

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 un'applicazione, *MIH-proxy*, che potesse sfruttare in modo costruttivo i servizi specificati dallo standard per creare un proxy che potesse scegliere su quale interfaccia 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 criptato ad alta affidabilità tra le applicazioni che possa sfruttare tutte le interfacce disponibili, sia per inviare, sia per ricevere.

Indice

1	Introduzione	1
1.1	La situazione odierna	1
1.2	Tipi di handovers	2
1.2.1	Horizontal handover	3
1.2.2	Vertical handover	3
1.2.3	Hard handover	3
1.2.4	Soft handover	3
1.3	Problematiche	4
2	Lo standard IEEE 802.21	5
2.1	Storia	5
2.2	Finalità	6
2.3	Architettura	7
2.3.1	MIHF	9
2.3.2	SAPs	10
2.3.3	MIH-User	11
2.4	Eventi e comandi	11
2.5	Esempio d'uso	14
2.6	Stato dell'arte	15
3	ODTONE	19
3.1	Descrizione	19
3.2	Funzionalità implementate	19
3.2.1	MIHF	20

3.2.2	MIH-Sap	21
3.2.3	MIH-Link-Sap 802.3	23
3.2.4	MIH-Link-Sap 802.11	24
3.2.5	La libreria libodtone	26
3.3	Compilazione ed esecuzione	27
3.3.1	Compilazione	27
3.3.2	Esecuzione	28
3.4	Risoluzione problemi	30
3.5	Sviluppi futuri	30
4	MIH-proxy	33
4.1	Descrizione	33
4.2	Funzionamento	34
4.2.1	Unidirezionale	34
4.2.2	Bidirezionale	35
4.3	Compilazione ed Esecuzione	37
4.3.1	Unidirezionale	38
4.3.2	Bidirezionale	40
4.4	Possibili utilizzi	42
4.5	Sviluppi futuri	42
5	Conclusioni	43
5.1	Dopo il lavoro svolto	43
5.2	Sviluppi futuri	43

Elenco delle figure

1.1	Handovers	2
2.1	IEEE 802.21 Timeline	6
2.2	Rapporto con gli altri standards	7
2.3	Architettura IEEE 802.21	8
3.1	Visione concettuale dell'MIHF di ODTONE	22
3.2	Visione concettuale del Link_SAP per 802.3	25
4.1	MIH-proxy unidirezionale	35
4.2	MIH-proxy bidirezionale senza phoxy	36
4.3	MIH-proxy bidirezionale con phoxy	37

Elenco delle tabelle

2.1	Lista degli eventi specificati nello standard	16
2.2	Lista dei comandi specificati nello standard	17

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. La figura 1.1 riassume graficamente quanto detto, mostrando i movimenti di un apparato mobile dotato di connettività Wi-Fi, WiMaX e 3G che, uscendo di volta in volta dal raggio di azione dell'*Access Point* a cui è attualmente connesso, adotta le azioni di *handover* necessarie per ripristinare la connettività, ristabilendo la connessione sull'interfaccia migliore tenendo conto delle prestazioni del *link* ed il consumo energetico associato.

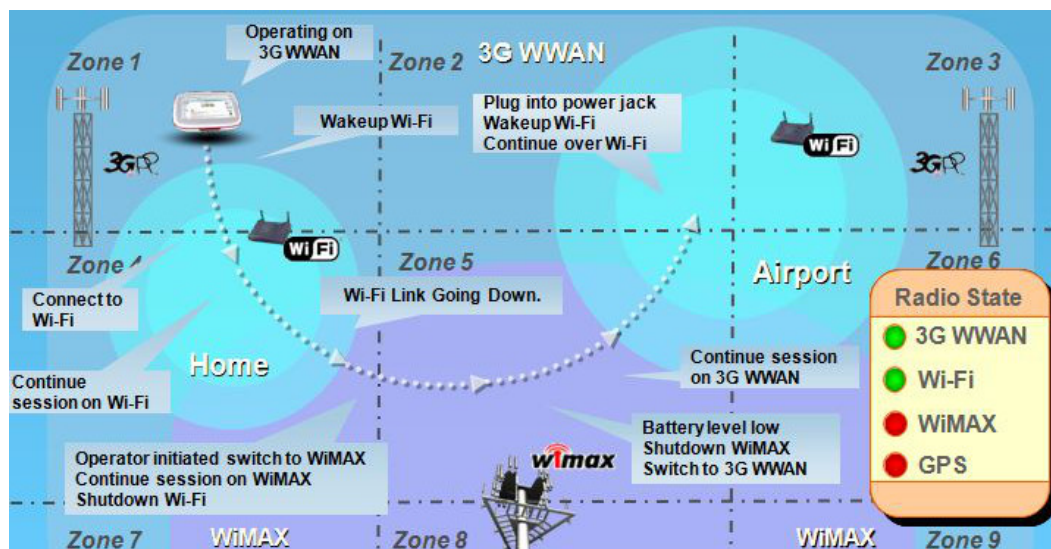


Figura 1.1: Handovers

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 questa 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 BTS¹, quando esce dal raggio di azione della prima, si parla di un *handover* orizzontale, i.e. la tecnologia adoperata a livello *data link* non cambia. Stessa tipologia di *handover* 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.

1.2.2 Vertical handover

Se è necessario, invece, cambiare il livello *data link* 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 solo quando si esce dalla zona di copertura dell'attuale *access point*, in modo da poter mantenere attiva la connessione, anche cambiando il mezzo trasmissivo e, di conseguenza, il livello *data link*.

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, quando avrà stabilito con successo un nuovo collegamento. 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

¹Base Transceiver Station

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 *access points* della stessa rete, al fine di mantenere lo stato della sessione.
3. come debba avvenire effettivamente il passaggio tra *access points* di reti diverse, al fine di mantenere lo stato della sessione.

Lo standard IEEE 802.21, preso in esame in questo lavoro, mira a risolvere il primo problema, fornendo dei servizi *media-independent* per prendere le dovute decisioni di *handover* in un modo standardizzato ed indipendente dalla tecnologia utilizzata al momento, ma non stabilisce come effettivamente debba avvenire il passaggio. Per il secondo problema è necessario stabilire opportune specifiche di come debba effettivamente avvenire l'*handover* tra *access points* della stessa rete, ad esempio come lo standard IEEE 802.11r-2008[1] per il passaggio da un *access point* Wi-Fi all'altro. Per il terzo problema è possibile adottare soluzioni come il *Mobile IP*[2] dell'IETF², il quale permette il *routing* dei pacchetti indipendentemente dalla posizione fisica del *device* associandogli un indirizzo IP permanente.

²Internet Engineering Task Force

Capitolo 2

Lo standard IEEE 802.21

In questo capitolo sarà esposto lo standard IEEE 802.21[3], illustrandone le componenti e le loro funzionalità, al fine di avere un quadro completo e poter comprendere il suo reale obiettivo, 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. Come possiamo notare dalla figura 2.1, 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 ECSG¹
- marzo 2004: creazione IEEE 802.21 WG²

¹Executive Committee Study Group

²Working Group

- 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[4]
- gennaio 2009: pubblicazione dello standard

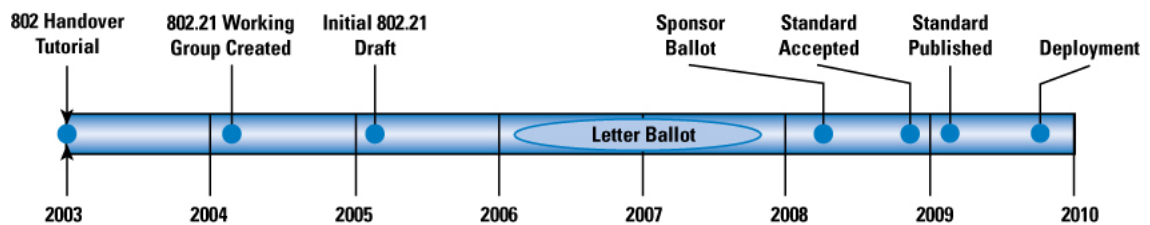


Figura 2.1: IEEE 802.21 Timeline

2.2 Finalità

Lo standard IEEE 802.21 si pone l'obiettivo di aiutare i nodi mobili a prepararsi ad eventuali azioni di *handover* da un *access point* all'altro, ma non specifica come debba avvenire la migrazione. Come sintetizzato dalla figura 2.2, esso si pone da intermediario tra gli standards coinvolti in azioni di *handover*: 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, il quale ha bisogno di apposite specifiche. L'applicazione utente che usufruisce dei servizi offerti da questo standard è in grado di conoscere lo stato delle connessioni disponibili attraverso la ricezione di eventi dal proprio MIHF³, quali *link_up* e *link_down*, e può, ad esempio, richiedere esplicitamente informazioni aggiuntive su una

³Media Independent Handover Function

interfaccia inviando delle specifiche richieste ad un particolare SAP⁴, come l'RSSI⁵ di una propria interfaccia 802.11 attualmente connessa. Lo standard IEEE 802.21 da solo non basta per gestire interamente i meccanismi di *handover*, poiché, ad esempio, per riuscire ad effettuare una migrazione *seamless* 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[2], altrimenti, dopo aver instaurato con successo la nuova connessione, il *peer* remoto non potrebbe sapere il nuovo indirizzo del nodo a cui era connesso precedentemente.

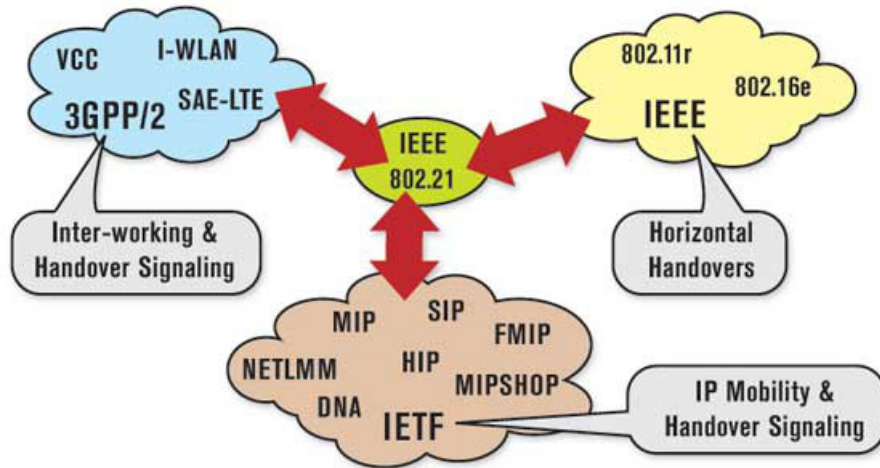


Figura 2.2: Rapporto con gli altri standards

2.3 Architettura

Lo standard IEEE 802.21 definisce più entità, rappresentate graficamente in figura 2.3, ognuna con il proprio specifico compito, al fine di fornire tutti i servizi necessari a prendere decisioni di *handover* in modo *media-*

⁴Service Access Point

⁵Received Signal Strength Indication

independent, ovvero in modo indipendente dalla tecnologia utilizzata in quel preciso momento:

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

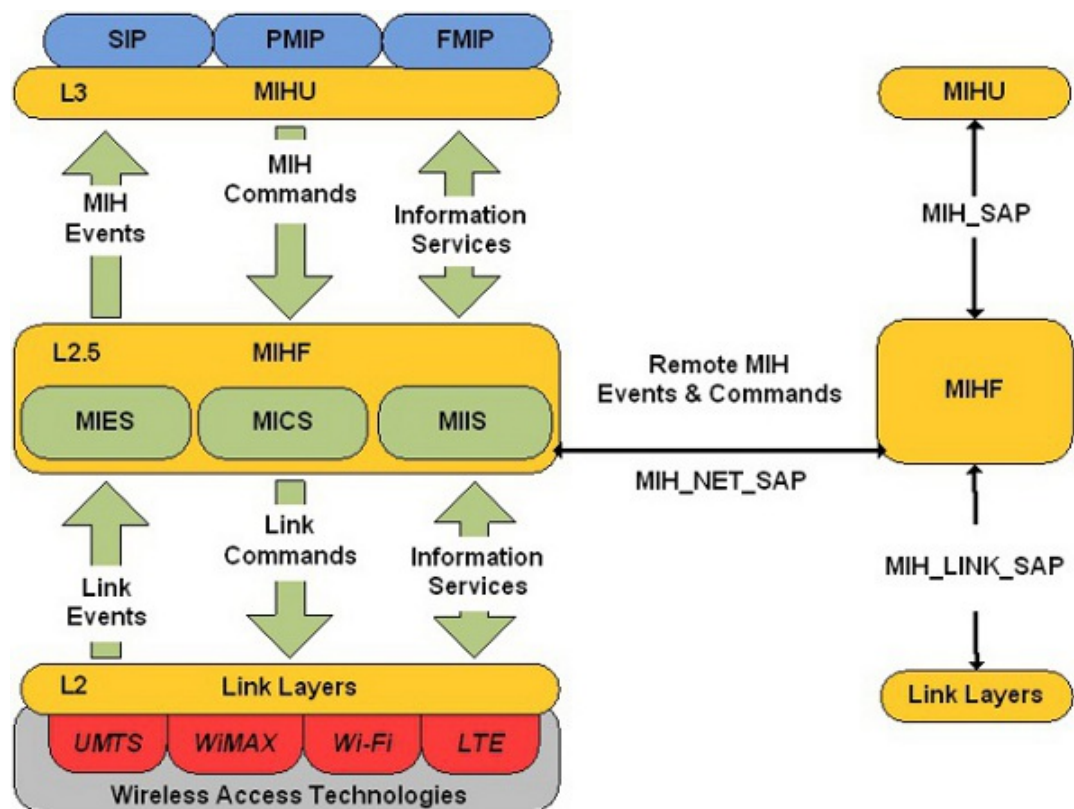


Figura 2.3: Architettura IEEE 802.21

2.3.1 MIHF

Questo componente è il più importante: i nodi mobili devono sottoscrivere 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. Ogni MIHF fornisce un insieme di servizi astratti ai livelli superiori indipendentemente dalle tecnologie adottate dalle singole interfacce gestite. 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 eventuali cambiamenti dello stato a livello *data link* agli strati superiori dello stack di rete del sistema locale o di sistemi remoti. Per un elenco completo di tutti gli eventi specificati nello standard IEEE 802.21 ed i loro significati si faccia riferimento alla tabella 2.1. 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*.
- 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 di 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. Per una lista completa di tutti i comandi previsti dallo standard IEEE 802.21 si faccia riferimento alla tabella 2.2. 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 la riconsegna delle 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 RDF⁶ basato su XML⁷, variants⁸ oppure TLV⁹. 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 e viceversa.

2.3.2 SAPs

Questo componente è incaricato di interagire direttamente con le interfacce di rete, al fine di esporre delle astrazioni 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. IEEE 802.3 od IEEE 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 il valore dell'intensità del segnale che assume valori differenti a seconda della tipologia dell'interfaccia interrogata.

⁶Resource Description Framework

⁷Extensible Markup Language

⁸tipi varianti, e.g. boost::variant [5]

⁹Type-length-value

2.3.3 MIH-User

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

2.4 Eventi e comandi

Lo standard IEEE 802.21 specifica una serie di eventi e di comandi per gestire tutta la fase preparatoria dell'*handover*. Gli eventi definiti, riassunti in tabella 2.1, servono a notificare alle entità interessate cambiamenti nello stato dei collegamenti disponibili e sono i seguenti:

- Link_Up: viene generato da un SAP quando l'interfaccia gestita da esso stabilisce una connessione di livello *data link*, segnalando la corretta connessione ad un *access point*.
- Link_Down: viene generato da un SAP quando la connessione instaurata attraverso l'interfaccia gestita viene interrotta, ovvero quando non è più attivo il collegamento con l'*access point*.
- Link_Detected: è generato quando è disponibile un nuovo *access point* di cui poter usufruire attraverso una propria interfaccia, ad esempio la scoperta di una nuova rete *wireless* durante una scansione.
- Link_Going_Down: può essere generato quando una particolare interfaccia sta per perdere contatto con il proprio *access point*, in modo da avvisare preventivamente l'MIH-User e consentire una gestione

dell'*handover* prima che la connessione cada effettivamente. I *triggers* che fanno scattare la generazione da parte di un SAP di questo evento sono diversi per ogni tecnologia gestita, a seconda delle caratteristiche intrinseche del mezzo trasmissivo, ad esempio valori minimi per l'RSSI, percentuale di pacchetti persi, etc.

- Link_Handover_Imminent: è utilizzato per comunicare a tutte le entità interessate che sta per essere eseguita una procedura di *handover* e, quindi, di prepararsi all'imminente passaggio.
- Link_Handover_Complete: è utilizzato per segnalare alle entità interessate che la procedura di *handover* è stata completata con successo.

I comandi previsti dallo standard IEEE 802.21, raccolti in tabella 2.2, servono a richiedere informazioni aggiuntive su specifiche interfacce, gestirne lo stato e la loro riconfigurazione oppure per iniziare le procedure di *handover*. Nel dettaglio, i comandi sono i seguenti:

- Link_Get_Parameters: permette di richiedere informazioni aggiuntive riguardo un collegamento specifico, come statistiche sui pacchetti persi, il *data rate* attuale, l'RSSI di interfacce wireless, etc.
- Link_Configure_Thresholds: permette di definire dei valori-soglia che facciano scattare la generazione di *report* sulle statistiche del collegamento per informare le entità interessate di eventuali cambiamenti nelle caratteristiche del *link*.
- Link_Capability_Discover: permettere di ottenere la lista di eventi e comandi supportati da un MIHF, per permettere all'MIH-User di definire dinamicamente gli opportuni *callbacks* a seconda delle funzionalità implementate da ogni singolo MIHF.
- Link_Event_Subscribe: permettere di sottoscrivere alla ricezione di eventi da parte dei SAPs richiesti per essere avvisati di eventuali cambiamenti di stato di una particolare interfaccia.

- Link_Event_Unsubscribe: permette di rescindere la sottoscrizione effettuata con il comando precedente.
- Link_Actions: permette di comandare esplicitamente l'interfaccia, ad esempio è possibile accendere o spegnere fisicamente una ricetrasmittente a seconda del suo utilizzo, in modo da ottimizzare i consumi energetici quando non è in uso.
- Net_Ho_Candidate_Query: in *handovers* iniziati dalla rete, segnala al nodo mobile di procedere al passaggio. Nella richiesta è anche allegata una lista di *networks* candidati, ovvero un elenco di altre reti disponibili a cui il nodo mobile dovrà collegarsi.
- Net_Ho_Commit: in *handovers* iniziati dalla rete, la ricezione di questo evento conferma l'esecuzione dell'*handover*, quindi il nodo mobile dovrà necessariamente procedere al passaggio ad un *network* comunicato precedentemente.
- N2n_Ho_Query_Resources: prima di confermare l'esecuzione del passaggio ad un altro network, l'attuale PoS¹⁰ contatta quello candidato per controllare che abbia le risorse per accettare un nuovo nodo mobile. Se il nuovo PoS si dichiara disponibile a ricevere un nuovo nodo mobile, il passaggio sarà confermato attraverso un successivo *commit*.
- N2n_Ho_Commit: il passaggio verso il nuovo PoS è confermato ed i nodi mobili devono provvedere all'esecuzione dell'*handover*.
- N2n_Ho_Complete: la procedura di *handover* verso un nuovo PoS 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.

¹⁰Point of Service

- Mn_Ho_Commit: in *handovers* iniziati dal nodo mobile, serve a comunicare che la procedura di *handover* è stata confermata.
- Mn_Ho_Complete: in *handovers* iniziati dal nodo mobile, serve a comunicare che la procedura di *handover* è stata completata.

2.5 Esempio d'uso

Si supponga di disporre di un dispositivo mobile dotato di più interfacce di rete *wireless* e che sia connesso tramite una di queste. Esso potrebbe richiedere al proprio MIHF, al quale deve essersi precedentemente sottoscritto, informazioni su tutti i collegamenti disponibili, indipendentemente dalla tipologia dell'interfaccia utilizzata in quel momento. In questo modo l'utente potrebbe venire a conoscenza preventivamente dell'esistenza di altri potenziali collegamenti tramite una differente tecnologia senza dover accendere fisicamente sul proprio dispositivo la ricetrasmittente appropriata per effettuare una scansione manuale, con un potenziale guadagno sui consumi energetici e maggior parsimonia per quanto riguarda lo sfruttamento dell'etere. Successivamente l'applicazione utente potrebbe essere informata di un deterioramento nelle prestazioni dell'interfaccia con la quale è attualmente connessa attraverso la ricezione dell'evento *Link_Going_Down* e quindi prepararsi ad azioni di *handover* richiedendo informazioni su altre reti disponibili con il comando *Mn_Ho_Candidate_Query* al fine di designare il prossimo PoA¹¹ a cui collegarsi, confermando l'eventuale operazione di *handover* con un *commit* impartendo il comando *Mn_Ho_Commit* per informare il proprio MIHF dell'imminente passaggio. Una volta completato, l'MIH-User dovrà confermare il successo dell'operazione con il comando *Mn_Ho_Complete*. La situazione appena descritta è riferita ad un *handover* iniziato dal nodo mobile, ma può anche accadere che il passaggio sia richiesto dal proprio MIHF, in questo caso saranno utilizzati i comandi *Net_Ho_Commit* e *Net_Ho_Commit* per segnalare che l'operazione è richiesta dalla rete stessa: in questo caso l'MIHF

¹¹Point of Access

invierà di propria iniziativa una lista di altri *networks* all'applicazione utente, richiedendo il trasferimento verso uno di essi. In questo modo l'MIH-User può recuperare tutte le informazioni utili per gestire le decisioni di *handover* in modo indipendente dalla tecnologia utilizzata per comunicare con il proprio MIHF e può, a questo punto, sfruttare questi dati per svariati compiti, come per implementare un proxy ad alta affidabilità oppure una sorta di *network manager* che di volta in volta si preoccupi di sottoscrivere 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*.

2.6 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*[6] e *ns-3*[7]. Queste simulazioni sono in grado di dimostrare come le funzionalità introdotte dallo standard possano essere utili per ridurre i tempi necessari per effettuare un *handover*[8], in particolare per quanto riguarda la scelta del nuovo *network*, grazie all'MIIS che permette di conoscere le reti disponibili senza dover far uno *scan* esplicito. Per quanto riguarda le implementazioni realmente eseguibili, la più utilizzata e studiata è ODTONE[9], la quale sarà analizzata nel dettaglio nel prossimo capitolo.

Nome	Descrizione
Link_Up	è generato quando una specifica interfaccia diventa attiva, i.e. viene stabilita una connessione di secondo livello
Link_Down	è generato quando cade una connessione di secondo livello di una specifica interfaccia, e.g. disconnessione da una rete <i>wireless</i>
Link_Detected	è generato quando è disponibile una nuova rete di cui usufruire, ad esempio la scoperta di una nuova rete <i>wireless</i> durante una scansione
Link_Going_Down	può essere generato da un SAP per informare le parti interessate che le prestazioni di un dato <i>link</i> stanno man mano degradando, secondo vari criteri, e.g. pacchetti persi, larghezza di banda, latenza, etc.
Link_Handover_Imminent	è utilizzato per comunicare alle entità interessate che sta per essere eseguita una procedura di <i>handover</i>
Link_Handover_Complete	è utilizzato per segnalare alle entità interessate che la procedura di <i>handover</i> è stata completata con successo

Tabella 2.1: Lista degli eventi specificati nello standard

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	permette di rescindere la sottoscrizione effettuata con il comando precedente
Link_Actions	permette di comandare esplicitamente l'interfaccia (e.g. <i>power_up</i> , <i>power_down</i>)
Net_Ho_Candidate_Query	in <i>handovers</i> iniziati dalla rete, segnala al nodo mobile di procedere al passaggio ad uno dei network candidati
Net_Ho_Commit	conferma all'esecuzione dell' <i>handover</i>
N2n_Ho_Query_Resources	prima di confermare l'esecuzione dell' <i>handover</i> su un altro network, l'attuale PoS 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 <i>handover</i> è stata completata
Mn_Ho_Candidate_Query	in <i>handovers</i> iniziati dal nodo mobile, serve per richiedere la lista dei network candidati per un eventuale passaggio
Mn_Ho_Commit	la procedura di <i>handover</i> è stata confermata
Mn_Ho_Complete	la procedura di <i>handover</i> 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[9], descrivendone le funzionalità attualmente implementate ed illustrando punto per punto tutti i passaggi richiesti per compilare ed eseguire per la prima volta con successo questa implementazione, partendo direttamente dal codice disponibile presso il repository ufficiale.

3.1 Descrizione

Essa è una implementazione open-source dello standard IEEE 802.21 rilasciata con licenza LGPLv3[10] ed è realizzata in C++ e resa multiplatforma tramite la libreria *Boost*[11]: è possibile infatti eseguire ODTONE su sistemi GNU/Linux, Windows, Android ed OpenWrt.[12]

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 sperimentale¹, fornisce solo le funzionalità principali oppure facilmente implementabili in chiave multi-platforma.

¹è 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 della scrittura, sono solo per sistemi GNU/Linux e Windows, limitatamente per interfacce 802.3 e 802.11, e supportano solo gli eventi *link_up* e *link_down*.

3.2.1 MIHF

La *Media Independent Handover Function* fornita è composta dai tre sotto-componenti principali 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 ne hanno richiesto la sottoscrizione. Più nel dettaglio è composto da:
 - 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 di recapitarli a destinazione. Più nel dettaglio è suddiviso in:
 - 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'IS² e inviare indietro le risposte.

Questi componenti forniscono i servizi basilari definiti nello standard. Oltre ad essi, come rappresentato graficamente in figura 3.1, ODTONE implementa altri componenti per fornire ulteriori funzionalità e arricchire quelle già esistenti:

²Information 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 od altre entità e li invia al modulo Service Access Controller.
- Service Access Controller: è responsabile dell'analisi degli *headers* di ogni messaggio ricevuto ed al loro smistamento ai servizi interessati. 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*[13] 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

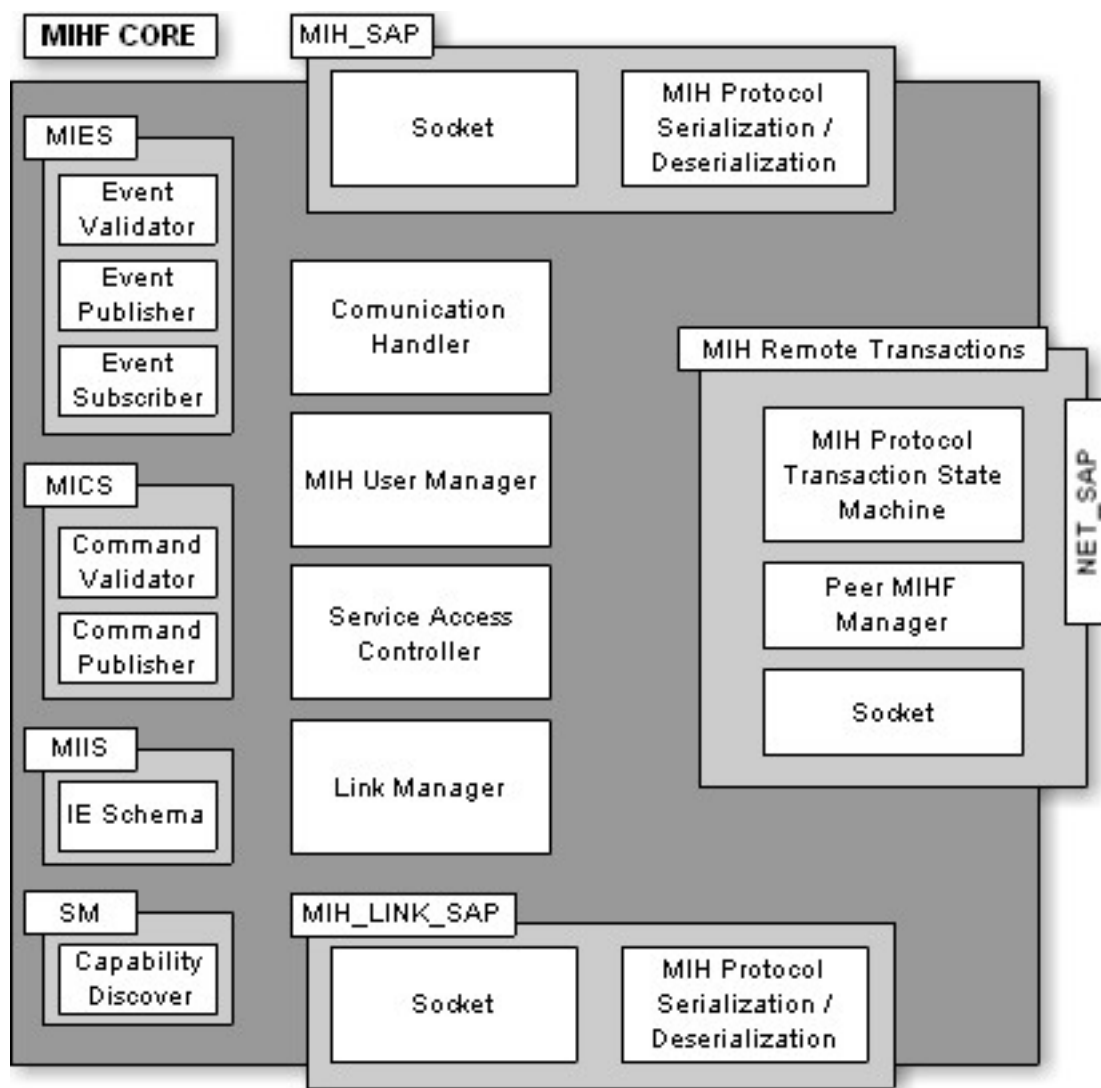


Figura 3.1: Visione concettuale dell'MIHF di ODTONE

- IEEE 802.22
- 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 (schematizzato in figura 3.2). 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 sottoscrivere 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.

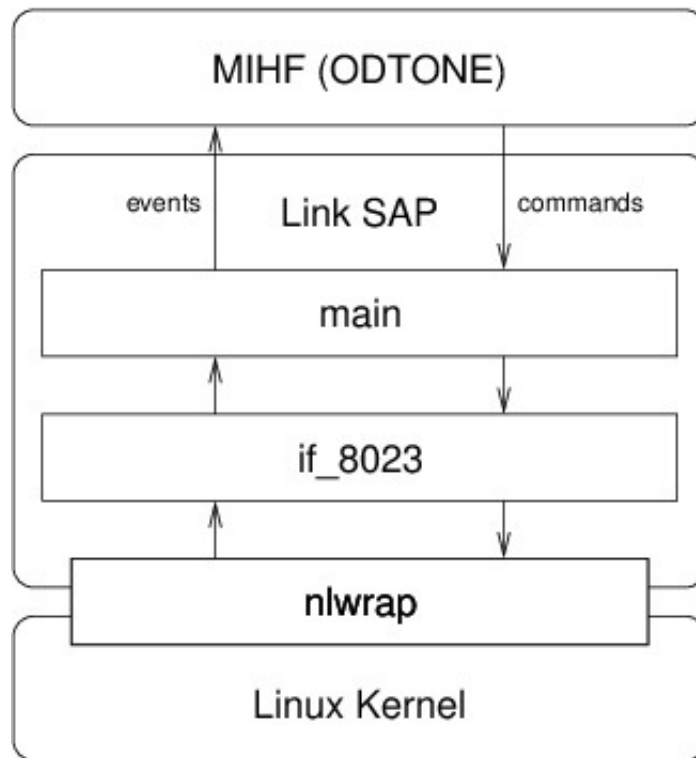


Figura 3.2: Visione concettuale del Link_SAP per 802.3

- Link_Down: generato quando il modulo *if_80211* riceve l'evento *NL80211_CMD_DISCONNECT* dal kernel.
- 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 sottoscrivere agli eventi richiesti.
- `Event_Unsubscribe`: permette di annullare l'iscrizione agli eventi specifici.
- `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_80211* certe operazioni sull'interfaccia. Al momento supporta i comandi:
 - *LINK_DISCONNECT*: richiede la disconnessione.
 - *LINK_POWER_UP*: accende l'interfaccia.
 - *LINK_POWER_DOWN*: 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 reti Wi-Fi disponibili.

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'`[14]. 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[15] *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[16]):

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_:
```

```
1 --- xtime.hpp    2014-02-18 03:26:56.291068347 +0100  
2 +++ xtime1.hpp  2014-02-18 03:27:23.679204146 +0100  
3 @@ -20,7 +20,7 @@  
4  
5  enum xtime_clock_types  
6  {  
7  -    TIME__UTC=1  
8  +    TIME__UTC_=1  
9  //    TIME__TAI,  
10 //    TIME__MONOTONIC,  
11 //    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 da utilizzare, occorre quindi una seconda istanza del proxy anche sul *peer* ricevente che si preoccupi di monitorare il traffico in arrivo su tutte le interfacce: è così possibile far comunicare due applicazioni che utilizzano bidirezionalmente il protocollo UDP oppure

TCP. In questo ultimo caso è necessario trasformare dapprima il flusso TCP in datagrammi UDP, instradarli sull'interfaccia corretta e ricostruire il flusso originario una volta a destinazione: questo compito è affidato a *phoxy*, un'applicazione separata da MIH-proxy. Si è preferito tenere separati gli applicativi per trasformare un flusso TCP in pacchetti UDP e per gestire l'invio e la ricezione sulle varie interfacce in modo da delegare compiti appropriati e coerenti ad ogni componente, i quali potrebbero avere anche un utilizzo concreto anche se utilizzati 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*[9]. Come discusso prima, ODTONE non fornisce tutte le funzionalità specificate dallo standard, ad esempio sono attualmente generabili solo alcuni eventi tra tutti quelli definiti nello standard.

4.2.1 Unidirezionale

La versione unidirezionale, rappresentata graficamente in figura 4.1, gestisce traffico in sola uscita, smistandolo sulle interfacce disponibili. Con una sola istanza dell'MIH-proxy la comunicazione può avvenire in un solo senso per limiti tecnici, poiché, supponendo traffico bidirezionale, nel caso cadesse un'interfaccia del trasmettitore, l'applicazione ricevente non potrebbe sapere che l'IP con cui stava precedentemente comunicando non è più disponibile e di conseguenza invierebbe pacchetti ad un indirizzo non più raggiungibile. Per conoscere lo stato delle interfacce locali e scegliere quali collegamenti utilizzare, 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.

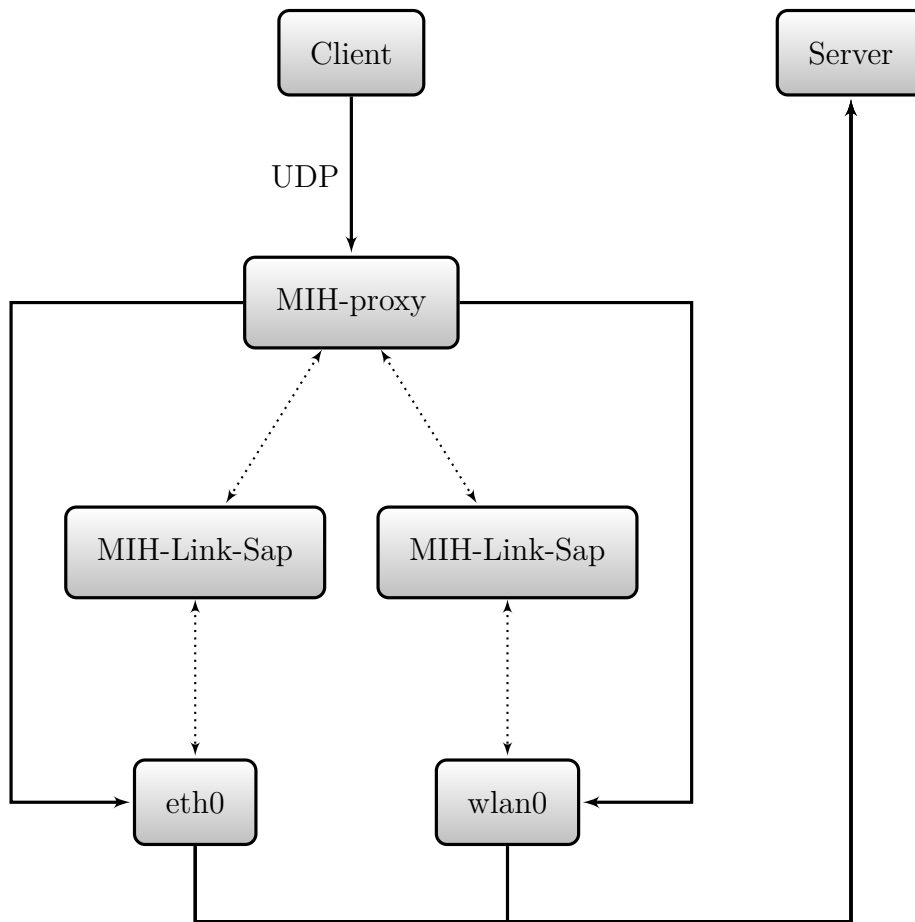


Figura 4.1: MIH-proxy unidirezionale

4.2.2 Bidirezionale

La versione bidirezionale (figura 4.2) 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.

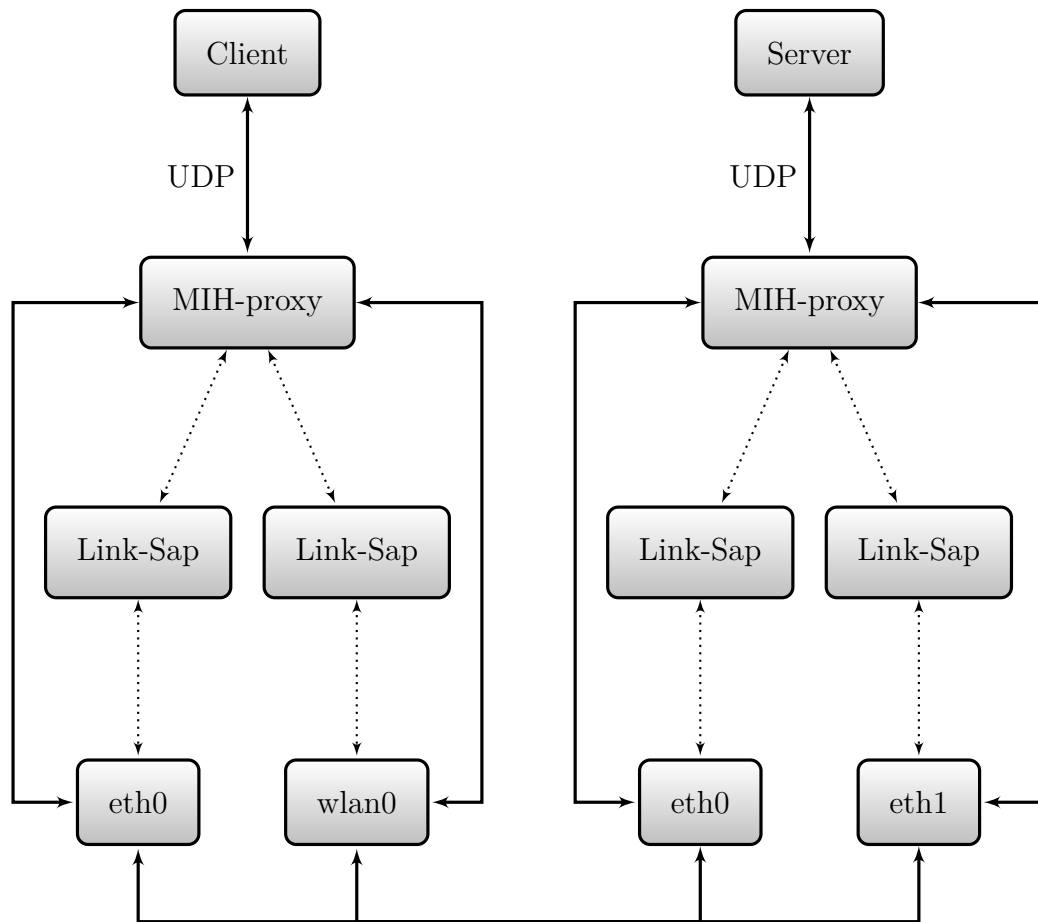


Figura 4.2: MIH-proxy bidirezionale senza phoxy

Per 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 occuperà di trasformare il flusso TCP in datagrammi UDP (figura 4.3), gestendo pacchetti persi, doppi o fuori ordine e ricostruendo il flusso originario prima di consegnarlo al destinatario. Inoltre è possibile richiedere una cifratura *AES256* nella conversione da flusso a datagrammi.

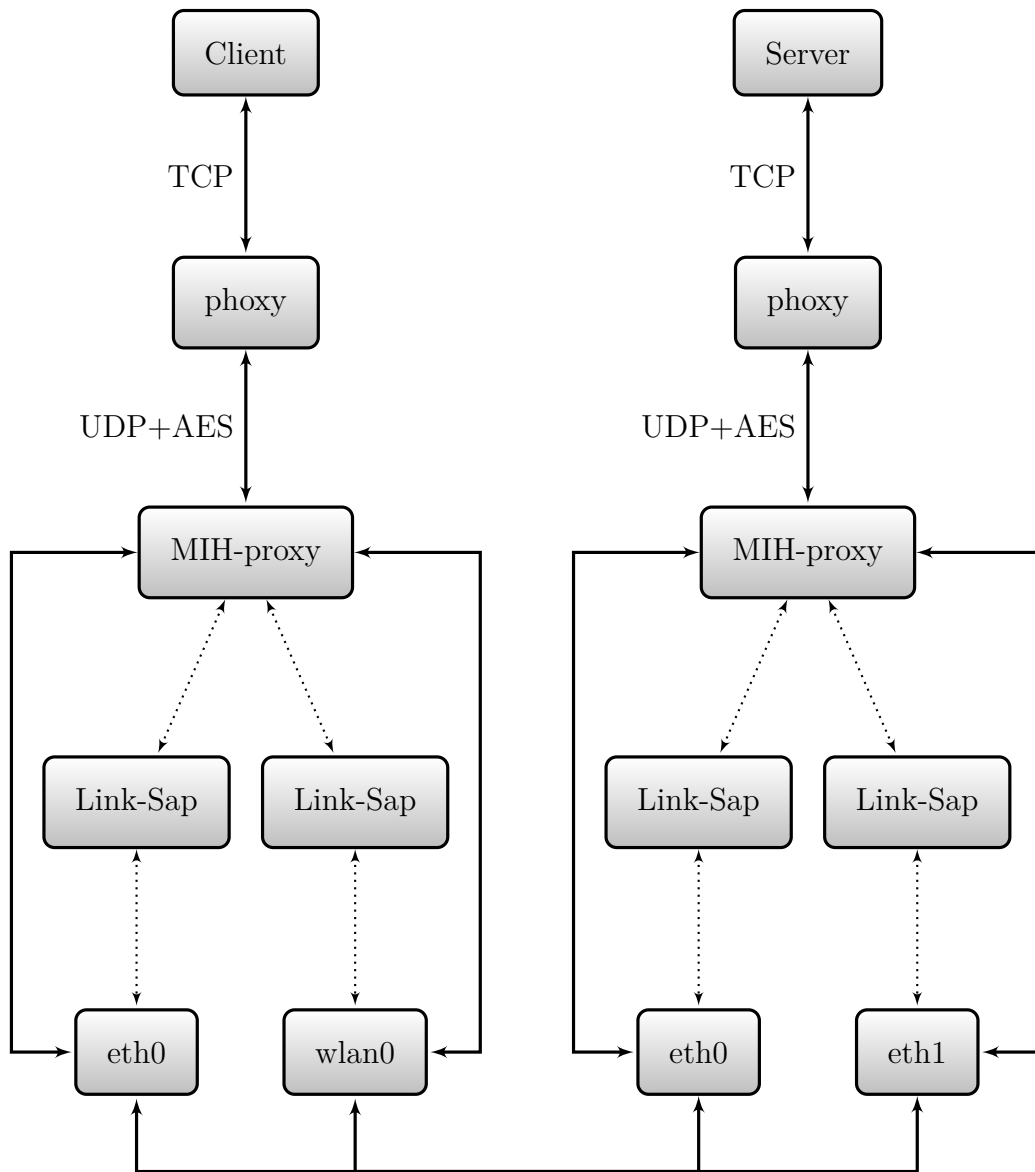


Figura 4.3: MIH-proxy bidirezionale con phoxy

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*:

```
1 class interfaces : boost::noncopyable {
2     private:
3         ...
4         boost::asio::ip::udp::endpoint dest =
5             boost::asio::ip::udp::endpoint(
6                 boost::asio::ip::address::from_string(
7                     "127.0.0.1"), 9999);
8         ...
9 }
```

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*[17] 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 appropriati¹. 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(
```

¹se 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* basterà fare:

```
$ pushd phoxy
```

```
$ make
```

```
$ popd
```

Per lanciarlo:

```
$ ./proxy -t <portTCP> -u <portUDP> -r <IPtoSEND> -p <portTosendUDP>  
[-e] [-c <ipTtoconnect>] [-b <BUFSIZE=200000>] [-m <MAXPKTS=1>] [-w  
<waitTtoresend=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 implementare, 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 di nuovi algoritmi di *scheduling* e la convivenza di quelli esistenti.

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, in modo tale da poter testare le applicazioni MIH-User con tutte le tecnologie utilizzate nella realtà: 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 operativo Android. Oltre ad aumentare le tecnologie supportate, bisogna completare i SAPs esistenti implementando la generazione degli eventi mancanti, come l'evento *link_going_down*, in modo da poter sfruttare al completo le funzionalità specificate dallo standard. Per quanto riguarda l'IEEE 802.21 è necessario ora insistere sul suo effettivo *deployment* e completare il lavoro integrandolo con tutti i meccanismi necessari per renderlo efficace, a partire dall'adozione di soluzioni di mobilità per quanto riguarda l'indirizzo IP e la standardizzazione di specifiche per l'effettiva esecuzione dell'*handover* tra una tecnologia e l'altra, che hanno bisogno di meccanismi per effettuare passaggi sia tra stessa tecnologia (*horizontal handover*) sia tra tecnologie diverse (*vertical handover*).

Bibliografia

- [1] IEEE Computer Society. Ieee standard for information technology – local and metropolitan area networks – specific requirements– part 11: Wireless lan medium access control (mac) and physical layer (phy) specifications amendment 2: Fast basic service set (bss) transition. *IEEE Std 802.11r-2008 (Amendment to IEEE Std 802.11-2007 as amended by IEEE Std 802.11k-2008)*, Luglio 2008.
- [2] Mobile ipv4. <https://tools.ietf.org/html/rfc5944>. [Online; ultimo accesso 25 Febbraio 2014].
- [3] IEEE Computer Society. Ieee standard for information technology – local and metropolitan area networks – part 21: Media independent handover services. *IEEE Std 802.21*, Gennaio 2008.
- [4] Ieee balloting procedure. <http://standards.ieee.org/develop/balloting.html>. [Online; ultimo accesso 25 Febbraio 2014].
- [5] Boost::variant. http://www.boost.org/doc/libs/1_55_0/doc/html/variant.html. [Online; ultimo accesso 25 Febbraio 2014].
- [6] Ns2. <http://www.isi.edu/nsnam/ns/>. [Online; ultimo accesso 25 Febbraio 2014].
- [7] ns-3. <https://www.nsnam.org/>. [Online; ultimo accesso 25 Febbraio 2014].

-
- [8] Adriano Tavares da Silva Pinho. Implementation of the iee 802.21 in the network simulator 3. Master's thesis, Universidade do Porto, 2008.
 - [9] ODTONE Team. Odtone, an open source multi-platform implementation of the iee 802.21 protocol. <http://atnog.github.io/ODTONE/>, 2009-2014. [Online; ultimo accesso 25 Febbraio 2014].
 - [10] Inc. Free Software Foundation. Gnu lesser general public license. <https://www.gnu.org/licenses/lgpl.html>, Giugno 2007. [Online; ultimo accesso 25 Febbraio 2014].
 - [11] Boost Community. Boost library. <http://www.boost.org>. [Online; ultimo accesso 25 Febbraio 2014].
 - [12] ODTONE Team. Odtone: Getting started. <http://atnog.github.io/ODTONE/documentation/odtone/getStart.html>, 2013. [Online; ultimo accesso 25 Febbraio 2014].
 - [13] Netlink protocol library suite. <http://www.carisma.slowglass.com/~tgr/libnl/>. [Online; ultimo accesso 25 Febbraio 2014].
 - [14] ODTONE Team. Odtone v0.6 changelog. <http://atnog.github.io/ODTONE/documentation/odtone/changelog.html>, 2013. [Online; ultimo accesso 25 Febbraio 2014].
 - [15] Debian - the universal operating system. <http://www.debian.org>. [Online; ultimo accesso 25 Febbraio 2014].
 - [16] Snapshot del repository ufficiale debian. <http://snapshot.debian.org/archive/debian/20140217T220529Z/>. [Online; ultimo accesso 25 Febbraio 2014].
 - [17] Netcat. <http://netcat.sourceforge.net>. [Online; ultimo accesso 25 Febbraio 2014].