'Aibo di destinazione della sua presenza, sia per saggiare lo stato di attività dell'Aibo di destinazione. Al ricevimento di un Hi message, infatti, TelepatiUdp sull'AIBO che riceve il messaggio, risponderà con un **HiReply** message, un messaggio di risposta al saluto. E' possibile impostare TelepatiUdp in modo che effettui un polling continuo della presenza degli altri Aibo (guardare Feature: Polling degli Aibo).

Durante il normale funzionamento TelepatiUdp tiene aggiornato un array contenente lo stato di attività di ogni altro Aibo presente in rete e, in caso avvengano modifiche nello stato di un destinatario, ne informa l'observer attraverso un **AiboInfoPkg** (guarda Utilizzo per ulteriori informazioni).

E' possibile impostare un package in modo da richiedere una conferma dell'avvenuta ricezione da parte del destinatario, attivando all'atto della costruzione del package la **richiesta di Ack**, in questo caso TelepatiUdp dopo aver spedito il package attenderà la ricezione di un messaggio di conferma Ack, se non viene ricevuto TelepatiUdp rispedirà il package e continuerà a rispedirlo fino al ricevimento dell'Ack. La spedizione di un messaggio con richiesta di Ack blocca la connessione di spedizione dati fino al ricevimento del Ack stesso, resta attiva la connessione di controllo per permettere agli oggetti TelepatiUdp degli Aibo di gestire la rispedizione del messaggio.

Allo **spegnimento** dell'Aibo verranno mandati dei Bye message a tutti gli Aibo presenti in rete per informarli della sua intenzione di scollegarsi dalla rete; alla ricezione di un Bye message verrà modificato l'array contenente lo stato di attività degli Aibo in rete impostando come *byed* (quindi inattivo) l'Aibo che lo ha spedito, questo per evitare possibili errori a livello di planner causati da una scorretta conoscenza degli agenti con cui cooperare.

I messaggi scambiati via rete e scambiati attraverso l'Inter-object comunication sono contenuti nella libreria Packages (in /TelepatiUdp/Packages).

3.4.3. Strutura dei sorgenti

Di seguito è riportata la struttura della directory di TelepatiUdp:

```
+-- MS \
 |      |
 |     +-- OPEN-R \
 |                 |
 |                +-- MW \
 |                        |
 |                        +-- CONF \
 |                        +-- OBJS \
```

3 Il numero di porta è specificato nel file Telepati.h dalla costante TELEPATI_PORT

```
 |
+-- TelepatiUdp \
       |
      +-- Doc \
```

In [TelepatiUdp] è contenuto il codice sorgente dell'oggetto OPEN-R. I file sono:

- Makefile: il makefile di Telepati.
- TelepatiUdp.h: l'header file di Telepati.
- TelepatiUdp.cc: il sorgente C++ di Telepati.
- stub.cfg: il file stub.cfg di Telepati per la dichiarazione dei servizi OPEN-R.
- telepatiudp.ocf: imposta alcune caratteristiche dell'oggetto tra cui l'esecuzione il Kernel o User mode, la dimensione dello stack, dello Heap…
- UDPConnection.h: header di UDPConnection.cc, contiene gli oggetti UDPConnection, la classe base, e UDPRecvConnection e UDPSendConnection, classi che ereditano da UDPConnection, che raprresentano le connessioni UDP in entrata e in uscita e offre i metodi per inizializzare le connessioni.
- UDPConnection.cc: Implementa gli oggetti UDPConnection e UDPRecvConnection e UDPSendConnection.
- telepatiudp_doc.cfg: il file di configurazione di Doxygen per la generazione della documentazione della classe.
- EtherStat.h: header file della classe EtherStat per la stampa delle statistiche di Rete (guarda Stampare informazioni sulla rete)
- EtherStat.cc: sorgente della classe EtherStat.

In [Doc] è contenuta la documentazione del codice di TelepatiUdp in formato HTML generata automaticamente da **Doxygen**[4].

In [MS] è contenuta la struttura delle directory da copiare nella memory stick. Più in dettaglio in MS/OPEN-R/MW/CONF saranno presenti i file:

- CONNECT.CFG: connette i vari servizi dei vari oggetti caircati sull'Aibo.
- OBJECT.CFG: elenca gli oggetti da caricare all'avvio dell'Aibo.
- AIBONET.CFG: una lista di indirizzi IP degli Aibo con cui comunicare.

Nella directory MS/OPEN-R/MW/OBJS sono presenti gli oggetti da caricare all'avvio dell'Aibo, tra i quail l'oggetto **telepati.bin**.

Per compilare TelepatiUdp è sufficiente andare nella direcotory TelepatiUdp\ lanciare il comando **make**. Lanciando il comando **make install** verrà copiato il file telepati.bin nella direcotory MS/OPEN-R/MW/OBJS. Lanciando il comando **make clean** verranno cancellati i vari file creati durante la compilazione lasciando intatti i vari sorgenti. Con il comando **make doc** viene rigenerata la documentazione di TelepatiUdp (in questo caso è necessario aver installato e configurato Doxygen).

4 Applicazione per la generazione automatica della documentazione della classe a partire dai commenti presenti nel codice. Distribuito con licenza GPL e scaricabile liberamente da www.doxygen.org.

3.4.4. Feature: Polling degli Aibo

E' possibile, in fase di compilazione, impostare TelepatiUdp in modo che effettui un **Polling** continuo degli altri Aibo in rete per rilevare il loro stato di attività. Questa feature è utile nel caso in cui l'ambiente in cui agiscono gli Aibo sia ambio e quindi si corre il rischio di uscire dal campo d'azione dell'Access Point rendendo quindi impossibile lo scambio di messaggi.

Il Polling viene attivato in fase di compilazione definendo TELEPATI_POLLING (opzione gcc -DTELEPATI_POLLING, nel Makefile è presente commentata l'opzione per attivarlo). Il polling consiste nel mandare ogni 5 secondi[5] un Hi message a tutti gli Aibo specificati nel file AIBONET.CFG e attendere il ricevimento di un HiReply message. Lo stato degli Aibo è memorizzato in un array di aiboState chiamato **aibolpActive**. Per chiarire i possibili passaggi di stato di un Aibo, di seguito è riportato l'Automa a Stati Finiti degli stati degli aibo (per chiarezza del diagramma non sono stati riportati gli eventi che non modificano lo stato corrente):

STATI		**EVENTI**	
Nome	**Descrizione**	**Nome**	**Descrizione**
INACTIVE	Aibo inattivo	Recv.msg	Ricevuto un messaggio dall'Aibo
ACTIVE	Aibo Attivo	Recv.Hi	Ricevuto un Hi message dall'Aibo
WAITINGREPLY	In attesa di un HiReply dall'Aibo	Recv.reply	Ricevuto un HiReply dall'Aibo
BYED	L'Aibo ha spedito un Bye message	Recv.bye	Ricevuto un By message dall'Aibo
		Send.Hi	Spedito un Hi message all'Aibo
		BoneTimer	Scaduto il timer di attesa della risposta
		Network error	Errore di rete

5 L'intervallo di tempo è specificato dalla variabil TELEPATI_BONE_TIME nell'header file di Telepati.

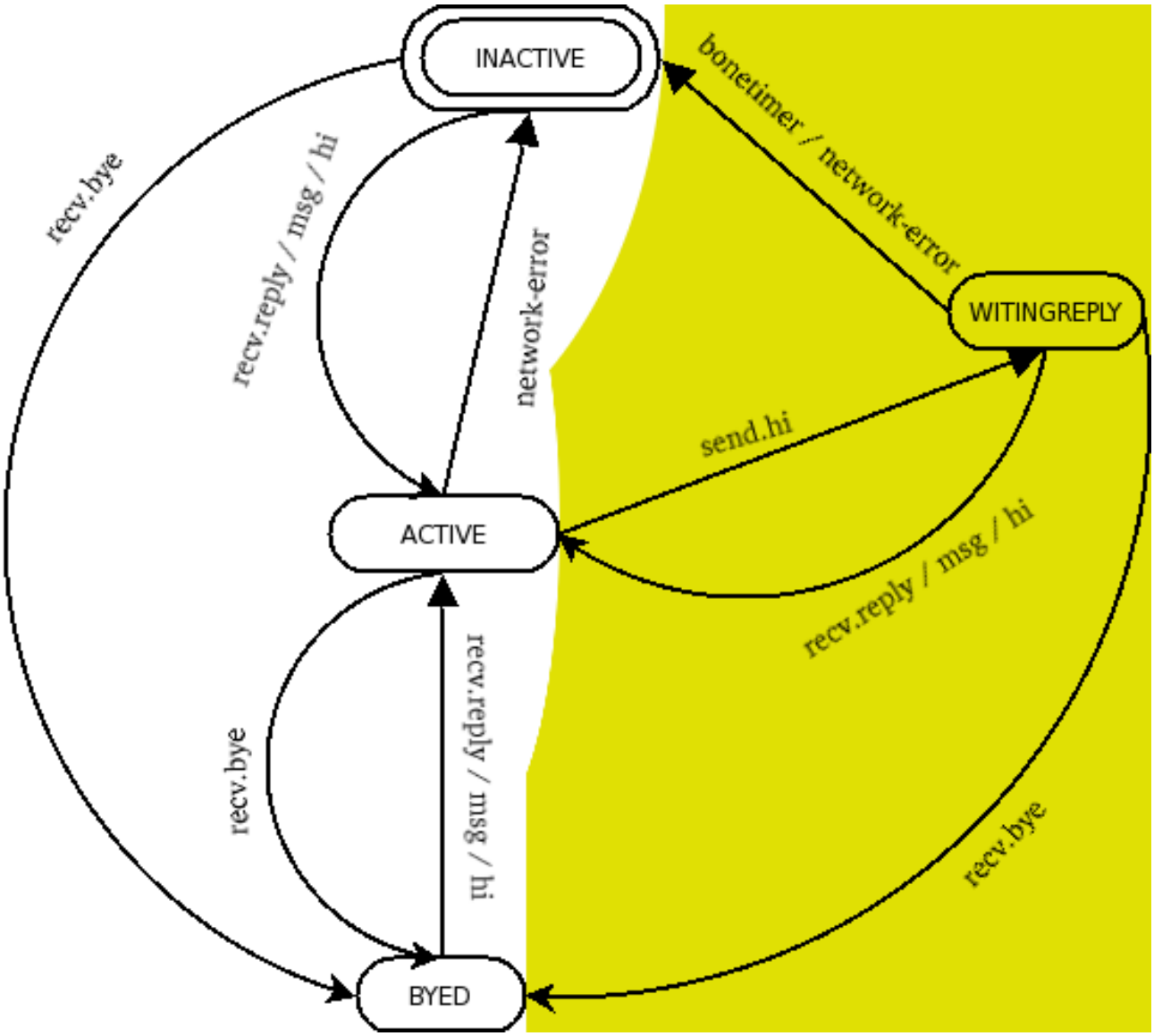


Figura 3 Automa degli Stati degli Aibo

E' da notare che gli stati ACTIVE e WAITINGREPLY sono stati attivi, se un Aibo si trova in uno di questi stati viene considerato attivo, mentre INACTIVE e BYED sono due stati di inattività.

Nel caso in cui non venga attivato il Polling, l'Automa che rappresenta gli stati dell'Aibo corrisponde alla parte con sfondo bianco della figura.

3.4.5. Servizi offerti da Telepati Udp

I servizi messi a disposizione da TelepatiUdp sono:

- Subject= *TelepatiUdp.ReceiveMsg.byte.S* : per la ricezione dei messaggi
- Observer= *TelepatiUdp.SendMsg.byte.O* : per la spedizione dei messaggi

3.5. Telepati Tcp

3.5.1. Descrizione

La versione Tcp è stata implementata sostanzialmente per ovviare ai problemi di perdita dei pacchetti che si è riscontrata con la versione Udp.

Anch'essa utilizza i TelepatiPkg come struttura dei messaggi scambiati, e a parte le differenze implementative da un punto di vista dell'utilizzo è del tutto simile alla versione Udp.

E' stato in parte impegnativa la gestione dei messaggi utilizzando il protocollo TCP. Stando all'RFC 713 il protocollo TCP è *orientato ai dati* di conseguenza tale protocollo accumula byte e li spedisce quando lo ritiene opportuno, è stato necessario quindi implementare alcune funzioni che permettano di ricostruire il messaggio nel caso venga spezzato in più spedizioni, situazione che si è riscontrata spesso nel caso di messaggi di grosse dimensioni ed è stato necessario implementare funzioni che, vice versa, scorrano i dati ricevuti in modo da interpretare i vari messaggi che sono stati accorpati in un unico pacchetto spedito, questo sostanzialmente a causa del fatto che nell'implementazione TCP di open-r non è possibile né impostare il flag PUSH che forza la spedizione dei dati finora accumulati dal TCP nè impostare l'URGENT POINT che similmente al flag PUSH forza il TCP a spedire i dati accumulati e forza anche il ricevente ad analizzare i dati marcati urgenti.

3.5.2. Come funziona

All'avvio TelepatiTcp effettua il parsing del file di configurazione AIBONET.CFG e dati *n* Aibo con cui comunicare crea *2n* Endpoint, per ogni Aibo si avrà, quindi, una connessione in ricezione ed una in spedizione. Il calcolo delle porte TCP per la comunicazione è abbastanza prolisso, si basa sul numero di righe del file di configurazione, se l'*i-esimo* Aibo vuole comunicare con il *j-esimo*, dove per *i-esimo* e *j-esimo* ci si riferisce alla sua posizione del file AIBONET.CFG, la porta a cui dovrà connettersi sarà data da:

```
PORTA = TELEPATI_BASE_PORT + (<riga del i-esimo> * <numero_totale_righe>)+ <riga del j-esimo>.
```

Dove TELEPATI_BASE_PORT è il numero di porta base da cui parte la numerazione (di default è 30000).

Dopo aver impostato le porte di comunicazione ed inizializzato le Endpoint, mette in ascolto le *n* connessioni in ricezione ed avvia la procedura di *connect* sulle *n* connessioni di spedizione, TelepatiTcp ritenterà una connessione, non anadata a buon fine, per un numero di volte specificato dalla variabile TELEPATI_MAX_CONNECT_ERROR.

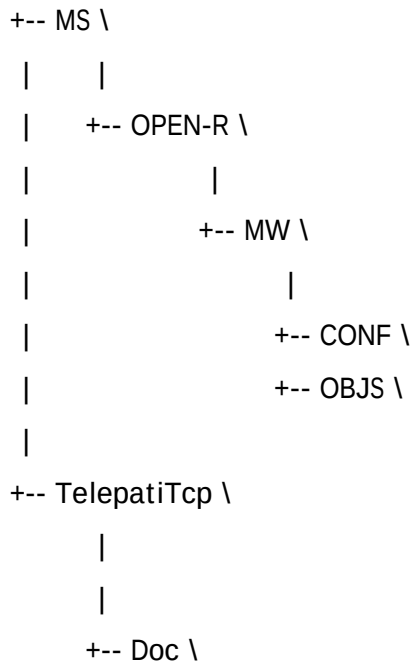
Stabilita la connessione, per ogni connessione, TelepatiTcp manderà un **Hi** message e resterà in attesa di riceve un **HiReply**, ricevuta la risposta l'Aibo verrà impostato come attivo, verrà mandato un AiboInfoMsg all'observer e potrà iniziare la comunicazione con l'Aibo.

Come la versione UDP TelepatiTcp tiene aggiornato un array contenete lo stato di ogni Aibo, stato che viene continuamente mantenuto aggiornato.

Al termine, prima dello spegnimento, TelepatiTcp manderà un **Bye** message ad ogni Aibo connesso per informarlo della sua intenzione di disattivarsi, quindi chiuderà le connessioni.

3.5.3. Struttura dei sorgenti

Di seguito è riportata la struttura della directory di TelepatiTcp:



In [TelepatiTcp] è contenuto il codice sorgente dell'oggetto OPEN-R. I file sono:

- Makefile: il makefile di Telepati.
- TelepatiTcp.h: l'header file di TelepatiTcp.
- TelepatiTcp.cc: il sorgente C++ di TelepatiTcp.
- stub.cfg: il file stub.cfg di TelepatiTcp per la dichiarazione dei servizi OPEN-R.
- telepatitcp.ocf: imposta alcune caratteristiche dell'oggetto tra cui l'esecuzione il Kernel o User mode, la dimensione dello stack, dello Heap…
- TCPConnection.h: header di TCPConnection.cc, contiene gli oggetti TCPConnection, la classe base, e TCPRecvConnection e TCPSendConnection, classi che ereditano da TCPConnection, che raprresentano le connessioni TCP in entrata e in uscita e offre i metodi per inizializzare le connessioni.
- TCPConnection.cc: Implementa gli oggetti TCPConnection e TCPRecvConnection e TCPSendConnection.
- telepatitcp_doc.cfg: il file di configurazione di Doxygen per la generazione della documentazione della classe.
- EtherStat.h: header file della classe EtherStat per la stampa delle statistiche di Rete (guarda Stampare informazioni sulla rete)
- EtherStat.cc: sorgente della classe EtherStat.

In [Doc] è contenuta la documentazione del codice di TelepatiTcp in formato HTML generata automaticamente da **Doxygen**.

In [MS] è contenuta la struttura delle directory da copiare nella memory stick. Più in dettaglio in MS/OPEN-R/MW/CONF saranno presenti i file:

- CONNECT.CFG: connette i vari servizi dei vari oggetti caircati sull'Aibo.
- OBJECT.CFG: elenca gli oggetti da caricare all'avvio dell'Aibo.

- AIBONET.CFG: una lista di indirizzi IP degli Aibo con cui comunicare.

Nella directory MS/OPEN-R/MW/OBJS sono presenti gli oggetti da caricare all'avvio dell'Aibo, tra i quail l'oggetto **teletcp.bin**.

Per compilare TelepatiTcp è sufficiente andare nella direcotory TelepatiTcp\ lanciare il comando **make**. Lanciando il comando **make install** verrà copiato il file telepati.bin nella direcotory MS/OPEN-R/MW/OBJS. Lanciando il comando **make clean** verranno cancellati i vari file creati durante la compilazione lasciando intatti i vari sorgenti. Con il comando **make doc** viene rigenerata la documentazione di TelepatiTcp (in questo caso è necessario aver installato e configurato Doxygen).

3.5.4. Servizi offerti da TelepatiTcp

I servizi messi a disposizione da TelepatiTcp sono:

- Subject= *TelepatiTcp.ReceiveMsg.byte.S* : per la ricezione dei messaggi
- Observer= *TelepatiTcp.SendMsg.byte.O* : per la spedizione dei messaggi

4. Utilizzo di Telepati

4.1. Comunicazione tra Telepati e gli altri oggetti OPEN-R

*** aggiungi commento sul fatto che telepati manda un messaggio alla volta agli obsever a prescindere dal numero di messaggi che deve spedirgli ***

L'utilizzo delle due versioni di Telepati è sostanzialmente identico, le uniche differenze riguardano i **nomi dei servizi** offerti e i **nomi degli oggetti** da caricare all'avvio degli Aibo.

Lo scambio di package tra Telepati e gli altri oggetti OPEN-R avviene attraverso i metodi offerti dalle classi `OObserver` e `OSubject`.

La spedizione di messaggi a Telepati avviene sfruttando i metodi: SetData e NotifiyObserver (di OSubject), getMsg e getSize (di TelepatiPkg):

```
TelepatiPkg* package=new DummyPkg();
subject[xxx]->SetData(package.getMsg(), package.getSize());
subject[xxx]->NotifyObserver();
```

Dove *getMsg()* restituisce un array di byte che rappresenta il TelepatiPkg e rappresenta quindi il messaggio da spedire, *getSize()* ritorna la dimensione (in byte) dell'array restituito da getMsg().

La ricezione dei messaggi avviene invocando il metodo statico TelepatiPkg::createPkg(byte*) sull'array di byte restituito dal metodo Data() della classe OSystemEvent:

```
TelepatiPkg* pkg=TelepatiPkg::createPkg((byte*)event.Data(0));
```

Il quale ricostruisce l'oggetto a partire dall'array di byte (l'array restituito da *getMsg()*) e lo restituisce.

In Telepati è stata implementata una coda dei messaggi da spedire agli observer ad esso collegato. Questa coda ha il compito di accumulare i vari messaggi che Telepati deve mandare ai suoi observer, nel caso (per esempio durate uno stream video) in cui ci siano piu' messaggi da spedire contemporaneamente, e mandarne uno alla volta; verrà quindi mandato un messaggio, a questo punto Telepati attenderà il ricevimento di un Assert Ready e al suo ricevimento spedirà il secondo…

E' stato implementato per evitare di dover utilizzare nel lato observer del codice simile a:

```
for(int i=0; i < event.NumOfData(); i++){
        TelepatiPkg* msg=TelepatiPkg::createPkg((byte*)event.Data(i));
        …
}
```

Il codice sopra riportato viene utilizzato il open-r nel caso siano stati spediti più messaggi ad un unico oggetto e permette di scorrere il buffer e interpretare (con il metodo Data(int i)) i singoli messaggi.

La coda è stata implementata per semplificare maggiormente l'utilizzo di Telepati da parte dei vari oggetti open-r che devono utilizzarlo.

4.2. AiboInfoPkg (Come sono rappresentati gli Aibo in rete)

L'AiboInfoPkg è il package che permette ad un oggetto OPEN-R di utilizzare Telepati per la comunicazione; esso permette di conoscere il numero di Aibo presenti in rete (metodo *getNumOfAibo() di* AiboInfoPkg), di conoscere lo stato di attività di ogni Aibo (metodo *isActive(int i) di* AiboInfoPkg) e di conoscere lo stato di attivià di tutti gil Aibo in rete (metodo *isAllActive()*). Se l'i-esimo Aibo risulta inattivo non è possibile mandargli un messaggio.

In rete gli Aibo sono identificati semplicemente da un intero (p.es. Se in rete sono presenti 3 Aibo, gli Aibo saranno 0, 1 e 2) sarà poi **Telepati a mappare l'intero con il relativo indirizzo IP** dell'Aibo di destinazione. E' presente anche l'indirizzo speciale *TelepatiPkg::ADDR_BROADCAST* che se specificato come Aibo di destinazione farà spedire a Telepati il messaggio in **broadcast** a tutti gli Aibo presenti nel file di configurazione AIBONET.CFG.

Non appena lo stato di un Aibo cambia Telepati spedisce un AiboInfoPkg all'oggetto observer per informarlo del nuovo stato, di conseguenza è consigliabile tenere un riferimento all'oggetto restituito per poter controllare lo stato di attività di ogni Aibo. Al ricevimento di un AiboInfoPkg da parte dell'oggetto observer Telepati risponderà con un'altro AiboInfoPkg con i valori aggiornati.

4.3. Spedizione e ricezione di messaggi via rete

La spedizione dei messaggi avviene costruendo il package voluto, impostando i valori corretti per l'Aibo di destinazione (tutti i package permettono nel costruttore di definire l'Aibo di destinazione), e mandando il package appena creato a Telepati (al servizio SendMsg) il quale spedirà il messaggio attraverso la rete e nel caso il messaggio richieda un Ack attenderà che il ricevente risponda con un messaggio di conferma.

Alla ricezione di un messaggio dalla rete Telepati manderà il relativo package al suo observer (guarda Comunicazione tra Telepati e gli altri oggetti OPEN-R per l'inter-object comunication) e si occuperà di mandare, se richiesto, un messaggio di Ack al mittente.

4.4. Come utilizzare Telepati (Quick Reference)

Da un punto di vista di un oggetto OPEN-R che intende utilizzare Telepati per la comunicazione:

1. Includere la libreria Package e il suo relativo header file (Package.h) nell'oggetto.
2. Impostare il file /MS/OPEN-R/MW/CONF/CONNECT.CFG collegando i subject e gli observer dell'oggetto con i relativi observer e subject offerti da Telepati, bisogna fare attenzione alla versione di Telepati utilizzata in quanto le due versioni, TCP ed UDP, hanno nomi di servizi differenti.

3 Impostare /MS/OPEN-R/MW/CONF/OBJECT.CFG in modo che venga caricato sia l'oggetto stesso sia l'oggetto Telepati: **telepati.bin** nel caso dell'Udp, **teletcp.bin** nel caso del Tcp.

4 In DoStart() mandare un AiboInfoPkg a Telepati e attendere la risposta contenente un altro AiboInfoPkg con i valori degli Aibo impostati correttamente

5 Ora è possibile comunicare con gli altri Aibo creando il package voluto e mandandolo a Telepati sfruttando la comunicazione inter-object.

4.5. Esempio di utilizzo

Come esempio è stata implementata la classe **TeleLed** che ha il compito di accendere dei led sul muso dell'Aibo. Questa classe riceve dalla rete, attraverso Telepati versione Udp (come detto in precedenza l'utilizzo delle due versioni e' sostanzialmente identico), un messaggio di tipo *DummyLed*.

Sono presenti in rete due Aibo di IP 192.168.254.2 e 192.168.254.3 e su entrambi vengono caricati Telepati e TeleLed. All'avvio ogni Aibo richiede a Telepati un AiboInfoPkg contenente gli Aibo attivi nella rete, non appena uno di essi, che chiameremo AIBO1, diventa attivo invia un Hi message all'altro AIBO che chiameremo AIBO2. Quando AIBO2 risponde con un HiReply, AIBO1 invia ad AIBO2 un messaggio DummyLed che farà accendere alcuni led sul suo muso.

Di seguito è riportato l'Header file di TeleLed:

```
#ifndef TELELED_H_DEFINED
#define TELELED_H_DEFINED

#include "../Telepati/Packages/Packages.h"
#include <OPENR/OObject.h>
#include <OPENR/OSubject.h>
#include <OPENR/OObserver.h>
#include <OPENR/RCRegion.h>
#include "def.h"

/*Indirizzi delle primitive di controllo dei Led*/
static const char* const LED3_LOCATOR[] = {
    "PRM:/r1/c1/c2/c3/la-LED3:la", // Face light1
    "PRM:/r1/c1/c2/c3/lb-LED3:lb", // Face light2
    "PRM:/r1/c1/c2/c3/lc-LED3:lc", // Face light3
    "PRM:/r1/c1/c2/c3/ld-LED3:ld", // Face light4
    "PRM:/r1/c1/c2/c3/le-LED3:le", // Face light5
    "PRM:/r1/c1/c2/c3/lf-LED3:lf", // Face light6
    "PRM:/r1/c1/c2/c3/lg-LED3:lg", // Face light7
    "PRM:/r1/c1/c2/c3/lh-LED3:lh", // Face light8
    "PRM:/r1/c1/c2/c3/li-LED3:li", // Face light9
    "PRM:/r1/c1/c2/c3/lj-LED3:lj", // Face light10
    "PRM:/r1/c1/c2/c3/lk-LED3:lk", // Face light11
    "PRM:/r1/c1/c2/c3/ll-LED3:ll", // Face light12
    "PRM:/r1/c1/c2/c3/lm-LED3:lm", // Face light13
    "PRM:/r1/c1/c2/c3/ln-LED3:ln"  // Face light14
};

class TeleLed : public OObject {
 public:

  TeleLed();
  virtual ~TeleLed();

  OSubject*    subject[numOfSubject];
  OObserver*   observer[numOfObserver];

  virtual OStatus DoInit   (const OSystemEvent& event);
```

```
  virtual OStatus DoStart  (const OSystemEvent& event);
  virtual OStatus DoStop   (const OSystemEvent& event);
  virtual OStatus DoDestroy(const OSystemEvent& event);

  /*Avviata al ricevimento di un messaggio dalla Rete*/
  void Receive(const ONotifyEvent& event);

  /*Avviata al ricevimento dell'AR da parte di OVirtualRobotComm*/
  void Ready(const OReadyEvent& event);

 private:
  /* Apre le primitive di controllo dei Led*/
  void OpenPrimitives();
  /* Crea un nuovo vettore CommandVectonData, contenente i comandi da mandare ai
Led*/
  void NewCommandVectorData();
  RCRegion* FindFreeRegion();
  void SetLED3Value(RCRegion* rgn, sword intensity, OLED3Mode mode);

  static const size_t NUM_LEDS           = 14;
  static const size_t NUM_COMMAND_VECTOR = 2;
  static const int    MAX_LEVEL          = 15;
  static const int    POSITIVE_SLOPE     = 1;
  static const int    NEGATIVE_SLOPE     = -1;

  OPrimitiveID          ledID[NUM_LEDS];
  RCRegion*             region[NUM_COMMAND_VECTOR];

  int aibos;  /* numero di aibo nel campo*/
  AiboInfoPkg* info; /* AiboInfoPkg aggiornato*/

};
#endif  //TELELED_H_DEFINED:
```

Di seguito è riportato il file stub.cfg di TeleLed contenente i servizi resi disponibili:

```
ObjectName: TeleLed
NumOfOSubject: 2
NumOfOObserver: 1
Service: "TeleLed.TesterSend.byte.S", null, null
Service: "TeleLed.TesterReceive.byte.O", null, Receive()
Service: "TeleLed.Blink.OCommandVectorData.S", null, Ready()
```

E il file CONNECT.CFG:

```
TeleLed.TesterSend.byte.S TelepatiUdp.SendMsg.byte.O
TelepatiUdp.ReceiveMsg.byte.S TeleLed.TesterReceive.byte.O

TeleLed.Blink.OCommandVectorData.S OVirtualRobotComm.Effector.OCommandVectorData.O
```

Tralasciando, per semplificare la comprensione dell'esempio, gli altri metodi di TeleLed che riguardano l'inizializzazione dell'oggetto o la gestione dei Led, il metodo più importante per la comunicazione via rete è receive():

```
void TeleLed::Receive(const ONotifyEvent& event){
  static bool canSend=true;

  TelepatiPkg* msg=TelepatiPkg::createPkg((byte*)event.Data(0));
  pkg_type type=msg->getType();

  if(type == INFO)
    {
      info=(AiboInfoPkg*)msg;
      aibos=info->getNumOfAibo();
```

```
      for(int i=0; i<aibos; i++){
          if(info->isActive(i) && canSend)
            {
              TelepatiPkg* pkg=new DummyLedPkg(i, 50);
              subject[sbjTesterSend]->SetData(pkg->getMsg(), pkg->getSize());
              subject[sbjTesterSend]->NotifyObservers();
              canSend=false;
            }
      }
    }
  else if(type == DUMMYLED)
    {
      DummyLedPkg* dummypkg=(DummyLedPkg*)msg;
      RCRegion* rgn = FindFreeRegion();
      SetLED3Value(rgn, dummypkg->getColor(), oled3_MODE_B);
      subject[sbjBlink]->SetData(rgn);
      subject[sbjBlink]->NotifyObservers();
    }

  observer[obsTesterReceive]->AssertReady();
  return;
}
```

Al ricevimento di un messaggio dalla rete Telepati manderà il messaggio stesso a TeleLed, più precisamente al servizio TeleLed.TesterReceive.byte.O, che, guardando il file stub.cfg, farà eseguire all'oggetto TeleLed il metodo receive(); l'oggetto passato come argomento, *event,* conterrà il messaggio stesso. Grazie al metodo statico TelepatiPkg::createPkg() viene ricostruito, a partire dall'array restituito da *event.Data(0)*, il package del messaggio.

Se il package è un AiboInfoPkg allora verrà aggiornato il numero di aibo presenti in rete e verrà memorizzato il packge per usi futuri.

Per ogni Aibo in rete, se l'i-esimo Aibo è attivo gli verrà mandato, una volta sola (il bool *canSend* fa in modo che venga spedito il messaggio una sola volta) un messaggio DummyLed con lo scopo di fargli accedere i led sul muso.

Se viene ricevuto un DummyLed allora verranno accesi il Led sul muso dell'Aibo.

4.6. Debugging

Telepati offre due possibilità di debugging, entrambe sfruttano la console Telnet offerta da OPEN-R per questo scopo. Compilato Telepati con le impostazioni corrette sarà sufficiente collegarsi via Telnet all'IP dell'Aibo sulla porta 59000 per visualizzare sul PC le informazioni necessarie per l'individuazione degli errori.

4.6.1. Attivare il debug di Telepati

Per avviare il debug di Telepati, che sostanzialmente consiste nel nome dei metodi avviati e in alcuni altre informazioni aggiuntive, come il numero di pacchetti ricevuti o spediti, il tipo di pacchetto ricevuto… è necessario compilare Telepati definendo *OPENR_DEBUG* (passando a gcc l'opzione: -DOPENR_DEBUG).

Il Debug si basa su una macro offerta da OPEN-R, *OSYSDEBUG(("<stringa>")),* che viene attivata solo se viene definito OPENR_DEBUG e che stampa sulla console Telnet in ordine di esecuzione gli argomenti passati tra le doppie parentesi, argomenti che hanno la stessa forma della funzione printf di C.

Il nome dell'oggetto, TelepatiUdp o TelepatiTcp, verrà stampato in verde per rendere più immediata la lettura delle stringe di debug. Il colore viene definito dalla #define

TLPT_BEG_COLOR (negli header file di Telepati) ed è espresso come una normale sequenza di escape ANSI.

4.6.2. Stampare informazioni sulla rete

E' stata implementata la possibilità di attivare una sorta di monitor dello stato della rete Wireless sempre accessibile via Telnet; ogni 5 secondi Telepati stampa a video varie informazioni sullo stato della rete come il nome della rete wireless (ESSID), la qualità del segnale, il numero di pacchetti persi…

Per attivare questa funzione è necessario definire *OPENR_ETHERSTAT* (passando a gcc l'opzione -DOPENR_ETHERSTAT) e linkare la libreria *ERA201D1* (passando a gcc l'opzione -lERA201D1)[6].

Di seguito è riportato uno screen shot delle informazioni sulla rete visualizzate da Telepati durante la sua esecuzione:

```
[Ether Statistics]
  carrierSenseErrorCount  : 0
  lateCollisionCount      : 0
  excessiveCollisionCount : 0
  singleCollisionCount    : 0
  multipleCollisionCount  : 0
  deferredCount           : 0
  alignmentErrorCount     : 0
  fcsErrorCount           : 0
  frameTooLongCount       : 0
[WLAN Settings]
  port          : 0
  mode          : ETHER_WLAN_MODE_AP
  essid         : Aibonet
  channel       : 10
  encryption    : ETHER_WLAN_ENC_OFF
  rate          : invalid value
  rts           : 2347
  fragmentation : 2347
  sensitivity   : ETHER_WLAN_DENSITY_LOW
[WLAN Statistics]
  link             : 63
  signal           : 84
  noise            : 42
  invalidIDCount   : 0
  invalidEncCount  : 0
  invalidMiscCount : 0
Messaggi spediti: 1569  -- Messaggi ricevuti: Totale 1564 - Data 292
```

5. Dati sperimentali

5.1. Prestazioni

E' importante chiarire inizialmente che a causa della scarsa quantità di funzioni fornite per la gestione del procotollo TCP/IP i valori riguardo la spedizione dei pacchetti attraverso il TCP non sono del tutto veritieri. Non è possibile sapere con precisione il tempo impiegato per la spedizione, la funzione SendCont in TelepatiTcp viene chiamata nel momento in cui Ant ha copiato tutti i dati da spedire nel buffer di spedizione di IPStack e quindi quando il buffer delle connessioni di spedizione torna ad essere disponibile e non quando è stata completata effettivamente la spedizione del pacchetto. Per calcolare la velocità di spedizione con la

6 Nel file Makefile sono presenti, commentate, entrambe le opzioni di debug.

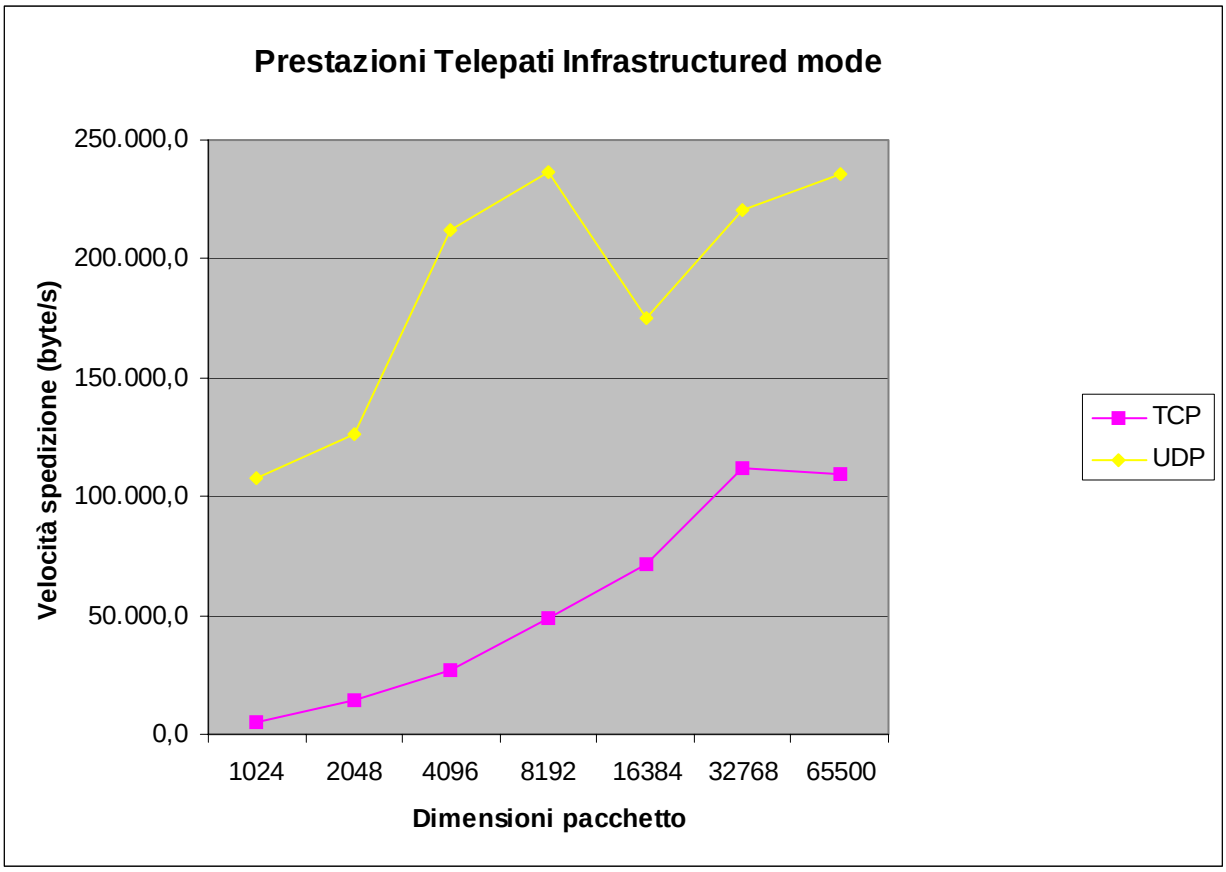
versione TCP sono state aggiunte alcune funzioni che al ricevimento di un dato pacchetto rispediscano al mittente un Ack onde informarlo dell'effettiva ricezione; il tempo di spedizione viene quindi calcolato dal momento in cui viene "mandato" il pacchetto ad IPStack al momento in cui viene ricevuto l'Ack.

Per quanto riguarda la versione UDP, utilizzando un metodo di spedizione **sliding window** di dimensione 1, permettendo quindi di spedire un solo pacchetto alla volta prima di ricevere l'Ack, è stato utilizzato lo stesso metodo della versione TCP per il calcolo del tempo di spedizione, con la differenza che nel caso dell'UDP il tempo trascorso è effettivamente il tempo impiegato per la spedizione.

Le prove sono state effettuate utilizzando due Aibo in comunicazione tra di loro, ogni Aibo all'avvio spedisce un pacchetto di dimensione variabile (guarda Dimensione Pacchetto nelle tabelle) e una volta spedito viene spento. Telepati è stato compilato con il Debug e il Polling disabilitato, mentre la Stampa delle statistiche di rete è rimasta attiva, questo per permettere di raccogliere i dati. Le prove sono state effettuate sia in modalità Infrastructured sia in modalità AdHoc e con entrambe le versioni di Telepati, TCP e UDP; di seguito le tabelle ed i grafici.

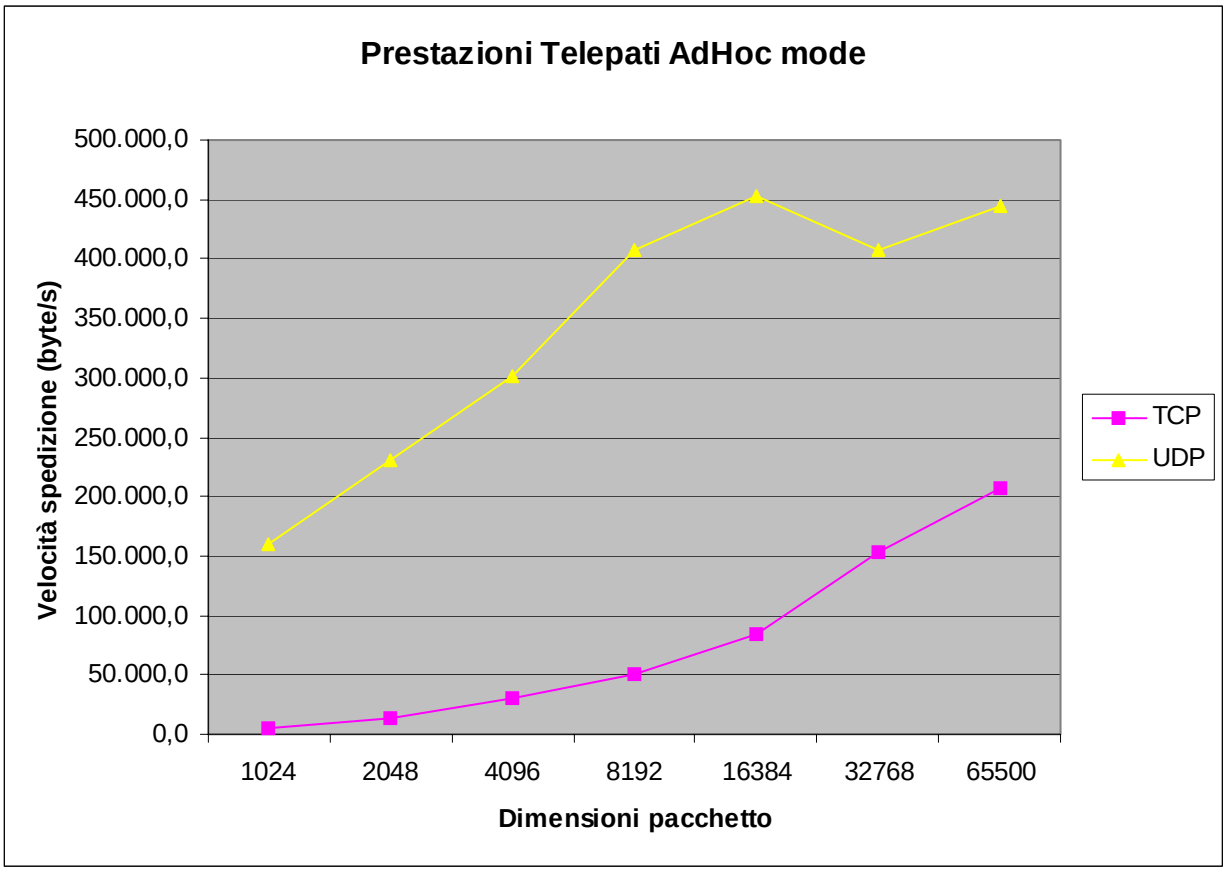
Modalità Infrastructured:

Modalità Infrastructured		
Dimensione Pacchetto	**TCP**	**UDP**
1024	5.399,3	107.647,1
2048	14.350,1	126.442,6
4096	27.093,6	212.400,8
8192	49.108,5	236.939,1
16384	71.630,6	175.161,4
32768	111.834,3	220.826,5
65500	109.084,1	235.976,9



Modalità AdHoc:

Modalità AdHoc		
Dimensione Pacchetto	**TCP**	**UDP**
1024	5.534,6	160.247,9
2048	13.022,7	229.940,2
4096	30.166,1	300.768,3
8192	50.019,1	406.683,8
16384	84.436,2	452.379,4
32768	152.814,2	407.962,7
65500	207.193,3	443.804,9



Si può facilmente notare che la versione UDP risulta sempre più veloce rispetto alla versione TCP, bisogna però tener presente che, come detto all'inizio del paragrafo, i valori della velocità di spedizione in modalità TCP non sono molto veritieri a causa della chiusura del codice delle librerie di gestione TCP/IP di OPEN-R.

5.2. Perdita di pacchetti spediti

[*** Cerca la causa per la perdita dei pacchetti: collisioni, troppo rumore, scarse risorse... sia in modalita' Adhoc sia in infastructured]

6. Aggiungere tipi di pacchetti *(libreria Packages)*

6.1. Struttura dei pacchetti

I pacchetti di Telepati sono tutti figli della classe TelepatiPkg e sono compresi nella libreria Packages in /lib/Packages; il file oggetto della libreria, *libPackages.a*, si trova in /lib mentre l'header della libreria (Packages.h) si trova in /include/Packages.

I pacchetti sono identificati dal tipo *enum* pkg_type di TelepatiPkg, i valori da 0 a 9 sono riservati per pacchetti speciali utilizzati da Telepati che quindi non possono essere utilizzati, i valori superiori a 9 possono, invece, essere utilizzati per la creazione di package personalizzati senza dover apportare modifiche all'oggetto Telepati.

La classe TelepatiPkg è una classe virtuale da ereditare nel caso si intenda creare un package personalizzato ed offre il metodo statico *TelepatiPkg* createPkg(const byte*)* che è

un'implementazione del SIMPLE FACTORY PATTERN; ha il compito di ricostruire l'oggetto del tipo corretto (AiboInfoPkg, DummyPkg…) a partire dall'array di byte che lo rappresenta (l'array restituito da *getMsg()*); questi due metodi, *createPkg()* e *getMsg(),* hanno un'importanza fondamentale nella struttura di Telepati, l'array di byte restituito da getMsg() rappresenta i dati che verranno poi spediti via rete da Telepati e rappresenta anche i dati che verrano spediti agli altri oggetti attraverso la comunicazione inter-object. Come detto la libreria Packages è fortemente imperniata sull'*ereditarietà* e sul *polimorfismo,* questi due concetti fondamentali nella programmazione orientata agli oggetti sono incompatibili con la comunicazione inter-object implementata in open-r; ai vari oggetti open-r non è possibile mandare oggetti polimorfici, se non utilizzando tecniche abbastanza contorte e sicuramente poco eleganti, mentre è possibile mandare tipi base (interi, float…), array e strutture; si è scelto, di conseguenza, di utilizzare l'array di byte come tipo di dati scambiati e fornire ogni oggetto della classe Packages di un metodo *pure virtual*, *getMsg()* appunto, che restituisce una rappresentazione compatibile con la comunicazione inter-object di open-r dell'oggetto stesso.

La documentazione di Packages e dei pacchetti utilizzati da Telepati è disponibile in formato HTML nella directory /lib/Package/Doc/html ed è stata creata con il tool **Doxygen**; il file di configurazione di Doxygen è *packages_doc.cfg*, per rigenerare la documentazione utilizzare il comando *make doc*.

6.2. Esempio di classe TelepatiPkg

Per spiegare come implementare un messagio da spedire via rete, quindi un oggetto della libreria Packages, si è scelto di utilizzare un esempio.

Di seguito è riportato il sorgente di un pacchetto Dummy, un pacchetto implementato principalmente per scopi di test ed è composto dall'header standard di un TelepatiPkg più una stringa, "Dummy message".

L'Header file di DummyPkg è il seguente:

```
#include "TelepatiPkg.h"
#include <string>

using namespace std;

/**implementa un messaggio dummy usato per testing.@n Struttura di
   "pkg data" (l'header e' standard):@n char[]= array di caratteri nel
   formato di stringhe di C.
*/
class DummyPkg : public TelepatiPkg {
 public:
  /** Costruttore ordinario. @param a aibo di destinazione @param ack
      true:se richiesto l'Ack; false: altrimenti*/
  DummyPkg(int a, bool ack=false);
  /**Costruisce l'oggetto a partire dall'array restituito da
     getMsg(). @param ptr puntatore all'array di byte. @see
     getMsg()*/
  DummyPkg(byte* ptr);

  /** Distruttore di Default.*/
  virtual ~DummyPkg();

  /** Clonatore. Restituisce un clone dell'oggetto su cui e'
      chiamato.*/
  virtual TelepatiPkg* clone();

  /** Ritorna un array che rappresenta l'oggetto. Costruisce un array
      che rappresenta l'oggetto, l'array corrisponde ai dati spediti
```

```
      via rete. @return array di byte. @see DummyPkg(byte* ptr)*/
  byte* getMsg();
  /** Stringa "Dummy message". @return stringa del messaggio*/
  inline string getString() { return str; };

 protected:
  string str;
};
```

Il package implementa due costruttori, uno standard:

```
DummyPkg::DummyPkg(int a, bool ack) : TelepatiPkg() {
  type=DUMMY;
  str="Dummy message";
  aibo=a;
  needAck=ack;
}
```

che crea un package contenente la stringa “Dummy message” e riceve come parametri int a che corrisponde all'indice dell'Aibo di destinazione, indice ricavato dall'AiboInfoPkg, e bool ack che viene utilizzato per impostare needAck quindi la necessità o meno di richiedere un Acknowledgment dopo la spedizione del message, di default è posto uguale a false, quindi non richiesto.

In generale il costruttore dovrà almeno:

- Richiamare il costruttore di default TelepatiPkg() che imposta i valori standard del pacchetto
- impostare int aibo: l'indice dell'Aibo di destinazione
- impostare pkg_type type: tipo di package
- impostare bool needAck (di default, nel costruttore TelepatiPkg(), a *false*): *true* se è richiesto l'Ack al ricevimento del message, *false* se non richiesto.

Il secondo costruttore ricostruisce l'oggetto a partire dall'array di byte restituito dal metodo getMsg (vedi oltre).

```
DummyPkg::DummyPkg(const byte* ptr) : TelepatiPkg(ptr){
  str="";
  for(int i=STD_HEADER_SIZE; i<size; i++){
    str+=ptr[i];
  }
}
```

Come si può notare il costruttore DummyPkg(byte* ptr) richiama all’inizio il costruttore TelepatiPkg(byte* ptr) costruttore che a partire dall'array di byte corrispondente al message, imposta i valori standard di TelepatiPkg (aibo, type, size, needAck e serialNum) scandendo l'header standard; questo costruttore andrebbe sempre richiamato all'inizio del costruttore ereditato.

Dev’essere poi implementata la funzione, definita *pure virtual* in TelepatiPkg.h, clone() che ha il compito di clonare l’oggetto su cui chiamata, quindi di creare un oggetto identico all’oggetto sui cui chiamata e restituirlo. Questa funzione è necessaria a Telepati sia per il broadcasting, sia per la coda di messaggi agli observer e sia per la gestione degli Ack nella versione UDP di Telepati. Un esempio della sua implentazione è il seguente:

```
TelepatiPkg* DummyPkg::clone(){
  DummyPkg* clone=new DummyPkg(aibo, needAck);
  clone->serialNum=serialNum;
  return clone;
}
```

Viene ricreato un oggetto di tipo DummyPkg, destinato allo stesso Aibo, paramentro aibo al costruttore, e con la stessa “necessità” di Ack, paramentro needAck al costruttore, quindi viene

copiato il serialNum dell'oggetto originale al clone e viene ritornato.

Quindi il distruttore:

```
DummyPkg::~DummyPkg(){
}
```

A questo punto dev'essere implementato il metodo pure virtual di TelepatiPkg, getMsg(). Questo metodo ha il compito di ritornare un array di byte rappresentante l'oggetto.

All'inizio di TelepatiPkg.h è riportata la struttura dell'header standard dei message, header che viene costruito dal metodo *byte* createStdHeader(size_t size)* che ritorna un array di byte di dimensione pari alla dimensione dell'header standard più' la dimensione passata come parametro che corrisponde alla dimensione dei dati aggiunti dal package (pkg data); imposta il valore di size del package e imposta i valori corretti dell'header standard nel message[7]. Sfruttando questo metodo il getMsg() di DummyPkg imposta i valori dell'header ed aggiungerà all'array la stringa "Dummy message). Ogni oggetto che eredita dalla classe TelepatiPkg avrà come membro un puntatore a byte, **byte* message**, che punta all'array di byte che rappresenta l'oggetto (all'array restituito da getMsg()). Per evitare leak di memoria viene salvato in message il puntatore a questo array, array che viene distrutto dal distruttore standard ~TelepatiPkg() e array che dev'essere distrutto ogni volta che ne viene ricreato un nuovo, come nel caso di getMsg().

```
byte* DummyPkg::getMsg(){
  delete [] message;
  const char *tmpStr=str.c_str();
  size_t tmpStrSize=str.size()+1; //stringa + \0
  message=createStdHeader(tmpStrSize);
  for(int i=0; i < tmpStrSize; i++)
    message[i+STD_HEADER_SIZE]=(byte)*(tmpStr+i);
  return message;
}
```

Come detto in precedenza, come prima istruzione viene distrutto il vecchio array di byte message per poi venir ricreato dalla funzione createStdHeader, salvato e restituito.

A questo punto è necessario modificare il file TelepatiPkg.cc che contiene il SIMPLE FACTORY PATTERN, il cui prototipo è *TelepatiPkg* createPkg(const byte*),* aggiungendo all'interno dello switch una linea del tipo:

```
case DUMMY:  //DUMMY è di tipo pkg_type
   return new DummyPkg(ptr);
   break;
```

che richiama il costruttore del pacchetto.

Come ultimo passaggio è necessario aggiungere all'interno del file Packages.h lo *#include* relativo all'header file DummyPkg.h.

Come si può notare l'implementazione di un package personalizzato è molto semplice e la classe TelepatiPkg offre tutti gli strumenti, costruttori e metodi, per permettere di gestire gli attributi ereditati.

7 è importante il valore size_t TelepatiPkg::STD_HEADER_SIZE che corrisponde alla dimensione in byte dell'header standard, valore necessario per poter aggiungere all'array di byte i vari campi presenti del pacchetto personalizzato.

6.3. I passi da seguire per la creazione di un nuovo pacchetto

Capita la struttura che deve rispettare un pacchetto personalizzato, riassumento, i passi da seguire sono:

1. Scrittura del codice del pacchetto.
2. Aggiungere in TelepatiPkg.cc, all'interno dello SWITCH, il CASE relativo al pacchetto implementato.
3. Aggiungere all'interno del file Packages.h lo *#include* relativo all'header file del pacchetto implementato.
4. Ricompilare la libreria, Telepati e gli altri Oggetti OPEN-R che dipendono dalla libreria Packages con il comando *make install* (che compilerà la libreria e installerà i file nelle directory corrette). E' consigliabile poi eseguire il comando *make doc* che rigenererà la documentazione della libreria.