SAPIENZA
UNIVERSITÀ DI ROMA

Un sistema di Deep Packet Inspection per OpenCAPWAP

Facoltà di Ingegneria dell'Informazione, Informatica, Statistica
Corso di laurea in Tecnologie Informatiche
Cattedra di

Candidato
Simone Rosignoli
n° matricola
1010152

Responsabile
Prof. Massimo Bernaschi

Corresponsabile
Dott. Emanuele Gabrielli

A/A 2010/2011

Indice

1 Introduzione **5**
- 1.1 Architetture delle reti WLAN . . . 5
- 1.2 Amministrazione di WLAN centralizzate: CAPWAP . . . 6
- 1.3 Qualità del Servizio . . . 6
- 1.4 Identificazione dei pacchetti . . . 7
- 1.5 Un caso reale: Provinciawifi e OpenCAPWAP . . . 7
- 1.6 Lavoro di tirocinio . . . 8
- 1.7 Struttura della relazione . . . 8

2 CAPWAP e OpenCAPWAP dalla teoria alla pratica **10**
- 2.1 Terminologia . . . 10
- 2.2 Modalità operative: Split e Local MAC . . . 11
- 2.3 Panoramica del protocollo . . . 12
- 2.4 Macchina a stati di CAPWAP . . . 15
- 2.5 Trasporto UDP . . . 17
- 2.6 OpenCAPWAP . . . 17
  - 2.6.1 Architettura AC-WTP . . . 17
  - 2.6.2 Demone AC . . . 18
  - 2.6.3 Demone WTP . . . 19
  - 2.6.4 Moduli di estensione . . . 20

3 Strumenti Utilizzati **21**
- 3.1 Hardware dei WTP . . . 21
- 3.2 OpenWrt . . . 22
- 3.3 Netfilter . . . 23
- 3.4 Connection Tracking System . . . 25
- 3.5 L7 Filter . . . 25
  - 3.5.1 Dettagli Tecnici . . . 26
  - 3.5.2 Definizione dei protocolli (File dei Pattern) . . . 27
  - 3.5.3 Come si utilizza . . . 28
- 3.6 Target ULOG . . . 28
- 3.7 Netlink Socket . . . 29
- 3.8 Setup di OpenWrt . . . 30

4 Layer 7 packet classifier per OpenCAPWAP **32**
4.1 Cos'è, cosa fa e come funziona? 32
4.2 WTPL7Agent 35
4.2.1 Funzionalità 35
4.2.2 Architettura 36
4.2.3 File di log e di configurazione 41
4.2.3.1 /etc/WTPL7Agent 41
4.2.3.2 l7pattern.conf 42
4.2.3.3 /var/log/WTPL7Agent.log 42
4.2.3.4 /var/log/WTPL7Log/wtpL7Flow.[...].log 43
4.3 ACL7Manager 44
4.3.1 Utilizzo 44
4.3.2 Opzioni 44
4.4 Integrazione con OpenCAPWAP 45
4.4.1 Connessione 46
4.4.2 Messaggi 46
4.4.3 Canale Universale 46
4.4.3.1 Generic Vendor Specific Payload 47
4.4.3.2 Protocollo di comunicazione AE <–> AC 48
4.4.3.3 Protocollo di comunicazione AE <–> WTP . . . 48
4.5 Formato dei messaggi utilizzati dal modulo L7 49

5 Test e Conclusioni **50**
5.1 Test 50
5.1.1 Test in laboratorio 50
5.1.1.1 Prima sessione di test 51
5.1.1.2 Seconda sessione di test 51
5.1.2 Test in produzione 52
5.2 Conclusioni e sviluppi futuri 54

Capitolo 1

Introduzione

Il proliferare della tecnologia Wi-Fi non conosce sosta ed è causato, e favorisce allo stesso tempo, la crescita del traffico Internet. Il successo della tecnologia wireless è giustificato dal fatto che soddisfa sia la crescente necessità di essere connessi che quella di poter mantenere la mobilità. E' ormai disponibile non solo per i computer tradizionali ma anche per smart-phone, tablet, console da gioco, televisori, player multimediali e automobili. Il crescente numero di dispositivi e la maturazione dello standard IEEE 802.11 Wireless LAN (WLAN) hanno portato all'implementazione di reti WLAN su larga scala. Amministrare e controllare un numero elevato di Access Points (AP) è un compito che si è rivelato non privo di problemi per gli amministratori di rete: come distribuire e mantenere consistente la configurazione su tutti i dispositivi? Come assegnare i canali radio di trasmissione/ricezione in modo da minimizzare le interferenze tra AP? Come affrontare i rischi legati alla sicurezza amplificati dalla natura del media wireless? Come regolare il traffico per massimizzare l'esperienza dell'utente (QoS) ? Per risolvere questi problemi sono state proposte diverse soluzioni proprietarie che, naturalmente, non sono utilizzabili in ambienti con dispositivi eterogenei. La necessità di una soluzione flessibile e con un elevato livello di interoperabilità ha portato alla definizione del protocollo *Control and Provisioning of Wireless Access Points* (CAPWAP) da parte dell'*Internet Engeneering Task Force* (IETF).

1.1 Architetture delle reti WLAN

Le reti WLAN 802.11 possono essere classificate in tre distinte famiglie:

- **Architetture WLAN autonome**: costituite da un singolo dispositivo configurato e controllato individualmente che implementa tutti i servizi dello standard IEEE 802.11. Il dipositivo è comunemente chiamato "Standalone Access Point" e può essere controllato utilizzando il protocollo SNMP.

- **Architetture WLAN centralizzate**: costituite da uno o più *Access Controller* (AC) la cui funzione è quella di amministrare, controllare e configurare i *Wireless Termination Point* (WTP) presenti nella rete. L'insieme completo delle funzioni richieste dallo standard IEEE 802.11 può essere realizzato in modo distribuito tra i WTP e gli AC.
- **Architetture WLAN distribuite**: costituite da un numero elevato di nodi che fungono da ricevitori, trasmettitori e ripetitori capaci di interconnettersi tra loro in modo autonomo per creare una rete wireless cooperativa non gerarchica. Un esempio sono le reti Mesh wireless.

1.2 Amministrazione di WLAN centralizzate: CAPWAP

Il protocollo CAPWAP nasce con l'obiettivo principale di risolvere i problemi della gestione delle reti wireless centralizzate, in particolare riguardo:

- **alla configurazione e al controllo delle risorse radio**: controllo della RF (Radio Frequenza) (es. identificazione dei radar, identificazione di rumore e interferenze, misurazioni), configurazione RF (es. ritrasmissione, selezione del canale, impostazione della potenza di trasmissione).
- **alla configurazione e all'amministrazione dei dispositivi WTP**: configurazione WTP (es. impostazione del SSID), gestione firmware WTP (es. caricamento automatico e aggiornamento firmware per garantire la consistenza delle configurazioni su tutta la rete).
- **alla gestione dello stato dei client** (stazioni, STA): database dello stato delle stazioni con informazioni necessarie per fornire servizi avanzati come la mobilità e il bilanciamento del carico.
- **alla sicurezza**: mutua autenticazione tra le entità della rete (es. tra AC e WTP).

Tra le funzioni definite da CAPWAP sulla base delle specifiche dello standard IEEE 802.11 vanno ricordate anche quelle per la gestione della QoS[1]. Tramite queste l'Access Controller può ridefinire le politiche di *Quality of Service* di uno o più WTP della WLAN. L'insieme delle funzioni definite non sono limitate alle sole presenti nel 802.11. Viene infatti prevista la possibilità di includere delle estensioni per nuovi servizi implementati da un determinato fornitore.

1.3 Qualità del Servizio

All'interno del traffico dati su Internet si registra un progressivo aumento di quello più *latency-sensitive*, come video, voip e giochi online. La sfida dei gestori

[1] http://tools.ietf.org/html/draft-ietf-capwap-protocol-binding-ieee80211-12#section-6.20

delle reti è quella di cercare di garantire i migliori livelli di qualità e servizio per gli utenti finali, che da parte loro, richiedono servizi sempre più affidabili e di alta qualità per supportare al meglio le loro attività di studio, lavoro e svago. Una possibile soluzione è quella di investire costantemente denaro per potenziare l'infrastruttura di rete e aumentare la larghezza di banda a disposizione. Ma la banda è una risorsa costosa e si rischia di lasciarla inutilizzata al di fuori dei periodi di picco del traffico; una soluzione più intelligente, non solo dal punto di vista economico, consiste nel fornire dinamicamente la banda necessaria a tutte quelle applicazioni sensibili alle interruzioni del flusso di rete, anche per piccole frazioni di secondo. A questo scopo è necessaria un'attenta amministrazione delle risorse disponibili con l'ausilio di policy e quota per singoli o gruppi di utenti. Regolare e amministrare la banda disponibile per caratterizzare il traffico di rete prende il nome di *Quality of Service* o più semplicemente **QoS**. Per realizzare la QoS è necessario:

- identificare i pacchetti che devono ricevere un trattamento privilegiato.
- assegnare questi pacchetti a specifiche classi con priorità e banda garantita appropriate.

1.4 Identificazione dei pacchetti

Un sistema efficace per l'identificazione dei pacchetti è la tecnica detta *Deep Packet Inspection* (**DPI**) che consiste nel guardare all'interno del contenuto dei pacchetti. In questo modo è possibile classificare traffico difficilmente identificabile con semplici regole basate solo sui parametri disponibili negli header di rete fino al livello di trasporto del modello ISO/OSI. Tanta potenza ha un costo computazionale che in larga parte dipende dalla quantità e dalla tipologia del traffico.

1.5 Un caso reale: Provinciawifi e OpenCAPWAP

Più di 600 Access Point installati sul territorio della Provincia di Roma[2], quasi 80.000 utenti registrati ed un tasso di crescita di circa 300 nuove utenze al giorno rendono la rete di *Provinciawifi* la più grande rete Wi-Fi gratuita mai costruita in Italia. La progettazione e la gestione tecnica della rete è stata affidata al *Consorzio Interuniversitario per il calcolo scientifico* (**CASPUR**). Il CASPUR si è occupato di progettare e realizzare un firmware personalizzato per gli apparati di accesso, l'infrastruttura centrale e l'infrastruttura di monitoraggio. Tramite questa architettura vengono forniti agli utenti i servizi di *autoprovisioning* delle credenziali di accesso e di connessione alla rete Internet. Grazie ad un sistema di *Authentication, Authorization and Accounting* (**AAA**)

[2] 121 Comuni, più di 4 milioni abitanti con una superficie di 5000 Km^2

è stato possibile sia definire delle politiche di utilizzo per limitare la navigazione degli utenti a 300 Mbyte trasferiti (oppure ad un'ora di navigazione) al giorno sia ottemperare alle norme che regolano la fornitura di un servizio di connessione Internet al pubblico. Tutti gli strumenti utilizzati per la realizzazione del progetto sono rigorosamente open source. Per la gestione centralizzata della WLAN di *Provinciawifi* è in sperimentazione l'utilizzo del protocollo CAPWAP nella sua implementazione open source denominata **OpenCAPWAP** realizzata dall'*Istituto per le Applicazioni del Calcolo del CNR* e l'*Università Campus Biomedico di Roma*.

1.6 Lavoro di tirocinio

Oggetto del lavoro di tirocinio è stata la progettazione di un sistema distribuito di identificazione di pacchetti tramite Deep Packet Inspection per reti WLAN centralizzate, amministrate utilizzando il protocollo CAPWAP. E' stato realizzato un prototipo funzionante dell'architettura Agent-Manager come estensione del progetto **OpenCAPWAP** con l'obiettivo di verificare la possibilità di utilizzare la DPI sui WTP della rete wireless anziché sul gateway Internet centrale. L'idea è di sfruttare la natura di orchestratore dell'Access Controller per la definizione e la distribuzione delle politiche di QoS sulla base dei dati raccolti dagli Agent distribuiti. Per il sistema da realizzare sono stati definiti alcuni obiettivi di alto livello:

- **Flessibilità operativa**: è necessario permettere tramite parametri di configurazione la modifica delle risorse utilizzate dell'Agent in previsione dell'evoluzione e della varietà degli apparati di accesso: da AP di fascia enterprise a AP estremamente compatti, leggeri ed economici.
- **Performance**: è necessario che le risorse degli AP siano sufficienti per garantire un livello di servizio soddisfacente anche con la DPI attiva.
- **Controllo**: data la natura distribuita dell'architettura è necessario individuare le funzionalità significative e permetterne la completa gestione da un'utility centralizzata di management.
- **Robustezza e affidabilità**: è necessario ridurre al minimo i malfunzionamenti e in caso di situazioni impreviste il sistema deve comportarsi in modo da garantire l'operatività. L'utilizzo del sistema di estensione per applicazioni esterne di OpenCAPWAP permette una separazione in moduli che favorisce questo comportamento.
- **Scalabilità e sicurezza**: queste caratteristiche si ereditano dalla realizzazione del progetto come moduli per OpenCAPWAP .

1.7 Struttura della relazione

Il resto di questo lavoro è strutturato come segue:

- *Capitolo 2, CAPWAP e OpenCAPWAP dalla teoria alla pratica*: panoramica del protocollo CAPWAP e dei principali componenti che caratterizzano l'implementazione di OpenCAPWAP.

- *Capitolo 3, strumenti utilizzati*: descrizione degli strumenti utilizzati nel progetto. Hardware, OpenWrt, Netfilter, Conntrack, L7-filter, ULOG, Netlink Socket.

- *Capitolo 4, layer 7 packet classifier per OpenCAPWAP*: presentazione dei moduli software WTPL7Agent e ACL7Manager. Viene illustrata l'architettura client-server, il design dell'Agent, dell'utility Manager e i dettagli di integrazione con OpenCAPWAP.

- *Capitolo 5, test e conclusioni*: vengono illustrate le diverse sessione di test in laboratorio, in produzione, i risultati raccolti, i possibili scenari di applicazione e gli sviluppi futuri.

Capitolo 2

CAPWAP e OpenCAPWAP dalla teoria alla pratica

In questo capitolo vengono illustrate le principali caratteristiche del protocollo CAPWAP e i componenti base della sua implementazione open source OpenCAPWAP.

2.1 Terminologia

- **Access Controller** (AC): Dispositivo che fornisce ai WTP l'accesso all'infrastruttura di rete.
- **CAPWAP Control Channel**: Flusso bidirezionale di rete definito dall'indirizzo IP dell'AC e del WTP, dalla porta di controllo dell'AC e del WTP, dal protocollo di trasporto (UDP o UDP-Lite) sul quale i pacchetti di controllo di CAPWAP sono inviati e ricevuti.
- **CAPWAP Data Channel**: Flusso bidirezionale di rete definito dall'indirizzo IP dell'AC e del WTP, dalla porta dati dell'AC e del WTP, dal protocollo di trasporto (UDP o UDP-Lite) sul quale i pacchetti dati di CAPWAP sono inviati e ricevuti.
- **Station** (STA): Dispositivo dotato di un'interfaccia per il media wireless (WM).
- **Wireless Termination Point** (WTP): Dispositivo dotato di un'antenna RF per la trasmissione e la ricezione del traffico delle stazioni che condividono l'accesso alla rete wireless.
- **Medium Access Protocol** (MAC): Protocollo per mediare l'accesso esclusivo al canale fisico di comunicazione condiviso. Nel caso in cui il mezzo condiviso sia un canale radio, il MAC è equivalente al Channel Access Protocol.

2.2 Modalità operative: Split e Local MAC

CAPWAP assume una configurazione di rete composta da WTP multipli che comunicano via IP con uno o più AC. I WTP sono visti come interfacce radio remote (RF) controllate tramite AC. Il protocollo supporta due modalità operative: Local Mac (LM) e Split Mac (SM).

- **Local MAC** (LM): Il livello 802.11 MAC risiede completamente sul WTP. Il vantaggio è che tutte le funzioni MAC sono eseguite velocemente, ma i WTP sono più complessi e quindi costosi.
- **Split MAC** (SM): Le funzioni 802.11 MAC sono separate in realtime e non realtime. Le funzioni realtime come la beacon generation, RTS, CTS e i messaggi ACK sono eseguite dal livello MAC del WTP. Le funzioni non realtime come le procedure di autenticazione, la frammentazione/deframmentazione dei frame sono eseguite dall'AC.

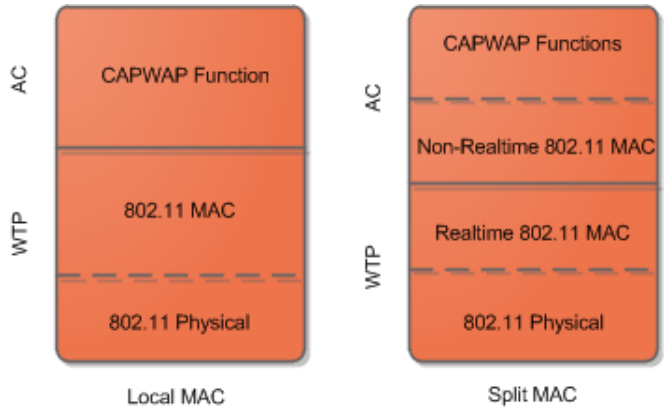


Figura 1: Architetture Local MAC e Split MAC.

Funzione	**Dispositivo**	
	Split MAC	**Local MAC**
Distribution Service	AC	WTP/AC
Integration Service	AC	WTP
Beacon Generation	WTP	WTP
Probe Response Generation	WTP	WTP
Power Management / Packet Buffering	WTP	WTP
Fragmentation / Defragmentation	WTP/AC	WTP
Association / Disassociation / Reassociation	AC	WTP/AC
IEEE 802.11 QoS		
Classifying	AC	WTP
Scheduling	WTP/AC	WTP
Queuing	WTP	WTP

Tabella 1: Associazione delle funzioni 802.11 per le architetture Split MAC e Local MAC[1].

2.3 Panoramica del protocollo

Il livello di trasporto di CAPWAP trasferisce due tipi di payload, CAPWAP Data message e CAPWAP Control message. I CAPWAP Data message incapsulano i frame wireless da instradare all'Access Controller. I CAPWAP Control messages contengono i messaggi di amministrazione scambiati tra WTP e AC.

Figura 2: CAPWAP Data packet.

Figura 3: CAPWAP Control packet.

I CAPWAP Control message e opzionalmente i CAPWAP Data message sono trasferiti in modo sicuro utilizzando il Datagram Transport Layer Security (DTLS). I pacchetti dati e controllo di CAPWAP sono inviati utilizzando porte UDP distinte. I pacchetti possono superare la lunghezza della Maximum Transmission Unit (MTU), quindi il payload dei messaggi CAPWAP può essere frammentato.

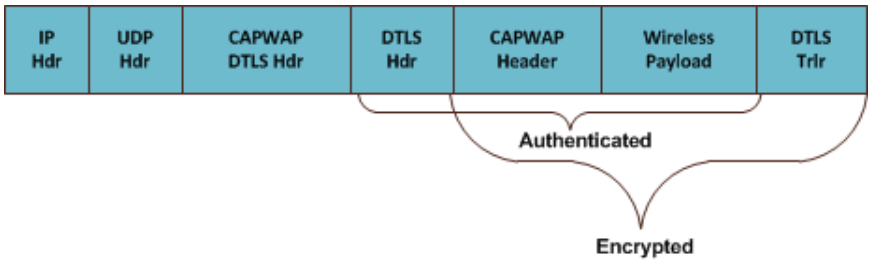


Figura 4: CAPWAP Data packet encrypted.

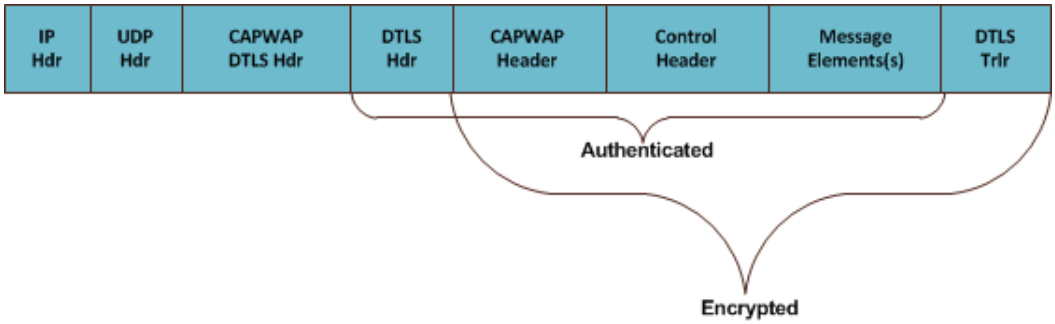
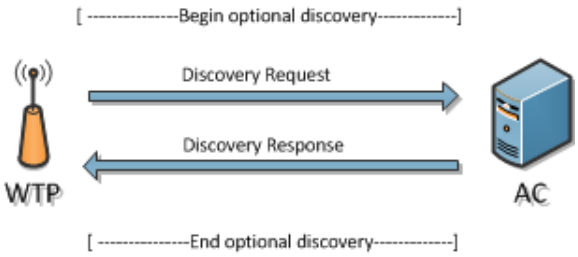
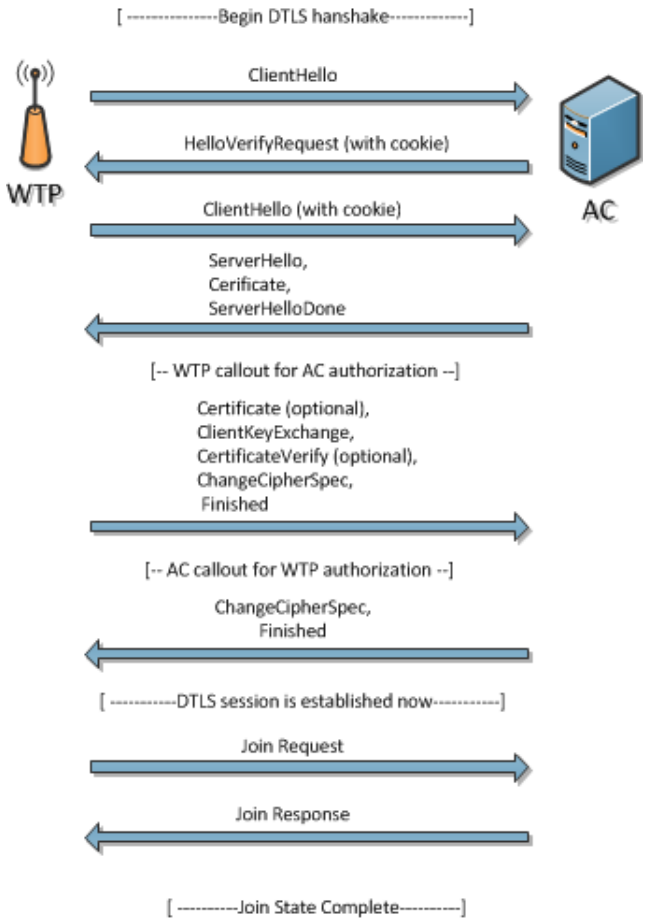


Figura 5: CAPWAP Control packet encrypted.

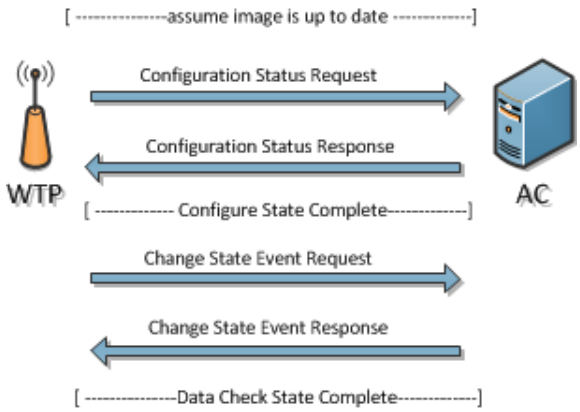
Il protocollo CAPWAP inizia con una fase di Discovery. I WTP inviano un messaggio di tipo Discovery Request. Qualsiasi Access Controller che riceve il messaggio risponde con uno di tipo Discovery Response.



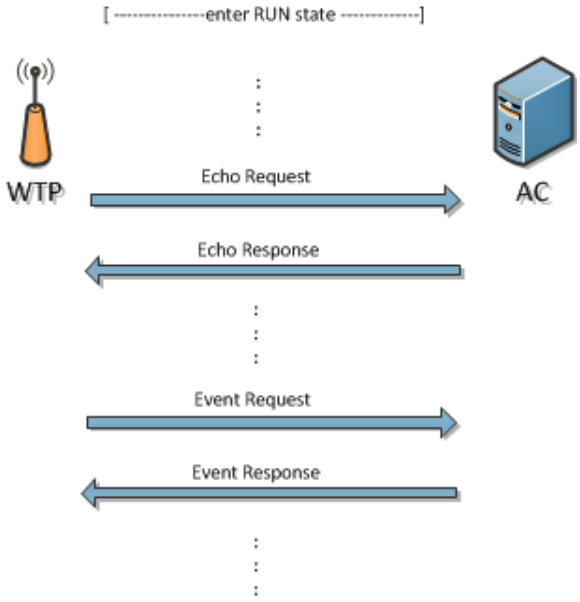
Dai messaggi di Discovery Response ricevuti i WTP selezionano un AC con cui stabilire una sessione sicura DTLS. Per stabilire una connessione sicura sono necessarie delle attività di pre-provisioning di alcuni paramentri sui WTP (es. la configurazione delle chiavi condivise o l'installazione di certificati).



Completata l'inizializzazione della sessione, inizia lo scambio della configurazione nel quale AC e WTP si accordano sulla versione. Durante questo scambio i WTP possono ricevere le impostazioni per il provisioning.



I WTP sono operativi e possono iniziare a scambiare i messaggi CAPWAP di tipo dati e controllo con l'AC. Il protocollo prevede la funzione di $keep--alive$ per mantenere attivo il canale di comunicazione tra WTP e AC. Se l'AC non risulta più attivo allora il WTP tornerà in fase di Discovery per cercarne uno nuovo.



2.4 Macchina a stati di CAPWAP

La macchina a stati raffigurata nel diagramma a blocchi in Figura 6 rappresenta il ciclo di vita di una sessione WTP-AC. L'automa risultante deriva dalla

giustapposizione della macchina a stati di CAPWAP con quella del DTLS. Sia il WTP che l'AC utilizzano la stessa macchina a stati. Segue una sintetica descrizione dei cambi di stato più significativi e degli eventi alla loro origine:

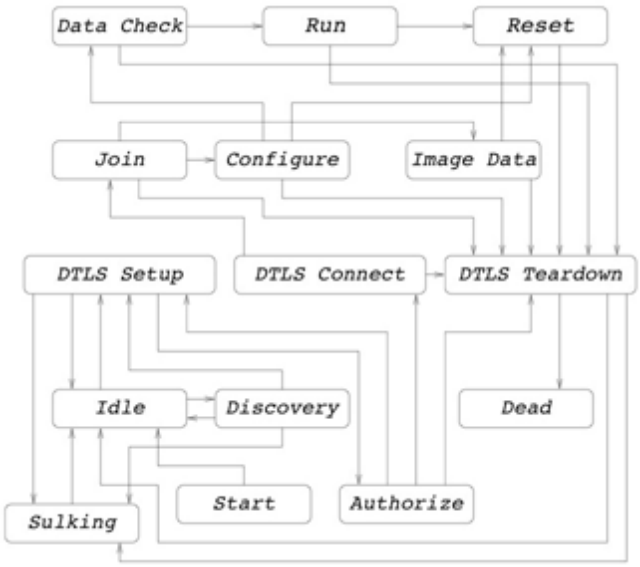


Figura 6: macchina a stati finiti di CAPWAP.

- **Idle -> Discovery**: Questa transizione avviene per supportare la fase di Discovery di CAPWAP.
- **Discovery -> DTLS Setup**: Questa transizione avviene per stabilire una sessione sicura DTLS.
- **DTLS Connect -> Join**: Questa transizione avviene quando una sessione DTLS è stata instaurata con successo.
- **Join -> Image Data**: Questa transizione avviene per il download dell'immagine del firmware.
- **Join -> Configure**: Questa transizione avviene per lo scambio dei parametri di configurazione.
- **Configure -> Data Check**: Questa transizione avviene quando il WTP e AC confermano la configurazione.
- **Data Check -> Run**: Questa transizione avviene quando è stato stabilito il collegamento tra i canali di controllo e dati; WTP e AC entrano nello stato di funzionamento normale.

- **Run -> Run**: Stato di funzionamento normale dei WTP e AC; diversi eventi portano a questo stato: *Configuration Update*, *Change State Event, Echo Request, Clear Config Request, WTP Event, Data Transfer, Station Configuration Request.*

2.5 Trasporto UDP

La comunicazione tra WTP e AC avviene utilizzando un modello client/server che sfrutta il protocollo di trasporto UDP. La sessione CAPWAP è iniziata dal WTP (client) sulla *well-known port* dell'AC (server). La porta per i messaggi di controllo dell'Access Controller è la 5246, mentre quella per i messaggi dati è la 5247.

2.6 OpenCAPWAP

Il progetto open source OpenCAPWAP[2] nasce con l'obiettivo di fornire un'alternativa libera e gratuita alle soluzioni commerciali dei principali vendor basate su CAPWAP. La piattaforma di riferimento sono i dispositivi hardware che eseguono il sistema operativo GNU/Linux. Le principali caratteristiche di OpenCAPWAP alla versione 0.93.3 sono:

- E' sviluppato interamente in C;
- E' multi-thread per offrire un buon compromesso tra efficienza e modularità del codice;
- E' caratterizzato da due demoni da distribuire rispettivamente sugli AC e sui WTP;
- Implementa il *binding* per lo standard 802.11;
- Al momento supporta unicamente l'architettura Local MAC;
- I messaggi di controllo e dati utilizzano la stessa porta di comunicazione;
- Integrazione di un *application management server* per la creazione di moduli di estensione di OpenCAPWAP come applicazioni esterne.
- Soddisfa i vincoli di sicurezza di CAPWAP utilizzando l'implementazione del protocolo DTLS di OpenSSL[3].

2.6.1 Architettura AC-WTP

In Figura 7 è riportata un esempio di architettura di gestione basata su OpenCAPWAP. I componenti principali sono:

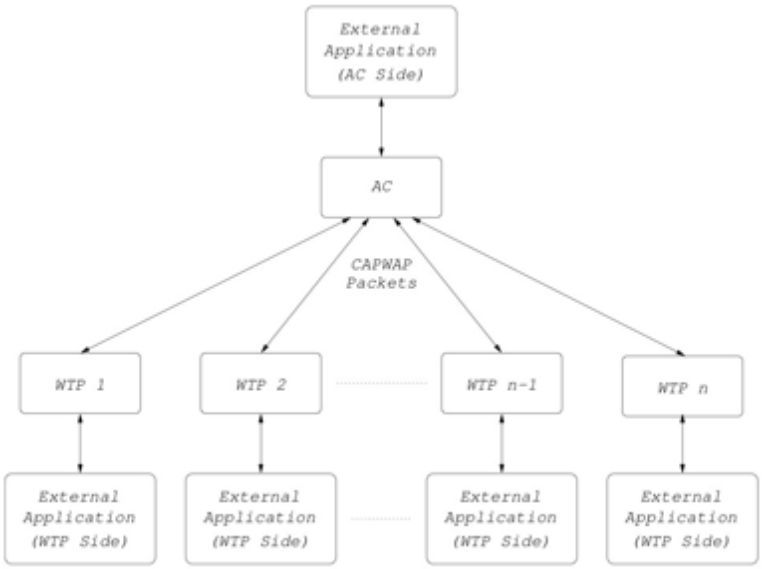


Figura 7: esempio di architettura di management di OpenCAPWAP.

- **Demone AC**: Amministra e configura i WTP della rete. Gestisce le comunicazioni con le applicazioni esterne.
- **Demone WTP**: Gestisce le comunicazioni con l'Access Controller e con le applicazioni esterne.
- **Applicazione Esterna** (lato AC): Può richiedere la lista dei WTP associati all'AC, scambiare dati con uno o più WTP.
- **Applicazione Esterna** (lato WTP): Può scambiare dati con le applicazioni esterne lato AC, inoltrati tarmite il demone WTP.

2.6.2 Demone AC

Il demone deve essere installato sui server che svolgono le funzioni di controller della rete. L'applicativo è un'implementazione multi-thread della macchina a stati di CAPWAP per sistemi GNU/Linux. L'architettura del demone è illustrata nella Figura 8 e si compone dei seguenti quattro thread:

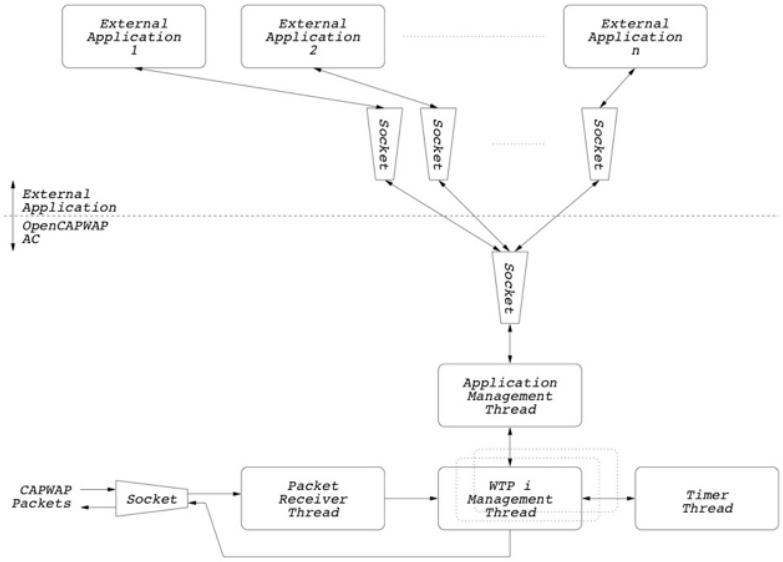


Figura 8: Architettura demone AC.

- **Packet Receiver**: Riceve e processa le richieste dei WTP. Se la richiesta è relativa ad una sessione valida la sposta nella lista del thread di gestione appropriato; altrimenti viene o stabilita una nuova sessione (e creato un nuovo thread di gestione) o scartata la richiesta.
- **WTP Management**: Per ogni sessione stabilita viene creato questo thread che ha il compito di gestire la comunicazione tra il WTP e AC tramite messaggi CAPWAP.
- **Timer**: Gestisce i timer e i timeout necessari ai thread del demone.
- **Application Management**: Interfaccia per applicazioni esterne che permette di implementare alcune funzioni del protocollo come moduli esterni al demone. Questa interfaccia verrà descritta in dettaglio nel Capitolo 4 nella sezione relativa all'integrazione del mio modulo software con OpenCAPWAP.

2.6.3 Demone WTP

Il demone deve essere installato sui WTP della rete. L'applicativo è un'implementazione multi-thread della macchina a stati di CAPWAP per sistemi GNU/Linux embedded. In particolare il progetto è stato sviluppato e testato con la distribuzione OpenWrt[4]. L'architettura del demone è illustrata nella Figura 9 e si compone dei seguenti tre thread:

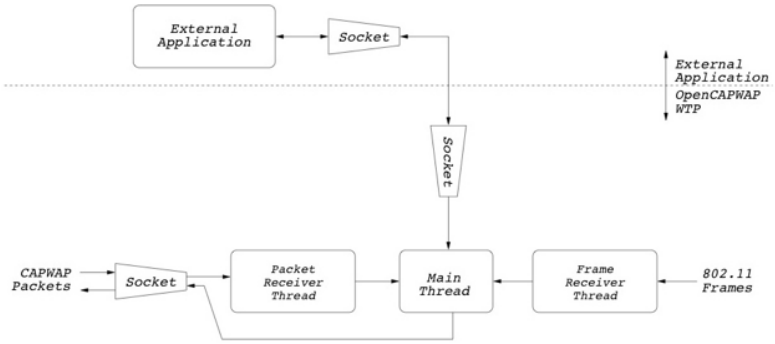


Figura 9: Architettura demone WTP.

- **Packet Receiver**: Legge i pacchetti dal socket UDP e li passa al Main thread tramite una lista condivisa.
- **Main Thread**: Gestisce la comunicazione con l'unico Access Controller a cui ha scelto di collegarsi. Gestisce la comunicazione con le applicazioni esterne lato WTP.
- **Frame Receiver**: Intercetta i pacchetti inviati dalle stazioni (STA) collegate al WTP e gestisce le statistiche relative al traffico radio.

2.6.4 Moduli di estensione

Al momento sono disponibili alcuni moduli di estensione che implementano nuove funzionalità di OpenCAPWAP per lo più sviluppati come lavoro di tirocinio da altri studenti del Corso di Laurea in Informatica dell'Università "La Sapienza". I moduli attualmente disponibili sono:

- **Hot-spots manager**[1]: Sistema di amministrazione integrale dei WTP della rete che utilizzano il sistema operativo OpenWrt, tramite l'invio di comandi utilizzando *OpenWrt's Unified Command Interface* (UCI) da un'unica console di gestione installata sugli AC; .
- **OpenCAPWAP update package**[5]: Sistema per l'aggiornamento del software di controllo dei WTP.
- **Sniffer traffico VoIP**[6]: Analizzatore per la cattura e l'analisi del traffico con l'obiettivo di individuare le comunicazioni VoIP.

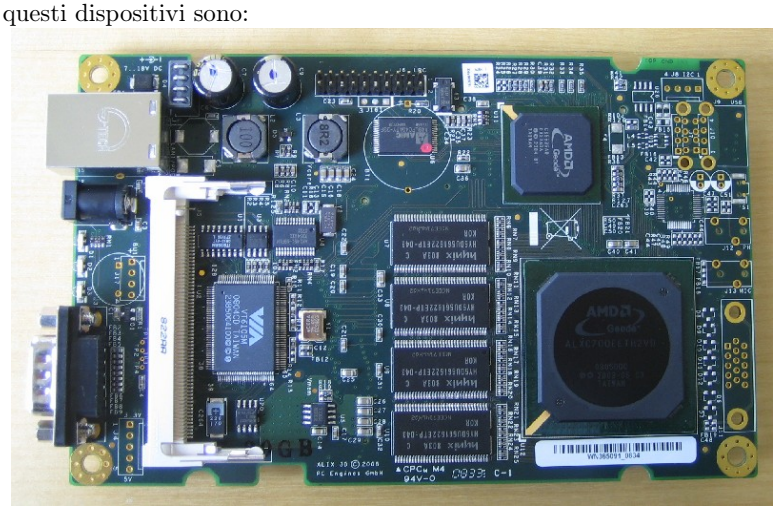
Capitolo 3

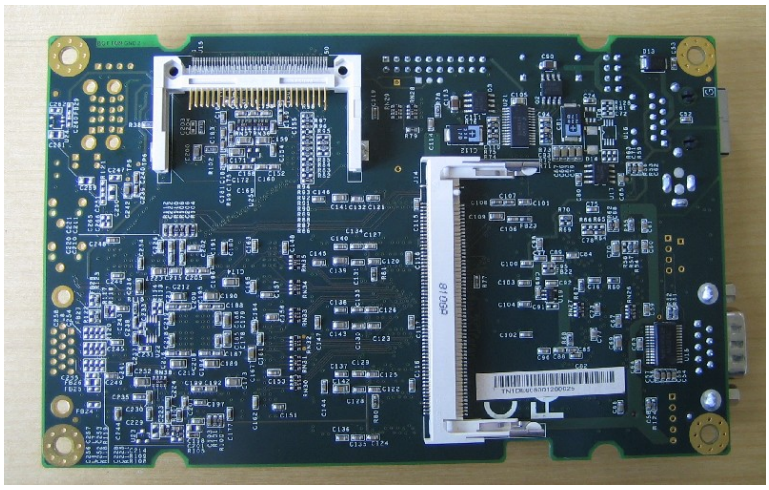
Strumenti Utilizzati

In questo capitolo vengono descritti la piattaforma hardware e gli strumenti software open source utilizzati per la realizzazione del progetto.

3.1 Hardware dei WTP

Sono state utilizzate le schede *alix3d1* della **PC Engines**[7] per la fase di sviluppo e per le fasi di test in laboratorio e in produzione. Le specifiche tecniche di questi dispositivi sono:





- **CPU**: 433 MHz AMD Geode LX700.
- **DRAM**: 128/256/512 MB DDR DRAM.
- **Storage**: Socket per CompactFlash.
- **Espansione**: 2 miniPCI slots, utilizzati per le schede WiFi 802.11 con chipset Atheros.
- **Connettività**: 1 scheda di rete Ethernet 10/100 con chipset Via VT6105M.
- **I/O**: porta seriale DB9.

I sistemi come l'*alix3d1*, costituiti da una sola scheda, sulla quale viene saldata la CPU, l'apparato radio, i socket miniPCI etc. vengono indicati con il nome di *Single Board Computer* (SBC). Le architetture più comuni per gli SBC sono x86, ARM e MIPS. La scelta dell'architettura x86 ha il vantaggio di avere un vero PC, anche se di piccole dimensioni, ma il consumo energetico è superiore a quello di un dispositivo con processore ARM o MIPS. Anche la quantità di memoria RAM non è comune. Quella disponibile in media nei dispositivi embedded è nell'ordine delle decine di Mbyte. La scelta del chipset Atheros per il sistema radio è dovuta al forte legame con la comunità open source per lo sviluppo del driver MadWiFi[8], il driver Linux più evoluto per dispositivi wireless, centrale nello sviluppo di alcune funzioni di OpenCAPWAP.

3.2 OpenWrt

OpenWrt[4] è una distribuzione Linux per sistemi embedded che dispone di un avanzato ambiente di cross'compilazione che automatizza processi complessi come il download dei sorgenti ed il loro patching. OpenWrt permette di installare

facilmente nuovo software mediante lo strumento dei *pacchetti*. Il packet manager si chiama *opkg* e permette l'installazione e l'aggiornamento del software tramite repository esterni come qualsiasi distribuzione Linux. La stabilità, la maturità e l'estendibilità del sistema hanno permesso l'adozione di OpenWrt anche in contesti industriali. La versione di riferimento del sistema operativo per lo sviluppo del progetto è **OpenWrt Kamikaze 8.09**.

3.3 Netfilter

E' il framework di packet filtering incluso nel kernel di Linux 2.4 e 2.6. Rappresenta un supporto fondamentale per il sistema operativo in termini di sicurezza e gestione dei pacchetti di rete. L'interfaccia utente per netfilter si chiama *iptables* e permette agli amministratori di sistema di definire le regole per i pacchetti in entrata, in transito e in uscita dalla macchina Linux. Brevemente si può riassumere il funzionamento di netfilter nel modo seguente:

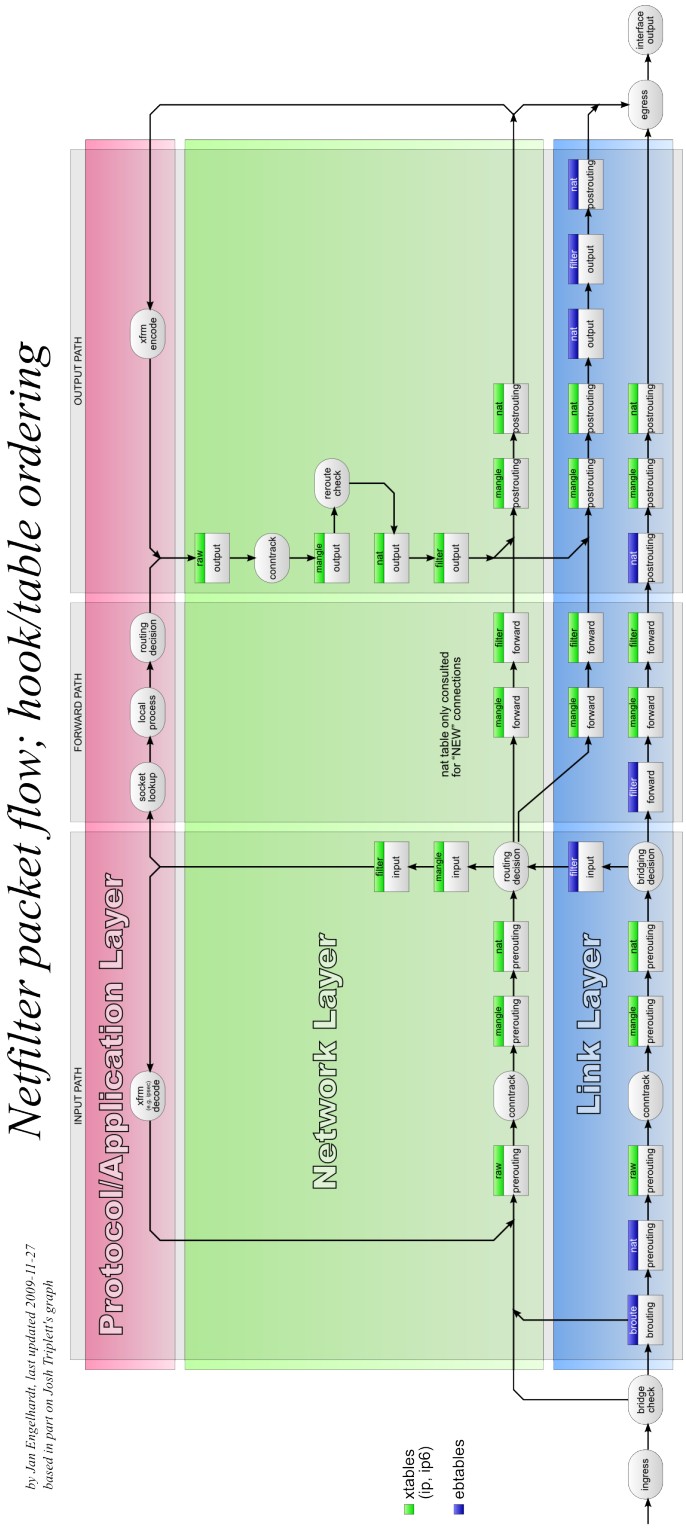
- Attraverso lo strumento iptables l'amministratore imposta le regole per i pacchetti IP che transitano sulla macchina;
- Netfilter analizza gli header IP di tutti i pacchetti che passano sulla macchina;
- Se viene identificato un pacchetto che corrisponde ad una delle regole impostate, il pacchetto viene manipolato secondo tale regola.

In realtà il processo è molto più complesso, netfilter utilizza delle tabelle che contengono delle regole chiamate *chain*. I pacchetti transitano attraverso le tabelle e le chain caricate da netfilter. I moduli di estensione di netfilter possono creare delle tabelle ad hoc con proprie chain. Gli amministratori, tramite iptables, possono creare nuove tabelle e chain in base alle proprie necessità.

Le tabelle di default caricate nel kernel sono tre[15]:

- **INPUT:** contiene le regole per i pacchetti destinati alla macchina.
- **FORWARD**: contiene le regole per i pacchetti da instradare verso altre macchine.
- **OUTPUT**: contiene le regole per i pacchetti generati dalla macchina.

Nella Figura è riportato il percorso che segue un pacchetto di rete tra le tabelle e le chain di netfilter:


Netfilter packet flow; hook/table ordering
by Jan Engelhardt, last updated 2009-11-27
based in part on Josh Triplett's graph
Protocol/Application Layer
Network Layer
Link Layer
INPUT PATH
FORWARD PATH
OUTPUT PATH
nat table only consulted for "NEW" connections
xtables (ip, ip6)
ebtables
ingress
bridge check
egress
interface output

3.4 Connection Tracking System

Il modulo di connection tracking di netfilter si chiama *ip_conntrack*, il suo compito è quello di memorizzare in una struttura dati in memoria lo stato delle connessioni. La struttura contiene l'indirizzo IP sorgente e destinazione, la coppia dei numeri di porta, il tipo di protocollo, lo stato e il timeout. Con queste informazioni è possibile definire politiche più intelligenti di filtraggio. Questa tecnica prende il nome di *stateful filtering*. I possibili stati di una connessione sono:

- **NEW**: una nuova connessione sta iniziando.
- **ESTABLISHED**: la connessione è stata stabilita.
- **RELATED**: i pacchetti sono correlati a connessioni esistenti.
- **INVALID**: i pacchetti non seguono il comportamento previsto per la connessione.

Il kernel di Linux mantiene e aggiorna in tempo reale lo stato delle connessioni gestite nel file */proc/net/ip_conntrack*. Il modulo di connection tracking si occupa esclusivamente di tracciare i pacchetti e non applica alcun tipo di filtro.

3.5 L7 Filter

L7 filter[9] è un classificatore di pacchetti per il kernel di Linux implementato come modulo di netfilter. Al contrario di altri classificatori non guarda semplicemente i valori negli header dei pacchetti, come ad esempio il numero di porta, ma applica espressioni regolari ai dati del livello applicazione del modello ISO/OSI per determinare quale protocollo viene utilizzato. L'indipendenza dal protocollo e dalla porta permette di classificare il traffico di applicazioni che non utilizzano porte standard o che possono essere configurate per utilizzare porte standard di altre applicazioni (es. filesharing con peer-to-peer su porta 80). Nel modello ISO/OSI (Figura 10) sono previsti sette livelli. Prima della nascita del progetto L7 si potevano definire solo le regole per i livelli:

- **Data Link**: iptables -A CHAIN **-m mac** ...
- **Network**: iptables -A CHAIN **-s IP_ADDRESS** ...
- **Transport**: iptables -A CHAIN **-p tcp –dport 80** ...

Nella definizione delle regole è ammessa una qualsiasi combinazione per i tre distinti livelli. L7 filter aggiunge a netfilter le funzionalità per identificare i pacchetti sulla base del contenuto dei dati del livello 7. Le regole per L7 possono essere combinate con quelle degli altri livelli e con i pacchetti identificati è possibile fare tutte le operazioni previste da iptables.

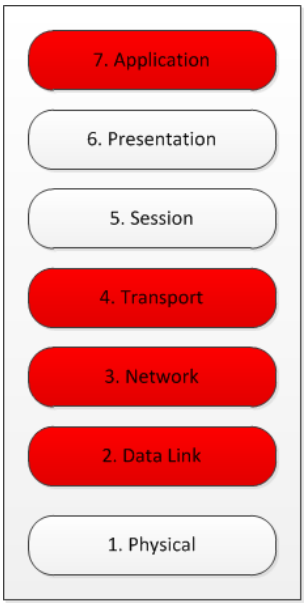


Figura 10: Modello ISO/OSI

- **Application**: iptables -A CHAIN **-m layer7** –**l7proto** [http | ftp | ...].

3.5.1 Dettagli Tecnici

L'architettura di L7 filter si basa su tre componenti principali:

- Una patch per il kernel di Linux, che permette al kernel di accedere il contenuto dei pacchetti;
- Una patch per iptables, per l'introduzione delle nuove opzioni per L7;
- Una collezione di file di pattern che contengono le espressioni regolari per i protocolli supportati[10].

Applicare le espressioni regolari ad ogni pacchetto di una connessione non è molto utile, anzi risulta un vero e proprio spreco di cicli macchina perché la maggior parte dei pacchetti di una connessione sono costituiti dal contenuto centrale di qualche file o trasmissione dati. La filosofia alla base di L7 filter è proprio quella di limitarsi a guardare il contenuto di un numero ristretto di pacchetti di ogni nuova connessione alla ricerca di un pattern caratteristico, come ad esempio **"220 ftp server ready"**, **"* ok"**, o **"HTTP/1.1 200 ok"**. Attraverso il modulo di connection tracking *ip_conntrack* tutti i pacchetti appartenenti ad una connessione identificata da l7 filter vengono automaticamente marcati. Per default l7 filter controlla i primi 10 pacchetti o

2KB. Questo limite può essere modificato in qualsiasi momento tramite il file proc: */proc/net/layer7_numpackets*. Poiché vengono esaminati solo alcuni pacchetti per connessione, l7 risulta ragionevolmente efficiente rispetto ad altri classificatori di pacchetti.

Lo pseudo codice del modulo di Netfilter per l'identificazione del protocollo è il seguente[11]:

```
/* This is the standard Netfilter match function, which returns
true on a match and false otherwise. Obviously, many details are
ignored here. */
int match(packet, protocol)
{
    if(regular expression for the protocol is not compiled yet)
       compile it and put it in a list of compiled regexps;
    else
       fetch the compiled pattern from the list;

     if(already classified this connection)
        if(classification matches one we're looking for)
           return true;
        else
           return false;

    if(seen too many packets with no match)
           return false;
    Append application layer data to data buffer;

    if(data buffer matches regexp for the protocol we're looking for)
       Mark the connection as identified;
       return true;
    else
       return false;
}
```

3.5.2 Definizione dei protocolli (File dei Pattern)

Le definizioni dei protocolli non sono altro che dei file che contengono le espressioni regolari e servono per indicare al kernel e a iptables le corrispondenze tra nomi e espressioni regolari: es. "ftp" corrisponde a "^220[\x09-\x0d -~]*ftp". I file sono contenuti per default nella directory */etc/l7-protocols* ma è possibile installarli in una directory diversa. La lista dei protocolli supportati è lunga e ne comprende oltre 100[10]. Inoltre è molto semplice aggiungere il supporto per un nuovo protocollo, infatti è sufficiente aggiungere un file con il pattern in */etc/l7-protocols*. Il formato del file è elementare:

- Nome del protocollo su una linea;

- Espressione regolare che definisce il protocollo sulla linea successiva.

L7 utilizza l'implementazione delle **Bell Version 8 regular expressions** di Henry Spencer[12]. Per la creazione delle espressioni regolari è necessario osservare il comportamento del protocollo per la ricerca di un pattern utile. Per questo si possono utilizzare degli analizzatori di protocollo di rete come **Wireshark**[13]. Nel sito **http://protocolinfo.org** sono documentati i metodi per identificare i protocolli di rete del livello applicazione.

3.5.3 Come si utilizza

L7 filter utilizza la sintassi standard delle estensioni per iptables[14]:

```
iptables -t [tabella] -A [chain] -m layer7
 --l7proto [nome protocollo] -j [azione]
```

Il parametro **nome protocollo** deve specificare il nome di uno dei file presenti nella directory contenente la definizione dei protocolli. E' possibile utilizzare l7 con qualsiasi opzione disponibile per iptables, dunque con il filtro si può:

- **Bloccare le applicazioni**: è una delle cose che è vivamente sconsigliato fare a causa dei falsi positivi che possono creare problemi ad altre applicazioni.
- **Traffic shaping**: limitare l'utilizzo della banda da parte di un'applicazione marcando in modo particolare i suoi pacchetti.
- **Accounting**: raccogliere dei dati sul consumo delle risorse da parte di una determinata applicazione.

In OpenWrt, come sarà illustrato nella sezione **3.8 Setup di OpenWrt**, il kernel e iptables hanno già i sorgenti patchati e sono disponibili in forma di pacchetti sia il modulo L7-filter e sia le definizioni dei protocolli.

3.6 Target ULOG

Tramite iptables è possibile registrare i pacchetti per il debug e l'analisi del traffico. ULOG è un tipo di target per iptables che permette di trasferire i pacchetti da kernel space a user space. Se un pacchetto corrisponde ad una regola di iptables ed è stato impostato il target ULOG, allora le informazioni del pacchetto vengono inviate in multicast tramite un netlink socket. Grazie al multicast uno o più processi in user space possono ricevere e lavorare con i pacchetti senza che un loro malfunzionamento possa interrompere la gestione del traffico da parte di netfilter. Il target ULOG è meno diffuso e usato del target LOG di iptables ma è molto più flessibile e estende il numero di applicazioni possibili. Le opzioni per il target ULOG sono:

- –**ulog-nlgroup**: imposta il gruppo netlink di multicast a cui deve essere inviato il pacchetto. I gruppi sono 32 numerati da 1 a 32. Il gruppo di default è 1.
- –**ulog-prefix**: imposta la stringa di prefisso alla riga di log, la lunghezza massima è di 32 caratteri.
- –**ulog-cprange**: imposta il numero di byte del pacchetto da trasferire in user space. Il valore di default è zero e indica che tutto il pacchetto deve essere copiato in user space.
- –**ulog-qthreshold**: imposta il numero di pacchetti da accodare nel kernel prima di trasferirli in user space.

Esempio di utilizzo del target ULOG:

```
iptables -A INPUT -p TCP --dport 22 -j ULOG --ulog-nlgroup 2
--ulog-prefix "SSH connection attempt: " --ulog-cprange 100
--ulog-cprange 100 --ulog-qthreshold 10
```

E' importante sottolineare che ULOG non interrompe l'attraversamento di una chain, i pacchetti una volta inviati al log continuano ad essere processati da netfilter secondo le regole successive.

3.7 Netlink Socket

I socket netlink[16] sono una flessibile interfaccia di comunicazione tra le applicazioni in user space e i moduli del kernel. Forniscono avanzate funzionalità di comunicazione come il supporto per il full-duplex, I/O bufferizzato, multicast e comunicazione asincrona che non sono disponibili negli altri meccanismi di IPC tra user e kernel space. La comunicazione per le applicazioni in user space avviene tramite le API standard per socket, mentre per i moduli del kernel devono essere utilizzate apposite API. I socket netlink utilizzano indirizzi della famiglia **AF_NETLINK** e il tipo è sempre **SOCK_RAW**:

```
socket_fd = socket(AF_NETLINK, SOCK_RAW, protocollo)
```

L'unico parametro variabile è il valore del protocollo, i cui tipi sono definiti nel file di header *include/linux/netlink.h*. Un sottoinsieme dei protocolli supportati dai socket netlink è composto da:

- **NETLINK_ROUTE**: i demoni di routing in user space aggiornano le tabelle di routing attraverso questo tipo di protocollo netlink.
- **NETLINK_FIREWALL**: riceve i pacchetti IPv4 inviati del firewall.
- **NETLINK_NFLOG**: canale di comunicazione tra iptables in user space e il modulo netfilter in kernel space.

- **NETLINK_ARPD**: per amministrare la tabella di arp da user space.

I socket netlink sono *connectionless* e funzionano in modo simile ai socket UDP. I messaggi sono inviati attraverso le funzioni di libreria *sendto* e *sendmsg*, mentre sono ricevuti attraverso le funzioni *recvfrom* e *recvmsg*. Rispetto ad altri sistemi di IPC come **system call**, **ioctls** e **file proc** i socket netlink sono:

- **Asincroni**: la comunicazione tramite socket fa si che i messaggi vengono aggiunti alla coda del socket netlink del ricevitore lasciandolo libero di decidere quando processare i messaggi.
- **Full duplex**: le sessioni possono essere iniziate sia dalle applicazioni in user space, come nel paradigma classico IPC, sia dal kernel permettendo la gestione di comunicazioni urgenti senza l'utilizzo del polling.
- **Multicast**: un processo può inviare un messaggio ad un indirizzo netlink di gruppo e un numero qualsiasi di processi può essere in ascolto su quel indirizzo.
- **Estendibili**: per nuove funzionalità non è necessario "modificare" il codice del kernel mettendo in pericolo la stabilità del sistema. É sufficiente aggiungere la definizione di un nuovo protocollo, sottoforma di costante, nel file netlink.h.

Nel presente progetto i socket netlink sono utilizzati per il trasferimento - tramite il target ULOG di iptables - dei pacchetti identificati da l7 filter in kernel space verso il mio agent nello spazio utente.

3.8 Setup di OpenWrt

Per la release **Kamikaze 8.09** è necessario installare i seguenti pacchetti per abilitare il supporto di l7 filter e del target ULOG:

- kmod-ipt-conntrack
- kmod-ipt-ulog
- kmod-ipt-filter
- iptables-mod-ulog
- iptables-mod-filter
- l7-protocols

Nelle architetture di rete dove l'interconessione delle reti WiFi avviene a livello 2, come nel caso del progetto *Provinciawifi*, è necessario abilitare il **bridge-netfilter** per permettere a netfilter di vedere i pacchetti IPv4, IPv6 e ARP scambiati in bridge, anche se incapsulati in 802.1Q VLAN. In OpenWrt la procedura consiste nell'abilitare il parametro del kernel **CONFIG_BRIDGE_NETFILTER=y**

e ricompilare la distribuzione con i relativi pacchetti utilizzando la *toolchain* appropriata per l'architettura del dispositivo. Il secondo passo è abilitare la modalità bridge di iptables impostando i seguenti parametri nel file */etc/sysctl.conf*:

- net.bridge.bridge-nf-call-arptables=1
- net.bridge.bridge-nf-call-ip6tables=1
- net.bridge.bridge-nf-call-iptables=1

Capitolo 4

Layer 7 packet classifier per OpenCAPWAP

In questo capitolo vengono illustrate le funzionalità e la struttura del sistema di identificazione dei pacchetti progettato per OpenCAPWAP.

4.1 Cos'è, cosa fa e come funziona?

E' un'estensione del progetto OpenCAPWAP che implementa una soluzione software client-server per la classificazione dei pacchetti di rete sui dispositivi WTP, come previsto dalla modalità operativa **Local MAC** supportata da OpenCAPWAP. Il sistema si compone di un'utility di amministrazione **ACL7Manager** che risiede sul server dell'Access Controller e di un agent multi-thread **WTPL7Agent** che risiede sui nodi amministrati. Lo scambio dei messaggi avviene utilizzando il protocollo CAPWAP. Gli agent processano i pacchetti e raccolgono statistiche sui protocolli identificati e possono salvare su file alcune informazioni relative ai pacchetti. Solo su richiesta inoltrano le informazioni al server di amministrazione. L'utility di amministrazione invia le richieste a uno o più nodi registrati sul server centrale, elabora i dati raccolti e sulla base dei risultati può inviare agli agent azioni dispositive per la modifica dei paramentri della QoS. Quest'ultima parte non è stata sviluppata all'interno del lavoro di tirocinio e costituisce uno degli sviluppi futuri del progetto.

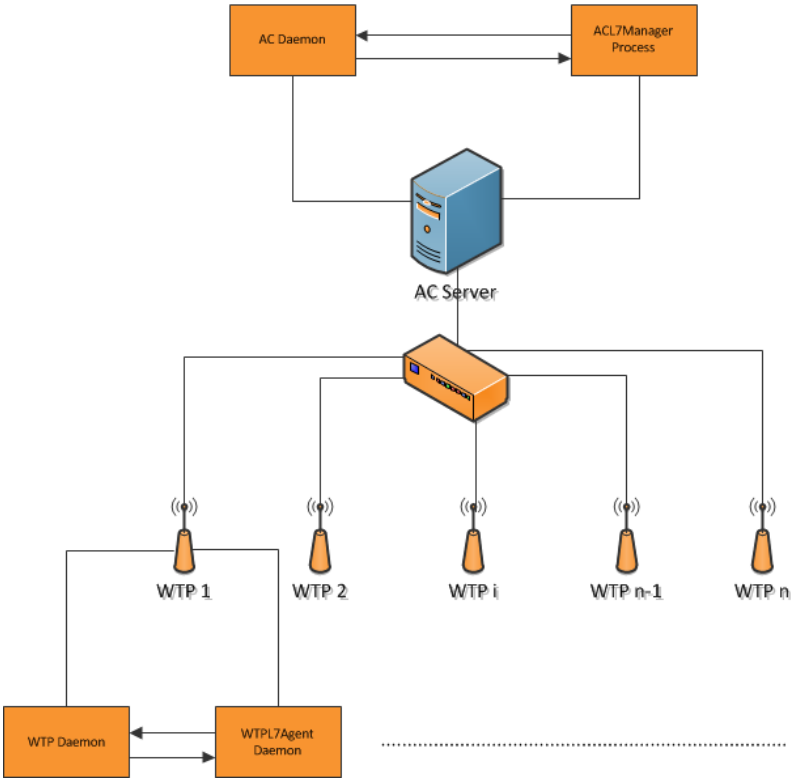


Figura 11: Esempio di architettura hardware con lo spiegamento sui sistemi del software OpenCAPWAP con il modulo L7 classifier.

Il sistema L7 classifier per OpenCAPWAP permette, da una console centralizzata di gestione, l'attivazione remota dei pattern L7 sui nodi amministrati e di collezionare dati e statistiche distinti per protocollo del livello applicazione. Lo scambio delle informazioni tra gli agent e il manager avviene attraverso messaggi incapsulati nel protocollo CAPWAP. Il flusso delle informazioni e l'elaborazione dei dati possono essere così schematizzati:

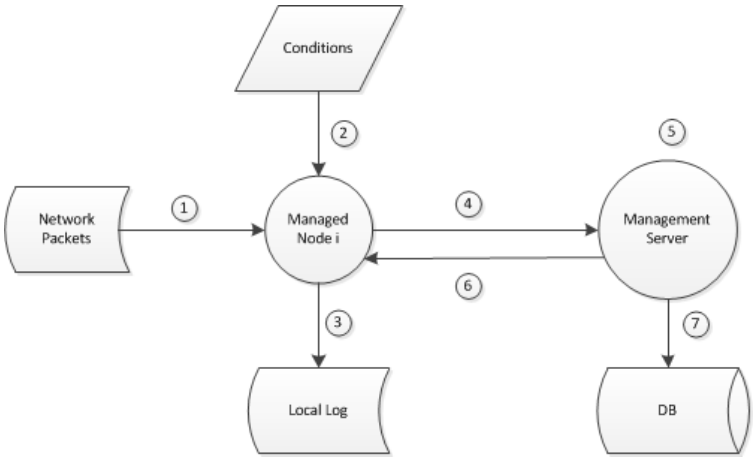


Figura 12: Schema del flusso delle informazioni e dell'elaborazione dei dati tra Agent e Manager.

1. **Analisi dei pacchetti di rete**: i pacchetti di rete originati dalle stazioni (STA) collegate al WTP vengono analizzati da netfilter. Quando un pacchetto corrisponde ad una regola L7, viene eseguita l'azione target ULOG che si occupa di copiare il pacchetto o una sua porzione nello spazio utente tramite il netlink socket.

2. **Valutazione delle condizioni**: per ogni pacchetto ricevuto tramite il netlink socket, al momento, possono essere eseguite azioni di raccolta delle statistiche e log su file del flusso di rete. L'eseguzione di un'azione avviene a seguito della valutazione della condizione di attivazione ad essa associata, amministrata tramite l'utility remota.

3. **Scrittura del dati di rete su file locali**: vengono salvati su file alcuni campi dell'intestazione dei pacchetti che hanno dato una corrispondenza per i pattern di protocollo su cui è stato abilitato il flag di log.

4. **Messaggi Agent-Manager**: i messaggi sono incapsulati nel protocollo CAPWAP. I messaggi inviati dall'Agent sono sempre in risposta a messaggi di richiesta da parte del Manager.

5. **Elaborazione sul server di amministrazione**: il server sulla base delle statistiche e dei dati del log del flusso di rete prende le decisioni su eventuali azioni di correzione da inviare ai nodi amministrati. Questa parte non è stata oggetto del lavoro di tirocinio, i dati vengono solo mostrati sul terminale dell'operatore.

6. **Messaggi Manager-Agent**: i messaggi sono incapsulati nel protocollo CAPWAP. La lista dei messaggi attualmente supportati dal sistema sono specificati nella sezione **4.5 - Formato dei messaggi utilizzati dal modulo L7**.

7. **Salvataggio dei dati nel database**: i log del flusso di rete possono essere salvati in un database per semplificare il lavoro di analisi. Questa parte non è stata oggetto del lavoro di tirocinio, i file di log vengono solo trasferiti sul file system del server di amministrazione.

4.2 WTPL7Agent

Il WTPL7Agent è un demone multi-thread scritto in C che deve essere installato sui WTP della rete. Il suo compito è quello di permettere l'amministrazione remota del sistema di deep packet inspection, la raccolta di statistiche, il log del flusso di rete, la visualizzazione delle connessioni attive e il trasferimento delle informazioni al server centrale di amministrazione.

4.2.1 Funzionalità

L'agent al momento implementa le seguenti funzionalità:

- **Restituzione della lista dei pattern**: viene inviata al server la lista dei pattern di protocollo disponibili sull'Agent. La lista contiene per ogni pattern: il nome, la descrizione, un flag booleano di attivazione e un flag booleano per il log del flusso di rete.
- **Attivazione di un pattern**: l'agent riceve il nome del pattern da attivare e inserisce la nuova regola di netfilter tramite il tool iptables. Restituisce al server un messaggio con l'esito dell'operazione.
- **Disattivazione di un pattern**: l'agent riceve il nome del pattern da disattivare e rimuove la regola da netfilter tramite il tool iptables. Restituisce al server un messaggio con l'esito dell'operazione.
- **Restituzione della lista delle statistiche**: viene inviata al server la lista delle statistiche raccolte per i pattern attivi. La lista riporta per ogni pattern attivo: il nome, il numero di pacchetti, la dimensione in byte dell'insieme dei pacchetti identificati e il timestamp dell'ultimo pacchetto ricevuto.
- **Attivazione del log del flusso di rete**: l'agent riceve il nome del pattern per cui si vuole attivare il log del flusso di rete e imposta il relativo flag di condizione nella struttura dati in memoria del pattern. Restituisce al server un messaggio con l'esito dell'operazione.
- **Disattivazione del log del flusso di rete**: l'agent riceve il nome del pattern per cui si vuole disattivare il log del flusso di rete e imposta il relativo flag di condizione nella struttura dati in memoria del pattern. Restituisce al server un messaggio con l'esito dell'operazione.

- **Restituzione della lista delle connessioni:** viene inviata al server la lista delle connessione attive sul WTP identificate dal modulo di connection tracking di netfilter *ip_ conntrack*. La lista è letta in user space dal file system Proc */proc/1/net/nf_ conntrack*.
- **Restituzione della lista dei file di log del flusso di rete**: viene inviata al server la lista dei file di log del flusso di rete memorizzati sul file system del WTP.
- **Trasferimento di un file di log del flusso di rete**: l'agent riceve il nome del file e avvia il trasferimento del file al server.
- **Cancellazione di un file di log del flusso di rete**: l'agent riceve il nome del file e lo elimina dal file system del dispositivo.

4.2.2 Architettura

In Figura 13 è riportato lo schema di sintesi delle relazioni tra le principali componenti software utilizzate nell'architettura, distinte sulla base dello spazio di indirizzamento di appartenenza (Kernel Space e User Space):

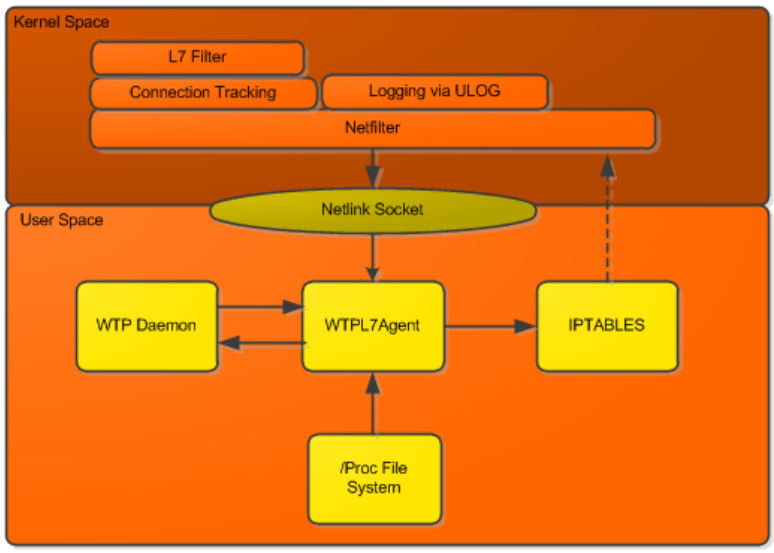


Figura 13: Relazioni tra le componenti software del sistema.

- **WTPL7Agent - WTP Daemon**: i messaggi scambiati tra Agent e Manager vengono incapsultati nel protocollo CAPWAP. Viene utilizzato l'application management server di OpenCAPWAP come proxy per i messaggi (WTPL7Agent <–> WTPDaemon <–> ... Network ... <–> ACDaemon <–> ACL7Manager).

- **WTPL7Agent - Iptables**: la configurazione delle regole per il sistema di Deep Packet Inspection che risiede in Kernel Space viene mediata tramite il comando (che gira nello spazio utente) Iptables.
- **WTPL7Agent - Proc File System**: la lista delle connessioni tracciate dal modulo di Connection Tracking viene letta in User Space dal Proc File System.
- **WTPL7Agent - Netlink Socket**: i pacchetti identificati dal sistema di Deep Packet Inspection vengono copiati in User Space attraverso un socket di tipo Netlink.

Internamente l'Agent si compone di quattro thread. Nella Figura 14 è riportato lo schema delle relazioni tra i thread e le principali strutture dati utilizzate:

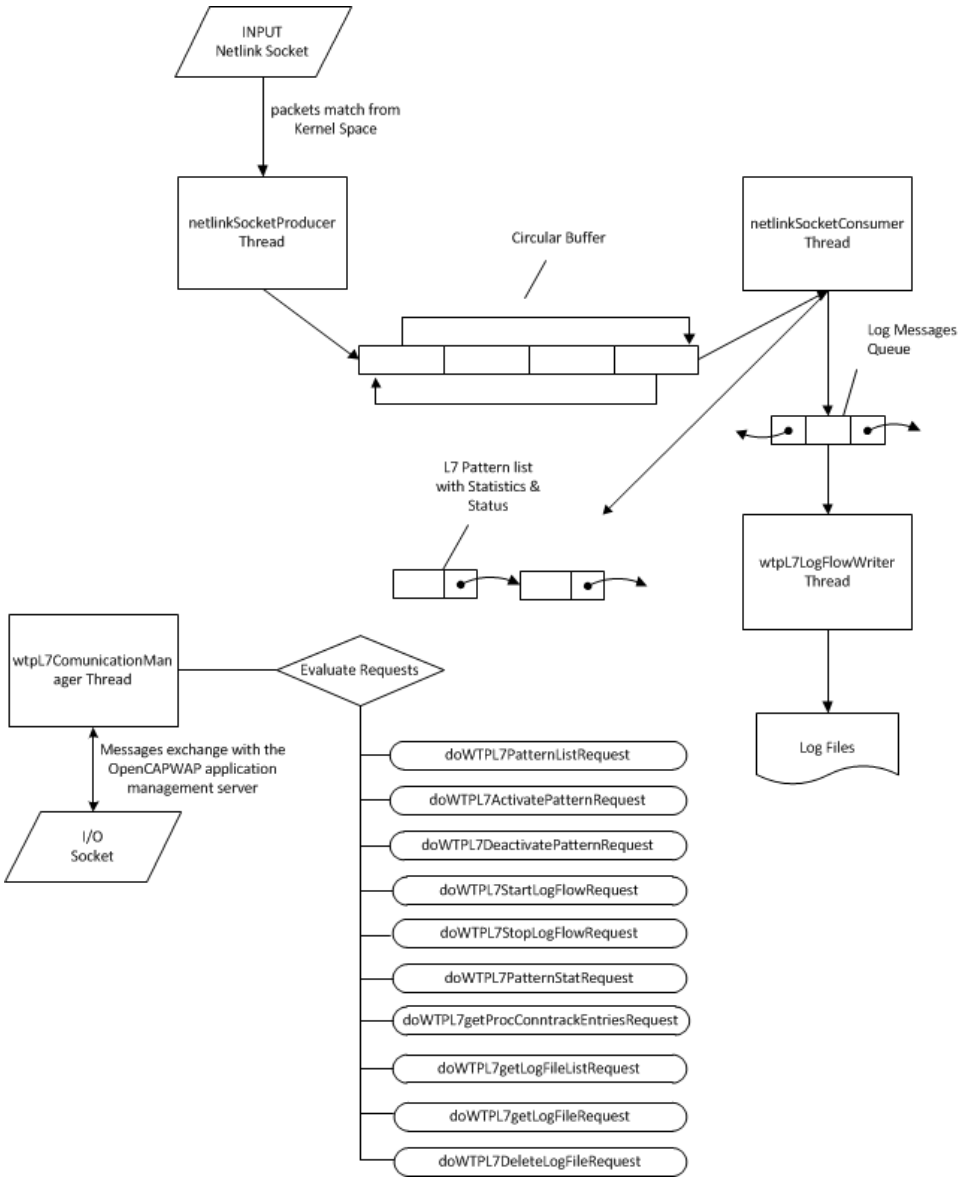


Figura 14: WTPL7Agent: schema dei thread e delle principali strutture dati.

In fase di avvio l'Agent esegue le attività necessarie per operare come demone e offrire i servizi in background, dopodiché inizializza il file di log, analizza il file di configurazione, inizializza il buffer circolare e la lista dei pattern L7

disponibili sul dispositivo. Come ultima operazione, crea i seguenti thread di lavoro:

- **wtpL7ComunicationManager**: questo thread apre un socket TCP sull'indirizzo di rete 127.0.0.1 e si mette in ascolto sulla porta 2911. Il thread resta in attesa delle richieste da parte dell'utility di amministrazione. Le richieste vengono instradate tramite l'application management server di OpenCAPWAP. Il thread valuta il tipo di richiesta, esegue le elaborazioni appropriate e invia la risposta all'Access Controller sempre tramite l'application management server di OpenCAPWAP. Al momento sono supportati i seguenti messaggi di richiesta:

  - **L7_PATTERN_LIST_REQUEST**: viene eseguita la funzione *doWTPL7PatternListRequest(int sockFd)* che si occupa di preparare il messaggio di risposta contenente la lista dei pattern di protocollo disponibili sul dispositivo. Il messaggio di risposta è di tipo **L7_PATTERN_LIST_RESPONSE**. La lista viene creata all'avvio dell'agent e popolata con i dati del file di configurazione dei pattern dove sono riportati i nomi e la descrizione dei pattern. La lista è definita dalle seguenti strutture dati:

```
struct listItem {
    struct l7PatternInfo *item;
    struct listItem *next;
    };
struct l7PatternInfo {
    char *name;
    char *description;
    int isActive;
    int isLogEnabled;
    struct l7Stat *stat;
    };
struct l7Stat {
    unsigned int packet_count;
    unsigned int packet_size;
    long timestamp_s;
    long timestamp_u;
    };
```

  - **L7_ACTIVATE_PATTERN_REQUEST**: viene eseguita la funzione *doWTPL7ActivatePatternRequest(int sockFd, char *request)* che si occupa di inserire la regola per netfilter tramite iptables e di abilitare il flag di attivazione per il pattern richiesto dal manager. Il messaggio di risposta è di tipo **L7_ACTIVATE_PATTERN_RESPONSE**.

- **L7_DEACTIVATE_PATTERN_REQUEST**: viene eseguita la funzione *doWTPL7DeactivatePatternRequest(int sockFd, char *request)* che si occupa di rimuovere la regola per netfilter tramite iptables e di disabilitare il flag di attivazione per il pattern richiesto dal manager. Il messaggio di risposta è di tipo **L7_DEACTIVATE_PATTERN_RESPONSE**.
- **L7_PATTERN_STAT_REQUEST**: viene eseguita la funzione *doWTPL7PatternStatRequest(int sockFd)* che si occupa di restituire la lista delle statistiche dei pattern attivi, raggruppate per pattern. Il messaggio di risposta è di tipo **L7_PATTERN_STAT_RESPONSE**.
- **L7_START_LOG_FLOW_REQUEST**: viene eseguita la funzione *doWTPL7StartLogFlowRequest(int sockFd, char *request)* che si occupa di abilitare il flag per il logging del flusso di rete per il pattern richiesto dal manager. Il messaggio di risposta è di tipo **L7_START_LOG_FLOW_RESPONSE**.
- **L7_STOP_LOG_FLOW_REQUEST**: viene eseguita la funzione *doWTPL7StopLogFlowRequest(int sockFd, char *request)* che si occupa di disabilitare il flag per il logging del flusso di rete per il pattern richiesto dal manager. Il messaggio di risposta è di tipo **L7_STOP_LOG_FLOW_RESPONSE**.
- **L7_SNAPSHOT_CONNTRACK_REQUEST**: viene eseguita la funzione *doWTPL7getProcConntrackEntriesRequest(int sockFd)* che si occupa di leggere e restituire i dati contenuti nel file */proc/1/net/nf_conntrack*. Il messaggio di risposta è di tipo **L7_SNAPSHOT_CONNTRACK**
- **L7_LOG_LIST_REQUEST**: viene eseguita la funzione *doWTPL7getLogFileListRequest(int sockFd)* che si occupa di restituire la lista dei file di log del flusso di rete salvati sul dispositivo. Il messaggio di risposta è di tipo **L7_LOG_LIST_RESPONSE**.
- **L7_LOG_RETR_REQUEST**: viene eseguita la funzione *doWTPL7getLogFileRequest(int sockFd, char *request)* che si occupa di restituire il file di log richiesto dal manager. Il messaggio di risposta è di tipo **L7_LOG_RETR_RESPONSE**.
- **L7_LOG_DELE_REQUEST:** viene eseguita la funzione *doWTPL7DeleteLogFileRequest(int sockFd, char *request)* che si occupa di eliminare il file di log richiesto dal manager. Il messaggio di risposta è di tipo **L7_LOG_DELE_RESPONSE**.

- **netlinkSocketProducer**: questo thread apre un socket netlink con protocollo NETLINK_NFLOG e con il gruppo di multicast specificato nel file di configurazione dell'agent. I dati letti dal socket vengono inseriti direttamente nel buffer circolare senza nessun tipo di elaborazione.

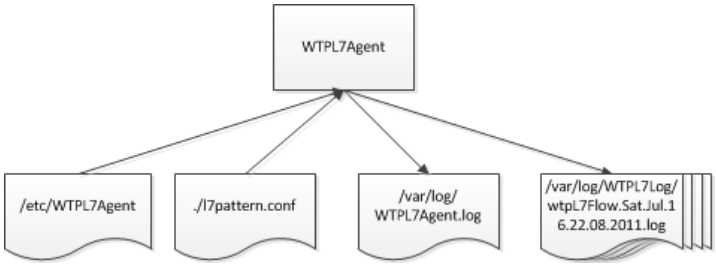
- **netlinkSocketConsumer**: questo thread legge i dati dal buffer circolare ed esegue l'elaborazione dei pacchetti restituiti dal socket netlink. Sulla base delle azioni abilitate per un certo pattern, l'agent può raccogliere

dati a fini statistici e salvare il contenuto dei campi nell'intestazione dei pacchetti catturati nella coda dei messaggi di log.

- **wtpL7LogFlowWriter**: questo thread legge i messaggi dalla coda dei messaggi di log e li scrive in un file di log del flusso di rete.

4.2.3 File di log e di configurazione

L'agent legge i dati da due file di configurazione. Un file contiene i parametri di configurazione e un altro contiene la lista e la descrizione dei pattern disponibili sul dispositivo. L'agent scrive tutte le informazioni di debug su un unico file, mentre crea nuovi file ad ogni attivazione di una sessione di log del flusso di rete.



4.2.3.1 /etc/WTPL7Agent

Il file deve essere copiato nella directory */etc*. I paramentri e i valori di default supportati sono:

- **NFLOG_BUFSIZE_DEFAULT**: Con questo parametro è possibile specificare la dimensione del buffer per il socket netlink di ricezione dei pacchetti identificati in kernel space. Il valore di default è 128KB.
- **NETLINK_MULTICAST_GROUP**: Con questo parametro è possibile specificare il gruppo di multicast netlink sul quale devono essere inviati e ricevuti i pacchetti. Esistono 32 gruppi netlink, sono numerati da 1 a 32. Il valore di default è 32.
- **NETLINK_THRESHOLD**: Con questo parametro è possibile specificare la soglia della coda netlink. Se un pacchetto viene identificato e sono già presenti in coda NETLINK_THRESHOLD pacchetti, allora la coda viene liberata nello spazio utente. Il valore di default è 50.
- **NETLINK_CPRANGE**: Con questo parametro è possibile specificare il numero di byte che devono essere copiati nello spazio utente per ciascun pacchetto identificato. Il valore 0 indica la copia dell'intero pacchetto. Il valore di default è 48 cioè viene copiata solo l'intestazione di ogni pacchetto.

- **L7_AGENT_IP_ADDRESS**: Con questo paramentro è possibile specificare l'indirizzo ip di binding del socket di comunicazione TCP dell'agent. Il valore di default è 127.0.0.1.
- **L7_AGENT_PORT**: Con questo paramentro è possibile specificare la porta di ascolto del socket di comunicazione dell'agent. Il valore di default è 2911.
- **MAX_CLIENT_QUEUE**: Con questo paramentro è possibile specificare la dimensione della coda delle richieste di connessione per il socket di comunicazione dell'agent. Il valore di default è 50.
- **BUFFER_SIZE**: Con questo parametro è possible specificare il numero degli slot utilizzati dal buffer circolare dall'agent. Il valore di default è 100.
- **CONNTRACK_PROC_FILE**: Con questo paramentro è possibile specificare il percorso del file di connection tracking all'interno del file system proc. Il valore di default è */proc/1/net/nf_conntrack*.

4.2.3.2 l7pattern.conf

Il file deve essere copiato nella stessa directory di installazione dell'agent. Il file contiene la lista dei pattern e la loro descrizione. Il nome del pattern e la descrizione si devono trovare sulla stessa riga separati dal carattere di tabulazione:

```
.
.
.
http-rtsp RTSP tunneled within HTTP
http HTTP - HyperText Transfer Protocol - RFC 2616
ident Ident - Identification Protocol - RFC 1413
imap IMAP - Internet Message Access Protocol (A common e-
mail protocol)
imesh iMesh - the native protocol of iMesh, a P2P application -
http://imesh.com
ipp IP printing - a new standard for UNIX printing - RFC 2911
irc IRC - Internet Relay Chat - RFC 1459
jabber Jabber (XMPP) - open instant messenger protocol - RFC
3920 - http://jabber.org
kugoo KuGoo - a Chinese P2P program - http://www.kugoo.com
.
.
.
```

4.2.3.3 /var/log/WTPL7Agent.log

In questo file vengono scritti tutti i messaggi di errore e di debug dell'agent.

4.2.3.4 /var/log/WTPL7Log/wtpL7Flow.[...].log

In questo file vengono salvate alcune informazioni dei pacchetti identificati dal sistema di deep packet inspection. Il nome del file è creato utilizzando il formato *prefisso + timeString + suffisso*:

- **prefisso**: wtpL7log
- **timeString**: stringa restituita dalla funzione *ctime_r()*. I caratteri ":" e spazio vengono sostituiti dal carattere ".".
- **suffisso**: .log

Per i pacchetti TCP vengono salvati i seguenti campi:

- **ulogPacket->timestamp_sec**: Secondi di arrivo del pacchetto.
- **ulogPacket->timestamp_usec**: Microsecondi di arrivo del pacchetto.
- **ulogPacket->prefix**: Prefisso utilizzato per la stampa dei messaggi di log. Per convenzione l'agent imposta come valore del prefisso il nome del pattern attivo. In questo modo è possibile utilizzare il campo per discriminare il tipo dei pacchetti ricevuti.
- **iph->id**: Identificativo univoco di ciascun datagramma inviato da un host.
- **iph->tot_len**: Lunghezza totale del datagramma espressa in bytes.
- **iph->protocol**: Specifica il tipo di protocollo trasportato da IP.
- **iph->saddr**: Indirizzo IP sorgente.
- **iph->daddr**: Indirizzo IP destinazione.
- **tcph->source**: Porta TCP sorgente.
- **tcph->dest**: Porta TCP destinazione.
- **tcph->seq**: Numero di sequenza TCP.

Per i pacchetti UDP vengono salvati i seguenti campi:

- **ulogPacket->timestamp_sec**: Secondi di arrivo del pacchetto.
- **ulogPacket->timestamp_usec**: Microsecondi di arrivo del pacchetto.
- **ulogPacket->prefix**: Prefisso utilizzato per la stampa dei messaggi di log. Per convenzione l'agent imposta come valore del prefisso il nome del pattern attivo. In questo modo è possibile utilizzare il campo per discriminare il tipo dei pacchetti ricevuti.
- **iph->id**: Identificativo univoco di ciascun datagramma inviato da un host.

- **iph->tot_len**: Lunghezza totale del datagramma espressa in bytes.
- **iph->protocol**: Specifica il tipo di protocollo trasportato da IP.
- **iph->saddr**: Indirizzo IP sorgente.
- **iph->daddr**: Indirizzo IP destinazione.
- **udph->source**: Porta UDP sorgente.
- **udph->dest**: Porta UDP destinazione.
- **udph->len**: Lunghezza del pacchetto UDP (header + data).

4.3 ACL7Manager

Il programma ACL7Manager è una semplice utility testuale di amministrazione che permette di inviare i comandi agli agent remoti e di ricevere i dati di risposta. Il programma deve essere installato sull'Access Controller della rete. La logica dell'applicativo è molto semplice: analizza le opzioni passate dalla linea di comando, apre una connessione con l'application management server di OpenCAPWAP in ascolto sulla porta TCP 1235, invia il messaggio di richiesta, riceve il messaggio di risposta, elabora e stampa le informazioni ricevute e chiude la connessione con il server.

4.3.1 Utilizzo

```
ACL7Manager --wtp=''WTP INDEX'' [-hlsfcWwadLSFD]
Esempio di utilizzo: ./ACL7Manager --wtp=0 --activate=''http''
```

4.3.2 Opzioni

Le opzioni al momento supportate sono:

- –**help, -h**: mostra la schermata di help.
- –**list, -l**: richiede la lista e lo stato dei pattern L7 disponibili sul/sui WTP. Non sono richiesti parametri. Invia un messaggio di tipo **L7_PATTERN_LIST_REQUEST** e ne aspetta uno di tipo **L7_PATTERN_LIST_RESPONSE**.
- –**stat, -s**: richiede la lista delle statistiche raccolte per i pattern L7 attivi sul/sui WTP. Non sono richiesti parametri. Invia un messaggio di tipo **L7_PATTERN_STAT_REQUEST** e ne aspetta uno di tipo **L7_PATTERN_STAT_RESPONSE**.
- –**wtpList, -W**: richiede la lista dei WTP registrati con l'AC. Non sono richiesti parametri. Invia un messaggio di tipo **LIST_MSG**.

- –**listLogFile, -f**: richiede la lista dei file dei log del flusso di rete salvati sul/sui WTP. Non sono richiesti parametri. Invia un messaggio di tipo **L7_LOG_LIST_REQUEST** e ne aspetta uno di tipo **L7_LOG_LIST_RESPONSE**.
- –**conntrack, -c**: richiede la lista delle connessione tracciate dal modulo di connection tracking sul/sui WTP. Non sono richiesti parametri. Invia un messaggio di tipo **L7_SNAPSHOT_CONNTRACK_REQUEST** e ne aspetta uno di tipo **L7_SNAPSHOT_CONNTRACK_RESPONSE**.
- –**wtp, -w**: specifica il WTP a cui inviare la richiesta. É richiesto come parametro l'identificativo del WTP. Per inviare la richiesta a tutti i WTP registrati sull'AC è necessario passare come parametro il valore **-1**.
- –**activate, -a**: richiede l'attivazione di un pattern L7 sul/sui WTP. Il parametro deve specificare il nome del pattern di protocollo da attivare. Invia un messaggio di tipo **L7_ACTIVATE_PATTERN_REQUEST** e ne aspetta uno di tipo **L7_ACTIVATE_PATTERN_RESPONSE**.
- –**deactivate, -d**: richiede la disattivazione di un pattern L7 sul/sui WTP. Il parametro deve specificare il nome del pattern di protocollo da disattivare. Invia un messaggio di tipo **L7_DEACTIVATE_PATTERN_REQUEST** e ne aspetta uno di tipo **L7_DEACTIVATE_PATTERN_RESPONSE**.
- –**startLogFlow, -L**: richiede l'attivazione del log del flusso di rete per un pattern di protocollo L7 sul/sui WTP. Il parametro deve specificare il nome del pattern di protocollo per cui attivare il log del flusso di rete. Invia un messaggio di tipo **L7_START_LOG_FLOW_REQUEST** e ne aspetta uno di tipo **L7_START_LOG_FLOW_RESPONSE**.
- –**stopLogFlow, -S**: richiede la disattivazione del log del flusso di rete per un pattern di protocollo L7 sul/sui WTP. Il parametro deve specificare il nome del pattern di protocollo per cui attivare il log del flusso di rete. Invia un messaggio di tipo **L7_STOP_LOG_FLOW_REQUEST** e ne aspetta uno di tipo **L7_STOP_LOG_FLOW_RESPONSE**.
- –**getLogFile, -F**: richiede il trasferimento di un file di log del flusso di rete L7 dal/dai WTP. Il parametro deve specificare il nome del file. Invia un messaggio di tipo **L7_LOG_RETR_REQUEST** e ne aspetta uno di tipo **L7_LOG_RETR_RESPONSE**.
- –**deleteLogFile, -D**: richiede l'eliminazione di un file di log del flusso di rete L7 dal/dai WTP. Il parametro deve specificare il nome del file. Invia un messaggio di tipo **L7_LOG_DELE_REQUEST** e ne aspetta uno di tipo **L7_LOG_DELE_RESPONSE**.

4.4 Integrazione con OpenCAPWAP

L'Application Management Server di OpenCAPWAP permette, attraverso un'interfaccia di comunicazione basata su socket di tipo stream, la creazione di moduli

di estensione del protocollo sotto forma di applicazioni esterne. Questo tipo di integrazione separa in modo netto l'architettura centrale del protocollo dalle sue estensioni offrendo una flessibilità notevole per la creazione di nuovi moduli, ad esempio non devono essere necessariamente scritti in C.

4.4.1 Connessione

Le applicazioni esterne devono essere installate sul server dell'AC ed aprire una connessione sulla porta TCP 1235 in ascolto su un socket locale. Quando un'applicazione ha completato la connessione con l'AC, riceve in risposta un valore intero di 4 byte con il quale viene notificato l'esito della connessione. Il valore 1 indica che la connessione è stata accettata, un valore diverso segnala il rifiuto della connessione, ad esempio perché la coda delle applicazioni è piena.

4.4.2 Messaggi

Le applicazioni esterne possono inviare tre tipi di messaggi all'Access Controller:

- **QUIT**: richiede la chiusura della connessione.
- **LIST**: richiede la lista dei WTP registrati.
- **CONFIGURATION UPDATE**: richiede servizi con un payload specifico ad una applicazione esterna lato WTP.

Il formato dei messaggi è il seguente:

- **QUIT e LIST:**

```
0         7
+-+-+-+-+-+
| cmd_msg |
+-+-+-+-+-+
```

- **CONFIGURATION UPDATE**:

```
0        7           15            23                                          X
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| cmd_msg |  msg_elem |  WTP Index  |         Message Specific Payload        |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

4.4.3 Canale Universale

Le applicazioni esterne che vogliono comunicare tramite OpenCAPWAP devono creare un canale di comunicazione ad hoc come illustrato nella *Guida alla creazione di un canale di comunicazione fra due applicazioni esterne nel protocollo OpenCAPWAP*[17]. Per ogni nuova applicazione deve essere modificato il

codice sorgente di OpenCAPWAP (demone AC e demone WTP) per il supporto dei nuovi valori del *msg_ elem* e del *Message Specific Paylod* di richiesta e di risposta definiti. Questo rallenta il processo di sviluppo dei moduli. Inoltre dal punto di vista sistemistico è necessario pianificare la distribuzione dei nuovi demoni di OpenCAPWAP sia sui dispositivi WTP che sui server AC della rete. La mia soluzione è stata la creazione di un canale "universale" che può essere utilizzato nella fase di sviluppo di un nuovo modulo e in via definitiva da tutte le estensioni che non hanno necessità particolari. E' stato definito un nuovo *msg_ elem* di tipo **MSG_ELEMENT_TYPE_VENDOR_GENERIC**. Tutti i moduli che vogliono utilizzare il canale universale devono solo creare un messaggio di **CONFIGURATION UPDATE** con il nuovo *msg_ elem* e rispettare una convenzione per i valori dei primi 32 bit del *Message Specific Payload* della richiesta e dei primi 48 bit del payload della risposta.

- Primi 32 bit del Message Specific Payload Request:

```
0                   15              31
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| wtp_module_ext_port | request_type  | ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

  - `wtp_module_exp_port: 16 bit con la porta dell'applicazione esterna lato WTP a cui OpenCAPWAP deve inoltrare i messaggi di richiesta.`
  - `request_type: 16 bit con il tipo di richiesta specifico per l'applicazione esterna.`

- Primi 48 bit del Message Specific Payload Response:

```
0               15     47
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| response_type | size  | ...
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

  - `response_type: 16 bit con il tipo di risposta specifico per l'applicazione esterna.`
  - `size: 32 bit con la dimensione del messaggio di risposta.`

4.4.3.1 Generic Vendor Specific Payload

É stato introdotto nel codice di OpenCAPWAP un nuovo tipo di *Vendor Specific Payload* chiamato *Generic*, definito dalla seguente struttura dati:

```
typedef struct {

    unsigned char *request;
    unsigned char *response;
    unsigned short int port;
    int len;
```

```
} CWVendorGenericValues;
```

4.4.3.2 Protocollo di comunicazione AE <–> AC

Descriviamo di seguito le convenzioni per la comunicazione tra l'applicazione esterna lato AC e l'AC stesso.

- **Invio della Richiesta**
  - Preparazione del *Message Specific Payload Request* rispettando la convenzione per i primi 32 bit.
  - Invio della dimensione del *Message Specific Payload Request*.
  - Invio del *Message Specific Payload Request*.
- **Ricezione della Risposta**
  - Ricezione del *Response Header* inviato dell'AC composto da tre interi: *wtpIndex*, *resultCode* e *payloadSize*.
  - Allocazione della memoria sulla base del *payloadSize*.
  - Ricezione della risposta dal socket per *payloadSize* bytes.
  - Parsing e elaborazione del *Message Specific Payload Response*.

4.4.3.3 Protocollo di comunicazione AE <–> WTP

Convenzioni per la comunicazione tra l'applicazione esterna lato WTP e il WTP stesso.

- **Ricezione della Richiesta**
  - Ricezione della dimensione del *Message Specific Payload Request*.
  - Viene allocata la memoria sulla base della dimensione del messaggio.
  - Ricezione del *Message Specific Payload Request*.
  - Parsing e elaborazione della richiesta.
- **Invio della Risposta**
  - Preparazione del *Message Specific Payload Response* rispettando la convenzione per i primi 48 bit.
  - Invio della dimensione del *Message Specific Payload Response*.
  - Invio del *Message Specific Payload Response*.
  - Invio di un valore intero per notificare al demone WTP la chiusura della connessione.

4.5 Formato dei messaggi utilizzati dal modulo L7

I messaggi di richiesta per i WTP inviati tramite l'application management server di OpenCAPWAP devono essere di tipo *Configuration Update Request.* I valori utilizzati dal mio modulo di estensione di OpenCAPWAP sono:

- **cmd_msg**: CONF_UPDATE_MSG.
- **msg_elem**: MSG_ELEMENT_TYPE_VENDOR_GENERIC.
- **WTP Index**: indice del WTP. Il valore **-1** deve essere utilizzato per indicare tutti i WTP della rete.
- **Message Specific Payload**: Il formato dei messaggi specifici è dettagliato nella sezione **Messages Specific Payload per il modulo L7** nel capitolo Appendice.

Capitolo 5

Test e Conclusioni

In questo capitolo vengono descritte le prove del modulo di deep packet inspection per OpenCAPWAP effettuate in laboratorio e in ambiente di produzione. Viene riepilogato brevemente il lavoro svolto con una riflessione sui possibili scenari di applicazione e sugli sviluppi futuri. I dispositivi hardware WTP utilizzati per i test sono quelli descritti nel **paragrafo 3.1**. Lo standard wireless utilizzato è stato l' IEEE 802.11g - 54 Mb/s, 2,4 GHz (compatibile con l'802.11b).

5.1 Test

I test in laboratorio sono stati effettuati presso l'*Università Campus Biomedico di Roma* mentre i test in produzione sono stati effettuati sulla rete di *Provinciawifi* utilizzando un WTP e un AC installati presso la sede del CASPUR.

5.1.1 Test in laboratorio

Sono state eseguite due distinte sessioni di test temporalmente distanti tra loro, in questo modo le prove in laboratorio sono state utilizzate per guidare lo sviluppo del modulo di estensione per OpenCAPWAP variando quando necessario i requisiti funzionali per il software sulla base dei risultati ottenuti in ciascuna sessione di test. Gli obiettivi del primo test sono stati quelli di verificare la sostenibilità del progetto rispetto alle risorse hardware dei WTP in dotazione e di identificare i casi d'uso più significativi da realizzare. L'architettura di rete ricreata appositamente per i test in laboratorio è mostrata nella **Figura 16** ed è composta da due postazioni client, una postazione server, da un dispositivo WTP e da un server AC. Per i test sono state utilizzate sessioni **FTP** per il trasferimento di file di grandi dimensioni e streaming video tramite il tool **VLC** in modalità client-server.

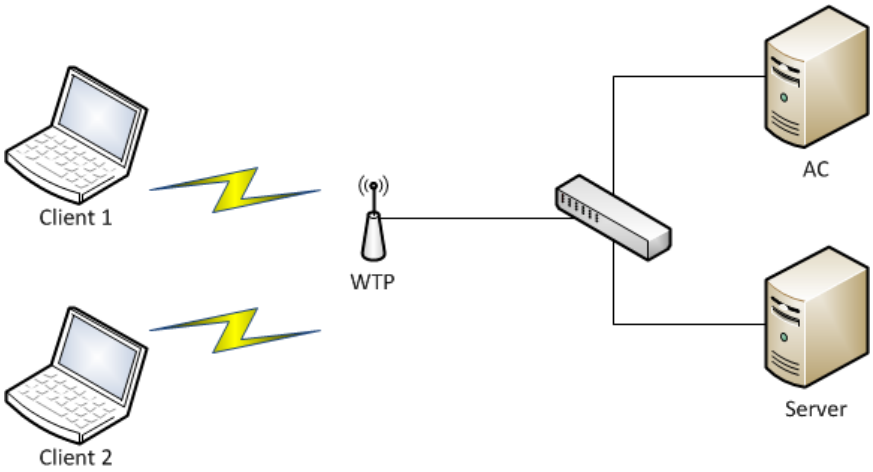


Figura 16: Architettura di rete per i test in laboratorio.

5.1.1.1 Prima sessione di test

Sul dispositivo WTP sono stati attivati i pattern di riconoscimento L7 **FTP** e **RTP** (Real-Time Transport Protocol), il numero di pacchetti e il volume del traffico identificato dal modulo è stato coerente con il traffico generato dai client e dal server. E' stato necessario modificare leggermente l'espressione regolare del pattern RTP per permettere l'identificazione dello streaming video di VLC. Nella prima versione l'agent è stato progettato come una semplice sonda di raccolta dati mantenuti in memoria finché non richiesti dall'utility di amministrazione. Nel caso in cui il buffer circolare dell'agent fosse diventato saturo, in ottica di un servizio **best effort,** i dati più vecchi venivano sovrascritti. L'utility di amministrazione richiedeva i dati all'agent ad intervalli regolari e una volta ricevuti li elaborava e li inseriva in un database SQLite. I dati erano composti da un singolo messaggio netlink di tipo multipart contenente un numero di pacchetti al più pari al valore **NETLINK_THRESHOLD (** 4.2.3.1 a pagina 41) e ciascun pacchetto era di dimensione pari a **NETLINK_CPRANGE** (4.2.3.1 a pagina 41). Nonostante le risorse del WTP fossero sostanzialmente adeguate, la soluzione di inviare tutti i pacchetti identificati da L7 al server di amministrazione si è rivelata non sostenibile anche perché nello scenario in cui il gateway internet viene colocato con il server AC si avrebbe un invio doppio di pacchetti (**N.B.** solo per quelli identificati da L7).

5.1.1.2 Seconda sessione di test

Sulla base dei feedback ricevuti dalla prima sessione di test, l'agent è stato modificato per eseguire direttamente in loco le elaborazioni per la raccolta di

statistiche e, solo su richiesta, il logging del flusso di rete dei pacchetti identificati dal modulo di deep packet inspection. Per ciascun pattern sono stati definiti dei contatori per tracciare il numero di pacchetti e la loro dimensione. Inoltre, è stata installata direttamente sui dispositivi WTP un'istanza del database SQLite in modo tale che l'agent potesse salvare direttamente il flusso di rete nel db, attraverso le API della libreria *sqlite3*. La dimensione dei paccheti copiati tramite il modulo **ulog** è stata impostata per default a 48 byte (solo l'intestazione dei pacchetti). L'utility di amministrazione è diventata interattiva, da linea di comando, con la possibilità di attivare e disattivare i pattern L7, il log del flusso di rete e di eseguire delle query SQL remote per la richiesta selettiva del flusso di rete. Sono stati quindi replicati i test eseguiti durante la prima sessione. Anche in questo caso il numero di pacchetti e il volume del traffico identificati dal modulo L7 sono stati coerenti con il traffico generato dai client e dal server. Dal punto di vista dell'utilizzo delle risorse c'è stato, naturalmente, un aumento dell'utilizzo della CPU. Il comando **top** riportava un utilizzo attorno al 30% mentre con la vecchia versione si registravano picchi al più del 10%. Il vero problema è stato però a discrepanza significativa delle statistiche raccolte dall'agent rispetto a quelle di netfilter. Si è registrata infatti una differenza di circa il 40% sul numero dei pacchetti identificati, troppo elevata anche in un ottica di servizio best effort. La successiva analisi ha evidenziato che il collo di bottiglia erano le operazioni di I/O. Le query di scrittura dei dati nel database e la scrittura delle informazioni di log su file venivano eseguite nello stesso thread di lettura e di elaborazione dei messaggi netlink presenti nel buffer circolare dell'agent, di conseguenza la velocità con cui veniva liberato il buffer circolare nello spazio utente non era più sufficiente per svuotare il buffer del socket netlink in kernel space ed era questo che provocava la perdita dei pacchetti. La soluzione è stata quella di eliminare SQLite e la relativa funzione per l'eseguzione delle query remote, di definire un sottoinsieme significativo dei campi di interesse per i pacchetti tcp e udp identificati da L7, di creare una coda di messaggi condivisa e un nuovo thread per la scrittura delle informazioni di log e del flusso di rete su file. La gestione effettiva delle scritture è stata delegata al sistema operativo, rinunciando all'utilizzo di operazioni esplicite di *flush*.

5.1.2 Test in produzione

Durante la riunione propedeutica tenuta al CASPUR per pianificare la messa in produzione del mio agent è stato identificato un ostacolo tecnico relativo al tipo di configurazione richiesta per le interfacce dei dispositivi WTP. Nella fase di sviluppo e in quella di test, per permettere a netfilter di vedere il traffico internet delle stazioni client (wireless) verso il gateway internet (cable), le interfacce wireless sono state configurate su una classe di rete diversa rispetto all'interfaccia ethernet del dispositivo WTP, con conseguente utilizzo del NAT. Questa configurazione è incompatibile con quella utilizzata sulla rete di *provinciawifi* dove le interfacce dei dispositivi sono configurate in *bridge*, per il supporto di alcune funzionalità avanzate, come ad esempio il roaming delle connessioni tra gli hotspot. La soluzione è stata quella di compilare una nuova immagine del firmware

abilitando il **bridge-netfilter** per permettere a netfilter di vedere i pacchetti IPv4, IPv6 e ARP scambiati in bridge, anche se incapsulati in 802.1Q VLAN. Per il test è stato utilizzato un solo dispositivo WTP installato presso la sede del CASPUR; sul quale veniva annunciata la rete di *provinciawifi*. Sono stati attivati i seguenti pattern L7: **flash**, **smtp**, **skypetoskype**, **pop3**, **jabber**, **http**, **dns**. Il sistema di monitoraggio è rimasto correttamente in funzione per sette giorni. Le statistiche raccolte sul dispositivo WTP tramite il tool iptables e tramite il mio agent sono di seguito riportate:

```
Chain FORWARD (policy ACCEPT 15M packets, 9588M bytes)  pkts bytes
2089K  285M       state NEW
76004 8081K ULOG  LAYER7 l7proto dns ULOG copy_range 48 nlgroup 32 prefix ‘dns’ ...
1162K 1108M ULOG  LAYER7 l7proto flash ULOG copy_range 48 nlgroup 32 prefix ‘flash’ ...
311   55809 ULOG  LAYER7 l7proto jabber ULOG copy_range 48 nlgroup 32 prefix ‘jabber’ ...
31924 4562K ULOG  LAYER7 l7proto skypetoskype ULOG copy_range 48 nlgroup 32 prefix ‘skypetoskype
4761  4660K ULOG  LAYER7 l7proto pop3 ULOG copy_range 48 nlgroup 32 prefix ‘pop3’ ...
370   160K  ULOG  LAYER7 l7proto smtp ULOG copy_range 48 nlgroup 32 prefix ‘smtp’ ...
6745K 6425M ULOG  LAYER7 l7proto http ULOG copy_range 48 nlgroup 32 prefix ‘http’ ...
```

mentre l’output delle statistiche raccolte dall’agent è il seguente:

```
Ricevuti 12 byte di response headerpayloadSize: 1114, wtpIndex: 0, resultCode: 0
Ricevuti 1114 byte di payload Tipo messaggio 7 Stat pattern size: 1114 Num pattern: 7
################################
Pattern name: flash
Pattern stat count: 1138758
Pattern stat pkt size: 51844062
Pattern stat time_s: 1306448313
Pattern stat time_u: 785003
################################
Pattern name: smtp
Pattern stat count: 365
Pattern stat pkt size: 17216
Pattern stat time_s: 1306423958
Pattern stat time_u: 324419
################################
Pattern name: skypetoskype
Pattern stat count: 31253
Pattern stat pkt size: 1483706
Pattern stat time_s: 1306501160
Pattern stat time_u: 715586
################################
Pattern name: pop3
Pattern stat count: 4673
Pattern stat pkt size: 214574
```

```
Pattern stat time_s: 1306499959
Pattern stat time_u: 363895
##############################
Pattern name: jabber
Pattern stat count: 301
Pattern stat pkt size: 14216
Pattern stat time_s: 1306443912
Pattern stat time_u: 9770
##############################
Pattern name: http
Pattern stat count: 6610210
Pattern stat pkt size: 314446857
Pattern stat time_s: 1306501156
Pattern stat time_u: 331465
##############################
Pattern name: dns
Pattern stat count: 74156
Pattern stat pkt size: 3559433
Pattern stat time_s: 1306501161
Pattern stat time_u: 348634
```

Durante la settimana di test non sono stati segnalati particolari disservizi o problemi di connettività da parte degli utenti connessi al WTP di test.

5.2 Conclusioni e sviluppi futuri

In questo lavoro di tirocinio è stato realizzato un sistema di identificazione distribuito di pacchetti tramite Deep Packet Inspection per reti WLAN centralizzate, amministrate utilizzando il protocollo CAPWAP. Sono stati utilizzati esclusivamente strumenti software open source. L'utilizzo della DPI per la classificazione del traffico consente di avere informazioni migliori sul traffico della propria rete che permettono di prendere le decisioni più appropriate in fase di controllo ed amministrazione. I gestori delle reti devono inoltre far fronte a questioni di natura non strettamente tecnica ma guidate da opportunità commerciali o vincoli di legge come ad esempio: fatturazione sulla base dei servizi utilizzati, pubblicità personalizzata, attuazione di filtri e restrizioni sui contenuti per motivi di copyright. La soluzione software sviluppata si è dimostrata utilizzabile in un contesto operativo reale. Le avanzate risorse hardware dei dispotivi WTP in dotazione sono riuscite a supportare il carico computazionale aggiuntivo dato dal modulo L7 di deep packet inspection. Sulla base di queste premesse sfruttare l'architettura centralizzata delle WLAN e il protocollo CAPWAP come framework di gestione e coordinamento di un sistema di DPI distribuito può risultare una valida alternativa rispetto a un sistema di DPI tradizionale. Infatti da una parte viene analizzato il traffico dati direttamente sui punti di accesso periferici della rete con una banda teorica di 54Mbs mentre dall'altra

viene analizzato il traffico dati su reti ad alta velocità con una banda nell'ordine dei Gigabit, attività per cui sono necessari dispositivi dedicati caratterizzati da costose architetture multi-core. Le caratteristiche più importanti di questo modulo di DPI per OpenCAPWAP sono la semplicità e l'indipendenza. Semplicità nella gestione del sistema e indipendenza dalla tecnologia di deep packet inspection. L'utilizzo del socket netlink come interfaccia per il trasferimento dei pacchetti identificati rende automaticamente l'agent predisposto a lavorare con qualsiasi motore di DPI purché questo possa inviare dati su un socket di tipo netlink. Tra i motori di DPI rilasciati con licenza open source, oltre a **l7-filter,** i più interessanti sono:

- **IPP2P**[22] specializzato per il riconoscimento del traffico peer-to-peer.
- **OpenDPI**[23] motore DPI rilasciato open source dalla società ipoque specializzata nello sviluppo di soluzioni avanzate per la gestione del traffico di rete.

Tralasciando i problemi di prestazioni è possibile avere più DPI engine attivi contemporaneamente. Ad esempio L7-filter, IPP2P e OpenDPI possono convivere e inviare i dati all'agent per la raccolta delle statistiche tramite un socket netlink.

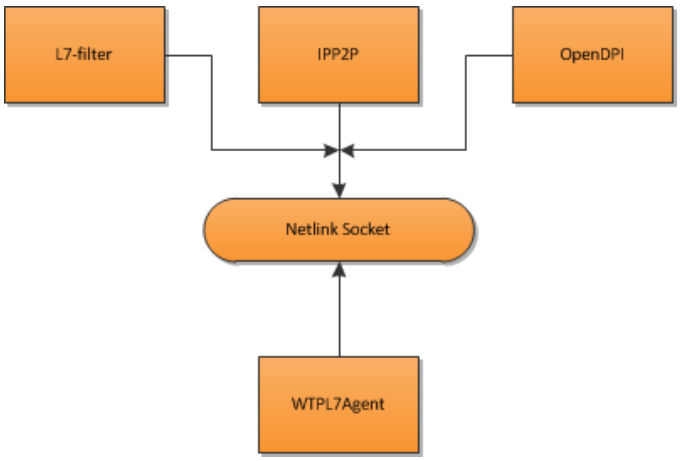


Figura 17: Architettura con DPI engine multipli.

Il modulo di L7 filter, e in generale i sistemi di deep packet inspection, poiché analizzano i dati contenuti nei pacchetti IP utilizzano molto la CPU e questo può introdurre problemi di latenza e perdita di pacchetti nella rete. Ovviamente la percentuale di CPU utilizzata è variabile e dipende sia dalla quantità e sia dal tipo di traffico che passa sul dispositivo. Poiché in una rete WLAN possono essere presenti tanti dispositivi WTP eterogenei per architettura, frequenza del processore e quantità di memoria, sarebbe interessante valutare i pattern disponibili per L7 su ciascun dispositivo. Il risultato sarà composto da due insiemi di

pattern per dispositivo: l'insieme *lightweight* e l'insieme *heavy*. Il primo conterrà tutti i pattern che una volta attivati sul dispositivo WTP non comportano un consumo eccessivo di CPU, mentre il secondo sarà costituito da tutti i pattern che non è possibile attivare sul WTP perché non sostenibili dalle risorse del dispositivo. Come conseguenza è possibile pensare di distribuire ulteriormente la DPI tra WTP e AC:

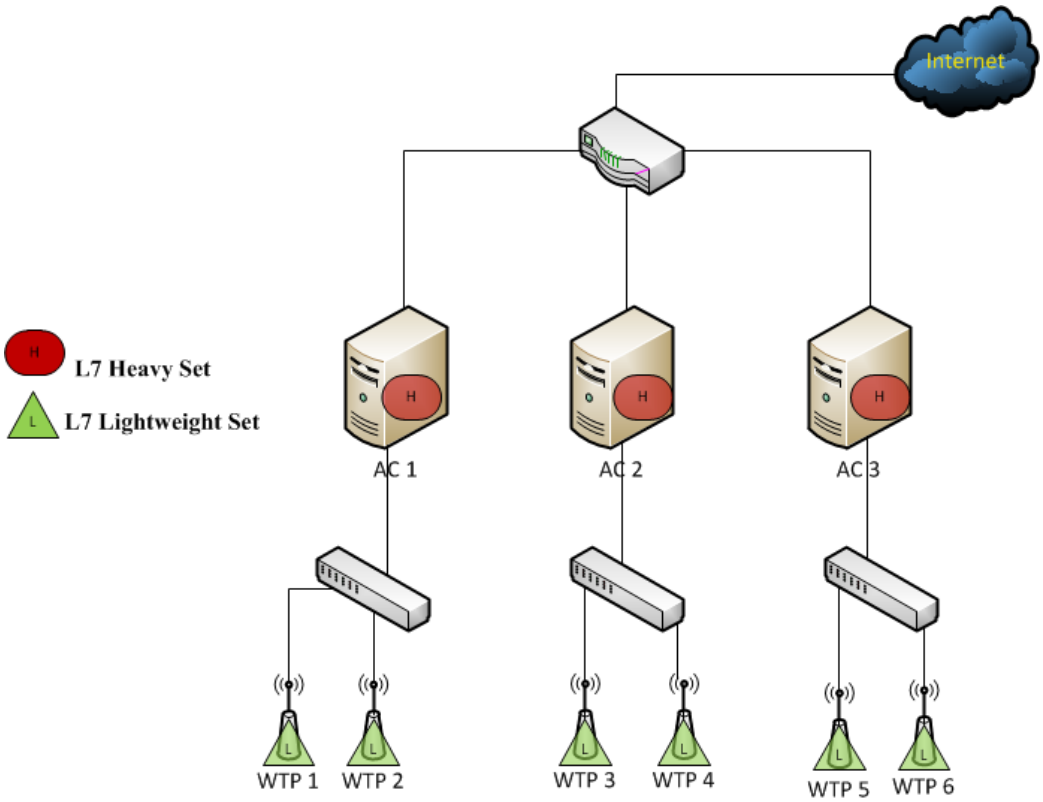


Figura 18: Architettura distribuita tra WTP e AC della rete.

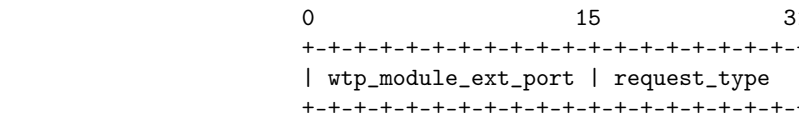
Nel caso in cui l'architettura preveda un solo Access Controller e questo coincida con il gateway Internet, si torna ad una architettura di DPI simile a quella tradizionale dove una parte dell'analisi del traffico viene fatta direttamente sul gateway. Alla luce dei risultati ottenuti si intende continuare la sperimentazione e lo sviluppo del progetto installando il sistema di analisi su tutti i WTP della rete di *provinciawifi*.

Appendice

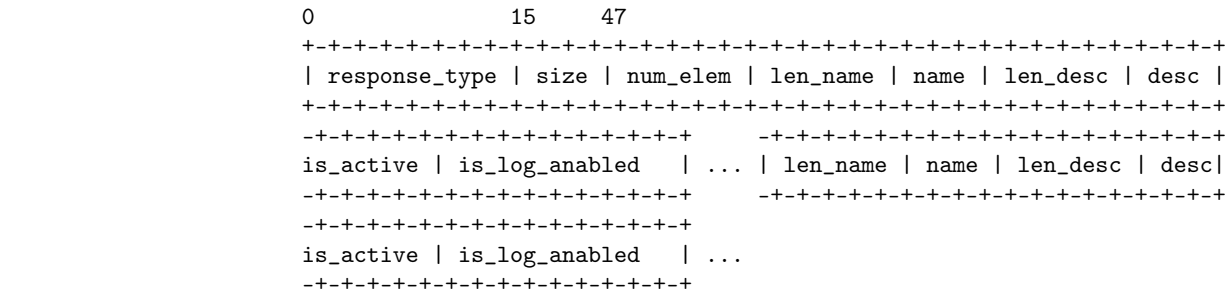
Messages Specific Payload per il modulo L7

L7_PATTERN_LIST

- Message Specific Payload Request. Request type: **L7_PATTER_LIST_REQUEST**

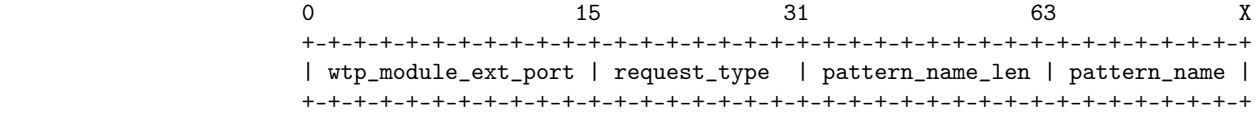


- Message Specific Payload Response. Response type: **L7_PATTER_LIST_RESPONSE**

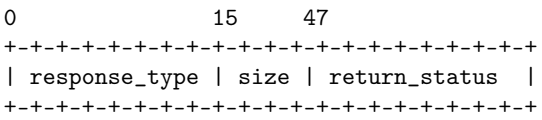


L7_ACTIVATE_PATTERN

- Message Specific Payload Request. Request type: **L7_ACTIVATE_PATTERN_REQUEST**

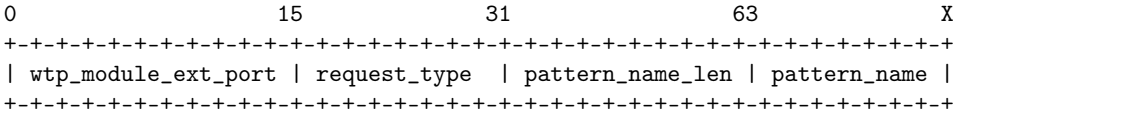


- Message Specific Payload Response. Response type: **L7_ACTIVATE_PATTERN_RESPONSE**

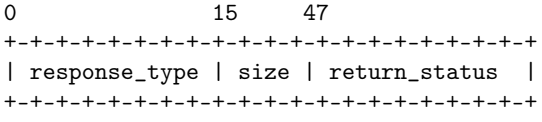


L7_DEACTIVATE_PATTERN

- Message Specific Payload Request. Request type: **L7_DEACTIVATE_PATTERN_REQUEST**

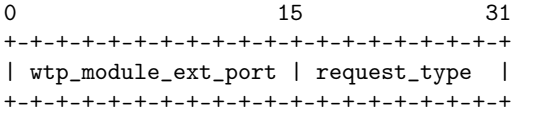


- Message Specific Payload Response. Response type: **L7_DEACTIVATE_PATTERN_RESPONSI**

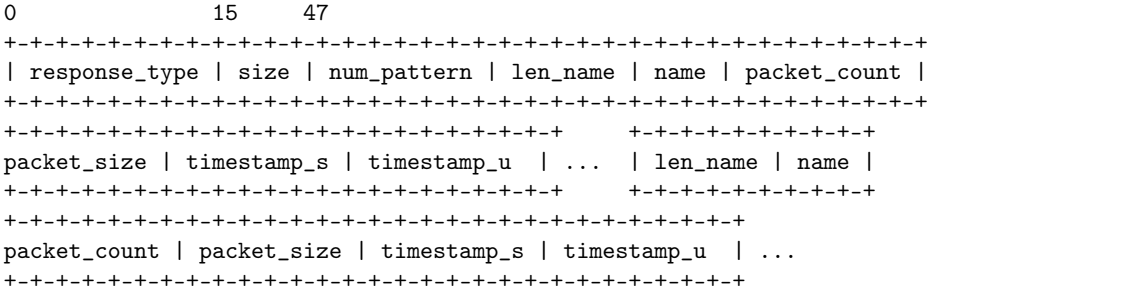


L7_PATTERN_STAT

- Message Specific Payload Request. Request type: **L7_PATTERN_STAT_REQUEST**

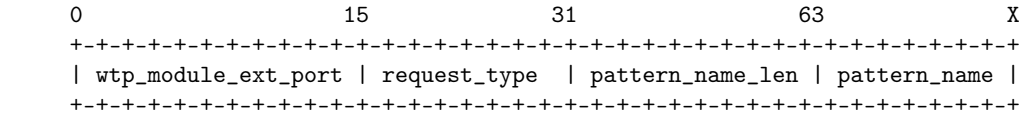


- Message Specific Payload Response. Response type: **L7_PATTERN_STAT_RESPONSE**

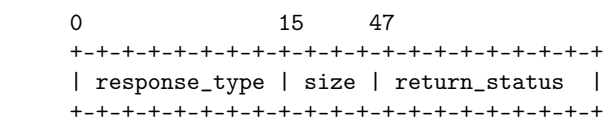


L7_START_LOG_FLOW

- Message Specific Payload Request. Request type: **L7_START_LOG_FLOW_REQUEST**

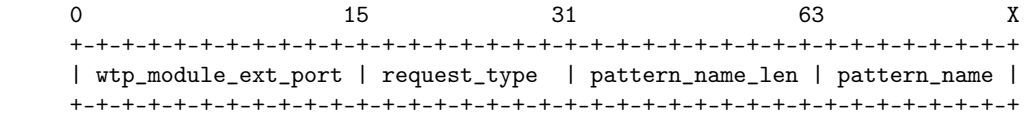


- Message Specific Payload Response. Response type: **L7_START_LOG_FLOW_RESPONSE**

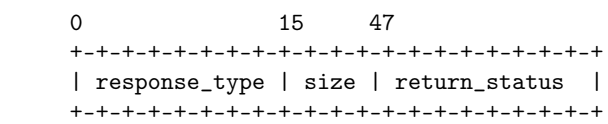


L7_STOP_LOG_FLOW

- Message Specific Payload Request. Request type: **L7_STOP_LOG_FLOW_REQUEST**

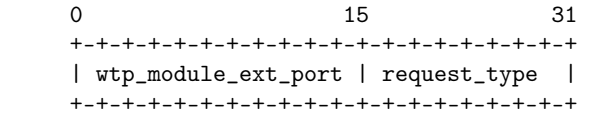


- Message Specific Payload Response. Response type: **L7_STOP_LOG_FLOW_RESPONSE**

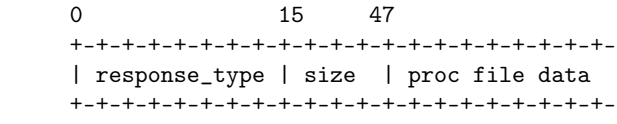


L7_SNAPSHOT_CONNTRACK

- Message Specific Payload Request. Request type: **L7_SNAPSHOT_CONNTRACK_REQUEST**

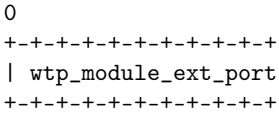


- Message Specific Payload Response. Response type: **L7_SNAPSHOT_CONNTRACK_RESPON**

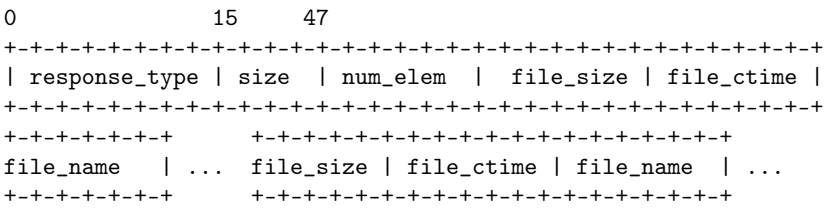


L7_LOG_LIST

- Message Specific Payload Request. Request type: **L7_LOG_LIST_REQUEST**

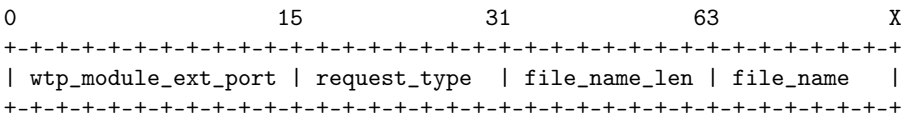


- Message Specific Payload Response. Response type: **L7_LOG_LIST_RESPONSE**

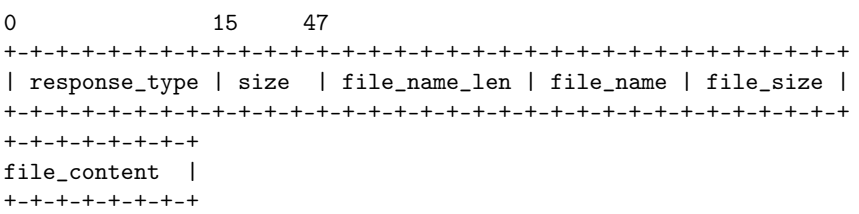


L7_LOG_RETR

- Message Specific Payload Request. Request type: **L7_LOG_RETR_REQUEST**

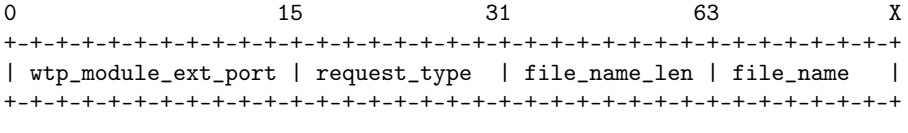


- Message Specific Payload Response. Response type: **L7_LOG_RETR_RESPONSE**



L7_LOG_DELE

- Message Specific Payload Request. Request type: **L7_LOG_DELE_REQUEST**



- Message Specific Payload Response. Response type: **L7_LOG_DELE_RESPONSE**

```
0               15     47
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
| response_type | size | return_status  |
+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+-+
```

Bibliografia

[1] M. Bernaschi, F. Cacace, A. Davoli, D. Guerri, M. Latini, L. Vollero: An Architecture for Monitoring and Management of WiFi Hot-Spots based on the CAPWAP protocol.

[2] http://sourceforge.net/projects/open-capwap/

[3] http://www.openssl.org/

[4] http://www.openwrt.org/

[5] D.Capitella: Disegno e implementazione di un sistema di aggiornamento per OpenCAPWAP.

[6] E.Agostini: Progettazione e realizzazione di uno sniffer di traffico VoIP per OpenCAPWAP.

[7] http://www.pcengines.ch/alix3d1.htm

[8] http://madwifi.org/

[9] http://l7-filter.sourceforge.net/

[10] http://l7-filter.sourceforge.net/protocols

[11] http://l7-filter.sourceforge.net/technicaldetails

[12] http://l7-filter.sourceforge.net/V8regex

[13] http://www.wireshark.org/

[14] man iptables

[15] http://www.netfilter.org/documentation/HOWTO/packet-filtering-HOWTO.html

[16] N. Horman: Understanding and programming with netkink sockets

[17] E. Agostini: Guida alla creazione di un canale di comunicazione fra due applicazioni esterne nel protocollo OpenCAPWAP.

[18] W. Richard Stevens: TCP Illustrated, Volume 1.

[19] W. Richard Stevens, Bill Fenner, Andrew M. Rudoff: UNIX Network Programming Volume 1.

[20] Kay A. Robbins, Steven Robbins: Unix System Programming

[21] L. Gheorghe: Designing and implementing Linux firewall and QoS.

[22] http://www.ipp2p.org/

[23] http://www.opendpi.org/