UNIVERSITÀ DEGLI STUDI DI TRENTO

Dipartimento di Ingegneria e Scienza dell'Informazione

Corso di Laurea in Informatica

Elaborato finale

Tecniche di classificazione di pacchetti con deep packet e flow inspection per IoT

Supervisore

Roberto Passerone

Laureando

Gennaro Cirillo

Anno accademico 2016/2017

Indice

Sommario **2**

1 Introduzione **3**

2 Deep Packet Inspection **3**
2.1 Generazione dei pacchetti . . . 4
2.1.1 Cattura dei pacchetti: Wireshark . . . 4
2.2 Espressioni regolari e flex . . . 5
2.3 Il protocollo MQTT . . . 7
2.3.1 Riconoscimento MQTT . . . 8
2.4 Il protocollo COAP . . . 9
2.4.1 Riconoscimento COAP . . . 10
2.5 Il protocollo AMQP . . . 12
2.5.1 Riconoscimento AMQP . . . 12
2.6 Risultati . . . 13

3 Deep Flow Inspection **15**
3.1 Tstat . . . 15
3.2 Weka . . . 17
3.2.1 Gli algoritmi di apprendimento . . . 19
3.3 Alberi decisionali . . . 20
3.4 Clusterizzazione . . . 20
3.4.1 K-means . . . 21
3.5 Generazione dei flussi . . . 21
3.6 Svolgimento e Parametri utilizzati . . . 22
3.7 Risultati . . . 22

Bibliografia **23**

Sommario

Oggigiorno, internet, domina parte della nostra vita, utilizziamo la rete per comunicare, controllare mail e navigare. Da qui siamo giunti all'esigenza e all'opportunità di collegare alla rete gli oggetti. Possiamo immaginare di rendere interattive delle cose apparentemente inanimate, ad esempio, di collegare mediante dei sensori, automobili, radio e telecamere. Basta guardarci attorno, per capire che nel futuro utilizzeremo sempre di più queste tecnologie. È sotto gli occhi di tutti ormai, sono anni che si parla di automobili in grado di guidare autonomamente senza l'ausilio di un intervento umano, oppure, basti pensare che col solo utilizzo del nostro smartphone è possibile comandare oggetti a centinaia di kilometri di distanza. Possiamo immaginare un mondo, dove sarà possibile rendere intelligente un frigorifero che a sua volta ci dirà quali alimenti ha al proprio interno e di ordinare online nel caso mancasse qualcosa, o una sveglia in grado di suonare prima in caso di traffico. L'obiettivo degli oggetti connessi è, in generale, quello di semplificarci la vita, rendendo automatici i processi o mettendoci a disposizione informazioni su cose che ci circondano. Con lo sviluppo di nuove tecnologie e infrastrutture di comunicazione, questa visione di internet sarà sempre più realtà. La nuova tecnologia 5G che nei prossimi anni andrà ad implementare il parco delle reti attualmente disponibili, è stata pensata, per un futuro utilizzo da parte dei dispositivi IoT. Una delle novità introdotte, sono per l'appunto i cosiddetti protocolli dell'internet delle cose. Questi protocolli, hanno la funzione di far comunicare gli oggetti con il mondo esterno, essi viaggiano in rete congiuntamente a protocolli cosiddetti tradizionali. Lo scopo di questo lavoro è quello di riconoscere, filtrare e smistare i pacchetti che usufruiscono di protocolli tradizionali, rispetto a quelli che utilizzano protocolli dell'internet delle cose. Questo tipo di ispezione dei pacchetti, ha come scopo principale, l'identificare anomalie dei protocolli, intrusioni e propagazione di virus, per ottimizzare il traffico sulle reti e per raccogliere dati statistici sull'utilizzo delle reti stesse. In futuro, si potrebbero prospettare utilizzi da parte degli internet service provider per effettuare una tariffazione diversificata in base ai servizi che questi offrono, andando ad analizzare il traffico e cercando di capire che utilizzo si stia facendo della rete, potendo così addebbitare costi differenziati. Verranno analizzati i criteri e metodologie di distinzione dei vari pacchetti, il loro funzionamento e i vantaggi di queste soluzioni. Inoltre, verrà analizzata una soluzione in grado di riconoscere i flussi senza andare a controllare i bytes all'interno dei singoli pacchetti ma utilizzando un algoritmo di machine learning. Le tecniche messe sotto i riflettori sono rispettivamente, il deep packet inspection e il deep flow inspection. Si tratta di due tecniche per ispezionare i pacchetti, in generale il deep packet inspection, controlla all'interno del segmento intere stringhe di bytes, mentre il deep flow inspection tramite l'analisi dei flussi di rete individua le comunicazioni IoT e non.
Per generare i pacchetti da ispezionare, sono state utilizzate, inizialmente delle librerie specifiche per ogni tipo di protocollo, successivamente, data l'esigenza di avere un gran numero di flussi abbiamo utilizzato un software in grado di simulare dei sensori, così da avere appunto un buon numero di campioni per la nostra analisi. Per quanto riguarda il software, per le espressioni regolari si è utilizzato uno scanner lessicale. Per generare le statistiche utilizzate nel deep flow inspection si è utilizzato un software di analisi delle reti, che, dato in ingresso un file di cattura dei pacchetti genera per ogni flusso delle stime dei parametri. L'analisi del dataset è stata sottoposta a un software di machine learning, contenente numerosi algoritmi pronti da applicare e in grado di fornirci una metodologia di riconoscimento. Infine verranno analizzate sotto il punto di vista delle performance le due tecniche confrontando rispettivamente i pro e i contro.

1 Introduzione

La prima metodologia utilizzata è quella del packet inspection. Sono stati analizzati tre tipi di protocolli, rispettivamente AMQP (Advanced Message Queuing Protocol), MQTT (Message Queue Telemetry Transport), COAP (Constrained Application Protocol). Con l'ausilio di librerie specifiche per i singoli protocolli siamo stati in grado di generare i pacchetti, che in seguito abbiamo catturato con il software "Wireshark". Per ogni tipo di protocollo ne abbiamo analizzato l'header andando così a marcare le prime differenze in termini di contenuti. Successivamente abbiamo provveduto a creare delle macchine a stati che sono state in grado di riconoscere i singoli pacchetti. Una volta avuto ben chiare le differenze fra i vari protocolli siamo giunti alla creazione di espressioni regolari in grado di idenficare le singole peculiarità. Un'espressione regolare è una sequenza di simboli (quindi una stringa) che identifica un insieme di stringhe[15]. Per la scrittura di queste espressioni ci siamo avvalsi di un software di analisi lessicale "flex", che, dato in ingresso uno stream di caratteri esegue delle azioni. Dopo esserci soffermati sul packet inspection, abbiamo messo sotto i riflettori altri due problemi, il primo, in cui ci sia l'eventualità che questi pacchetti siano criptati e quindi non possano essere ispezionati al loro interno, il secondo, invece, quando dei dispositivi IoT utilizzano protocolli non definiti per essi, ma per altri scopi. In questi casi non è possibile fare un analisi del pacchetto, ma si va ad analizzare l'andamento dei flussi così da riconoscerne il tipo. Per fare tutto ciò abbiamo raccolto dei dati riguardanti comunicazioni IoT e comunicazioni tradizionali. Il traffico IoT è stato generato mediante un software di simulazione di sensori "mimic". Grazie a questi sensori simulati siamo stati in grado di raccogliere i flussi necessari per la nostra analisi. Raccolti i flussi, tramite il software tstat abbiamo generato le statistiche per le varie comunicazioni. Abbiamo così evidenziato caratteristiche come la lunghezza dei pacchetti, il tempo di arrivo e di comunicazione. Con l'ausilio di un software "weka", abbiamo diviso in cluster le feature e tramite l'algoritmo "j48" si è generato un albero decisionale in grado di analizzare i flussi e di condurci ad un risultato finale.

2 Deep Packet Inspection

Con l'avanzare dell'avanguardia tecnologica e lo sviluppo dell'internet delle cose, siamo giunti a dover progettare dispositivi che permettano il corretto funzionamento di tutte le apparecchiature di ultima generazione che giornalmente utilizziamo. La Deep Packet Inspection (DPI), ha lo scopo di inviduare e filtrare all' interno di una rete il contenuto dei pacchetti in base a criteri ben precisi. Questo tipo di ispezione dei pacchetti è stata presa in considerazione principalmente dagli Internet Service Provider per identificare anomalie dei protocolli, intrusioni e propagazione di virus, per ottimizzare il traffico sulle reti e per raccogliere dati statistici sull'utilizzo delle reti stesse. Inoltre nell'ultimo periodo aziende di tutti i settori economici dato l'aumento del consumo di banda si stanno attrezzando con dispositivi che permettono di analizzare la tipologia di traffico e di definire priorità sull'utilizzo della banda[4]. Nel dettaglio, molte soluzioni, oggi, richiedono l'applicazione di metodologie di Packet Inspection come, ad esempio, sistemi di rilevazione e prevenzione delle intrusioni di rete oppure nel caso in cui gli switch e i firewall di livello applicativo forniscano contenuti filtrati, bilanciati e monitorati[14][24]. Spesso nel deep packet inspection è necessaria la scansione dei singoli bytes, anche se nel nostro caso con la creazione di espressioni regolari abbiamo generato dei pattern in grado di riconoscere intere stringhe. I casi in cui abbiamo dovuto scansionare i singoli bytes sono stati per l'appunto quando abbiamo avuto casi di header variabili senza delle regole fisse. Infatti siamo dovuti ricorrere a dei contatori che venivano decrementati scansionando i singoli byte.

2.1 Generazione dei pacchetti

In primo luogo abbiamo analizzato le peculiarità e caratteristiche dei singoli protocolli IoT. Abbiamo generato i singoli pacchetti ed ispezionato gli header in modo tale da marcarne le differenze. La generazione dei pacchetti è avvenuta per mezzo di codice python e grazie a librerie specifiche per ogni singolo tipo di protocollo[25]. Per quanto riguarda la generazione di pacchetti MQTT abbiamo utilizzato la libreria paho, essa ci fornisce numerose funzioni fra cui quella di creare un client tramite una propria classe. Inoltre, grazie ad un altro metodo è possibile creare la connessione e il messaggio da inviare. Infine il programma si collega al broker iot.eclipse.org restando in ascolto sulla porta 1883. Il codice implementato è il seguente:

```
import paho.mqtt.client as mqtt
# The callback for when the client receives a CONNACK response from the server.
def on_connect(client, userdata, flags, rc):
    print("Connected with result code "+str(rc))
# Subscribing in on_connect() means that if we lose the connection and
# reconnect then subscriptions will be renewed.
    client.subscribe("$SYS/#")
# The callback for when a PUBLISH message is received from the server.
def on_message(client, userdata, msg):
    print(msg.topic+" "+str(msg.payload))
client = mqtt.Client()
client.on_connect = on_connect
client.on_message = on_message
client.connect("iot.eclipse.org", 1883, 60)
client.loop_forever()
```

Per quanto riguarda COAP abbiamo generato in maniera del tutto analoga a MQTT i pacchetti. Anche in questo caso grazie ad una libreria "Coapthon" siamo stati in grado di generare il server e il relativo client. Per AMQP invece abbiamo utilizzato la libreria pika. In questo caso abbiamo costruito 2 file rispettavimente, un ricevitore e mittente in grado di mandare i messaggi.
Rispettivamente i file del ricevitore e del mittente:

```
#!/usr/bin/env python
import pika
connection = pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))
channel = connection.channel()
channel.queue_declare(queue='hello')
def callback(ch, method, properties, body):
    print(" [x] Received %r" % body)
channel.basic_consume(callback,queue='hello',no_ack=True)
print(' [*] Waiting for messages. To exit press CTRL+C')
channel.start_consuming()
##Send
#!/usr/bin/env python
import pika
connection = pika.BlockingConnection(pika.ConnectionParameters(host='localhost'))
channel = connection.channel()
channel.queue_declare(queue='hello')
channel.basic_publish(exchange='',routing_key='hello', body='Hello World!')
print(" [x] Sent 'Hello World!'")
connection.close()
```

2.1.1 Cattura dei pacchetti: Wireshark

Per la cattura dei pacchetti è stato utilizzato un software di sniffing, Wireshark. Lo sniffing, in informatica e nelle telecomunicazioni è l'attività di intercettazione passiva dei dati che transitano in

una rete telematica[20]. Wireshark ci fornisce un interfaccia con la quale è possibile osservare tutto il traffico presente sulla rete. Possiamo selezionare l'interfaccia di rete dalla quale il software deve catturare i pacchetti in tempo reale. I dati mostrati dal software ci mostrano tutti i livelli di rete per i quali i pacchetti sono implementati e ci fornisce l'indicazione del protocollo al livello applicativo utilizzato. Inoltre ci mostra gli indirizzi ip del mittente e del destinatario oltre che a farci vedere i valori dei singoli bytes contenuti all'interno dei singoli pacchetti. Raccolti i dati, li abbiamo esportati, così da avere dei pacchetti di prova con cui fare il packet inspection. Dei pacchetti catturati vengono visualizzati tutti i bytes in esadecimale dal livello IP al livello applicativo. Con i filtri integrati nel software abbiamo la possibilità di filtrare i pacchetti per protocollo, per indirizzo destinazione o sorgente. È possibile salvare ed esportare i file di dump in molteplici formati (pcap,pcapng,csv,raw...). Di seguito esempi di pacchetti rispettivamente AMQP, MQTT E COAP.

```
0x01,0x00,0x00,0x00,0x00,0x00,0x03,0x77,0x55,0x75,0xCE ;
0x30,0x04,0x44,0x43,0x66,0x53;
0x43,0x02,0x42,0x56,0x54,0xAA,0x56,0X4A,0x33,0x44,0x55,0x43;
```

2.2 Espressioni regolari e flex

Per questa analisi abbiamo utilizzato un software di analisi lessicale chiamato "flex". Flex è un generatore automatico di scanner, che dato in ingresso dei caratteri compie delle azioni in base all'input. Inoltre questo software crea una vera e propria macchina a stati, atta appunto, al riconoscimento di uno stream di input. Questo programmma esegue l'analisi e la tokenizzazione dei caratteri mediante l'utilizzo di una macchina a stati deterministica, accettando solo espressioni regolari. Queste macchine sono un sottoinsieme della collezione delle macchine di Turning. La sintassi è appunto basata su espressioni regolari. Flex usualmente ha complessità O(n) in base alla lunghezza dell'input, cioè esegue un numero costante di operazioni per ogni simbolo di ingresso. Un' applicazione tipica di flex è quella della scrittura di compilatori, il programma scritto dall'utente è lo stream di caratteri in ingresso. Lo scanner analizza l'ingresso e spezza in token l'input, che viene passato successivamente al parser il quale ha il compito di riconoscere le vere e proprie regole grammaticali[16]. Nel nostro caso esso ci ha permesso di poter costruire delle espressioni regolari e con l'ausilio del codice in C abbiamo utilizzato dei contatori per le parti variabili. Il codice assume la seguente forma:

```
Definizioni
%%
regole
%%
codice
```

Nella prima parte dell'intestazione, all'interno dei due simboli % % vengono definite le librerie e tutto il codice che deve essere eseguito prima dello scanner. Nella seconda parte invece, denominata regole, delimitata da %% %% troviamo la parte dedicata alla definizione delle espressioni regolari. Nell'ultima parte invece viene scritto il codice e il main. Le espressioni regolari possono assumere molteplici forme, inoltre sono presenti dei pattern specicifici in base a quello che vogliamo indicare[5]. Nella nostra analisi sono state utilizzate oltre ai pattern di riconoscimento anche dei token di start cioè delle condizioni specifiche affinchè una determinata espressione venga attivata o meno. Un token di start deve essere posto all inizio del pattern che si vuole utilizzare, delimitato dai simboli maggiore e minore, ad esempio <start>. Esse sono dichiarate nella prima parte dedicata alle definizioni, e vengono attivate nella parte dedicata al codice. L'attivazione viene per mezzo della particella BEGIN seguita dal nome del token di start precedentemente dichiarato. Di seguito la tabella dei pattern:

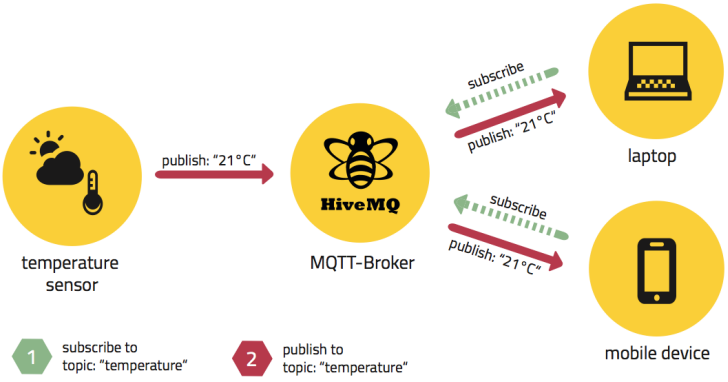
Pattern	Descrizione
'x'	Controlla il carattere x
'.'	Controlla tutti in caratteri eccetto fine linea
'[xyz]'	Una classe di caratteri, vengono controllati rispettivamente i caratteri 'x', 'y' e 'z'.
'[abj-oZ]'	Classe di caratteri con intervallo, si riconosce una 'a' una 'b', e un carattere da 'j' a 'o' o fino a 'Z'.
'[Â-Z]'	Classe di caratteri negata, in questo caso si riconoscono tutti i caratteri eccetto quelli maiuscoli
'[Â-Z/n]'	Tutti i caratteri eccetto lettere maiuscole e a capo.
'r*'	Riconosce 0 o più r dove r è un'espressione regolare.
'r+'	Riconosce una o più r
'r?'	Riconosce zero o più r (r è opzionale)
'r{2,5}'	Riconosce da due a cinque r
'r{2,}'	Riconosce due o più r
'r{4}'	Riconosce esattamente 4 r
'{name}'	Espansione della definizione di nome
'"[xyz]"foo"'	Riconosce la strinza ""[xyz]"foo"
'0'	Carattere nullo.
'123'	Carattere con valore ottale 1, 2 o 3.
'x2a'	Carattere in esadecimale con valore 2a
'rs'	Espressione regolare r seguita dall'espressione s.
'r|s'	Riconosce una r o una s.
'r/s'	Riconosce una r solo se seguita da una s
'r̂'	Riconosce una r solo se all'inizio della linea
'r$'	Riconosce una r solo se alla fine della linea
'<s>r'	Riconosce una r solo nel caso in cui la start condition s è valida
'<*>r'	Riconosce una r in tutte le start condition.
'<<EOF>>'	Pattern di fine file

Ogni espressione regolare è seguita da un blocco di codice che viene attivato solamente nel caso in cui quella determinata espressione regolare viene riconosciuta in input. Le espressioni regolari vengono valutate sequenzialmente ed inoltre se vengono incontrati casi nel quale l'input può ricadere all'interno di due espressioni, flex abilita quella in grado di coprire il maggior numero di caratteri. Per la compilazione del file in ambiente linux tramite shell eseguiamo
$ flex file.l
Viene generato il file lex.yy.c il quale dovrà essere compilato digitando gcc lex.yy.c -lfl. Fatto ciò viene generato un file eseguibile. È possibile eseguire il programma in 2 modi, eseguendolo semplicemente e quindi potendo poi avere la possibilità, tramite tastiera, di immettere una stringa da fare analizzare oppure dando in input un file di testo da scansionare. Il programma inizialmente cerca una corrispondenza nei pattern. Una volta determinata la corrispondenza, il token è disponibile nel puntatore di caratteri generale yytext e la la sua lunghezza nella variabile globale intera yyleng. Successivamente viene eseguito il codice associato a quella espressione predisponendo lo scanner ad analizzare la prossima stringa. Nel caso in cui nessuno schema sia stato scelto, viene eseguita la regola di default e viene scansionata la stringa seguente. Yytext può essere definito sia come un puntatore di char sia come un array. Il file di output lex.yy.c contiene la routine di scansione yylex(), con al proprio interno un numero di tabelle per utilizzare i token corrispondenti. Ogni volta che viene chiamato, yylex() esegue la scansione dei token dal file di input globale yyin proseguendo finchè non si raggiunge un finale. Quando lo scanner riceve comunicazione da parte dell'input di end-of-file, flex va a controllare la funzione yywrap(). Se yywrap () restituisce false, si presume che la funzione sia passata avanti e viene impostato yyin per puntare a un altro file di input e la scansione continua. Se restituisce true, lo scanner termina, restituendo 0 al suo chiamante.

2.3 Il protocollo MQTT

MQTT è un protocollo di messaggistica leggero di tipo publish-subscribe posizionato in cima a TCP/IP. È stato disegnato per le situazioni in cui è richiesto un basso impatto e dove la banda è limitata. Il modello su cui si basa è publish/subscribe e richiede un dispositivo chiamato broker. Il broker è responsabile della distribuzione dei messaggi ai client destinatari. A livello pratico viene utilizzato ad esempio da facebook per le chat[19]. Come è possibile vedere, un sensore di temperatura collegato al broker manda ad intervalli regolari tramite il pacchetto publish i dati relativi alle proprie rilevazioni. Con i nostri dispositivi, laptop e smartphone, tramite la sottoscrizione del topic possiamo ricevere i dati pubblicati sul broker dal sensore. Il topic identifica in maniera univoca i dati che vogliamo andare a visionare.

Figura 2.1: Comunicazione MQTT



Ogni pacchetto si presenta in questa forma, un header fisso, un header variabile e il payload.
Nel protocollo MQTT sono stati predisposti 13 tipi di pacchetti rispettivamente:

- Connect serve per mandare dal client una richiesta di connesione al server.
- Connack è l'acknowledgment dell'avvenuta connesione.
- Publish è spedito dal server al client o viceversa per trasportare un messaggio applicativo.
- Puback è l'acknowledgment del pacchetto publish.
- Pubrec è un pacchetto di risposta al pacchetto puback.
- Pubrel è un pacchetto di risposta al pacchetto pubrec.
- Pubcomp è un pacchetto di risposta al pacchetto pubrel.
- Subscribe questo pacchetto viene inviato dal client al server per creare una o più sottoscrizioni.
- Suback è un pacchetto spedito dal server al client come acknowledgment al pacchetto subscribe.
- Unsubscribe questo pacchetto viene inviato dal client al server, per annullare la sottoscrizione ad un topic.
- Unsuback è un pacchetto spedito dal server al client come acknowledgment al pacchetto unsubscribe.
- Pingreq è un pacchetto inviato dal client al server. Può essere utilizzato in 3 casi, per indicare al server che il client è attivo in assenza di altri pacchetti di controllo inviati dal client al server, per richiamare il server che risponda per confermare che è vivo, oppure per esercitare la rete, per indicare che la connessione di rete è attiva.

- Pingresp è un pacchetto spedito dal server al client come acknowledgment al pacchetto pingreq.
- Disconnect è il pacchetto di controllo finale spedito dal client al server per chiudere la connessione.

Il primo byte di ogni pacchetto ne indica il tipo, mentre il secondo byte ne indica la lunghezza rimanente. La lunghezza può variare da uno a quattro bytes in base 128. Inoltre è presente un header variabile che cambia di volta in volta in base al tipo di pacchetto. Per quanto riguarda il payload, in questa analisi non è stato preso in considerazione[8].

2.3.1 Riconoscimento MQTT

Figura 2.2: Header MQTT

Bit	7	6	5	4	3	2	1	0
byte 1	MQTT Control Packet type				Flags specific to each MQTT Control Packet type			
byte 2...	Remaining Length							

Un espressione regolare per riconoscere un pacchetto MQTT ad esempio può assumere la seguente forma:

```
"<INITIAL>\x10({mqtt_init})\x00\x04\x4d\x51\x54\x54\x04." dove:
<INITIAL> : Rappresenta il token di start, questo token ci permette
di essere sicuri di trovarci nello stato iniziale della lettura.
\x10:  Rappresenta il primo byte ed indica il tipo di pacchetto MQTT.
In questo caso è un pacchetto connect.
{mqtt_init}: Rappresenta la lunghezza in questa forma:

digit       [\x00-\x7F]
alpha       [\x80-\xFF][\x00-\x7F]
beta        [\x80-\xFF][\x80-\xFF][\x00-\x7F]
gamma       [\x80-\xFF][\x80-\xFF][\x80-\xFF][\x00-\x7F]
mqtt_init   ({digit}|{alpha}|{beta}|{gamma})

\x00\x04\x4d\x51\x54\x54\x04.: Questa stringa rappresenta l'header fisso
del pacchetto connect di mqtt.
```

Nel dettaglio la lunghezza ha la seguente interpretazione, se il primo byte riservato alla lunghezza presenta un valore fra 0 e 127 in questo caso non sono presenti altri bytes e quella è la lunghezza rimanente del pacchetto. Se, invece, il primo byte presenta un valore maggiore di 127 è presente un secondo byte, inoltre, se questo è minore o uguale a 127 la lunghezza è finita, altrimenti è presente un altro byte così fino ad arrivare alla lunghezza massima, che si compone di 3 bytes maggiori di 128 e un byte fra 0 e 127. Naturalmente i valori dei bytes essendo in base 128, devono essere moltiplicati rispettivamente per 128, 16384, 2097152.

In questo caso il riconoscimento terminerebbe e darebbe esito positivo. Oltre a ciò, per i pacchetti Publish, Subscribe, Unsubscribe, in quanto presentano un header variabile contentente informazioni che cambiano da situazione a situazione si è adottata una tecnica leggermente differente di riconoscimento, infatti, si sono utilizzati dei contatori per contare il numero di bytes per le parti variabili. Ad esempio:

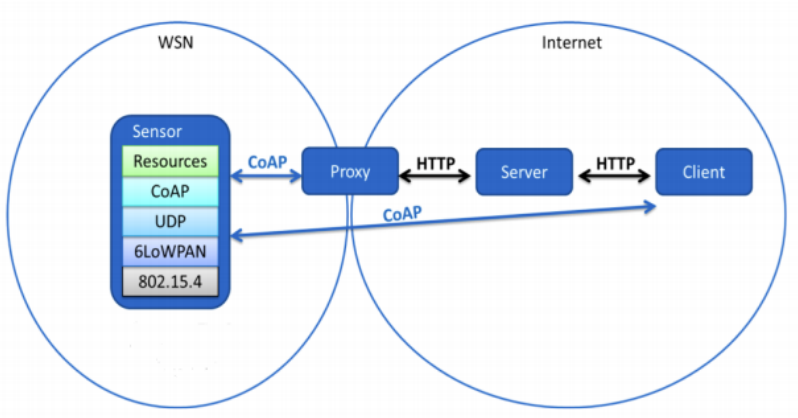
```
 <INITIAL>\x30({digit}) : Questa espressione regolare rappresenta
l'inizio del pacchetto Publish, viene salvata la lunghezza in una variabile,
e attivato un token di start che ci permette di attivare un altra espressione
regolare e di valutare i restanti byte.
Di seguito il pezzo di codice che viene attivato.
<MQTT1>.* {
int lenght = strlen(yytext);
lunghezza_mqtt = lunghezza_mqtt - lenght;
if ( lunghezza_mqtt == 0 )
{
  printf("%s\n", "E' mqtt, pacchetto Publish");
}
BEGIN 0;
}
```

<MQTT1> rappresenta il token che viene attivato, il restante sono tutti i bytes del pacchetto. I restanti bytes del pacchetto vengono contati e salvati in una variabile, successivamente si sottrae la lunghezza appena calcolata alla lunghezza letta nel secondo byte del pacchetto. Se questa differenza come risultato è uguale a 0 probabilmente ci troviamo di fronte a un pacchetto MQTT di tipo Publish. Questa tecnica dei contatori è possibile grazie all'ausilio del codice C che ci permette di salvare la lunghezza e poi sottrarla.

2.4 Il protocollo COAP

Per quanto concerne il protocollo COAP, esso è un protocollo a livello applicativo, specializzato per il trasferimento web. E' stato progettato per architetture machine-to-machine, come la smart energy e domotica. COAP, si basa su un modello richiesta/risposta fra due punti e include concetti chiave del web come URI. Inoltre questo protocollo è stato progettato per interfacciarsi facilmente con HTTP per l'integrazione col web soddisfacendo particolari requisiti come il supporto multicast, o la semplicità in ambienti vincolati[3]. Utilizza come protocollo a livello di trasporto UDP.

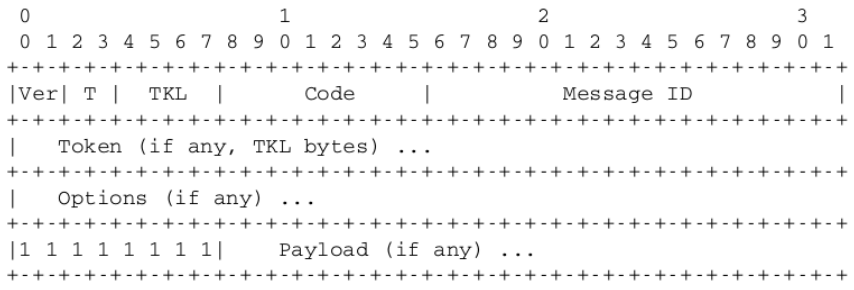
Figura 2.3: Comunicazione COAP



L'header è composto anche in questo caso da una parte fissa, da una parte variabile e dal payload. I primi due bit indicano la versone del protocollo, i due bit seguenti invece indicano il tipo messaggio che può essere in base al valore, Confirmable (0), Non-confirmable (1), Acknowledgement (2), o Reset (3). Seguono poi 4 bit che indicano il token-lenght, qui si indica appunto la lunghezza del campo token che segue, essa può assumere valori da 0 a 8 bytes. Successivamente si incontra il campo codice, esso riserva 3 bit alla classe del messaggio, e i restanti 5 al dettaglio. La classe può indicare una richiesta, una risposta positiva, una risposta errata del client o un errore del server. Infine per quanto riguarda la parte fissa del protocollo gli ultimi due bytes indicano l'id del messaggio. L'id viene utilizzato per rilevare duplicazioni di messaggio o per confrontare il tipo di messaggio ACK/RESET con il messaggio Confirmable/Non-confirmable. Per quanto riguarda la parte variabile abbiamo appunto, il

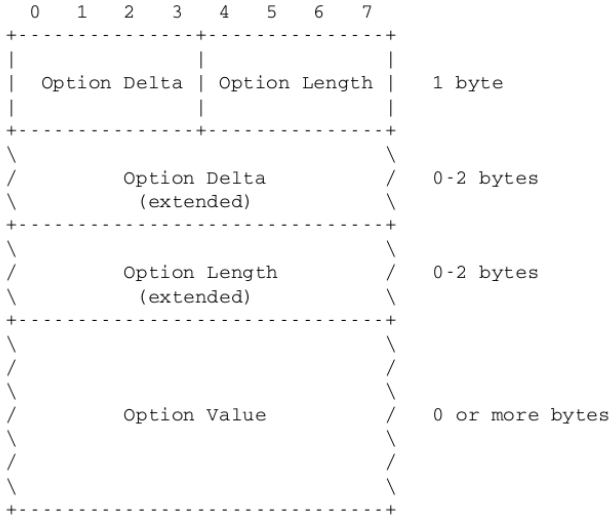
campo token che può variare in base alla lunghezza descritta nei primi 4 bit del primo byte. Il campo token viene utilizzato per correlare richieste e risposte.

Figura 2.4: Header COAP



Seguono infine le opzioni, esse possono essere molteplici oppure non comparire. Le opzioni sono codificate nel seguente modo, nel primo byte è presente un campo Option Delta di 4 bit mentre per i restanti 4 è presente il campo Option Lenght. Se il campo option delta presenta un valore fra 0 e 12 allora quello è l'option delta. Se invece presenta il valore 13 indica che c'è un byte che segue che indica l'option delta. Infine se si presenta il valore 14 si va ad indicare che sono presenti due bytes che seguono che indicano l'option delta. Il valore 15 è riservato per usi futuri. In maniera del tutto analoga funziona il campo Option Lenght, se esso ha un valore fra 0 e 12 allora quella è la lunghezza del opzione, se invece è presente il valore 13 abbiamo un byte aggiuntivo se abbiamo il valore 14 ne abbiamo 2[3].

Figura 2.5: Opzioni COAP



2.4.1 Riconoscimento COAP

Il riconoscimento avviene anche in questo caso per mezzo di espressioni regolari. L'espressione regolare è la seguente:

```
"<INITIAL>{coap_init}(\x00|\x01|\x02|\x03|\x04|[\x41-\x45]|[\x80-\x86]|
\x8C|\x8D|\x8F|[\xA0-\xA5])[\x00-\xFF][\x00-\xFF] {
lunghezza_coap = ( ( yytext[0] ) & 0x0f ) ;
if ( lunghezza_coap == 0 )
{
BEGIN COAP_OPTION;
} else {
BEGIN COAP_TOKEN;
}
```

```
dove:
<INITIAL>: è il token di start
{coap_init}: sono tutti i possibili valori del primo byte
(\x00|\x01|\x02|\x03|\x04|[\x41-\x45]|[\x80-\x86]|\x8C|\x8D|\x8F|[\xA0-\xA5]):
sono tutti i possibili valori del codice
[\x00-\xFF][\x00-\xFF] : sono gli ultimi due bytes che identificano l'id del messaggio.
```

Una volta valutata la prima espressione, si salva in una variabile la lunghezza del campo token. Se non sono presenti token, si attiva un espressione regolare che controlla se ci sono opzioni, altrimenti, si attiva l'espressione regolare che conta la lunghezza del token, per poi passare al campo opzioni. Per quanto riguarda le opzioni possono avere lunghezze variabili, per questo motivo sono state create più espressioni regolari per ogni singolo caso, successivamente, siamo andati a salvare in una variabile la lunghezza dell'opzione, inoltre, poi abbiamo attivato un altra espressione regolare che conta i singoli byte, se la differenza da come risultato 0 molto probabilmente si tratta di un pacchetto COAP.
Ad esempio:

```
<COAP_OPTION>([\x00-\x0C])|([\x10-\x1C])|([\x20-\x2C])|([\x30-\x3C])|([\x40-\x4C])
|([\x50-\x5C])|([\x60-\x6C])|([\x70-\x7C])|([\x80-\x8C])|([\x90-\x9C])
|([\xA0-\xAC])|([\xB0-\xBC])|([\xC0-\xCC]) {
  lunghezza_coap = ( ( yytext[0] ) & 0x0f );
  BEGIN 0;
  BEGIN COAP_COUNTER;
  }
```

Questa espressione regolare viene attivata tramite il token di start <COAP-OPTIONS> abilitato nella precedente espressione. Tutti gli altri bytes sono i valori che l'opzione può assumere. Di seguito viene salvata la lunghezza nella variabile lunghezza-coap, e viene attivato il token di start <COAP-COUNTER>.

```
<COAP_COUNTER>[\x00-\xFF] {
 lunghezza_coap --;
 if ( lunghezza_coap == 0 )
 {
  BEGIN 0;
  BEGIN COAP_OPTION;
 }
```

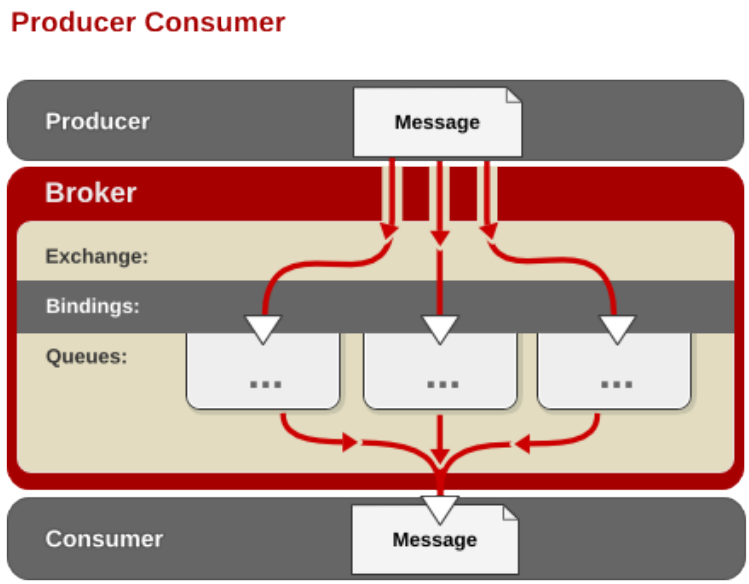
Questa è l'espressione che viene attivata, successivamente, si sottrae uno alla lunghezza totale e finchè la lunghezza non è uguale a zero si procede a sottrarre i bytes. Infine si ritorna alle espressioni che valutano le opzioni, nel caso non ne fossero più presenti il riconoscimento termina. Il riconoscimento può terminare in due modi, o non è presente il payload e quindi se la lunghezza del pacchetto è zero si termina il riconoscimento positivamente, oppure se è presente il payload viene valutato il primo byte che è sempre 0xFF.
Di seguito le ultime due espressioni:

```
<<COAP_OPTION><<EOF>> {
if ( lunghezza_coap == 0 ) {
printf("%s\n","E' Coap" );
BEGIN 0;
}}
<COAP_OPTION>\xFF {
printf("%s\n","E' coap" );
}
```

2.5 Il protocollo AMQP

AMQP è un protocollo a livello applicativo, progettato per supportare in modo efficiente un'ampia gamma di applicazioni di messaggistica e modelli di comunicazione. Fornisce una comunicazione a flussi controllata, con garanzie di consegna dei messaggi, come una volta sola, almeno una volta e esattamente una volta. Presuppone un protocollo sottostante di livello affidabile di trasporto come il protocollo TCP. Esso è stato progettato per supportare un ampia gamma di applicazioni nel campo della messaggistica e della comunicazione di pattern. Le caratteristiche definite da AMQP, sono l'orientamento ai messaggi, coda, routing (incluso il point-to-point e publish-and-subscribe), affidabilità e sicurezza[1]. I vantaggi di AQMP sono quelli di essere flessibile, aperto e interoperabile, esso è in grado di connettersi tra organizzazioni e tecnologie, garantendo un alto livello di sicurezza. Come è possibile vedere, il messaggio viene prodotto, mandato al broker inserito all'interno di una coda, e poi recapitato al destinatario.

Figura 2.6: Esempio di comunicazione AMQP



Esso è formato da una parte fissa e dal payload. Il primo byte indica il tipo di frame, i possibili valori sono 4. "METHOD","HEADER","BODY","HEARTBEAT". I seguenti 2 bytes indicano il canale, esso può assumere valori da 0 a 65535. I restanti 4 bytes indicano la lunghezza rimanente del pacchetto. Il pacchetto di apertura della connessione invece è composto dalla stringa "AMQP0091"[9].

Figura 2.7: Header AMQP

2.5.1 Riconoscimento AMQP

Per il riconoscimento dei pacchetti viene attivata la seguente espressione regolare:

```
<INITIAL>(\x01|\x02|\x03|\x04)[\x00-\xFF][\x00-\xFF]
[\x00-\xFF][\x00-\xFF][\x00-\xFF][\x00-\xFF]{

BEGIN AMQP;
```

```
lunghezza_amqp = lunghezza_amqp + ( ( ( yytext[3] ) * 16777216 ) +
( ( yytext[4] ) * 65536 ) + ( ( yytext[5] ) * 256 ) + yytext[6] ) ;

}
```

```
dove:
<INITIAL>: è il token di start
(\x01|\x02|\x03|\x04): primo byte del pacchetto
[\x00-\xFF][\x00-\xFF][\x00-\xFF][\x00-\xFF][\x00-\xFF][\x00-\xFF]: sono rispettivamente
 2 byte per i canale e 4 per la lunghezza del pacchetto.
```

Valutata questa espressione, si salva la lunghezza del pacchetto in una variabile e viene attivata una seconda espressione che va a contare i restanti bytes. Se la differenza fra la lunghezza salvata e i bytes che sono stati contati è 0 probabilmente ci troviamo di fronte ad un pacchetto AMQP. Come nell' esempio:

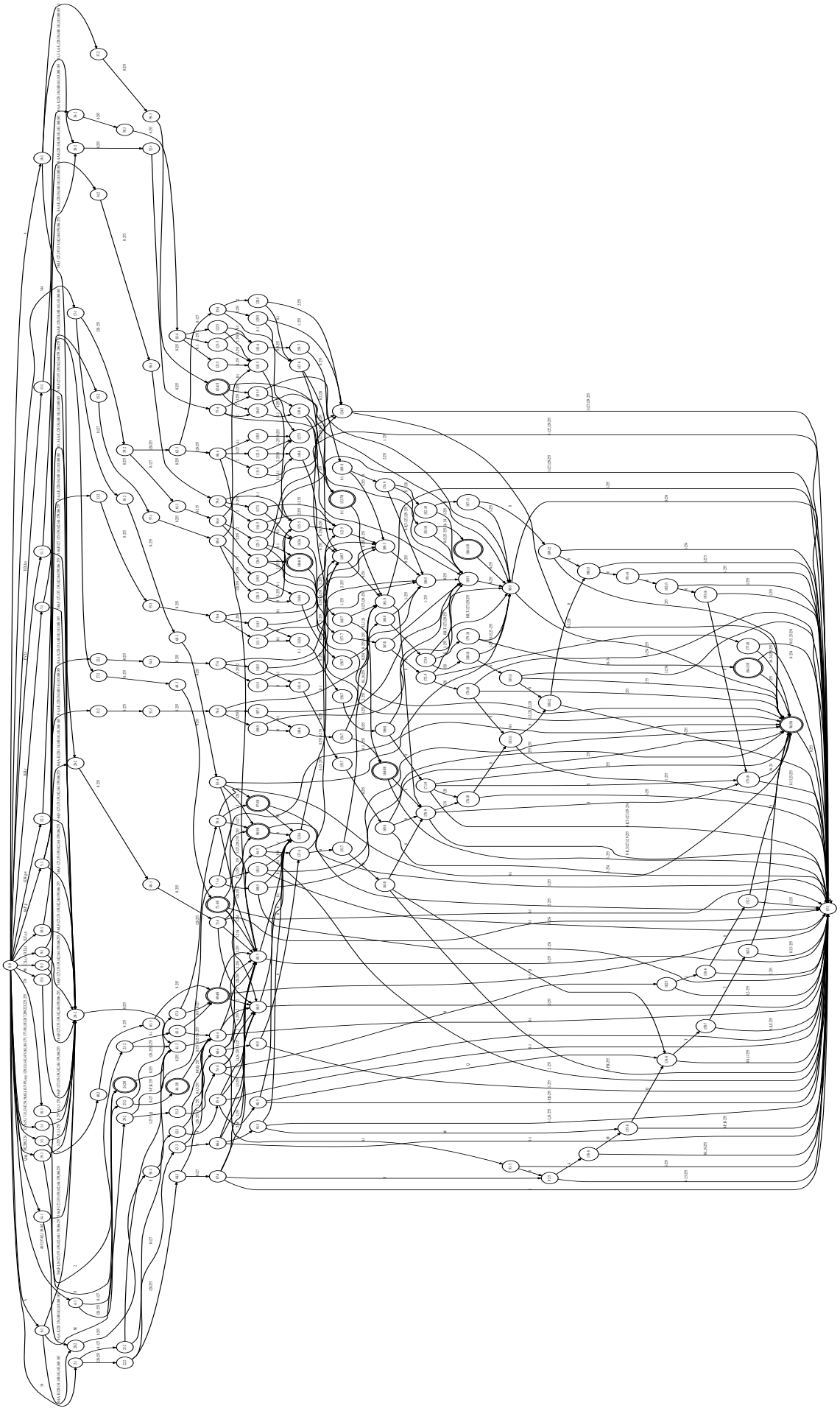
```
<AMQP>.* {
int lenght = strlen(yytext);
lunghezza_amqp = lunghezza_amqp - lenght;
if ( lunghezza_amqp == 0 )
{printf("%s\n", "E' amqp");}
BEGIN 0;
}
```

Questa è l'espressione che viene attivata tramite il token di start AMQP e si controlla la lunghezza.

2.6 Risultati

I risultati avuti dall'utilizzo delle espressioni regolari per il riconoscimento sono buoni, infatti facendo un analisi così approfondita del pacchetto è molto difficile che la scansione non vada a buon fine. In prima analisi, già con il primo byte del pacchetto possiamo indirizzare il riconoscimento su una strada rispetto che su un' altra. Inoltre per quei pacchetti che potrebbero avere un interpretazione ambigua, si è appunto utilizzata la tecnica dei contatori, con i quali, leggendo il campo lunghezza del pacchetto si è potuto andare nel dettaglio potendo così contare il numero di bytes e andando a fare un confronto fra il campo e l'effettiva lunghezza del pacchetto. Inoltre a partire dalle espressioni regolari tramite il software "Regex Processor" è stata generata la macchina a stati deterministica. Questa macchina a stati tramite delle tecniche di riduzione è stata ottimizzata così da avere la dimensione minore possibile. Per l'appunto sono state accorpate le transazioni adiacenti fra di loro. Gli automi sono stati rappresentati mediante vettori di adiacenze. La macchina a stati ha 194 stati, la tabella completa delle transizioni ha 49664 elementi, mentre quella compressa, ne ha solo 1610. Dopo che si compila l'eseguibile va da 107092 byte per il primo a 8876 byte per il secondo. L'eseguibile generato con flex invece ha dimensione 57516 byte, pero' utilizza interi a 16 bit per le tabelle, mentre nelle precedenti si hanno interi a 8 bit. Inoltre il parser con flex esegue i controlli con i contatori che con le regular expression normali non è possibile fare. A livello di prestazioni la prima versione senza riduzioni è più performante in quanto ogni stato ha la propria tabella, però occupa maggiore memoria. La tabella compressa è più lenta in quanto ogni volta deve trovare lo stato in un altra tabella, ma occupa meno memoria.

Figura 2.8: Automa generato



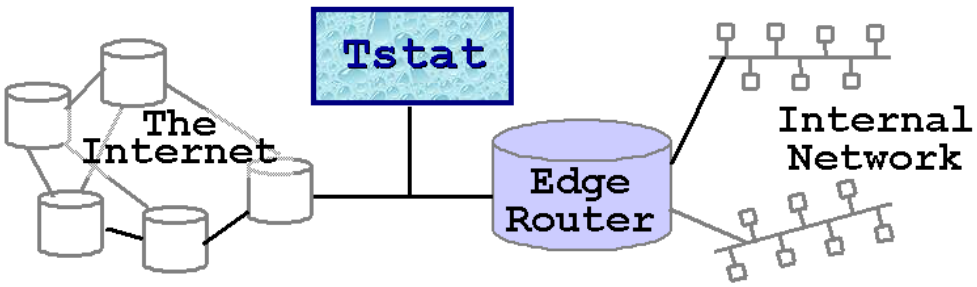
3 Deep Flow Inspection

Il Deep Flow inspection, contrariamente al Deep Packet Inspection non va a controllare il contenuto dei singoli pacchetti, ma va ad analizzare l'andamento dei flussi. Per la realizzazione di questa analisi ci viene in aiuto l'intelligenza artificiale, infatti grazie ad algoritmi di machine learning possiamo individuare e riconoscere interi flussi di rete. Questa tecnica viene utilizzata principalmente in due casi, quando i pacchetti da ispezionare sono criptati e quindi non è possibile leggerne il contenuto, oppure, quando dei dispostivi IoT utilizzano dei protocolli non progettati per essi, ad esempio, se un sensore al posto di usare mqtt utilizza http per una trasmissione. Per questo tipo di analisi tramite wireshark abbiamo, in primo luogo, raccolto pacchetti IoT e non, in secondo luogo con un software di analisi, "tstat" abbiamo generato le statistiche riguardanti, il numero di pacchetti, il numero di bytes, inter arrival time e round trip time massimo, medio e minimo. Generato il dataset, con un software di machine learning "weka" abbiamo creato dei cluster per dividere le feature e infine un albero decisionale in grado di stabilire se un flusso appartiene ad una comunicazione IoT oppure a una comunicazione tradizionale.

3.1 Tstat

Tstat, uno strumento che, a partire dalle librerie software standard, offre ai responsabili di rete e ai ricercatori informazioni importanti su indici di prestazioni classici e dati statistici sul traffico Internet. Si possono generare statistiche in due modi, sia con la cattura in diretta dei pacchetti, oppure analizzando tracce a livello di pacchetti precedentemente registrati, supportando diversi formati di dump, come quelli supportati dalla libreria libpcap e molti altri[10].
Il programma va richiamato con la seguente riga di comando:
$ tstat -[option] file_dump
Uno schema su come si colloca tstat nell' apparato di rete.

Figura 3.1: Tstat in un apparato di rete



L'output generato dal programma sono dei log contenenti le relative statistiche. Inoltre, tstat nella raccolta dei dati e delle statistiche divide in dei sotto flussi tutto il flusso di rete che è stato catturato. Un flusso si individua dal momento in cui viene aperta l connessione fino al momento di chiusura della stessa. Nel dettaglio i file log generati sono i seguenti:

- "log_tcp_complete" e "log_tcp_nocomplete" : in questi due file si riportano i dati di tutte le connessioni tcp, come detto in precedenza una connessione viene individuata dal momento in cui il primo segmento SYN è osservato fino alla chiusura con un segmento FIN/ACK o RST oppure se non vengono rilevati pacchetti dopo l'apertura della connessione per un tempo di default di 10 secondi oppure ancora se sono passati più di 5 minuti dall'ultimo pacchetto in generale. Inoltre tstat scarta tutte quelle connessioni per le quali il three way handshake non viene osservato.

Detto ciò una connessione correttamente aperta e chiusa viene registrata nel file tcp_complete in caso contrario viene salvata nel file tcp_noncomplete.

- "log_udp_complete" : questo file riporta tutte le coppie di flussi UDP rilevate. Un flusso UDP viene a configurarsi nel momento in cui il primo segmento del protocollo viene individuato per una coppia di socket UDP e termina nel momento in cui non vengono rilevati pacchetti per 10 secondi dall' apertura della connessione oppure trascorsi tre minuti e venti dall' ultimo pacchetto in generale.

- "log_video_complete" : in questo file vengono tracciati tutti i flussi video tcp. Le connessioni vengono suddivise come RTMP, TLS che sono associate ad esempio a youtube e connessioni HTTP.

- "log_http_complete" : viene prodotto un log con le informazioni riguardanti le richieste e le risposte http. Questo file non è generato di default ma deve essere specificato al momento del esecusione del programma.

- "log_mm_complete" : riporta le statistiche per i flussi RTP e RTCP.

- "log_skype_complete" : qui vengono riportate le statistiche riguardandi skype.

- "log_chat_complete, log_chat_messages": in questo caso vengono prodotte le statistiche riguardanti messenger e le chat in generale.

All interno dei singoli file di log troviamo i valori raccolti in base alla tipologia di trasmissione utilizzata come appena illustrato. Le colonne sono raggruppate sulla base della direzione del traffico, cioè da client al server e dal server al client. I valori generati generati per tcp sono i seguenti:

C2S	S2C	Short description	Unit	Long description
1	15	Client/Server IP addr	-	IP addresses of the client/server
2	16	Client/Server TCP port	-	TCP port addresses for the client/server
3	17	packets	-	total number of packets observed form the client/server
4	18	RST sent	0/1	0 = no RST segment has been sent by the client/server
5	19	ACK sent	-	number of segments with the ACK field set to 1
6	20	PURE ACK sent	-	number of segments with ACK field set to 1 and no data
7	21	unique bytes	bytes	number of bytes sent in the payload
8	22	data pkts	-	number of segments with payload
9	23	data bytes	bytes	number of bytes transmitted in the payload, including retransmissions
10	24	rexmit pkts	-	number of retransmitted segments
11	25	rexmit bytes	bytes	number of retransmitted bytes
12	26	out seq pkts	-	number of segments observed out of sequence
13	27	SYN count	-	number of SYN segments observed (including rtx)
14	28	FIN count	-	number of FIN segments observed (including rtx)
29		First time abs	ms	Flow first packet absolute time (epoch)
30		Last time abs	ms	Flow last segment absolute time (epoch)
31		Completion time	ms	Flow duration since first packet to last packet
32		C first payload	ms	Client first segment with payload since the first flow segment
33		S first payload	ms	Server first segment with payload since the first flow segment
34		C last payload	ms	Client last segment with payload since the first flow segment
35		S last payload	ms	Server last segment with payload since the first flow segment
36		C first ack	ms	Client first ACK segment (without SYN) since the first flow segment
37		S first ack	ms	Server first ACK segment (without SYN) since the first flow segment
38		C Internal	0/1	1 = client has internal IP, 0 = client has external IP
39		S Internal	0/1	1 = server has internal IP, 0 = server has external IP
40		C anonymized	0/1	1 = client IP is CryptoPAn anonymized
41		S anonymized	0/1	1 = server IP is CryptoPAn anonymized
42		Connection type	-	Bitmap stating the connection type as identified by TCPL7 inspection engine (see protocol.h)
43		P2P type	-	Type of P2P protocol, as identified by the IPP2P engine (see ipp2p_tstat.h)
44		HTTP type	-	For HTTP flows, the identified Web2.0 content (see the http_content enum in struct.h)

Un ulteriore funzionalità che tstat ci mette a disposizione è quella di generare istogrammi. Un istogramma rappresenta la distribuzione empirica di un indice specifico considerando un periodo di misurazione fisso. Tstat per queste misurazioni salva numerosi file che corrispondono di default a 5 minuti di flusso analizzato. Il traffico viene suddiviso come in precedenza in base alla direzione cioè, se esso va dal client al server o viceversa. I parametri analizzati in questo caso sono molteplici.

- IP Layer: statistiche relative agli indirizzi ip e i protocolli ip
- TCP Segments: statistiche ringuardanti i segmenti tcp
- TCP Flows: statistiche relative ai flussi TCP
- UDP Layer: statistiche relative ai flussi UDP
- Streaming Flows: statistiche relative ai flussi streaming
- RTCP Flows: statistiche relative al protocollo RTCP
- HTTP Flows: statistiche relative al protocollo HTTP
- Profile: profilo della macchina dove è in esecuzine Tstat

3.2 Weka

Weka è un software, che, dato in input un dataset è in grado di generare numerosi algoritmi di machine learning. Esso utilizza un ambiente completamente scritto in java, permettendoci così di avere un' interfaccia con la quale è possibile risolvere problemi di classificazione e regressione, di clusterizzare, di estrarre regole di associazione e di estrarre relativi grafici[21]. Inoltre in sostituzione all'interfaccia grafica è possibile utilizzare direttamente anche il linguaggio di programmazione java. Un modo per l'utilizzo di Weka è quello di applicare gli algoritmi di machine learning al dataset per analizzare in maniera più approfondita i dati, oppure quello di utilizzare modelli per predire i risultati. La modalità più semplice per utilizzare weka è quella di sfruttare la propria interfaccia grafica chiamata "explorer". In primo luogo, nella prima schermata vengono osservati 6 pannelli selezionabili con varie palette in alto. All'interno della prima interfaccia troviamo come prima cosa la possibilità di caricare il dataset. Il dataset si predispone in formato "arff"(Attribute Relationship File Format) esso è un formato per database. E' composto principalmente da due sezioni, l'intestazione e i dati. Nella prima riga dell'intestazione troviamo il nome della relazione, a seguire vengono descritti gli attributi con i propri nomi e il tipo. I tipi di attributi possono essere molteplici:

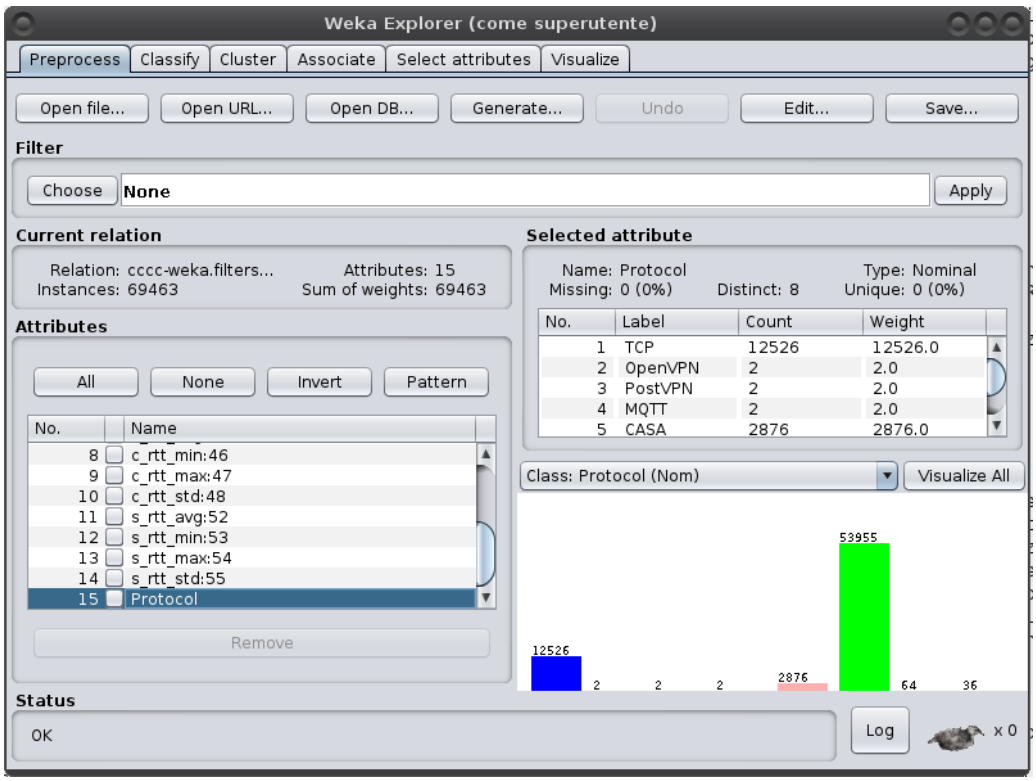
- Numeric è un tipo di attributo che rappresenta un float.
- Nominal rappresenta un numero predefinito di valori.
- String rappresenta una stringa, tipicamente usato nella classificazione di testi.
- Date appresenta una data, internamente descritta in floating point.
- Relational è un tipo di attributo che può contenere al proprio interno altri attributi.

Infine nell'intestazione troviamo le classi di appartenenza dei dati esplicitati nella sezione seguente. Successivamente troviamo i dati, essi vengono descritti sequenzialmente, dove, ogni dato numerico viene diviso da una virgola e, dove infine si trova la classe di appartenenza. Un esempio di file arff che abbiamo utilizzato è il seguente.

```
@relation 'cccc-weka.filters'
@attribute c_pkts_all:3 numeric
@attribute c_bytes_all:9 numeric
@attribute s_pkts_all:17 numeric
@attribute s_bytes_all:23 numeric
@attribute durat:31 numeric
```

```
@attribute inter-arrival-time numeric
@attribute c_rtt_avg:45 numeric
@attribute c_rtt_min:46 numeric
@attribute c_rtt_max:47 numeric
@attribute c_rtt_std:48 numeric
@attribute s_rtt_avg:52 numeric
@attribute s_rtt_min:53 numeric
@attribute s_rtt_max:54 numeric
@attribute s_rtt_std:55 numeric
@attribute Protocol {TCP,OpenVPN,PostVPN,MQTT,CASA}
@data
5,572,5,296,607.911,60.7911,0.269658,0.198,0.362,0.0839,0.1153,0.038,0.228,0.0998,TCP
5,354,5,1193,282.466,28.2466,1.622613,0.324,4.15,2.1890,0.1323,0.039,0.269,0.1209,VPN
5,490,5,296,602.203,60.2203,0.31199,0.219,0.468,0.135,0.1193,0.036,0.246,0.1115,TCP
5,354,5,1193,333.994,33.3994,0.31199,0.239,0.388,0.0745,0.1236,0.04,0.231,0.097,MQTT
5,612,5,296,578.331,57.8331,0.272991,0.215,0.375,0.0886,0.1689,0.05,0.347,0.157,TCP
5,572,5,296,644.661,64.4661,0.615646,0.189,1.455,0.7269,0.1066,0.041,0.2,0.0830,TCP
5,354,5,1193,300.728,30.0728,0.254658,0.148,0.398,0.128,0.1233,0.037,0.251,0.112,TCP
5,599,5,296,586.181,58.6181,0.253325,0.211,0.332,0.0681,0.1163,0.038,0.223,0.095,CASA
5,354,5,1193,325.747,32.5747,0.278991,0.182,0.41,0.1171,0.1323,0.061,0.248,0.1010,TCP
```

Come è possibile vedere i dati sono in ordine e corrispondono all'attributo presente nell' intestazione, mentre alla fine troviamo la classe di appartenenza. Una volta caricato il file, weka provvede alla creazione di un istogramma nel quale vengono visualizzati gli attributi e la classe di appartenenza graficamente. Esso nel dettaglio ci mostra l'occorrenza e quante volte compaiono i singoli attributi, inoltre ci mostra in base al totale la classe di appartenenza. La classe è un punto saliente, in quanto è quell' attributo che istruisce il sistema dati quei parametri in ingresso. Inoltre è possibile filtrare i dati principalmente per due tipi di apprendimento, supervisionato e non. I tipi di filtri sono molteplici, ad esempio è possibile normalizzare i dati oppure renderli discreti dividendoli in bin. Di seguito la schermata principale di weka.



3.2.1 Gli algoritmi di apprendimento

Weka fornisce molteplici algoritmi di apprendimento. Nella scheda di classificazione possiamo scegliere come prima cosa l'algoritmo di apprendimento che desideriamo utilizzare.
Gli algoritmi sono i seguenti:

- Classificatori bayesiani sono disponibili gli algoritmi NaiveBayes, NaiveBayesSimple, NaiveBayesMultinomial.
- Funzioni, all' interno di questa categoria troviamo gli algoritmi di regressione lineare e di regressione logistica.
- Regole, qui troviamo gli algoritmi, DecisionTable, JRip, OneR, PART.
- Alberi, gli algoritmi per la costruzione di alberi decisionali sono i seguenti, DecisionStump, HoeffdingTree, J48, LMT, RandomForest, RandomTree, RepTree.

Una volta scelto l'algoritmo, possiamo scegliere se il dataset precendentemente caricato venga utilizzato interamente per il training set, se invece lo si vuole dividere in fold, oppure se si vuole utilizzarne solo una percentuale. Il training set è quel dataset utilizzato appositamente per l'apprendimento, mentre il validation set è quel dataset utilizzato esclusivamente per la convalida e il test del modello che viene creato. Scelta la modalità e come dividere il dataset fra training set e validation possiamo eseguire l'algoritmo. Nel nostro esempio abbiamo eseguito l'algoritmo j48. L'output è formato nel seguente modo, in primo luogo troviamo la lista degli attributi e come questi sono stati utlizzati nell'albero, in secondo luogo segue l'albero vero e proprio dove vengono mostrati i parametri utilizzati per discriminare il cammino nei nodi. Successivamente viene mostrato un sommario dove al proprio interno vengono mostrate le statistiche.

```
Correctly Classified Instances          12048               96.1379 %
Incorrectly Classified Instances          484                3.8621 %
Kappa statistic                           0.9608
Mean absolute error                       0.0012
Root mean squared error                   0.0248
Relative absolute error                   6.2703 %
Root relative squared error              25.0411 %
Total Number of Instances                12532
```

In primo luogo troviamo le istanze completamente classificate, quelle errate, il grado di accuratezza e affidabilità, l'errore medio assoluto, l'errore quadratico medio, l'errore relativo assoluto, l'errore quadratico relativo e infine il numero totale delle istanze. Segue poi il dettaglio dell' accuratezza in base alle classi. Esso si compone di 9 colonne e ci fornisce un dettaglio in base ad ogni tipo di classe utlizzata.

TP Rate	FP Rate	Precision	Recall	F-Measure	MCC	ROC Area	PRC Area	Class
0,950	0,000	0,938	0,950	0,944	0,944	1,000	0,981	cluster0
0,942	0,000	0,970	0,942	0,956	0,955	1,000	0,981	cluster1
0,982	0,001	0,974	0,982	0,978	0,978	1,000	0,996	cluster2
0,966	0,001	0,957	0,966	0,961	0,961	1,000	0,986	cluster3

Le colonne indicano in ordine, la percentuale di flussi correttamente identificati, i falsi positivi, la precisione, la sensibilità, l'accuratezza, il coefficiente di correlazione di Matthews, la curva ROC (Receiver Operating Characteristic), la curva PRC, ed infine la classe di appartenza. Infine nei risultati viene mostrata la matrice di confusione, essa restituisce una rappresentazione dell'accuratezza di classificazione statistica. Ogni colonna della matrice rappresenta i valori predetti, mentre ogni riga rappresenta i valori reali. Nel nostro caso la tabella è utilizzata con i valori di "veri positivi"/"falsi positivi" e "falsi negativi"/"veri negativi"[18].

```
=== Confusion Matrix ===

     a     b     c     d     e     f     g     h   <-- classified as
 12524     0     0     0     0     2     0     0 |     a = TCP
     0     2     0     0     0     0     0     0 |     b = OpenVPN
     0     0     2     0     0     0     0     0 |     c = PostVPN
     1     0     0     0     0     0     0     1 |     d = MQTT
     0     0     0     0  2872     4     0     0 |     e = CASA
     0     0     0     0     0 53955     0     0 |     f = CINA
     0     0     0     0     0     0    57     7 |     g = INTEL
     0     0     0     0     0     0     1    35 |     h = BOSCH
```

Un altra sezione del programma da menzionare è quella che riguarda la clusterizzazione. Anche in questo caso, è possibile dare in input il dataset e sottoporlo a clusterizzazione. Dall'interfaccia è possibile selezionare vari tipi di algoritmi di clusterizzazione. Gli algoritmi disponibili sono rispettivamente, Canopy, Cobweb, EM, FarthestFirst, FilteredClustered, HiearchicalClusterer, MakeDensityBasedClusterer, SimpleKMeans. Scelto l'algoritmo anche in questo caso è possibile effettuare la scelta su come dividere il dataset. È possibile utilizzare l'intero file per il training set, oppure se dividerlo in fold o utilizzarne solo una percentuale per l'addrestamento. L'output generato, comprende nel caso di utilizzo dell'algoritmo SimpleKMeans, una tabella che indica il centroide utilizzato per ogni cluster e una tabella nella quale si indicano quanti flussi sono stati assegnati a un cluster rispetto che ad un altro. Per concludere è possibile vedere anche graficamente l'assegnazione e la distribuzione dei vari cluster, potendo esportare il file riuscendo così a creare il dataset nel quale i cluster assumono la funzione di classe.

3.3 Alberi decisionali

Nel machine learning un albero di decisione è un modello predittivo, dove ogni nodo interno rappresenta una variabile, un arco verso un nodo figlio rappresenta un possibile valore per quella proprietà e una foglia il valore predetto per la variabile obiettivo a partire dai valori delle altre proprietà, che nell'albero è rappresentato dal cammino (path) dal nodo radice (root) al nodo foglia. Normalmente un albero di decisione viene costruito utilizzando tecniche di apprendimento a partire dall'insieme dei dati iniziali (data set), il quale può essere diviso in due sottoinsiemi: il training set sulla base del quale si crea la struttura dell'albero e il test set che viene utilizzato per testare l'accuratezza del modello predittivo così creato[11]. Nel nostro caso abbiamo preso in esame l'algoritmo J48, esso è un implementazione scritta in java dell' algoritmo C45. L'algoritmo C45 è stato sviluppato da Ross Quinlan ed è utilizzato per generare un albero decisionale sulla base del concetto di entropia informativa. Il training set contiene dati già classificati, ogni ingresso corrisponde ad un vettore con dei valori e una classe che indentifica le feature. Ad ogni nodo dell'albero C45 sceglie l'attributo che divide maggiormente i campioni in una classe. Il criterio di suddivisione è il guadagno di informazioni normalizzato (differenza di entropia). L'attributo con il più alto guadagno di informazioni normalizzato viene scelto per prendere la decisione. L'algoritmo poi si ripete su tutti i sotto alberi. I casi base sono principalmente 3, quando tutti i campioni appartengono alla stessa classe, in questo caso non sussiste l'esigenza di andare a creare ulteriori nodi, quando nessuna caratteristica delle feature non fornisce alcun guadagno di informazioni, oppure infine quando compare un' istanza di una classe non ancora incontrata[12].

3.4 Clusterizzazione

Un altra tecnica utilizzata è quella della clusterizzazione. Dividere in cluster significa utilizzare delle tecniche che ci permettono la selezione e il raggruppamento delle feature simili fra loro. Le similitudini generalmente sono concepite in termini di distanza in uno spazio multidimensionale. Le tecniche si dividono principalmente in due metodologie, cioè, dal basso verso l'alto (bottom up) oppure dall'alto verso il basso (top down). Per le tecniche bottom up l'idea di base è quella nella quale tutti gli elementi siano divisi in cluster singoli. L'algoritmo poi provvederà ad unire i cluster più vicini fino ad ottenerne un numero prefissato, oppure finchè la distanza minima fra i cluster non superi una certa soglia. Per quanto riguarda invece le tecniche top down si pongono tutti gli elementi in unico cluster così poi da iniziare la divisione in tanti cluster di dimensioni inferiori. L'algoritmo continua a

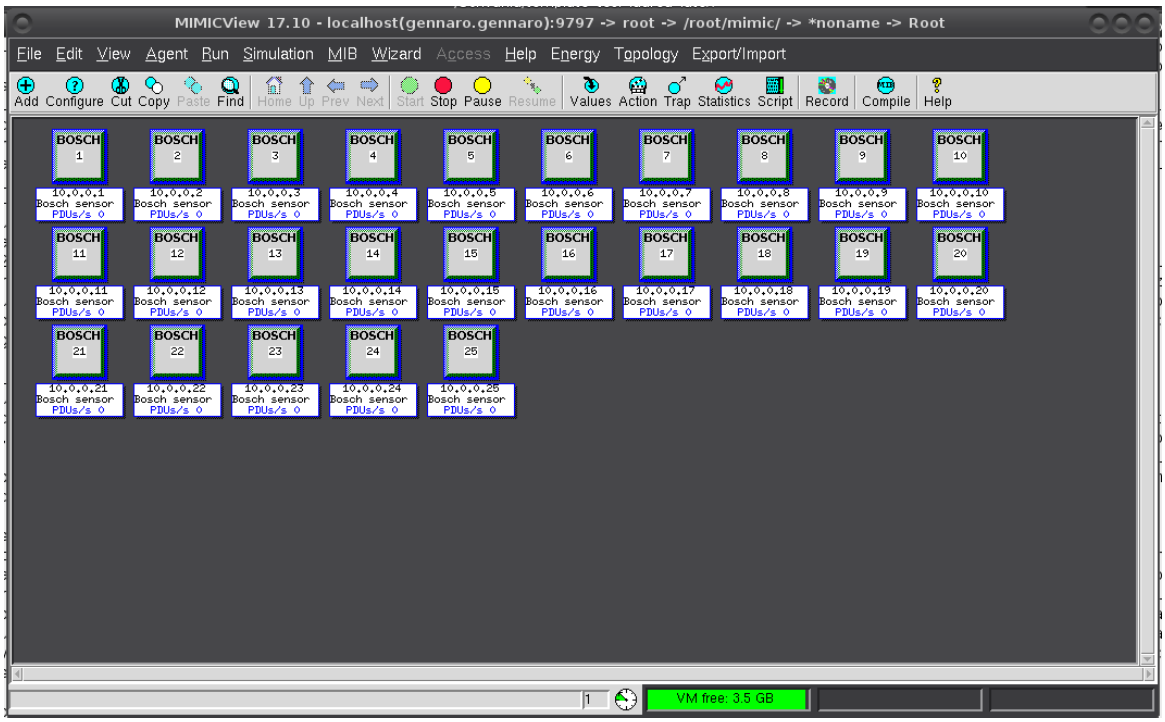
dividere finchè non si raggiunge un numero prefissato di cluster. Esistono varie classificazioni per gli algoritmi di clustering. Una prima categorizzazione dipende dalla possibilità che un elemento possa o meno essere assegnato a più cluster, possiamo avere, il clustering esclusivo, cioè ogni elemento può essere assegnato solo ad un sono gruppo oppure il clustering non esclusivo in cui un elemento può appartenere a più cluster con gradi di appartenenza diversi. Un altra distinzione fra gli algoritmi è quella delle metodologie per dividere lo spazio, principalmente abbiamo due distinzioni, il clustering partizionale in cui per definire l'appartenenza ad un gruppo viene utilizzata una distanza da un punto rappresentativo del cluster, avendo prefissato il numero di gruppi della partizione risultato, oppure il clustering gerarchico, in cui viene costruita una gerarchia di partizioni caratterizzate da un numero (de)crescente di gruppi, visualizzabile mediante una rappresentazione ad albero , in cui sono rappresentati i passi di accorpamento/divisione dei gruppi[13].

3.4.1 K-means

L'algoritmo K-means è un algoritmo di clustering partizionale che permette di suddividere un insieme di oggetti in K gruppi sulla base dei loro attributi[17]. L'obiettivo che l'algoritmo si prepone è di minimizzare la varianza totale inter-cluster. Ogni cluster viene identificato mediante un centroide o punto medio. L'algoritmo segue una procedura iterativa. Inizialmente crea K partizioni e assegna ad ogni partizione i punti d'ingresso o casualmente o usando alcune informazioni euristiche. Quindi calcola il centroide di ogni gruppo. Costruisce quindi una nuova partizione associando ogni punto d'ingresso al cluster il cui centroide è più vicino ad esso. Quindi vengono ricalcolati i centroidi per i nuovi cluster e così via, finché l'algoritmo non converge[2].

3.5 Generazione dei flussi

In primo luogo, ci siamo occupati della raccolta dei flussi di rete. Come base per avere un buon data-set, abbiamo utilizzato, per quanto riguarda i flussi IoT e non, dati provenienti da sensori situati nei pressi dell'università a Povo. Inoltre, per avere un buon numero di flussi non IoT, abbiamo reperito online tramite delle repository delle catture di pacchetti[6]. Per quanto riguarda la cattura dei flussi IoT, abbiamo utilizzato un software in grado di simulare dei sensori in tempo reale. Il software in questione prende il nome di "Mimic Simulator" http://gambitcomm.com/. Nell'immagine di seguito la schermata dell'interfaccia principale. esso ci ha consentito tramite un iterfaccia grafica di poter creare e simulare 2 tipi di sensori, rispettivamente sensori Intel e sensori Bosch[7].



Tramite la paletta "add" è possibile aggiungere i device, nel nostro caso la scelta è ricaduta sui sensori Bosch. La configurazione dei sensori prevede l'assegnazione di un indirizzo ip privato per ogni singolo sensore, per quanto concerne invece la configurazione per la parte mqtt la configurazione prevede l'immissione di un indirizzo ip per il broker, la porta, eventuali username e password. Per quanto riguarda invece la generazione del payload abbiamo utilizzato dei file di default in grado di generare un contenuto per ogni singolo pacchetto. Inizialmente abbiamo utilizzato un broker, in locale "mosquitto", in un secondo momento per avere dati più veritieri in merito a statistiche come il round trip time abbiamo connesso il computer con un ip pubblico e abbiamo fatto mandare i dati a dei broker mqtt pubblici. Raccolti tutti i dati abbiamo generato le statistiche con tstat e generato il dataset.

3.6 Svolgimento e Parametri utilizzati

Per lo svolgimento del deep flow inspection abbiamo preso in esame l'algoritmo j48 esso è stato scelto dopo un attenta analisi di esperimenti analoghi, nei quali sono stati messi a confronto l'efficacia di molteplici algoritmi. Siamo arrivati a conclusione che questo algoritmo ha un efficacia maggiore nel riconoscere i vari flussi, in quanto esso riesce ad avere una precisione molto alta e un numero di falsi positivi molto basso[23]. Per quanto riguarda la scelta dei parametri da prendere in cosiderazione anche qui c'è stata un analisi di esperimenti analoghi[22]. In primo luogo il parametro che non abbiamo preso in considerazione è la porta, in quanto essa in una prima analisi individua esattamente il protocollo di livello applicativo che si sta utilizzando ma è facilmente manipolabile, in quanto è possibile avviare comunicazioni anche utilizzando porte diverse da quelle conosciute. I parametri salienti che abbiamo deciso di utilizzare sono nel dettaglio:

- Numero di pacchetti dal client al server.
- Numero di bytes dal client al server.
- Numero di pacchetti dal server al client.
- Numero di bytes dal server al client.
- Durata in millisecondi del flusso (Ogni flusso viene individuato ogni qualvolta venga aperta una connessione e poi chiusa).
- Inter-arrival-time medio, questo è il tempo che mediamente intercorre fra ogni pacchetto.
- Round-trip-time medio, massimo, minimo,deviazione standard, questo è il tempo che intercorre fra l'invio di un pacchetto e la relativa risposta.

La prima fase è stata quella di raccogliere i pacchetti e di generare le relative statistiche. Successivamente siamo stati in grado di creare un dataset aggiungendo per ogni flusso la relativa classe, nella nostra analisi ne abbiamo create due, una per i flussi IoT e un altra per flussi tradizionali. La fase di classificazione si divide principalmente in due fasi, la prima di training, cioè quella in cui si va ad addestrare l'albero e a scegliere le migliori feature da utilizzare per il riconoscimento, la seconda quella di testing nella quale si va a testare appunto il modello creato. Inoltre successivamente è stata utilizzata la tecnica della clusterizzazione. Al posto di decidere a priori la classe del flusso, abbiamo deciso di creare dei cluster in grado di raggruppare i flussi con caratteristiche simili fra loro. Quindi al posto di utilizzare la divisione in classi con l'idea IoT o non IoT, abbiamo utilizzato i cluster per la classificazione.

3.7 Risultati

In primo luogo abbiamo analizzato i cluster che abbiamo creato con l'algoritmo SimpleKMeans. Il nostro obiettivo è stato quello di far rientrare nello stesso cluster i flussi IoT e di far ricadere in tutti gli altri le comunicazioni tradizionali. Il numero di cluster generati è appunto il giusto compromesso per arrivare al risultato appena citato, abbiamo fatto delle prove generando rispettivamente 10, 20, 30, 40, 50 e 100 cluster così da trovare la dimensione giusta. Abbiamo osservato le differenze fra i vari insiemi cercando di trovare dei punti di contatto e similitudini. Il dataset utilizzato contiene circa 70

mila flussi. Abbiamo potuto osservare, che le comunicazioni simulate con "mimic", dato che i sensori pubblicavano e quindi inviavano dati ad intervalli regolari, che la quantità di bytes inviata dal client al server era nettamente superiore a quella inviata dal server a client, dato che il server si occupa in questo caso solo di inviare pacchetti di acknowledgements. Sono state calcolate le percentuali di flussi iot in ogni cluster e messo in relazione il numero totale di flussi non IoT con il totale presente nel cluster. Successivamente abbiamo provveduto alla creazione dell'albero di decisione andando ad analizzare i parametri presi in considerazione dall'algoritmo. Di seguito i dati che siamo andati a generare, con la divisione in 30 cluster.

```
=== Summary ===

Correctly Classified Instances       69540               99.8693 %
Incorrectly Classified Instances        91                0.1307 %
Kappa statistic                          0.9985
Mean absolute error                      0.0002
Root mean squared error                  0.0089
Relative absolute error                  0.2666 %
Root relative squared error              5.163  %
Total Number of Instances            69631

=== Detailed Accuracy By Class ===

TP Rate  FP Rate  Precision  Recall   F-Measure  MCC      ROC Area  PRC Area  Class
1,000    0,000    1,000      1,000    1,000      1,000    1,000     1,000     cluster0
0,999    0,000    1,000      0,999    0,999      0,999    1,000     1,000     cluster1
1,000    0,000    1,000      1,000    1,000      1,000    1,000     1,000     cluster2
1,000    0,000    1,000      1,000    1,000      1,000    1,000     1,000     cluster3
0,986    0,000    0,995      0,986    0,991      0,991    1,000     0,988     cluster4
0,875    0,000    1,000      0,875    0,933      0,935    0,999     0,880     cluster5
0,941    0,000    0,970      0,941    0,955      0,955    0,999     0,943     cluster6
0,994    0,000    0,991      0,994    0,993      0,993    1,000     0,997     cluster7
0,999    0,000    0,997      0,999    0,998      0,998    1,000     0,998     cluster8
0,999    0,000    0,997      0,999    0,998      0,998    1,000     1,000     cluster9
1,000    0,000    0,964      1,000    0,982      0,982    1,000     0,967     cluster10
0,998    0,000    0,997      0,998    0,998      0,998    1,000     0,997     cluster11
1,000    0,000    1,000      1,000    1,000      1,000    1,000     1,000     cluster12
1,000    0,000    1,000      1,000    1,000      1,000    1,000     1,000     cluster13
0,996    0,000    1,000      0,996    0,998      0,998    1,000     1,000     cluster14
0,991    0,000    0,991      0,991    0,991      0,991    1,000     0,996     cluster15
0,800    0,000    1,000      0,800    0,889      0,894    1,000     0,848     cluster16
1,000    0,000    1,000      1,000    1,000      1,000    1,000     1,000     cluster17
1,000    0,000    0,999      1,000    1,000      1,000    1,000     1,000     cluster18
1,000    0,000    1,000      1,000    1,000      1,000    1,000     1,000     cluster19
0,000    0,000    0,000      0,000    0,000      0,000    0,987     0,030     cluster20
0,943    0,000    1,000      0,943    0,971      0,971    1,000     0,985     cluster21
0,997    0,000    0,994      0,997    0,995      0,995    1,000     0,996     cluster22
1,000    0,000    0,999      1,000    1,000      0,999    1,000     1,000     cluster23
0,997    0,000    0,996      0,997    0,997      0,996    1,000     0,997     cluster24
0,997    0,000    0,998      0,997    0,997      0,997    1,000     0,999     cluster25
0,902    0,000    0,958      0,902    0,929      0,930    0,998     0,923     cluster26
0,997    0,000    0,998      0,997    0,997      0,997    1,000     0,999     cluster27
1,000    0,000    1,000      1,000    1,000      1,000    1,000     1,000     cluster28
0,999    0,000    0,999      0,999    0,999      0,999    1,000     1,000     cluster29
Weighted Avg.    0,999    0,000    0,999      0,999    0,999      0,999    1,000     0,999
```

Bibliografia

[1] Advanced message queuing protocol. https://en.wikipedia.org/wiki/Advanced_Message_Queuing_Protocol.

[2] Clustering. http://mining2007.awardspace.com/sara/K-means.htm.

[3] The constrained application protocol (coap). https://tools.ietf.org/html/rfc7252.

[4] Deep packet inspection. http://www.xech.it/index.php?option=com_content&view=article&id=101&Itemid=119&lang=it.

[5] Manual-flex. ftp://ftp.gnu.org/old-gnu/Manuals/flex-2.5.4/html_mono/flex.html.

[6] Mawilab. http://www.fukuda-lab.org/mawilab/index.html.

[7] Mimic-simulator. http://www.gambitcomm.com/site/mimic-simulator.php.

[8] Mqtt version 3.1.1. http://docs.oasis-open.org/mqtt/mqtt/v3.1.1/os/mqtt-v3.1.1-os.html.

[9] Oasis advanced message queuing protocol(amqp) version 1.0. http://docs.oasis-open.org/amqp/core/v1.0/os/amqp-core-complete-v1.0-os.pdf.

[10] Tstat. http://tstat.polito.it/.

[11] Wikipedia-albero di decisione. https://it.wikipedia.org/wiki/Albero_di_decisione.

[12] Wikipedia-c4.5 algorithm. https://en.wikipedia.org/wiki/C4.5_algorithm.

[13] Wikipedia-clustering. https://it.wikipedia.org/wiki/Clustering.

[14] Wikipedia-deep packet inspection. https://it.wikipedia.org/wiki/Internet_delle_cose.

[15] Wikipedia-espressioni regolare. https://it.wikipedia.org/wiki/Espressione_regolare.

[16] Wikipedia-flex(software). https://en.wikipedia.org/wiki/Flex_(lexical_analyser_generator).

[17] Wikipedia-k-means. https://it.wikipedia.org/wiki/K-means.

[18] Wikipedia-matrice-di-confusione. https://it.wikipedia.org/wiki/Matrice_di_confusione.

[19] Wikipedia-mqtt. https://it.wikipedia.org/wiki/MQTT.

[20] Wikipedia-sniffing. https://it.wikipedia.org/wiki/Sniffing.

[21] Wikipedia-weka. https://it.wikipedia.org/wiki/Weka.

[22] Michael L. Crogan Andrew W. Moore, Denis Zuev. Discriminators for use in flow-based classification. *Department of Computer Science Research Reports*, 2005.

[23] Markus Rupp Markus Laner, Philipp Svoboda. Detecting m2m traffic in mobile cellular networks. *International Conference on Systems, Signals and Image Processing*, 2014.

[24] M. Padmavathy T. Nalini. Deep packet inspection with regular expressionmatching. *International Journal of Computer Applications Technology and Research*, 2(2):137–140, 2013.

[25] Michele Welponer. Implementazione di una rete neurale multilayer perceptron in np-5. 2017.