Alma Mater Studiorum·Università di Bologna

SCUOLA DI SCIENZE
Corso di Laurea in Informatica

IEEE 802.21: Media Independent Handover

Tesi di Laurea in Reti di Calcolatori

Relatore:
Chiar.mo Prof.
Vittorio Ghini

Presentata da:
Francesco Soncina

Sessione III
2012/2013

IEEE 802.21: Media Independent Handover

Francesco Soncina

18 Marzo 2014

I'm personally convinced that computer science has a lot in common with physics. Both are about how the world works at a rather fundamental level. The difference, of course, is that while in physics you're supposed to figure out how the world is made up, in computer science you create the world. Within the confines of the computer, you're the creator. You get to ultimately control everything that happens. If you're good enough, you can be God. On a small scale.

Just for Fun

Linus Torvalds, David Diamond

Abstract

In questo lavoro, dopo un'introduzione sul panorama contemporaneo, si è analizzato lo standard IEEE 802.21, illustrandone i motivi che hanno portato al suo sviluppo, la *timeline* del processo di standardizzazione, gli obbiettivi del *working group*, l'architettura del sistema specificato e le sue funzionalità, con particolare riguardo all'utilità in applicazioni reali, al fine di darne un giudizio completo sulla sua effettiva efficacia. Dopo aver citato qualche esempio di possibile applicazione dello standard e descritto lo stato attuale dell'arte, si è studiata una sua implementazione *cross-platform* chiamata ODTONE, descrivendone i vari componenti e le loro funzionalità, ma anche sottolineando le attuali mancanze per arrivare ad una implementazione completa sotto tutti i punti di vista. Successivamente si è studiata ed implementata una semplice applicazione, *MIH-proxy*, che potesse sfruttare in modo costruttivo le informazioni accessibili tramite lo standard per creare un proxy che potesse scegliere su quale interfacce instradare i pacchetti a seconda dello stato attuale di tutti i collegamenti, realizzato in versione unidirezionale e bidirezionale. In particolare questa applicazione è in grado di restare in ascolto di cambiamenti di stato delle interfacce di rete, e.g. quando viene stabilita una connessione oppure cade, e, di conseguenza, stabilire di volta in volta quali collegamenti utilizzare per inviare dati. Nella versione bidirezionale è anche possibile far comunicare tra loro applicazioni che normalmente utilizzerebbero il protocollo di trasporto TCP attraverso un ulteriore componente, *phoxy*, che si preoccupa di convertire, in modo trasparente, un flusso TCP in datagrammi UDP eventualmente cifrati. Sarà quindi possibile creare un collegamento ad alta affidabilità tra le applicazioni che possa sfruttare tutte le interfacce disponibili, sia per inviare, sia per ricevere.

Indice

Elenco delle figure

Elenco delle tabelle

Capitolo 1

Introduzione

In questo capitolo verranno esposte le motivazioni e gli obbiettivi di questo lavoro, a partire dalla situazione odierna per descriverne i problemi che hanno portato allo sviluppo dello standard IEEE 802.21.

1.1 La situazione odierna

Il mondo di oggi è in continua evoluzione, ogni giorno abbiamo nuovi dispositivi con hardware sempre migliore non solo sotto un punto di vista qualitativo, bensì anche quantitativo, nel senso che i nuovi *devices* hanno sempre più elettronica a bordo per poter sfruttare le nuove tecnologie, in particolare riguardo alla connettività, diventata negli ultimi anni un aspetto fondamentale della vita di tutti i giorni. Il fatto che ormai la maggioranza dei nuovi dispositivi sia dotata di più interfacce di rete spiana la strada a molte opportunità, ma, allo stesso tempo, pone anche degli ostacoli lungo il cammino. Chiunque possieda uno smartphone recente può connettersi alla rete in più modi, come Wi-Fi, GSM, UMTS, HSDPA, LTE, etc. Se, per esempio, siamo attualmente connessi tramite una connessione wireless ed usciamo dalla zona di copertura dell'*access point*, il *device* si collegherà automaticamente attraverso un'altra tecnologia, attivando le ricetrasmittenti adatte oppure se usciamo dalla zona di copertura della nostra attuale connessione GSM, il

dispositivo si connetterà autonomamente ad un'altra torretta. L'atto di cambiare l'*access point* oppure addirittura la tecnologia utilizzata per ristabilire la connessione è detto *handover* e può essere più tipi.

1.2 Tipi di handovers

I due esempi sopra citati descrivono rispettivamente uno scenario di *handover* verticale ed orizzontale, a seconda della tecnologia utilizzata prima e dopo il passaggio. Oltre a questo classificazione, è possibile descrivere altri due tipi di *handover*, a seconda che le connessioni aperte al momento del passaggio vengano o meno preservate nel passaggio.

1.2.1 Horizontal handover

Nel caso del telefonino che cambia la propria BTS[1], si parla di un *handover* orizzontale, i.e. la tecnologia adoperata a livello *datalink* non cambia. Stesso scenario se, muovendoci, ci si collega man mano agli *access points* Wi-Fi con miglior ricezione e ciò può anche essere effettuato utilizzando due interfacce 802.11 contemporaneamente in modo da avere potenzialmente entrambe le connessioni attive al momento del passaggio, ma si parlerà sempre di un *handover* orizzontale.

1.2.2 Vertical handover

Se è necessario, invece, cambiare il livello *datalink* durante il passaggio, allora si tratta di un *handover* verticale. Sono i più utili da un punto di vista applicativo poiché permettono di sfruttare le diverse caratteristiche intrinseche di ogni particolare tecnologia, ovvero si potrebbe essere interessati ad utilizzare una connessione Wi-Fi qualora disponibile e passare su HSDPA quando si esce dalla zona di copertura dell'attuale *access point*, in modo da poter mantenere attiva la connessione, indipendentemente dal mezzo trasmissivo.

[1]Base Transceiver Station

1.2.3 Hard handover

Si tratta di un *handover* dove le connessioni attive prima del passaggio vengono chiuse, incaricando il dispositivo stesso di ripristinarle da zero successivamente. Viene anche definito come *break-before-make*.

1.2.4 Soft handover

In questo caso le connessioni aperte vengono mantenute anche dopo il passaggio, in modo da preservare lo stato della rete e consentire un passaggio trasparente ed indolore all'utente. Questo è realizzato stabilendo una connessione verso il nuovo *access point* preventivamente, in modo che, prima che il collegamento salti, si abbia già un altro canale pronto all'uso. Viene anche definito come *make-before-break*.

1.3 Problematiche

I problemi principali da gestire sono:

1. come e quando debbano effettuarsi le decisioni di *handover*.
2. come debba avvenire effettivamente il passaggio tra una tecnologia e l'altra.
3. come mantenere lo stato delle connessioni prima e dopo il passaggio.

Lo standard IEEE 802.21, preso in esame in questo lavoro, mira a risolvere il primo di questo punti, fornendo dei servizi *media-independent* per prendere le giuste decisioni di *handover* in un modo standardizzato ed indipendente dalla tecnologia utilizzata al momento, ma non stabilisce come effettivamente debba avvenire il passaggio tra una particolare tecnologia ed un'altra. Per il secondo problema è necessario stabilire metodi di passaggio specifici per ogni coppia di tecnologie, ad esempio lo standard GAN[2] per le operazioni di *handover* tra Wi-Fi e GSM, ma anche tra stesse tecnologie come lo

[2]https://en.wikipedia.org/wiki/Generic_Access_Network

standard IEEE 802.11r. Per l'ultimo problema è possibile studiare soluzioni come il *Mobile IP*[1] dell'IETF[3], il quale permette il *routing* dei pacchetti indipendentemente dalla posizione fisica del *device* mantenendo un indirizzo IP permanente.

[3]Internet Engineering Task Force

Capitolo 2

Lo standard IEEE 802.21

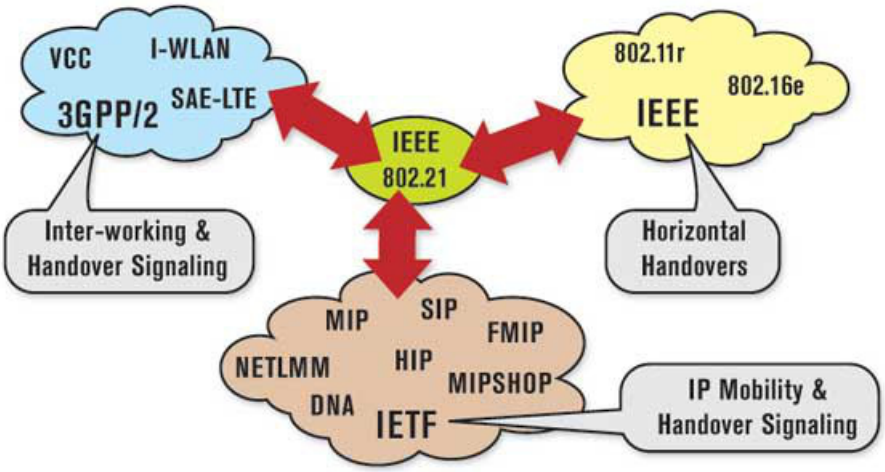

Figura 2.1: Rapporto con gli altri standards

In questo capitolo sarà esposto lo standard IEEE 802.21[2], illustrandone le componenti e le loro funzionalità, al fine di avere un quadro completo e poter comprendere il suo reale obbiettivo, ovvero creare un meccanismo standardizzato per poter prendere più facilmente decisioni di *handover*, in modo che sia *media-independent*, ovvero indipendente dal mezzo trasmissivo utilizzato.

2.1 Storia

Con il continuo diffondersi di nuove tecnologie e dispositivi dotati di più interfacce di rete, è diventato necessario dover formalizzare alcune funzionalità per facilitare un passaggio indolore da una rete all'altra. Il *working group* cominciò effettivamente i lavori nel marzo 2004 e la prima versione ufficiale dello standard fu pubblicata nel gennaio 2009. Gli eventi principali sono stati i seguenti:

- marzo 2003: creazione IEEE 802.21 ECSG[1]
- marzo 2004: creazione IEEE 802.21 WG[2]
- settembre 2004: analisi dei requisiti
- ottobre 2004: raccolta delle proposte
- maggio 2005: formulazione di una proposta unica
- luglio 2005: inizio discussione della proposta
- luglio 2007: IEEE 802 Sponsor Ballot[3]
- gennaio 2009: pubblicazione dello standard

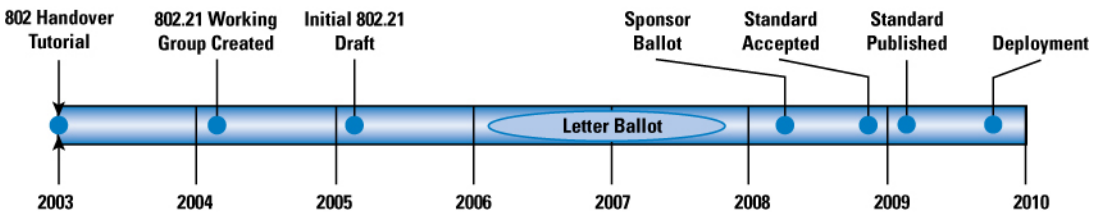

Figura 2.2: IEEE 802.21 Timeline

[1]Executive Committee Study Group
[2]Working Group

2.2 Finalità

Lo standard IEEE 802.21 si pone l'obbiettivo di aiutare i nodi mobili a prepararsi ad eventuali azioni di *handover* da una rete ad un'altra, ma non specifica come deve avvenire la migrazione. Sono infatti definite tutte le funzionalità per acquisire informazioni sullo stato delle varie interfacce, ma non è specificato come e quando debba effettivamente avvenire l'eventuale passaggio. L'utente è in grado di conoscere lo stato delle connessioni disponibili attraverso la ricezione di eventi dal proprio MIHF[3], quali *link_up* e *link_down*, e può richiedere esplicitamente informazioni aggiuntive su una interfaccia inviando delle specifiche richieste ad un particolare SAP[4], come l'RSSI[5] di una propria interfaccia 802.11 attualmente connessa. Lo standard 802.21 da solo non basta per gestire i meccanismi di *handovers*, poiché, ad esempio, per riuscire ad effettuare una migrazione *seamlessly* delle connessioni attualmente attive, mantenendo quindi lo stato dei flussi aperti prima e dopo l'*handover*, sarebbe necessaria anche una migrazione dell'indirizzo di terzo livello attraverso, ad esempio, Mobile IP[1], altrimenti, anche dopo aver instaurato con successo la nuova connessione, il *peer* remoto non potrebbe sapere il nuovo indirizzo del nodo a cui era connesso precedentemente.

2.3 Architettura

Lo standard IEEE 802.21 definisce più entità, ognuna con il proprio specifico compito:

- MIHF: implementa il core delle funzionalità offerte dallo standard.
- MIH-SAP: fornisce una astrazione *media-independent* per tutti i tipi di interfacce.

[3]Media Independent Handover Function
[4]Service Access Point
[5]Received Signal Strength Indication

- MIH-LINK-SAP: fornisce una astrazione *media-specific* per una singola tecnologia.
- MIH-NET-SAP: permette la gestione di MIHF remoti.
- MIH-User: è l'entità che si sottoscrive ad un MIHF per usufruire dei servizi offerti.

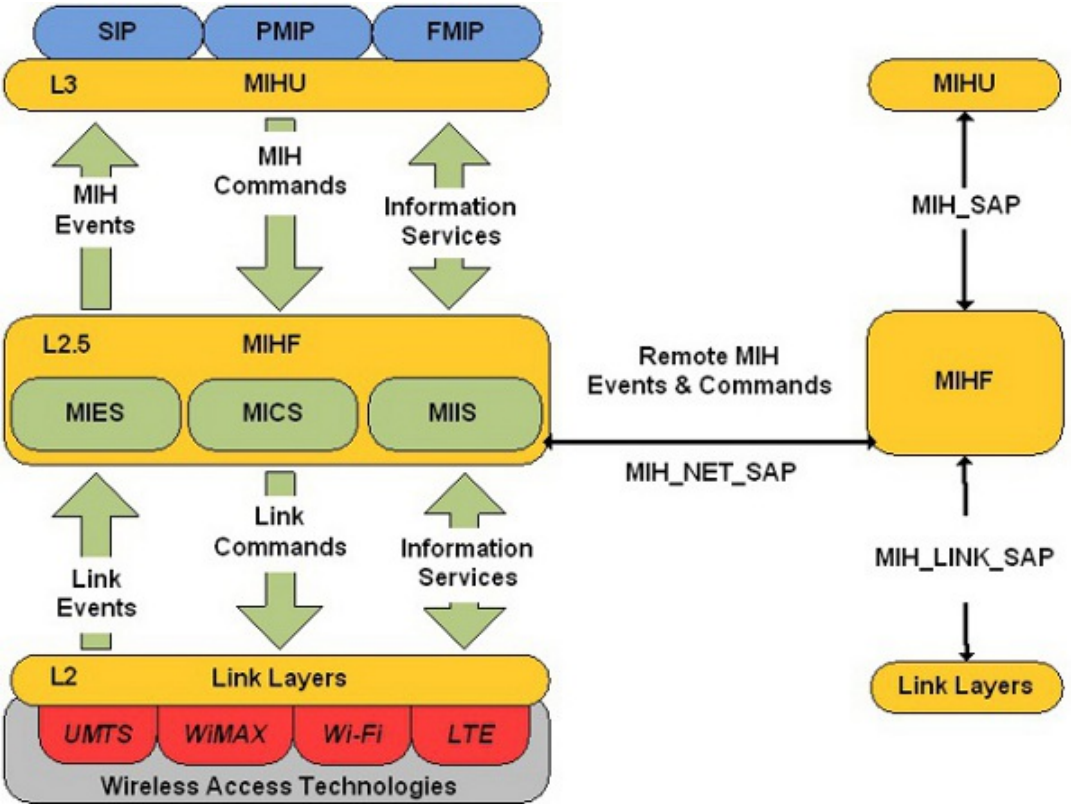

Figura 2.3: Architettura IEEE 802.21

2.3.1 MIHF

Questo componente è il più importante: i nodi mobili devono sottoscriversi ad un MIHF, il quale invierà loro una conferma con informazioni sui collegamenti disponibili, come i rispettivi indirizzi di livello due ed il tipo di

tecnologia, ed un elenco di eventi e comandi supportati. In sostanza, l'MIHF fornisce un insieme di servizi astratti ai livelli superiori indipendentemente dalle tecnologie adottate dalle singole interfacce. L'implementazione della Media Independent Handover Function è formata da tre sottocomponenti:

- MIES (Media Independent Event Services): ha il compito di propagare gli eventi a tutte le parti interessate come cambiamenti dinamici dello stato a livello *data link*, i.e. il secondo strato dello stack di rete, agli strati soprastanti del sistema locale o di sistemi remoti. Gli eventi possono essere generati dall'MIHF stesso oppure dai livelli sottostanti. Il flusso dei dati generati da questo componente ha quindi un verso prettamente *bottom-up*.

Nome	Descrizione
Link_Up	il collegamento L2 torna attivo
Link_Down	il collegamento L2 cade
Link_Detected	nuovo collegamento disponibile
Link_Going_Down	il collegamento sta per saltare
Link_Handover_Imminent	sta per essere eseguita una procedura di *handover*
Link_Handover_Complete	la procedura di *handover* è stata completata

Tabella 2.1: Lista degli eventi specificati nello standard

- MICS (Media Independent Command Services): è incaricato di propagare i comandi ricevuti dagli strati più alti dello stack di rete verso il basso per aiutare il nodo mobile ad eseguire più facilmente procedure di *handover*, ad esempio una richiesta esplicita per la riconfigurazione per un'interfaccia, per comunicare ad un MIHF remoto di procedere con l'esecuzione di un *handover* oppure una richiesta di informazioni più dettagliate sullo stato di un collegamento specifico, e.g. richiedere il valore RSSI di un'interfaccia wireless. I comandi possono essere originati sia dall'MIH-User, sia dall'MIHF stesso. La destinazione dei comandi può essere per i livelli inferiori dello stack locale oppure di

uno stack remoto, propagando il comando al peer MIHF appropriato. Il flusso dei dati è prettamente *top-down*.

- MIIS (Media Independent Information Services): è la componente incaricata di gestire richieste di informazioni e le opportune risposte. Ad esempio, un nodo mobile può essere interessato ad informazioni aggiuntive riguardo uno specifico link, come l'RSSI di una interfaccia 802.11, inviando una richiesta all'MIHF a cui è sottoscritto tramite il MICS ed aspettando una risposta dall'MIIS. Lo scambio di informazioni è effettuato secondo un modello RDF[6] basato su XML[7], variants[8] oppure TLV[9]. In questo caso non è possibile stabilire un verso per il flusso delle informazioni, poiché uno strato superiore dello stack potrebbe richiedere informazioni circa un *layer* sottostante o viceversa.

2.3.2 SAPs

Questo componente è incaricato di interagire con le interfacce di rete, al fine di esporre delle primitive di servizi di basso livello di una singola interfaccia ad un MIHF, al quale deve essersi preventivamente registrato. I Service Access Points possono essere di due tipi: *media-independent* e *media-specific*. I primi forniscono servizi astratti per tutte le possibili tipologie di interfacce di rete, come la generazione di eventi *link_down* e *link_up*, e, di conseguenza, offrono un minor numero di servizi. I secondi forniscono servizi specifici per una singola tecnologia, e.g. 802.3 o 802.11, e per questo motivo sono più espressivi dei primi: ad esempio, in seguito ad una richiesta *get_link_parameter* è naturale aspettarsi risposte differenti per ogni famiglia di tecnologie, come l'RSSI che assume valori differenti a seconda della tipologia dell'interfaccia interrogata.

[6] https://en.wikipedia.org/wiki/Resource_Description_Framework
[7] https://en.wikipedia.org/wiki/Extensible_Markup_Language
[8] https://en.wikipedia.org/wiki/Variant_type
[9] https://en.wikipedia.org/wiki/Type-length-value

2.3.3 MIH-User

L'entità che si sottoscrive ad un MIHF è definita MIH-User. Esso invia una richiesta di *capability_discover* e riceve come risposta una lista di interfacce, eventi e comandi disponibili. Una volta ricevuto questo messaggio, l'utente deciderà a quali interfacce è interessato ed invierà di conseguenza una richiesta di sottoscrizione per ogni interfaccia scelta. In seguito l'utente potrà ricevere eventi generati dal cambio di stato di un collegamento ed inviare richieste e comandi al proprio MIHF che si preoccuperà di interagire a sua volta con i SAPs che gestiscono le interfacce interpellate.

2.4 Esempio d'uso

Si supponga di disporre di un dispositivo mobile con più interfacce di rete wireless e che sia connesso tramite una di queste. Esso potrebbe richiedere informazioni all'MIHF a cui è registrato su tutti i collegamenti disponibili, indipendentemente dalla tipologia delle singole. In questo modo l'utente potrebbe conoscere preventivamente lo stato di un potenziale collegamento tramite una differente tecnologia senza dover accendere fisicamente sul proprio dispositivo la ricetrasmittente adatta, con un potenziale guadagno sui consumi di elettricità e maggior efficienza nello sfruttamento dell'etere. Si potrebbe anche implementare una sorta di *network manager* che di volta in volta si preoccupi di sottoscriversi ad ogni MIHF e di decidere l'interfaccia migliore secondo un *tradeoff* tra consumo energetico e qualità del segnale, sempre supponendo di aver a disposizione soluzioni di *IP mobility*.

2.5 Stato dell'arte

Attualmente è possibile testare lo standard in due modi: eseguendo una sua implementazione oppure simulandolo. La maggior parte della letteratura esistente si concentra su simulazioni all'interno di *ns-2*[4]. Per quanto riguarda

le implementazioni esistenti, la più utilizzata è ODTONE[5], la quale sarà analizzata nel dettaglio successivamente.

Nome	Descrizione
Link_Get_Parameters	permette di richiedere informazioni aggiuntive riguardo un collegamento specifico
Link_Configure_Thresholds	permette di definire dei valori-soglia che facciano scattare degli eventi
Link_Capability_Discover	permettere di ottenere la lista di eventi e comandi supportati da un MIHF
Link_Event_Subscribe	permettere di registrarsi per essere avvisati di cambiamenti di stato di una particolare interfaccia
Link_Event_Unsubscribe	permettere di la sottoscrizione effettuata con il comando precedente
Link_Actions	permette di comandare esplicitamente l'interfaccia (e.g. *power_up*, *power_down*)
Net_Ho_Candidate_Query	in *handovers* iniziati dalla rete, segnala al nodo mobile di procedere al passaggio ad uno dei network candidati
Net_Ho_Commit	conferma all'esecuzione dell'*handover*
N2n_Ho_Query_Resources	prima di confermare l'esecuzione dell'*handover* su un altro network, l'attuale PoS[10] contatta il candidato per controllare che abbia le risorse per accettare un nuovo nodo mobile
N2n_Ho_Commit	il passaggio verso il nuovo PoS è confermato
N2n_Ho_Complete	la procedura di *handover* è stata completata
Mn_Ho_Candidate_Query	in *handovers* iniziati dal nodo mobile, serve per richiedere la lista dei network candidati per un eventuale passaggio
Mn_Ho_Commit	la procedura di *handover* è stata confermata
Mn_Ho_Complete	la procedura di *handover* è stata completata

Tabella 2.2: Lista dei comandi specificati nello standard

Capitolo 3

ODTONE

In questo capitolo verrà descritta, analizzata e testata l'implementazione denominata ODTONE[5], descrivendone le funzionalità attualmente implementate ed illustrando punto per punto tutti i passaggi richiesti per compilare ed eseguire con successo questa implementazione, partendo direttamente dal codice disponibile presso il repository ufficiale.

3.1 Descrizione

Esso è una implementazione open-source dello standard IEEE 802.21 rilasciata con licenza LGPLv3[6] ed è realizzato in C++ e reso multipiattaforma tramite la libreria *Boost*[7]: è possibile infatti eseguire ODTONE su sistemi Linux, Windows, Android ed OpenWrt.

3.2 Funzionalità implementate

Questo progetto mira a fornire una implementazione dell'MIHF compatibile *out-of-the-box* con tutti i principali sistemi operativi esistenti e, di conseguenza, essendo tuttora in stato sperimentale[1], fornisce solo le funzionalità principali oppure facilmente implementabili in chiave multi-piattaforma.

[1] è attualmente disponibile la versione 0.6

ODTONE è un'implementazione quasi completa, ovvero il core supporta praticamente tutto lo standard mentre i SAPs forniti, al momento, sono solo per sistemi Linux e Windows e limitatamente per interfacce 802.3 e 802.11.

3.2.1 MIHF

La *Media Independent Handover Function* fornita è composta dai tre sottocomponenti definiti dallo standard IEEE 802.21:

- *Media Independent Event Service* (MIES): ha il compito di vagliare gli eventi ricevuti per controllare che siano conformi allo standard e di propagarli alle entità che hanno richiesto la sottoscrizione. Più nel dettaglio:
  - Event Validator: controlla che l'evento sia conforme allo standard.
  - Event Subscriber: si occupa delle sottoscrizioni agli eventi.
  - Event Publisher: si occupa di propagare gli eventi agli interessati.
- *Media Independent Command Service* (MICS): ha il compito di vagliare i comandi ricevuti e recapitarli a destinazione. Più nel dettaglio:
  - Command Validator: controlla che i comandi ricevuti siano conformi allo standard.
  - Command Publisher: propaga i comandi ai diretti interessati.
- *Media Independent Command Service* (MIIS): ha il compito propagare le richieste all'IS[2] e inviare indietro le risposte.

Questi componenti forniscono i servizi basilari definiti nello standard. In aggiunta, ODTONE implementa altri componenti per fornire ulteriori funzionalità e arricchire quelle già esistenti:

[2]Information Service

- Service Manager: è responsabile della gestione delle richieste *Capability_Discover*, le quali servono per richiedere la lista di eventi e comandi supportati dall'MIHF.
- Communication handler: raccoglie i messaggi ricevuti da tutti i SAPs o altre entità e li invia al modulo Service Access Controller.
- Service access controller: è responsabile di analizzare l'*header* dei messaggi ricevuti e farli gestire dai dovuti servizi. La decisione è presa in base all'identificativo del tipo di messaggio ed ai *callbacks* registrati.
- Link manager: gestisce le informazioni sui Link SAPs disponibili.
- MIH-User manager: gestisce le informazioni su MIH-Users registrati.
- Peer MIHF manager: gestisce le informazioni su MIHFs remoti.
- Transaction state machine controller: mantiene gli stati di ogni transazione con MIHFs remoti ed è responsabile di inviare messaggi di *acknowledge* ed individuare messaggi duplicati.

3.2.2 MIH-Sap

Fornisce un'astrazione *media-independent* per un gran numero di interfacce, ma supporta solo la generazione di eventi *link_up* e *link_down*. Su sistemi Linux utilizza *rtnetlink* e la libreria *libnl*[8] per gestire le interfacce mentre su Windows utilizza le *SDK Libraries* ufficiali. Le tecnologie supportate sono:

- IEEE 802.3
- IEEE 802.11
- IEEE 802.16
- IEEE 802.20
- IEEE 802.22

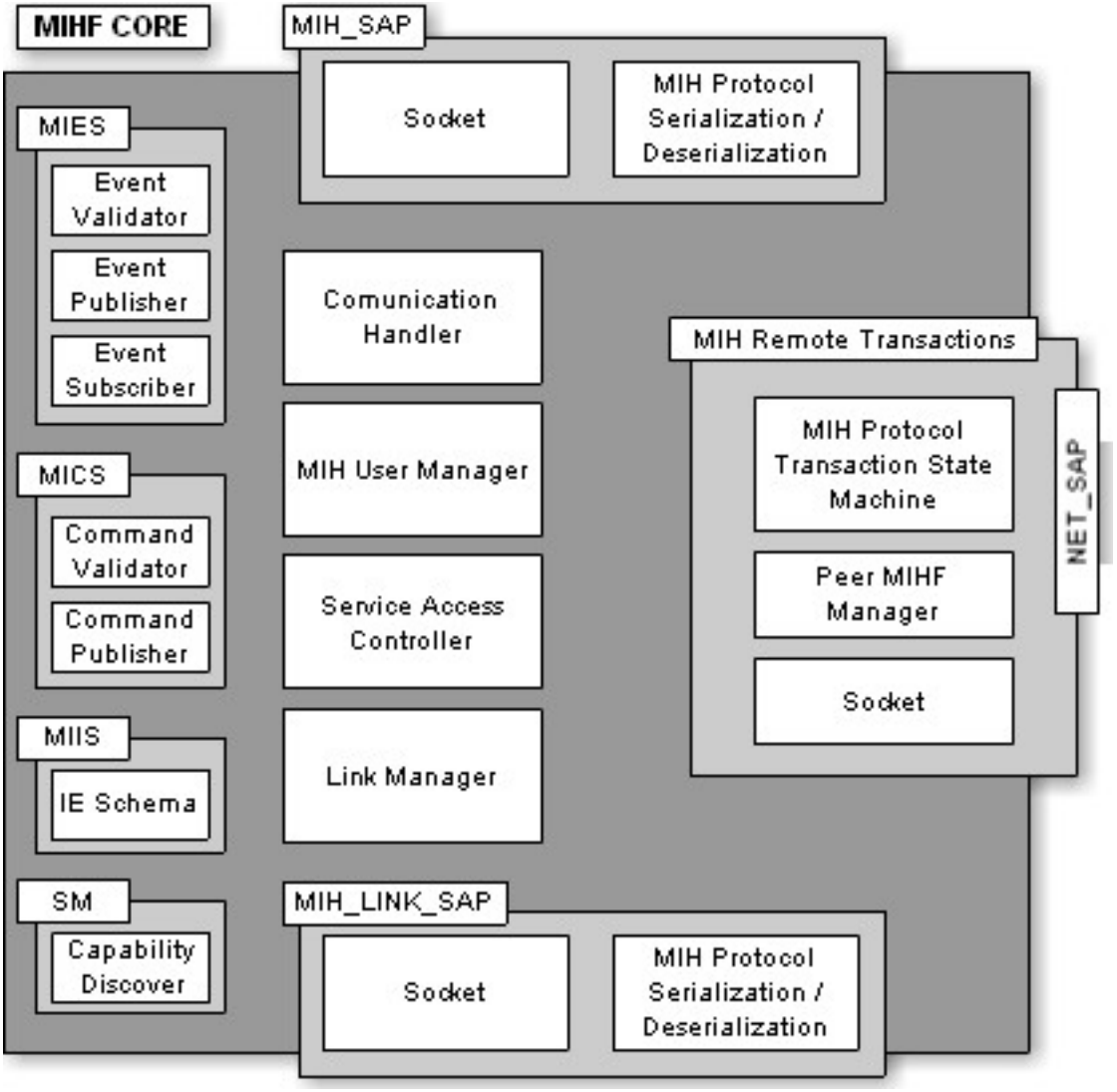

Figura 3.1: Visione concettuale del MIHF di ODTONE

- 3GPP GSM
- 3GPP GPRS
- 3GPP EDGE
- 3GPP CDMA2000
- 3GPP CDMA2000-HRPD
- 3GPP UMTS

3.2.3 MIH-Link-Sap 802.3

Fornisce un SAP specifico per la tecnologia IEEE 802.3. Supporta la generazione dei seguenti eventi:

- Link_Up: generato quando viene stabilita una connessione a livello *datalink*, i.e. viene settato il flag *IF_OPER_UP*.
- Link_Down: generato quando la connessione viene interrotta, i.e. viene disattivato il flag *IF_OPER_UP*.
- Link_Parameters_Report: generato in risposta ad una richiesta *Link_Get_Parameters*, allo scadere di un timer periodico impostabile oppure quando scatta uno dei *triggers* specificati con la richiesta *Link_Configure_Thresholds*. Al momento supporta solo i parametri *DATA_RATE* e *PACKET_ERROR_RATE*, ottenuti tramite il comando Netlink *RTNL_GET_LINK*. Si noti che il dato sui pacchetti persi è una statistica *all-time*.

Supporta i seguenti comandi:

- Capability_Discover: risponde con una lista di eventi e comandi supportati dal SAP.
- Event_Subscribe: permette di sottoscriversi agli eventi richiesti.

- Event_Unsubscribe: permette di annullare l'iscrizione agli eventi specificati.
- Link_Get_Parameters: permette di richiedere esplicitamente l'invio di un *Link_Parameters_Report*.
- Link_Configure_Thresholds: permette di configurare dei valori-soglia che quando oltrepassati faranno generare uno specifico evento.
- Link_Action: permette di richiedere al kernel tramite il modulo *if_8023* certe operazioni sull'interfaccia. Al momento supporta solo i comandi:
  - *LINK_DISCONNECT*: richiede la disconnessione.
  - *LINK_POWER_UP*: accende l'interfaccia.
  - *LINK_POWER_DOWN*: spegne l'interfaccia.

  Per poter usufruire di queste funzionalità bisogna avviare il SAP tramite un utente che abbia la capability *CAP_NET_ADMIN*, e.g. root.

3.2.4 MIH-Link-Sap 802.11

Fornisce un SAP specifico per la tecnologia IEEE 802.11. Supporta la generazione dei seguenti eventi:

- Link_Detected: al termine di ogni scan delle reti 802.11 disponibili, il kernel invia un messaggio *NL80211_CMD_NEW_SCAN_RESULTS*. Per ogni *entry* ottenuta, il SAP esegue una chiamata *NL80211_CMD_GET_SCAN* per ottenere le informazioni da scrivere nell'evento.
- Link_Up: generato quando il modulo *if_80211* riceve l'evento *NL80211_CMD_CONNECT* dal kernel.
- Link_Down: generato quando il modulo *if_80211* riceve l'evento *NL80211_CMD_DISCONNECT* dal kernel.

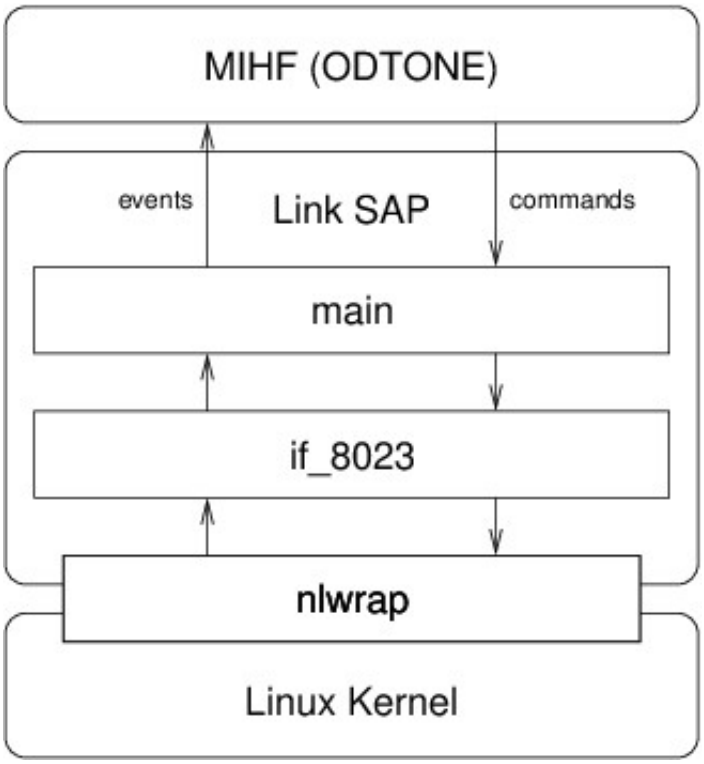

Figura 3.2: Visione concettuale del Link_SAP per 802.3

- Link_Parameters_Report: generato in risposta ad una richiesta *Link_Get_Parameters*, allo scadere di un timer periodico impostabile oppure quando scatta uno dei *triggers* specificati con la richiesta *Link_Configure_Thresholds*. Al momento supporta i parametri *DATA_RATE*, *PACKET_ERROR_RATE* e *SIGNAL_STRENGTH*, ottenuti tramite il comando Netlink *RTNL_GET_LINK*. Si noti che il dato sui pacchetti persi è una statistica *all-time* e *SIGNAL_STRENGTH* non è altro che l'RSSI.

Supporta i seguenti comandi:

- Capability_Discover: risponde con una lista di eventi e comandi supportati dal SAP.

- Event_Subscribe: permette di sottoscriversi agli eventi richiesti.

- Event_Unsubscribe: permette di annullare l'iscrizione agli eventi specificati.
- Link_Get_Parameters: permette di richiedere esplicitamente l'invio di un *Link_Parameters_Report*.
- Link_Configure_Thresholds: permette di configurare dei valori-soglia che quando oltrepassati faranno generare uno specifico evento.
- Link_Action: permette di richiedere al kernel tramite il modulo *if_8023* certe operazioni sull'interfaccia. Al momento supporta i comandi:
  - *LINK_DISCONNECT*: richiede la disconnessione.
  - *LINK_POWER_UP*: accende l'interfaccia.
  - *LINK_POWER_UP*: spegne l'interfaccia.
  - *LINK_LOW_POWER*: mette l'interfaccia in modalità a basso consumo energetico. (non supportato da vecchie versioni del kernel)
  - *LINK_SCAN*: richiede esplicitamente uno scan della rete.

  Per poter usufruire di queste funzionalità bisogna avviare il SAP tramite un utente che abbia la capability *CAP_NET_ADMIN*, e.g. root.

3.2.5 La libreria libodtone

Per poter scrivere un programma che interagisca con l'MIHF, quindi un MIH-User oppure un SAP, è possibile utilizzare la libreria inclusa *libodtone*, la quale fornisce tutte le funzioni necessarie per comunicare correttamente con il core: è disponibile come unica libreria solo dalla versione 0.6 di ODTONE ed è l'unione delle precedenti librerie 'base', 'mih', 'sap' e 'net'[9]. Fornisce tutte le classi *helper* necessarie per facilitare lo sviluppo delle applicazioni utente, come servizi di debug, gestione di liste, log, gestione delle eccezioni e generazione di numeri pseudo-casuali. Definisce inoltre tutti i tipi di dato del

protocollo MIH specificati nello standard e fornisce delle classi per facilitare la creazione dei messaggi, il loro *parsing*, la loro ricezione ed il loro l'invio. Per gli ultimi due, i servizi sono implementati sulla base della classe della libreria Boost *boost::asio* che fornisce comunicazione tramite I/O asincrono, unica tipologia di comunicazione offerta dalla libreria: è necessario infatti eseguire qualsiasi operazione asincronicamente, registrando gli opportuni *callbacks* tramite la funzione *boost::bind()*.

3.3 Compilazione ed esecuzione

Vengono di seguito illustrati tutti i passaggi per compilare ed eseguire per la prima volta ODTONE recuperando il codice dal repository ufficiale su un sistema Debian[10] *Wheezy*.

3.3.1 Compilazione

La procedura per compilare ODTONE dal repository ufficiale è la seguente (al momento c'è un problema con le dipendenze da risolvere manualmente[11]):

1. prelevare la prima parte di dipendenze necessarie:
   ```
   # apt-get update
   # apt-get install build-essential git realpath cmake autoconf
   automake libboost-all-dev
   ```

2. prelevare l'ultima versione:
   ```
   $ git clone https://github.com/ATNoG/ODTONE.git odtone
   ```

3. prelevare i sottomoduli:
   ```
   $ git submodule update -init
   ```

4. passare momentaneamente al ramo *testing* di Debian:
   ```
   # vi /etc/apt/sources.list
   ```
   e sostituire la parola *stable* o *wheezy* in *testing*, e.g.:
   ```
   deb http://mi.mirror.garr.it/mirrors/debian/ testing main
   ```

5. prelevare la seconda parte di dipendenze necessarie:
   ```
   # apt-get update
   # apt-get install librdf0-dev libnl-3-dev
   libnl-route-3-dev libnl-genl-3-dev
   ```

6. ritornare al ramo *stable* di Debian:
   ```
   # vi /etc/apt/sources.list
   ```
   e.g.:
   ```
   deb http://mi.mirror.garr.it/mirrors/debian/ stable main
   ```

7. procedere alla compilazione:
   ```
   $ cd odtone
   $ cmake .
   $ make -j2
   ```

3.3.2 Esecuzione

Appena finita la compilazione, è necessario rendere disponibili le librerie appena compilate al sistema.

1. creare dei links simbolici in /usr/lib:
   ```
   # ln -s $(realpath lib/odtone/libodtone.so)
   /usr/lib/libodtone.so.0.5
   # ln -s $(realpath lib/external/libnl/nlwrap/libnlwrap.so)
   ```

```
/usr/lib/libnlwrap.so.0.5
```

2. eseguire per primo l'MIHF:
   ```
   $ ./src/mihf/odtone-mihf
   ```

3. eseguire un SAP per ogni interfaccia, inserendo l'indirizzo MAC appropriato:

   802.11:
   ```
   # ./app/sap_80211_linux/odtone-sap_80211
   --link.link_addr <MAC>
   ```

   802.3:
   ```
   # ./app/sap_8023/odtone-sap_8023 --link.link_addr <MAC>
   ```

4. infine eseguire l'MIH-User d'esempio fornito:
   ```
   $ ./app/mih_usr/odtone-mih_usr --dest mihf1
   ```

Una volta che ogni componente sarà avviato, è possibile eseguire l'MIH-User che invierà una richiesta *capability_discover* per sapere quali interfacce sono disponibili e, per ognuna di queste, una richiesta di sottoscrizione. Ora è possibile scollegarsi dalla rete wireless oppure staccare fisicamente il cavo CAT5 per veder l'MIH-User ricevere l'evento *link_down* e segnalarlo su *stdout*. Una volta ripristinato il collegamento, vedremo l'MIH-User ricevere un evento *link_up*. Per realizzare il programma MIH-proxy, descritto nel prossimo capitolo, è stato utilizzato come base per il codice l'MIH-User ufficiale.

3.4 Risoluzione problemi

Può capitare che la compilazione non vada a buon fine per degli errori della libreria Boost. In tal caso bisogna modificare il file xtime.hpp della libreria Boost:
`# vi /usr/include/boost/thread/xtime.hpp`
sostituendo TIME_UTC con TIME_UTC_:

```
--- xtime.hpp   2014-02-18 03:26:56.291068347 +0100
+++ xtime1.hpp  2014-02-18 03:27:23.679204146 +0100
@@ -20,7 +20,7 @@

 enum xtime_clock_types
 {
-    TIME_UTC=1
+    TIME_UTC_=1
 //    TIME_TAI,
 //    TIME_MONOTONIC,
 //    TIME_PROCESS,
```

Per quanto riguarda l'esecuzione, se dobbiamo eseguire più istanze dello stesso SAP, bisogna modificare il file .conf contenuto nella directory dove si trova l'eseguibile assegnando differenti indirizzi per distinguere l'istanza del SAP e differenti porte su cui mettersi in ascolto per la ricezione di messaggi dall'MIHF.

3.5 Sviluppi futuri

Per quanto riguarda lo sviluppo di ODTONE, la priorità maggiore rimane arricchire l'insieme dei SAPs disponibili poiché ufficialmente viene fornito un SAP generico per tutte le interfacce, ma che supporta solo gli eventi *link_up*

e *link_down*, e dei LINK_SAP specifici solo per 802.3 e 802.11 per sistemi Linux e Windows. Bisogna quindi completare il lavoro implementando il supporto per altre tecnologie, specialmente per la famiglia 3GPP, e completare la lista di comandi ed eventi supportati da ogni SAP.

Capitolo 4

MIH-proxy

In questo capitolo verrà presentato il programma MIH-proxy realizzato per testare le potenzialità di ODTONE tramite un'applicazione reale, descrivendone la logica ed il funzionamento. Saranno inoltre descritti tutti i passi necessari per la configurazione e per l'esecuzione, al fine di creare prima un flusso unidirezionale che possa sfruttare tutte le interfacce disponibili per inviare dati e poi anche bidirezionale in modo che possa sfruttare tutti i collegamenti anche per ricevere.

4.1 Descrizione

MIH-proxy è un proxy ad alta affidabilità che permette di sfruttare più interfacce di rete per inviare e ricevere traffico ed è disponibili in due varianti: unidirezionale e bidirezionale. Nel primo, il trasmettitore può sfruttare più interfacce per inviare pacchetti UDP verso un peer remoto, il quale, per ovvie ragioni, non deve considerare l'IP sorgente, poiché i dati in arrivo potrebbero provenire da più IP sorgenti. Nel secondo, sia il trasmettitore, sia il ricevente possono aver più interfacce di rete su cui può passare traffico bidirezionale, occorre quindi una seconda istanza del proxy anche sul ricevente che si preoccupi di monitorare il traffico in arrivo su più interfacce ed è così possibile anche far collegare due applicazioni che usano TCP trasformando dapprima

il flusso in pacchetti UDP, instradandolo sull'interfaccia corretta e ricostruendo il flusso originario una volta a destinazione. Si è deciso di tenere separati l'applicativo per trasformare un flusso TCP in pacchetti UDP e quello per gestire l'invio e ricezione sulle varie interfacce per delegare compiti appropriati e coerenti ad ogni componente, i quali potrebbero avere anche un utilizzo concreto anche se presi singolarmente.

4.2 Funzionamento

Entrambe le versioni del proxy sfruttano una implementazione open-source dello standard IEEE 802.21 per conoscere lo stato dei links disponibili. L'implementazione utilizzata è *ODTONE*[5]. Sfortunatamente è solo un'implementazione parziale dello standard, ad esempio implementa solo alcuni degli eventi definiti nello standard tra quelli generabili da ogni interfaccia.

4.2.1 Unidirezionale

La versione unidirezionale gestisce solo un traffico in uscita da più interfacce per limiti tecnici, ovvero, supponendo traffico bidirezionale, nel caso cadesse un'interfaccia del trasmettitore, il ricevente non saprebbe mai che l'IP da cui stava ricevendo non è più disponibile, trattandosi di pacchetti UDP, e quindi continuerebbe ad inviare ad un IP non raggiungibile. Per conoscere lo stato delle interfacce locali disponibili, il proxy rimane in ascolto di eventi *link_down* e *link_up* inviati dall'*MIHF* quando c'è un cambiamento di stato in una interfaccia, in modo da sapere attraverso quali interfacce poter inviare dati.

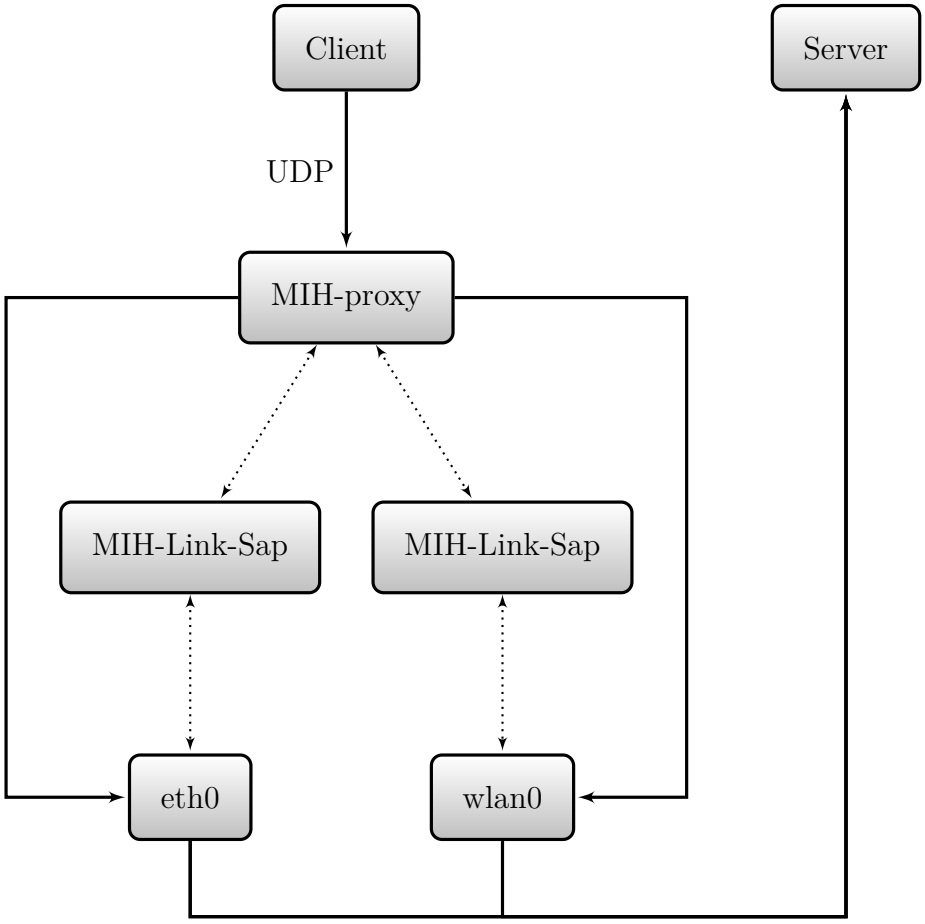

Figura 4.1: MIH-proxy unidirezionale

4.2.2 Bidirezionale

La versione bidirezionale necessita di due istanze del proxy, poiché ognuna deve poter inviare e ricevere su più interfacce. Per conoscere lo stato locale si comporta come la versione unidirezionale, per lo stato del peer remoto è stato necessario introdurre un sistema di *heartbeat* temporizzato con pacchetti vuoti per segnalare alla controparte i links funzionanti. Se dopo un certo periodo non si ricevono pacchetti da un dato IP, quella destinazione viene contrassegnata *down* e quindi non si invieranno più dati verso quell'IP fino a quando non sarà ricevuto un pacchetto di *heartbeat*. In questo modo ogni istanza è a conoscenza dello stato delle interfacce di rete locali e remote. Per

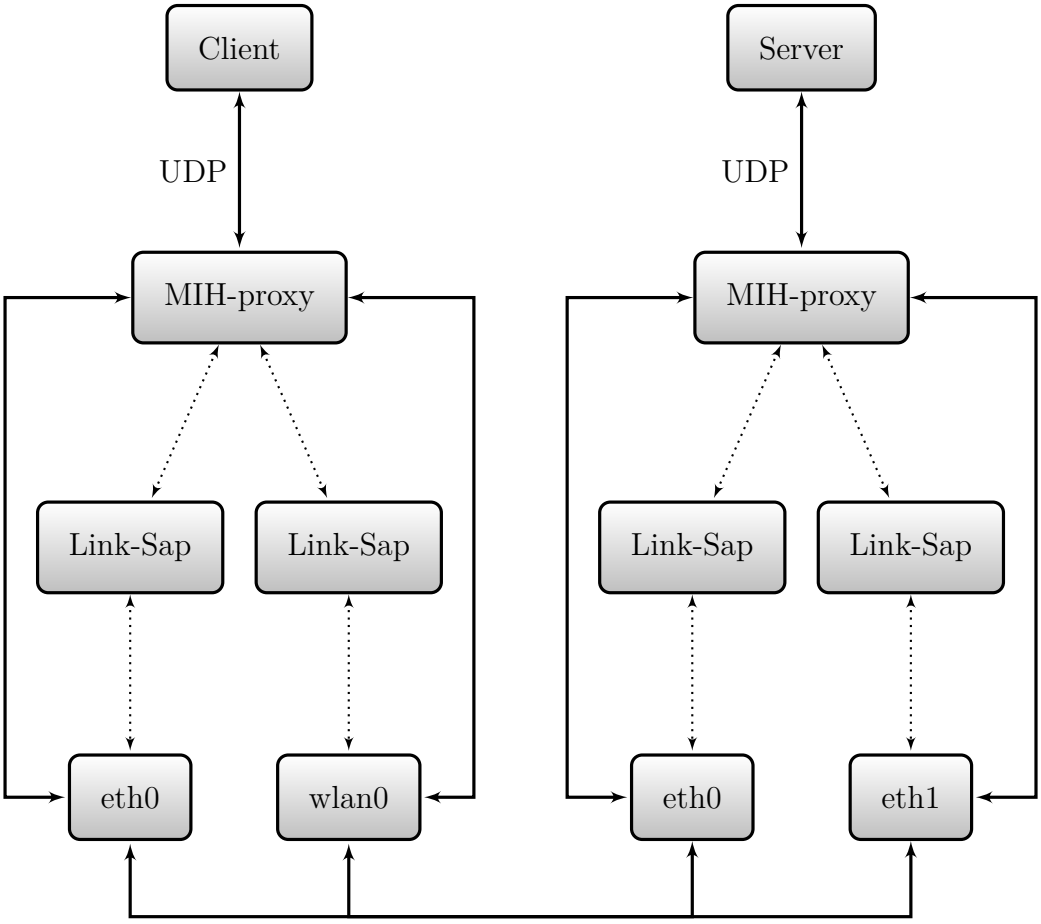

Figura 4.2: MIH-proxy bidirezionale senza phoxy

poter gestire il traffico generato da due applicazioni che si connettono tra loro tramite il protocollo TCP è possibile aggiungere un nuovo componente tra l'applicazione e il proxy in entrambi i peers che si occuperanno di trasformare il flusso TCP in datagrammi, gestire pacchetti persi, doppi o fuori ordine e ricostruire il flusso originario prima di consegnarlo al destinatario. Inoltre è possibile richiedere una cifratura *AES256* nella conversione da flusso a datagrammi.

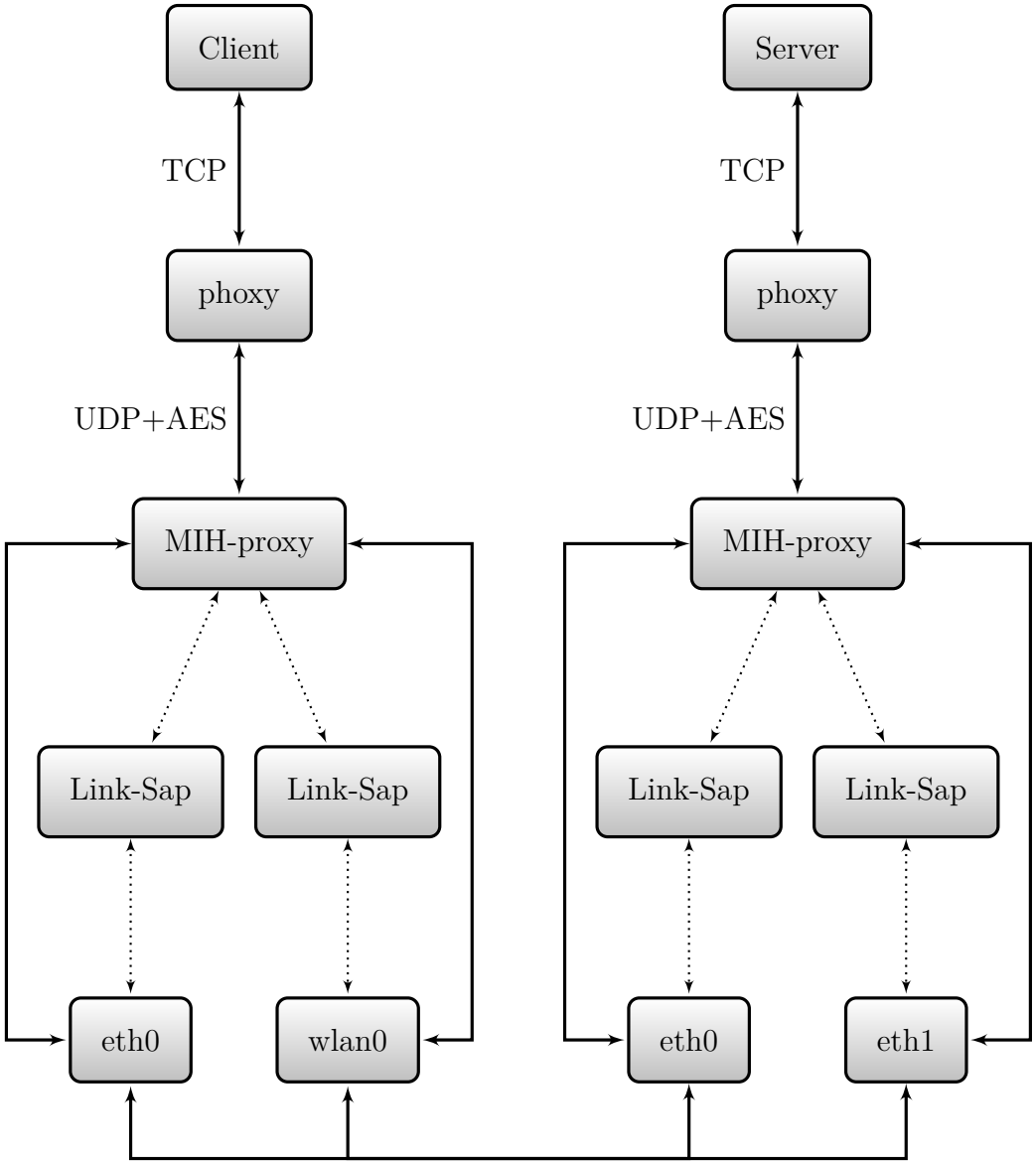

Figura 4.3: MIH-proxy bidirezionale con phoxy

4.3 Compilazione ed Esecuzione

Prima di tutto, bisogna recuperare il codice con il seguente comando:

```
$ git clone https://github.com/phra/802_21.git
```

Nel repository è contenuto tutto il codice necessario per eseguire MIH-proxy, quindi bisognerà seguire la stessa procedura per ODTONE per compilarlo

direttamente nella cartella oppure, se si ha a disposizione già ODTONE, basterà sostituire il file *app/mih_usr/mih_usr.cpp*.

4.3.1 Unidirezionale

Per compilare la versione unidirezionale dovremo rinominare il file *app/mih_usr/mih_usr_unidirectional.cpp*:

```
$ pushd app/mih_usr/
$ mv mih_usr.cpp mih_usr_bidirectional.cpp
$ mv mih_usr_unidirectional.cpp mih_usr.cpp
$ popd
```

Ora è necessario impostare i giusti IP per ogni MAC address, in modo che l'MIH-proxy possa sapere su quali indirizzi fare *bind(2)* per poter scegliere l'interfaccia desiderata, modificando coerentemente la funzione *interfaces::mac_to_ip()* in *app/mih_usr/mih_usr.cpp*.

```
1  std::string mac_to_ip(odtone::mih::mac_addr& mac) {
2      odtone::mih::mac_addr eth0mac("aa:bb:cc:dd:ee:ff");
3      odtone::mih::mac_addr eth1mac("ff:ee:dd:cc:bb:aa");
4      if (mac == eth0mac)
5          return "192.168.1.145";
6      else if (mac == eth1mac)
7          return "192.168.2.1";
8      log_(0,__FUNCTION__, " -> 127.0.0.1");
9      return "127.0.0.1";
10 }
```

Questa modifica manuale è obbligata poiché non esiste un modo *cross-platform* per conoscere gli IP associati ad una interfaccia. Bisogna poi impostare l'IP del destinatario nelle variabili private della classe *interfaces*:

```
class interfaces : boost::noncopyable {
    private:
        ...
        boost::asio::ip::udp::endpoint dest =
            boost::asio::ip::udp::endpoint(
                boost::asio::ip::address::from_string(
                    "127.0.0.1"),9999);
        ...
}
```

Successivamente bisognerà avviare come indicato nel capitolo precedente l'MIHF, ogni SAP per interfaccia ed il MIH-User modificato. Per testare il collegamento è fornito il server UDP *app/mih_usr/testproxy/udpserver.c* che stampa a schermo ciò che riceve indicando anche il mittente del datagramma. Non è possibile testarlo con *netcat*[12] poiché, in modalità UDP, riceve solo pacchetti dal primo mittente, quindi siccome il traffico potrebbe uscire potenzialmente da più interfacce, scarterebbe tutti i pacchetti non inviati dal primo mittente. Per compilare il server fornito basterà fare:

```
$ pushd app/mih_usr/testproxy
$ make
$ popd
```

e per metterlo in *listening* sulla porta locale 9999:

```
$ ./app/mih_usr/testproxy/udpserver 9999
```

Dopo aver eseguito tutti i passaggi basterà aprire *netcat* in modalità UDP per inviare pacchetti all'MIH-proxy e controllare il loro arrivo a destinazione:

```
$ nc -u 127.0.0.1 10000
```

Sarà ora possibile scollegare una interfaccia per segnalare al proxy di non utilizzare più quel collegamento e verificare che i pacchetti arrivino a destinazione tramite l'interfaccia rimasta attiva e successivamente ripristinare la connessione per controllare che il *link* venga effettivamente riutilizzato.

4.3.2 Bidirezionale

Per poter testare la versione bidirezionale, il procedimento è più complesso. Prima di tutto bisogna necessariamente avere a disposizione due computer ed ognuno dovrà avere in esecuzione MIHF ed i SAPs appropriati[1]. Bisognerà poi modificare la funzione *interfaces::mac_to_ip()* in modo coerente su entrambi i sistemi e siccome si potrebbero avere sia più interfacce per inviare, ma potenzialmente anche più destinazioni, bisognerà impostare queste ultime inizializzando le celle necessarie del vettore *interfaces::destinations* nel costruttore della classe:

```
1  #define LINK_TIMEOUT 10L /*timeout collegamento*/
2  #define HB_TIMEOUT 3 /*frequenza heartbeats*/
3  #define IP_SENDBACK "127.0.0.1" /*ip fruitore proxy o phoxy*/
4  #define PORT_SENDBACK 10001 /*porta fruitore proxy o phoxy*/
5  #define PORT_LSOCK 10000 /*porta per ricevere dal fruitore*/
6  #define PORT_DEST 11000 /*porta dell'altro proxy*/
7  #define PORT_DATA_SOCK 12000 /*porta invio di dati all'altro proxy*/
8  class interfaces : boost::noncopyable {
9      public:
10         interfaces() {
11             ...
12             destinations[1].up = true;
13             destinations[1].dest = new boost::asio::ip::udp::endpoint(
14                 boost::asio::ip::udp::endpoint(
15                     boost::asio::ip::address::from_string(
16                         "192.168.1.147"),PORT_DEST + 1));
17             destinations[0].up = true;
18             destinations[0].dest = new boost::asio::ip::udp::endpoint(
19                 boost::asio::ip::udp::endpoint(
20                     boost::asio::ip::address::from_string(
```

[1] se più SAPs per la stessa tecnologia bisogna editare il file .conf

```
21                          "192.168.2.2"),PORT_DEST));
22              ...
23          }
24      ...
25  }
```

Una volta che ogni sistema avrà in esecuzione tutte le istanze con gli indirizzi configurati correttamente sarà possibile eseguire *netcat* invocandolo nel seguente modo:

```
$ nc -up<PORT_SENDBACK> 127.0.0.1 <PORT_LSOCK>
```

Sarà ora possibile far comunicare le due istanze di *netcat* tra di loro in modo bidirezionale, ovvero potremo inviare dati dall'una all'altra e viceversa e, supponendo di aver due interfacce di rete per ogni elaboratore, sarà possibile disattivare al più una interfaccia per *peer*, anche contemporaneamente, senza far cadere la connessione tra i due processi. Se si vuole utilizzare questo sistema anche con processi basati su flussi TCP bisognerà aggiungere un ulteriore componente, *phoxy*, disponibile anch'esso attraverso il repository fornito, il quale si preoccuperà di trasformare il flusso TCP in datagrammi UDP prima di inviare i dati all'MIH-proxy, viceversa per la ricezione, gestendo il riordinamento dei pacchetti, il loro rinvio nel caso il pacchetto vada perso e l'eliminazione di doppi pacchetti. Inoltre è possibile cifrare il traffico attraverso l'algoritmo a cifratura simmetrica *AES256*, ovvero con chiavi a 256 bit. Per compilare *phoxy* bastare fare:

```
$ pushd phoxy
$ make
$ popd
```

Per lanciarlo:

```
$ ./proxy -t <portTCP> -u <portUDP> -r <IPtoSEND> -p <portTOsendUDP>
[-e] [-c <ipTOconnect>] [-b <BUFFSIZE=200000>] [-m <MAXPKTS=1>] [-w
<waitTOresend=3500000>] [-s <SIZEPKTS=10000>]
```

Per attivare la cifratura dovremo invocarlo con il parametro opzionale -e ed il programma chiederà all'utente di scrivere la password direttamente su *stdin*.

4.4 Possibili utilizzi

Una volta ottenuta la possibilità di instradare su più interfacce il traffico, è naturale chiedersi quali possibili utilizzi possa avere questo progetto: è possibile, ad esempio, stabilire una particolare politica di *scheduling* dei *sockets* da usare per effettuare *load balancing* del traffico sui vari canali disponibili oppure creare un canale ad alta affidabilità che utilizzi una interfaccia tra quelle disponibili, magari decidendone una preferita. Tutto ciò può essere effettuato semplicemente modificando le funzioni *interfaces::find_best_interface()* e *interfaces::find_best_destination()* nel codice.

4.5 Sviluppi futuri

In futuro si potrebbe pensare di aggiungere, ad esempio, una gestore aggiuntivo delle sole interfacce wireless in modo da utilizzare preferibilmente la connessione con il valore *RSSI* migliore, tenendo conto dell'entità dei diversi valori per ogni specifica tecnologia oppure implementare uno *scheduler* a priorità per l'instradamento del traffico. Inoltre si potrebbe delegare a dei moduli la gestione delle interfacce, in modo da facilitare l'implementazione e la convivenza di più algoritmi di *scheduling*.

Capitolo 5

Conclusioni

5.1 Dopo il lavoro svolto

Durante la realizzazione di questo lavoro è stato possibile apprezzare lo standard IEEE 802.21 sia a livello teorico, sia a livello pratico. Lo standard in sé specifica funzionalità molto utili per poter prendere decisioni di *handover* e, anche solo testando un sottoinsieme delle funzionalità teoricamente offerte attraverso l'implementazione open-source ODTONE, si è rivelato efficace per gestire a livello utente dei meccanismi di *handover* per un singolo flusso di dati e creare così un proxy ad alta affidabilità, sfruttando tutti i collegamenti disponibili. Per quanto riguarda ODTONE, la principale limitazione riscontrata sta nel fatto che i Link-SAPs forniti da ODTONE sono solo per le tecnologie 802.3 e 802.21, quindi non è possibile utilizzare nessun protocollo della famiglia 3GPP per eseguire dei test.

5.2 Sviluppi futuri

Bisognerebbe nel prossimo futuro completare ed ampliare il progetto ODTONE, in particolar modo riguardo i SAPs disponibili attualmente e gli eventi generabili, in modo da poter testare più fedelmente gli scenari del mondo odierno. Se fosse possibile testare anche una sola tecnologia non-IEEE 802 si

potrebbe, ad esempio, fare qualche esperimento con dispositivi mobili reali, gestendone le decisioni di *handover*, come uno *smartphone* di nuova generazione con un sistema Android. Per quanto riguarda lo standard è necessario ora insistere sul suo effettivo *deployment* e completare il lavoro integrandolo con tutti i meccanismi necessari per renderlo efficace, a partire da soluzioni di mobilità per quanto riguarda l'indirizzo IP e specifiche per l'effettiva esecuzione dell'*handover* tra una tecnologia e l'altra, che hanno bisogno di meccanismi per effettuare passaggi tra stessa tecnologia (*horizontal handover*) e tra tecnologie diverse (*vertical handover*).

Bibliografia

[1] Mobile ipv4. https://tools.ietf.org/html/rfc5944. [Online; ultimo accesso 25 Febbraio 2014].

[2] IEEE 802.21 WG. *Media Independent Handover Services*, 2008.

[3] Ieee balloting procedure. http://standards.ieee.org/develop/balloting.html. [Online; ultimo accesso 25 Febbraio 2014].

[4] Ns2. http://www.isi.edu/nsnam/ns/. [Online; ultimo accesso 25 Febbraio 2014].

[5] ODTONE Team. Odtone, an open source multi-platform implementation of the ieee 802.21 protocol. http://atnog.github.io/ODTONE/, 2009-2014. [Online; ultimo accesso 25 Febbraio 2014].

[6] Inc. Free Software Foundation. Gnu lesser general public license. https://www.gnu.org/licenses/lgpl.html, Giugno 2007. [Online; ultimo accesso 25 Febbraio 2014].

[7] Boost Community. Boost library. http://www.boost.org. [Online; ultimo accesso 25 Febbraio 2014].

[8] Netlink protocol library suite. http://www.carisma.slowglass.com/~tgr/libnl/. [Online; ultimo accesso 25 Febbraio 2014].

[9] ODTONE Team. Odtone v0.6 changelog. http://atnog.github.io/ODTONE/documentation/odtone/changelog.html, 2013. [Online; ultimo accesso 25 Febbraio 2014].

[10] Debian - the universal operating system. `http://www.debian.org`. [Online; ultimo accesso 25 Febbraio 2014].

[11] Snapshot del repository ufficiale debian. `http://snapshot.debian.org/archive/debian/20140217T220529Z/`. [Online; ultimo accesso 25 Febbraio 2014].

[12] Netcat. `http://netcat.sourceforge.net`. [Online; ultimo accesso 25 Febbraio 2014].