UNIVERSITA' DEGLI STUDI DI
NAPOLI FEDERICO II

Scuola Politecnica e delle Scienze di Base
Corso di Laurea in Ingegneria Informatica

Elaborato d'esame

Reti di Calcolatori I

Raccolta e analisi di tracce di traffico di applicazioni mobili

Anno Accademico 2018/2019

Professore

Prof. Antonio Pescapè

Gruppo – **38**

Marco Bocchetti **Matr. N46/003218**
Domenico Benfenati **Matr. N46/003380**

Indice

Capitolo 1

Classificazione del traffico

1.1 Introduzione

La diffusione degli smartphone e di altri tipi di dispositivi mobili ha portato negli ultimi due decenni ad un aumento crescente e sempre più diversificato del traffico di rete. Al 2016, solo sull'Apple App Store erano presenti più di 2 milioni di applicazioni con un totale di 130 miliardi di download [1].

Le tecniche di *classificazione del traffico Internet* — cioè quelle tecniche che mirano ad associare un flusso sulla rete con il tipo di applicazione che l'ha generato — sono nate innanzitutto come bisogno da parte dei provider di monitorare, priorizzare, ottimizzare, proteggere e prevenire determinate tipologie di traffico all'interno di una rete sempre più eterogenea e dinamica, ma si sono rivelate inoltre utili per condurre analisi sul comportamento delle applicazioni e dei servizi di terze parti nei confronti dell'utente. Spesso si parla anche di *identificazione del traffico Internet*, intendendo con questo tutti quegli approcci mirati ad individuare una specifica applicazione tra le tante.

La classificazione del traffico non solo fornisce informazioni utili per i provider, per i ricercatori e per le agenzie di sicurezza, ma solleva anche una serie di problematiche relative alla privacy degli utenti della rete. Con il termine *eavesdropping* ci si riferisce all'atto di intercettazione dei pacchetti sulla rete e della lettura del loro contenuto ad opera di terze parti (ad esempio, per motivi politici).

Le *classi* definiscono il livello di dettaglio del processo di classificazione: al livello più basso di dettaglio si collocano le classi di traffico (interactive, sensitive, undesired, etc.). Salendo di dettaglio si distinguono le categorie di applicazioni (chat, streaming, web, mail, file sharing, etc.) e le applicazioni specifiche (Skype, Facebook, Instagram, etc.). Al livello più alto di dettaglio il traffico viene classificato in base ai servizi interni ad un'applicazione (condivisione di foto, visualizzazione di video, chiamata, etc.).

L'oggetto di una classificazione può essere molteplice:

- Un pacchetto
- Una connessione TCP
- Un flusso (unidirezionale o bidirezionale)
- Un host
- Un altro tipo di oggetto di rete, come un *burst* o un *service burst*

Con il presente elaborato si espone il processo di generazione, di cattura, di classificazione e di validazione di tracce di traffico di applicazioni mobili. Gli oggetti della nostra classificazione saranno i biflussi, definiti dalla quintupla bidirezionale <*protocollo, indirizzo IP sorgente, indirizzo IP destinazione, porto sorgente, porto destinazione*>.

Nel primo capitolo passiamo in rassegna alcune delle tecniche di classificazione, con particolare enfasi sulle tecniche basate sugli algoritmi di machine learning. Nel secondo capitolo tratteremo gli strumenti utilizzati per lo svolgimento dell'elaborato.

Nel terzo capitolo descriveremo il processo di cattura delle tracce di traffico, della loro elaborazione mediante script e della validazione della classificazione. Nel quarto capitolo mostreremo i risultati ottenuti dall'analisi e dalla validazione delle tracce di traffico.

1.2 Analisi del traffico mobile

La presenza dei negozi virtuali come l'Apple App Store e il Google Play Store ha favorito negli ultimi anni la creazione, la diffusione e l'utilizzo di un numero sempre maggiore di applicazioni mobili.

Lo studio condotto da Bohmer et al. [10] ha dimostrato, analizzando il traffico di circa 4000 utenti nell'arco di 160 giorni, che gli utenti trascorrono in media un'ora al giorno sul loro smartphone, con una media di 72 secondi per ogni utilizzo, o di 37 secondi per un "accesso rapido". Dallo studio è inoltre emerso che:

- Ogni utente utilizza più volte il proprio dispositivo per poco tempo.
- Le sessioni che sono in maggioranza sono quelle brevi, e per ognuna di esse la quantita media di dati scambiati non supera il valore di 1 MB.
- In "modalità viaggio" gli utenti utilizzano soprattutto applicazioni multimediali.
- Google Maps è usata soprattutto all'imbrunire per controllare le condizioni di traffico.

Ancora, in *ProfileDroid: Multi-layer Profiling of Android Applications*, Gomez et al. [5] hanno osservato che:

- Le applicazioni spesso utilizzano le risorse del dispositivo senza preavviso, talvolta delegando ad altre applicazioni il compito di accedere a determinate risorse.
- Le applicazioni gratuite spesso hanno un costo che si riflette sul piano dati Internet, a causa dell'aumento del traffico generato dai servizi di advertising e di analytics.
- Google è presente quasi ovunque: di 27 applicazioni, 22 scambiano dati con Google. Tuttavia alcune applicazioni, come Amazon e Facebook, non hanno traffico Google.
- Le applicazioni comunicano con molte più sorgenti di traffico di quanto ci si aspetti. In particolare, le versioni gratuite tendono ad avere più traffico rispetto alle loro controparti a pagamento.

Riuscire a classificare il traffico mobile a partire dai nodi intermedi della rete solleva delle evidenti questioni di privacy. Impiegando metodi avanzati di machine learning, come quelli di *random forest* per l'apprendimento supervisionato, e quelli di *clustering gerarchico* per l'apprendimento non supervisionato, Conti et al. [11] sono riusciti non solo ad identificare traffico cifrato, ma anche a classificare delle specifiche azioni su Gmail, Facebook, Dropbox, Twitter, con un'accuratezza superiore al 95%.

In questo senso la classificazione del traffico mobile non è solo utile al monitoraggio e alla gestione del traffico della rete, ma pone anche le basi per poter lavorare a contromisure più efficienti per la protezione della privacy.

1.3 Evoluzione delle tecniche di classificazione

Nel corso degli anni diversi sono stati gli sforzi per classificare flussi di traffico su di una rete. Le prime tecniche di classificazione, note come *port-based classification*,

prevedevano la ricerca di numeri di porto noti, così come assegnati dallo IANA (Internet Assigned Numbers Authority). Tali tecniche, tuttavia, sono risultate essere spesso inaffidabili poiché, da un lato, non riescono ad individuare traffico trasportato su porti non noti, e, dall'altro, falliscono nei confronti delle applicazioni che mascherano il loro traffico su porti noti. Un ulteriore ostacolo agli approcci port-based è costituito dalla traduzione degli indirizzi di rete e dei porti (Network Address Translation – NAT).

Successivamente le tecniche di classificazione si sono basate sull'ispezione del payload dei pacchetti, ed in particolare sul confronto di parti del payload con *signatures* di applicazioni note. Questi approcci, noti con il nome di *deep packet inspection* (DPI), non riescono, tuttavia, ad essere scalabili con il crescere del numero (e delle funzionalità) delle applicazioni, e non sono in grado di classificare traffico cifrato o incapsulato tramite tunneling [2].

Lo stato dell'arte delle tecniche di classificazione è costituito da approcci di tipo statistico, detti anche *flow-feature-based*: analisi statistica sulla grandezza dei pacchetti, analisi statistica sulla frequenza dei pacchetti, pattern recognition mediante algoritmi di machine learning. Questi ultimi possono generalmente essere raggruppati in due categorie differenti:

- Apprendimento supervisionato: in cui il computer impara a classificare a partire da un insieme di *training data* dato in input all'algoritmo.
- Apprendimento non supervisionato: in cui il computer riconosce da solo le differenti classi e vi assegna i vari flussi di traffico (*clustering*).

Poiché le applicazioni evolvono in maniera estremamente dinamica, è necessario che l'insieme di training data sia mantenuto costantemente aggiornato. In questo senso gli algoritmi ad apprendimento non supervisionato scalano molto meglio rispetto a quelli ad apprendimento supervisionato.

La sfida principale in tutte queste tecniche di classificazione è la scarsità di *ground truth* (ovvero di traffico classificato) per l'addestramento ed il testing degli algoritmi. In [2] si descrivono diverse possibili soluzioni per risolvere il problema. Si potrebbe, ad esempio, pensare di spostare il software di analisi alla sorgente dei dati (*"move the code to the data"*). Alternativamente, utenti volontari potrebbero annotare e validare il traffico da loro generato, in maniera tale da fornire un riferimento per la costruzione di ground truth (*crowdsourcing*). È proprio in quest'ultimo contesto, infatti, che si colloca il presente elaborato.

Tra tutti gli approcci di tipo statistico, promettenti sono quelli basati su algoritmi *deep learning* (DP). Questi algoritmi sono in grado di ricavare le correlazioni intrinseche presenti nei dati in maniera efficiente, minimizzando lo sforzo di preprocessing e riducendo la necessità di conoscere il dominio del problema. Inoltre gli algoritmi di DP si prestano bene ad essere eseguiti in parallelo su GPU, fornendo risultati soddisfacenti in un tempo dell'ordine dei millisecondi.

1.4 Esempi di soluzioni software esistenti

Per concludere accenneremo a due esempi di soluzioni software esistenti per la classificazione del traffico di rete.

Libprotoident [3] è un software open-source che opera una classificazione di tipo *lightweight packet inspection* (LPI). Con questo approccio solo i primi quattro byte del payload vengono analizzati (in entrambe le direzioni), cercando un matching con signatures note. Per questo motivo, la classificazione fornita dalla libreria richiede un

utilizzo minore delle risorse rispetto a metodi DPI, ma risulta essere anche meno affidabile.

L7-filter [4] è un classificatore per il framework Netfilter di Linux che identifica i pacchetti in base ai dati di livello applicativo, tramite l'analisi di espressioni regolari. Nasce soprattutto come classificatore di software peer-to-peer che utilizzano numeri di porta impredicibili. È in grado di classificare con successo messaggi di Kazaa, HTTP, Citrix, BitTorrent, FTP, etc.

Nel prossimo capitolo tratteremo in dettaglio TIE, il software utilizzato per la classificazione delle tracce di traffico catturate.

Capitolo 2

Strumenti utilizzati

2.1 Strumenti per la cattura

Per generare il traffico mobile sono stati utilizzati due dispositivi Android con privilegi di root, collegati via USB ad un terminale per la cattura del traffico. Al termine di ciascuna sessione di cattura viene prodotto un file .pcap (contenente i pacchetti catturati), un file strace.log (contenente le chiamate al sistema operativo), ed altri file relativi agli identificativi e ai metadati dei processi.

2.2 Strumenti per la classificazione

Il filtraggio delle tracce di traffico .pcap viene svolto dal terminale dei comandi con *tshark*, specificando i vari campi secondo cui filtrare il traffico.

La classificazione viene effettuata con il software TIE (Traffic Identification Engine) [6]. TIE è in grado di utilizzare diverse tecniche di classificazione simultaneamente, ciascuna implementata come modulo software caricabile dinamicamente (plugin). È inoltre in grado di operare sia in modalità offline che in modalità realtime.

TIE decompone il traffico di rete in sessioni. Di default vengono classificati i biflussi in modalità offline, prendendo in input la traccia .pcap. Può inoltre generare una ground truth basandosi su tecniche DPI, oppure può addestrare un plugin di classificazione con un insieme di dati pre-classificati (identificati dall'estensione .gt). Per il nostro lavoro si è utilizzato il plugin nDPIng, che classifica il traffico con un approccio di tipo DPI, basandosi sulla dimensione del pacchetto e sul contenuto del payload. Altri plugin disponibili sono Port (classificazione port-based), L7 (basato su L7-Filter), e PortLoad (che opera una classificazione di tipo LPI).

Capitolo 3

Cattura e analisi del traffico mobile

3.1 Cattura del traffico

Per generare il traffico sono state utilizzate quattro applicazioni su Android (Twitter, Duolingo, Foursquare, Instagram) in 24 sessioni da circa 5 minuti ciascuna, mediante uno Xiaomi MI5 e un Google Nexus 7.

In ciascuna sessione vengono catturati in un file .pcap i pacchetti trasmessi, i quali potranno essere successivamente filtrati tramite Wireshark e classificati con TIE. Vengono inoltre memorizzate tramite *strace* le chiamate al sistema operativo (salvate nel file strace.log). Di questo file saranno di particolare interesse tutte quelle operazioni di lettura e scrittura su una socket, poiché permetteranno in alcuni casi di determinare il processo associato ad una particolare quintupla.

Per semplicità di trattazione, utilizzeremo un identificativo numerico per fare riferimento ad una specifica traccia di traffico catturata. Tale identificativo numerico è associato ad una determinata cartella della cattura secondo la seguente tabella:

ID	CARTELLA	ID	CARTELLA
0	1540304978	6	1540309814
1	1540306490	7	1540310612
2	1540306955	8	1540310983
3	1540307842	9	1542732994
4	1540308462	10	1542733688
5	1540309356	11	1542733921

3.2 Script per l'automazione delle analisi

Il processo di automazione per la classificazione e l'analisi delle tracce di traffico è stato implementato in parte in linguaggio di scripting per Bash e in parte in Python3.6. Nella descrizione dello script procederemo seguendo l'ordine di esecuzione dei moduli che lo compongono. L'intero script viene eseguito per ciascuna cartella di traffico catturato tramite lo script run.sh in Bash.

3.2.1 Filtraggio delle tracce

Il filtraggio delle tracce .pcap passa attraverso l'esecuzione dello script run_tshark.sh in Bash. Questo script prende in input il file .pcap da filtrare e genera in output due file in formato CSV.

Il primo file è un elenco di tuple in formato <*streamid*, *ipsrc*, *ipdst*, *proto*, *tcp_srcport*, *tcp_dstport*, *udp_srcport*, *udp_dstport*, *ssl_hostname*, *http_hostname*>, generato filtrando i pacchetti registrati nella traccia.

Il secondo file è un elenco di coppie in formato <*addresses*, *hostname*>, generato filtrando le risoluzioni DNS presenti nella traccia. Verrà utilizzato per ricavare informazioni circa gli hostname contattati, dove necessario.

3.2.2 Preprocessing

Prima di poter analizzare i biflussi è necessario preprocessare entrambi i file ottenuti dal filtraggio delle tracce.

Dal file delle quintuple è necessario estrarre le quintuple appartenenti a biflussi distinti, considerando solo le porte dell'header TCP nel caso il protocollo sia TCP, oppure considerando solo le porte dell'header UDP nel caso il protocollo sia UDP. La prima operazione si può effettuare raggruppando l'elenco in base al campo *streamid*, mentre la seconda operazione si può effettuare considerando il numero di protocollo contenuto nel campo *proto*. La funzione è stata implementata in maniera tale da poter aggiungere, eventualmente, il supporto per altri header di livello trasporto. Il risultato di questa operazione è un file contenente tutti i biflussi secondo il formato <*streamid*, *ipsrc*, *ipdst*, *proto*, *srcport*, *dstport*, *ssl_hostname*, *http_hostname*>. Questo file, che chiameremo d'ora in poi file delle quintuple, costituirà il punto di partenza per il resto dell'analisi.

Dal file delle risoluzioni DNS è necessario separare tra di loro gli indirizzi IP contenuti nel campo *addresses* della stessa riga. Il risultato di questa operazione è un file contenente un elenco di coppie secondo il formato <*address*, *hostname*>.

3.2.3 Matching con TIE e strace.log

Giunti a questo punto è possibile eseguire il matching dei biflussi preprocessati con la classificazione effettuata da TIE. Per ciascuna riga contenuta nel file delle quintuple viene cercata una riga corrispondente allo stesso biflusso nel file generato da TIE. Se il matching è positivo, vengono apposti alla riga del file delle quintuple l'identicativo del biflusso di TIE e la classificazione prodotta. L'identificativo verrà utilizzato come chiave esterna per velocizzare la correzione della classificazione al termine della validazione manuale.

Il matching con lo strace.log consiste nel trovare il processo associato ad un dato biflusso. Da Python questo potrebbe essere fatto in diversi modi. Una prima soluzione sarebbe quella di scorrere linea per linea il file strace.log e, per ciascuna linea, cercare un matching all'interno del file delle quintuple. Questo metodo, tuttavia, può risultare molto lento se il file strace.log è troppo lungo e ci sono troppe quintuple da trovare. Una possibile ottimizzazione sarebbe quella di bufferizzare l'elenco di quintuple dal file, operare il matching su tale buffer ed infine eliminare dal buffer le quintuple man mano che queste vengono trovate. Così facendo il numero di quintuple da controllare tende a decrescere ad ogni iterazione.

Una seconda soluzione è quella di avviare un sottoprocesso shell per eseguire il comando *grep* sul file. Questo metodo è, in generale, più veloce di quello precedente, poiché grep utilizza l'algoritmo Boyer-Moore per la ricerca di una stringa [7]. Si potrebbe pensare di migliorare ulteriormente questa soluzione utilizzando la stessa ottimizzazione che abbiamo descritto per il primo metodo.

Abbiamo sperimentato con entrambe le soluzioni, e abbiamo riscontrato un netto aumento delle prestazioni con grep (anche nell'ordine dei minuti).

Il risultato finale è un file in formato CSV, in cui ciascuna riga ha i seguenti campi <*streamid*, *ipsrc*, *ipdst*, *proto*, *srcport*, *dstport*, *ssl_hostname*, *http_hostname*, ***tieid***, ***tie_class***, ***process***>.

3.2.4 Generazione delle informazioni sugli IP

L'ultimo passo è quello di associare a ciascuna riga nel file delle quintuple informazioni riguardanti l'indirizzo IP contattato. Questo permetterà di fare considerazioni circa la validità della classificazione prodotta da TIE e circa i servizi di terze parti utilizzati.

Per fare ciò, ciascun indirizzo IP di destinazione nel file delle quintuple viene cercato all'interno del file preprocessato delle risoluzioni DNS. Se il matching è positivo, viene apposto l'hostname alla riga corrispondente alla quintupla, altrimenti viene lanciato un sottoprocesso shell per eseguire la risoluzione DNS inversa tramite comando *dig*, sul server DNS pubblico di Google (8.8.8.8). Infine viene eseguito un sottoprocesso shell per cercare informazioni circa l'organizzazione cui corrisponde l'indirizzo IP, tramite comando *whois*.

Poiché questa fase richiede una risposta da parte di altri server, può rappresentare un pericoloso collo di bottiglia per l'intero script se le risposte sono troppo lente e/o se ci sono troppe quintuple. È possibile ottimizzare questa fase per successive iterazioni dello script implementando una cache locale di informazioni sugli IP di interesse. In questo modo sarà necessario contattare server esterni solo se l'informazione è assente nella cache.

Non solo, poiché queste informazioni sono relative ad un singolo indirizzo IP piuttosto che ad una intera quintupla, si può pensare di implementare una cache condivisa, in maniera tale che tutte le esecuzioni dello script sulle varie tracce di traffico facciano riferimento alla stessa cache.

Faremo un'analisi più accurata delle prestazioni dell'intero script nel paragrafo seguente, confrontando i tempi di risposta con e senza cache IP.

Il risultato finale è un file in formato CSV, in cui ciascuna riga ha i seguenti campi <*streamid*, *ipsrc*, *ipdst*, *proto*, *srcport*, *dstport*, *ssl_hostname*, *http_hostname*, *tieid*, *tie_class*, *process*, ***dst_hostname***, ***dst_orgname***>.

3.2.5 Analisi delle prestazioni dello script

Come già osservato nel precedente paragrafo, la generazione delle informazioni sugli IP può rallentare notevolmente l'esecuzione dello script, poiché richiede una risposta da parte di server esterni. Per questo motivo, svolgere un'analisi accurata delle prestazioni dello script richiederebbe innanzitutto un'analisi delle prestazioni della rete attraverso la quale vengono generate le informazioni. Ciononostante, può essere interessante osservare le differenze (in senso qualitativo) dei tempi di esecuzione quando viene o non viene utilizzata una cache locale di informazioni sugli IP.

È stata dapprima misurata l'esecuzione dello script quando non viene utilizzata una cache di informazioni sugli IP. Questa condizione può essere simulata svuotando la cache prima di eseguire lo script su ciascuna traccia.

Successivamente è stata misurata l'esecuzione con la cache. Questa condizione corrisponde ad un primo avvio dello script, ovvero di una cache che si riempie man mano che le tracce vengono elaborate. Affinché la misurazione dei tempi di esecuzione sia corretta ad ogni iterazione, è necessario svuotare la cache prima di lanciare lo script sulla prima traccia.

Infine è stata misurata l'esecuzione pre-caricando la cache con tutti i dati. Questa condizione può essere simulata facendo una esecuzione completa dello script prima di avviare la misurazione dei tempi.

Nelle seguenti tre tabelle si riportano i dati ottenuti, rispettivamente nelle tre modalità di esecuzione appena descritte. Si riportano inoltre i tempi di risposta per il preprocessing delle risoluzioni DNS, per il preprocessing del file delle quintuple, e per il matching con i file TIE e strace.log.

Lo script è stato eseguito su una macchina virtuale Linux Ubuntu 64-bit con 2 CPU e 4 GB RAM dedicati, installata su una macchina Windows 10 con processore Intel Core i5-6198DU a 2.8 GHz, 8 GB RAM. La misura dei tempi di esecuzione è stata automatizzata in Python tramite il modulo *timeit*, facendo 10 iterazioni e mediando i risultati ottenuti.

INFORMAZIONI GENERALI				ESECUZIONE SENZA CACHE IPINFO (secondi)				
ID TRACCIA	MINUTI CATTURA	NUMERO QUINTUPLE	LINEE STRACE	DNS PREPROCESSING	QT PREPROCESSING	TIESTRACE	IPINFO	TOTALE
0	7	853	85060	0.0010	0.4686	14.7394	8.3943	23.6034
1	5	114	99790	0.0008	0.0498	1.7762	4.7999	6.6266
2	5	344	69847	0.0008	0.1676	4.7909	47.5319	52.4913
3	3	149	5	0.0007	0.2790	0.4340	21.7763	22.4900
4	12	1585	211501	0.0019	1.8811	65.5013	17.3074	84.6916
5	5	691	125564	0.0011	0.5644	18.1627	12.9773	31.7055
6	7	730	143693	0.0012	0.9632	21.6570	15.6214	38.2429
7	5	469	98630	0.0011	0.2158	8.9678	12.9445	22.1292
8	5	530	116378	0.0012	0.3541	12.3077	12.9080	25.5710
9	5	109	80832	0.0007	0.2909	3.0978	2.9982	6.3875
10	3	65	145655	0.0008	0.1773	1.8563	2.6801	4.7145
11	4	77	148606	0.0007	0.2379	2.4236	1.4451	4.1073
TOT				**0.012**	**5.650**	**155.715**	**161.384**	**322.761**

INFORMAZIONI GENERALI				ESECUZIONE NORMALE CON CACHE IPINFO (secondi)				
ID TRACCIA	MINUTI CATTURA	NUMERO QUINTUPLE	LINEE STRACE	DNS PREPROCESSING	QT PREPROCESSING	TIESTRACE	IPINFO	TOTALE
0	7	853	85060	0.0012	0.4769	14.6850	9.2457	24.4088
1	5	114	99790	0.0006	0.0457	1.7112	2.8148	4.5724
2	5	344	69847	0.0009	0.1792	4.9586	50.4772	55.6159
3	3	149	5	0.0007	0.2701	0.4295	12.8659	13.5662
4	12	1585	211501	0.0018	1.8466	64.4258	14.0836	80.3578
5	5	691	125564	0.0011	0.5738	18.3489	5.3008	24.2246
6	7	730	143693	0.0013	0.9510	21.5462	2.5395	25.0380
7	5	469	98630	0.0014	0.2324	8.9843	4.7358	13.9539
8	5	530	116378	0.0013	0.3409	12.0709	2.5016	14.9147
9	5	109	80832	0.0008	0.2847	3.0221	1.6621	4.9697
10	3	65	145655	0.0007	0.1695	1.8427	1.7376	3.7505
11	4	77	148606	0.0009	0.2451	2.4431	0.3105	2.9996
TOT				**0.013**	**5.616**	**154.468**	**108.275**	**268.372**

INFORMAZIONI GENERALI				ESECUZIONE CON PRECARICAMENTO CACHE IPINFO (secondi)				
ID TRACCIA	MINUTI CATTURA	NUMERO QUINTUPLE	LINEE STRACE	DNS PREPROCESSING	QT PREPROCESSING	TIESTRACE	IPINFO	TOTALE
0	7	853	85060	0.0012	0.4756	14.6638	0.0139	15.1545
1	5	114	99790	0.0008	0.0457	1.7263	0.0030	1.7758
2	5	344	69847	0.0008	0.1671	4.7497	0.0075	4.9252
3	3	149	5	0.0007	0.2670	0.4287	0.0046	0.7010
4	12	1585	211501	0.0017	1.8611	64.0010	0.0326	65.8964
5	5	691	125564	0.0012	0.5768	18.1494	0.0195	18.7469
6	7	730	143693	0.0014	0.9688	21.5788	0.0185	22.5675
7	5	469	98630	0.0013	0.2214	8.9150	0.0130	9.1507
8	5	530	116378	0.0012	0.3617	12.2049	0.0142	12.5821
9	5	109	80832	0.0009	0.2860	3.0206	0.0037	3.3112
10	3	65	145655	0.0008	0.1658	1.8245	0.0028	1.9938
11	4	77	148606	0.0007	0.2397	2.4560	0.0029	2.6993
TOT				**0.013**	**5.637**	**153.719**	**0.136**	**159.504**

Come è lecito aspettarsi, il tempo di esecuzione è principalmente funzione del numero di quintuple e del numero di linee nello strace.log.

L'utilizzo di una cache locale di informazioni sugli indirizzi IP permette in generale di velocizzare l'esecuzione dello script. Nelle condizioni in cui si ha il 100% delle cache hit, il tempo di fetching delle informazioni sugli indirizzi IP è praticamente trascurabile (0.136 secondi).

Si può pensare di migliorare ulteriormente le prestazioni dello script eseguendo in parallelo il matching con TIE, il matching con lo strace.log e la generazione delle informazioni sugli IP. Tuttavia, per prestazioni dello script ottimali, è comprensibile perché convenga non superare il limite di cinque minuti di utilizzo di un'applicazione.

3.3 Validazione manuale

Al termine dell'esecuzione dello script, ciascuna cartella della cattura contiene un file delle quintuple denominato qtuples.myfinal. A questo punto è possibile procedere con la validazione manuale della classificazione prodotta da TIE. La validazione viene effettuata a partire dal file delle quintuple, prendendo in considerazione solo le quintuple con campo *proto* pari a 6 (TCP), e apponendo, se necessario, la classificazione corretta al termine di ciascuna riga.

Per identificare l'applicazione associata ad una data quintupla, è sufficiente analizzare (dove presente) il valore del campo *process* ottenuto dal matching con lo strace.log. Ad esempio, la riga <*edge-mqtt.facebook.com, 13,* ***Facebook***, ***com.instagram.android***, *edge-mqtt-shv-01-mrs1.facebook.com, Facebook*> ha come processo com.instagram.android. In questo caso la classificazione prodotta da TIE (Facebook) è da considerarsi sbagliata.

In assenza dell'informazione sul processo, non è possibile classificare accuratamente un biflusso. Ad esempio, se si considera la riga <*..., edge-mqtt.facebook.com, 22, Other TCP,* ***UNKNOWN_PROCESS***, *edge-mqtt-shv-01-mxp1.facebook.com, Facebook*>, sarebbe sbagliato classificare il biflusso come Facebook, perché non è detto che i domini di Facebook vengano contattati esclusivamente dall'applicazione Facebook. In questo caso il biflusso non è classificabile con i metodi utilizzati e viene apposto UNKNOWN.

In questo senso, poiché lo strace.log della traccia num. 3 conteneva solo cinque linee (probabilmente a causa di qualche malfunzionamento durante la cattura), non è stato possibile validare la maggior parte dei suoi biflussi.

Quando è possibile classificare un biflusso, si possono determinare informazioni aggiuntive considerando i valori dei campi *ssl_hostname*, *http_hostname*, *dst_hostname*, e *dst_orgname*. Ad esempio, dalla riga <..., ***www.googleadservices.com***, *231, SSL_with_certificate,* ***com.duolingo***, *mil04s27-in-f2.1e100.net.;mil04s27-in-f130.1e100.net, Google LLC*> è possibile dedurre che Duolingo abbia usufruito dei servizi di GoogleAds. In questo caso si è apposto alla riga il valore Duolingo+GoogleAds.

In alcuni casi particolari è possibile dedurre i servizi interni ad un'applicazione. Ad esempio, se si considera la riga <***video.twimg.com***, *128, Twitter,* ***com.twitter.android***, *UNKNOWN_HOSTNAME, MCI Communications Services, Inc. d/b/a Verizon Business*>, è possibile ipotizzare che il biflusso sia associato alla visualizzazione di un video all'interno di Twitter (video.twimg.com). In questo caso la classificazione è stata lasciata invariata.

La validazione manuale produce un file denominato validated.txt in formato CSV, contenente i campi <*streamid*, *ipsrc*, *ipdst*, *proto*, *srcport*, *dstport*, *ssl_hostname*, *http_hostname*, *tieid*, *tie_class*, *process*, *dst_hostname*, *dst_orgname*, ***new_class***>.

3.4 Script per la correzione della classificazione

La validazione manuale ha prodotto un file validated.txt contenente nell'ultima colonna la classificazione corretta del biflusso. Scopo dello script per la correzione della classificazione (gt.py) è quello di generare un file formalmente identico a quello prodotto da TIE, ma con la classificazione aggiornata. In questo modo è possibile visualizzare velocemente le classificazioni corrette ed eventualmente usare il file generato come *training data* per TIE.

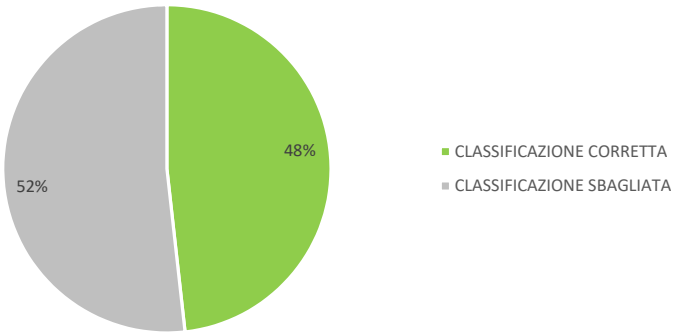
Il risultato finale è un file con estensione .gt.tie. Dove la classificazione è stata cambiata, è stato aggiunto un commento per segnare la classificazione che vi era precedentemente.

Capitolo 4

Risultati sperimentali

4.1 Google

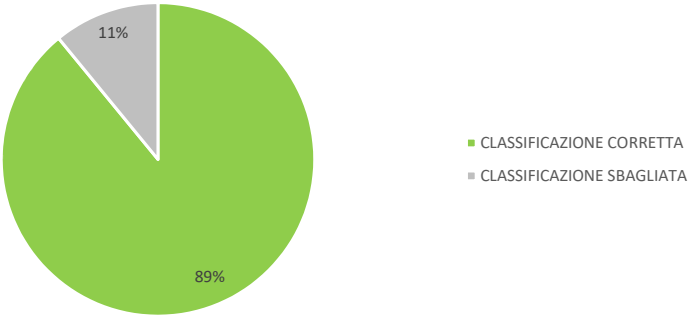
Il traffico associato a Google è stato principalmente osservato nella traccia num. 0, in cui si è scaricato e installato Twitter tramite Google Play. Si è dedotto che i pacchetti che contattavano i server cache di Google (*cache.google.com*) fossero associati al download dell'applicazione.
Nelle altre tracce, in assenza di un processo associato ad una data quintupla, è stato spesso necessario annullare la classificazione Google prodotta da TIE.



Non sono stati individuati biflussi di Google Play associabili all'utilizzo di servizi di terze parti, tuttavia i servizi di Google sono stati spesso utilizzati dalle altre applicazioni (in particolare GoogleAds).

4.2 Twitter

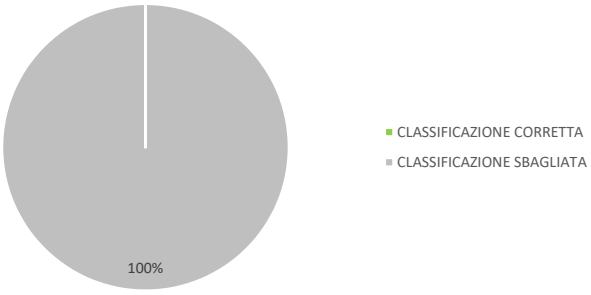
Le tracce num. 1, 2, 3 contengono traffico classificabile come Twitter. È interessante notare come Twitter venga spesso classificato correttamente da TIE. Questo ci porta a concludere che Twitter si presta abbastanza bene ad essere classificato con approcci di tipo DPI.

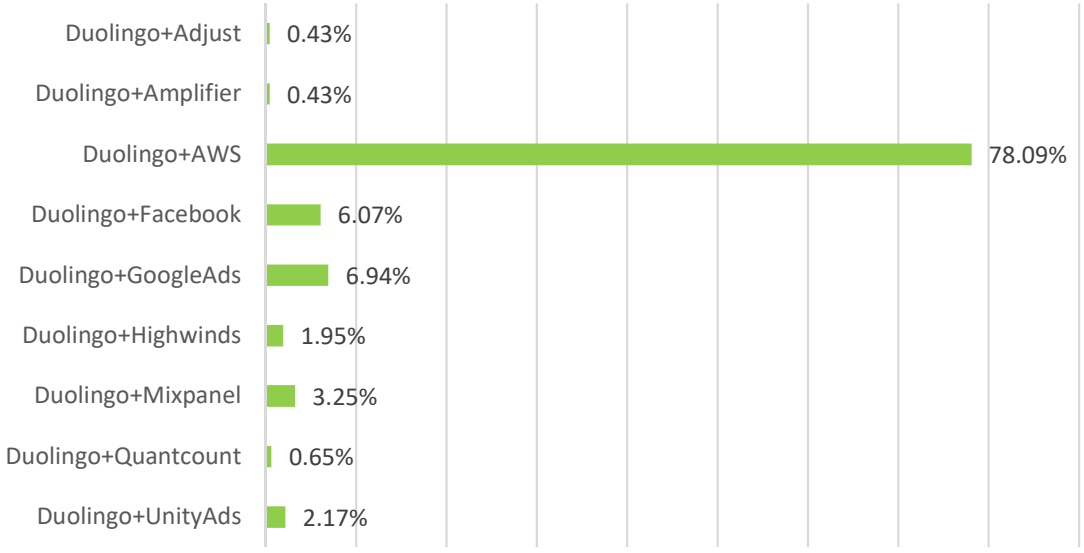


Durante l'esecuzione di Twitter sono stati osservati biflussi associabili a servizi di *analytics* (Twitter+TwitterAnalytics, 2%) e ad Amazon Web Services (Twitter+AWS, 4%). In alcuni casi è stato possibile dedurre delle specifiche azioni all'interno di Twitter (in particolare, la visualizzazione di video) con la tecnica descritta nel paragrafo 3.3, tuttavia sarebbe necessario confermare questo risultato con esperimenti più mirati.

4.3 Duolingo

Le tracce num. 4, 5, 6, 7, 8 contengono traffico classificabile come Duolingo. In alcuni casi TIE non è riuscito a fare alcuna classificazione, in altri casi la classificazione è stata sbagliata. Le cause vanno ricondotte all'assenza di signature note per Duolingo e alla presenza di traffico cifrato. Ad esempio, con riferimento alla traccia num. 4, TIE ha classificato come Facebook un biflusso che comunicava con *graph.facebook.com*, tuttavia dal confronto con lo strace.log si è trovato un match con il processo *com.duolingo*. In questo caso si è concluso che Duolingo avesse utilizzato servizi di terze parti di Facebook. Questo dimostra il limite delle tecniche basate su DPI.





Duolingo rappresenta un caso di studio interessante: i risultati ottenuti sono in accordo con la ricerca condotta da Seneviratne et al. [8], secondo cui le applicazioni gratuite tendono a comunicare con un numero elevato di servizi di advertising e di analytics (*trackers*). Duolingo utilizza soprattutto gli Amazon Web Services (AWS): l'applicazione, infatti, è dipendente da Amazon DynamoDB per lo storage, da Amazon EC2 per il computing, da Amazon ElastiCache per le performance, da Amazon Redshift per i servizi di data analytics, ed altro [9].

Durante l'analisi delle tracce di Duolingo sono stati anche individuati biflussi associati all'esecuzione di Google Play (*com.android.vending*), nonostante non fosse presente tra le applicazioni in background.

4.4 Instagram

Le tracce num. 9, 10, 11 contengono traffico classificabile come Instagram. Anche qui TIE non è riuscito a classificare correttamente alcun biflusso. Molti biflussi sono stati classificati da TIE (erroneamente) come Facebook, poiché comunicavano con domini di Facebook. Il confronto con lo strace.log ci ha nuovamente permesso di discriminare la provenienza dei biflussi.

Il 30% del traffico classificato come Instagram è associato ad un utilizzo dei servizi di Facebook (Instagram+Facebook), coerentemente col fatto che Instagram è posseduto da Facebook.

4.5 Foursquare

Non ci sono pervenute tracce contenenti traffico generato con Foursquare. Nonostante ciò ipotizziamo di ottenere dei risultati simili a quelli di Duolingo, sia in termini di percentuali di biflussi classificati correttamente sia in termini di utilizzo di servizi di terze parti.

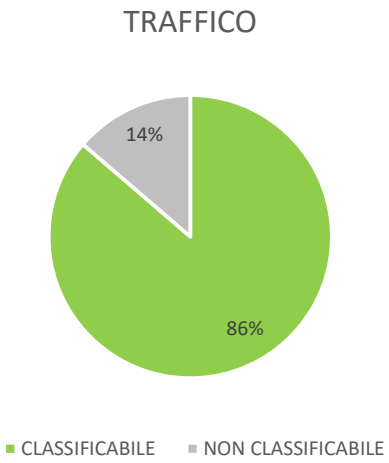
4.6 Booking

Sebbene l'applicazione Booking non sia stata utilizzata durante le sessioni di cattura, nelle tracce num. 0, 1 sono state individuate delle quintuple associate al processo *com.booking:bookingcomprocess* con hostname di destinazione *iphone-xml-l.booking.com*, mentre nelle tracce 4, 5, 6, 7, 8 sono state individuate delle quintuple con processo sconosciuto e hostname di destinazione ancora *iphone-xml-l.booking.com*.

Gomez et al. [5] hanno dimostrato come gli *Android intents* permettano alle applicazioni di accedere indirettamente a risorse non autorizzate delegando applicazioni autorizzate. Non è chiaro, dunque, se la presenza di biflussi di Booking sia dovuta ad un processo non completamente terminato o se sia dovuta all'esecuzione di *Android intents* da parte di Google Play, Twitter o Duolingo. In tutti i casi TIE non è stato in grado di classificare correttamente i biflussi di Booking.

Conclusioni

La validazione manuale delle tracce di traffico ha mostrato come i metodi di classificazione basati su approcci di tipo DPI diano risultati quasi sempre inesatti o incompleti. Questo è dovuto al fatto che il payload della maggior parte dei pacchetti è cifrato. Quando ciò accade TIE *"cerca di fare il suo meglio"* per far combaciare parti del pacchetto con signatures di applicazioni note. È stato ad esempio osservato come biflussi che contattavano domini associati con Facebook venissero classificati come Facebook, sebbene fossero in realtà associabili ad altre applicazioni.



Sono due le considerazioni da fare a proposito della classificabilità delle tracce di traffico. In primo luogo ciò che permette di fare una classificazione più accurata rispetto a TIE è il matching con il file strace.log. Quando il matching è positivo, infatti, è possibile quasi immediatamente dedurre l'applicazione associata ad una quintupla. Le informazioni riguardanti gli indirizzi IP contattati permettono di dedurre approssimativamente i servizi di terze parti utilizzati dall'applicazione, ma non consentono di determinare l'applicazione in assenza di un matching con lo strace.log.

In secondo luogo bisogna considerare il fatto che le tecniche di classificazione basate sul matching con lo strace.log hanno un campo di applicazione molto limitato. Non possono, ad esempio, essere utilizzate dagli operatori di rete per monitorare il traffico, poiché le informazioni riguardanti le chiamate al sistema operativo sono tipicamente presenti solo sugli end-systems.

Ciononostante queste tecniche risultano essere uno strumento semplice e veloce per la generazione di una ground truth da parte degli utenti finali, permettendo l'addestramento degli algoritmi di machine learning. Il passo successivo è quello di integrare le tecniche già note con algoritmi di apprendimento non supervisionato, in maniera tale da poter rispondere efficacemente alle esigenze di una rete Internet sempre più eterogenea e sempre più in cambiamento.

Bibliografia

[1] Techcrunch, *https://techcrunch.com/2016/06/13/apples-app-store-hits-2m-apps-130b-downloads-50b-paid-to-developers/*, acceduto l'ultima volta il 02/12/2018

[2] Dainotti, Alberto, Antonio Pescapè, Kimberly C. Claffy, *Issues and future directions in traffic classification.*, IEEE network 26.1, 2012

[3] Shane Alcock, Richard Nelson, *Libprotoident: Traffic Classification Using Lightweight Packet Inspection*, 2012

[4] L7-Filter, *https://l7-filter.sourceforge.net/*, acceduto l'ultima volta il 03/12/2018

[5] Xuetao Wei, Lorenzo Gomez, Iulian Neamtiu, Michalis Faloutsos, *ProfileDroid: Multi-layer Profiling of Android Applications*, 2012

[6] TIE, *http://tie.comics.unina.it/*, acceduto l'ultima volta il 03/12/2018

[7] FreeBSD, *https://lists.freebsd.org/pipermail/freebsd-current/2010-August/019310.html*, acceduto l'ultima volta il 06/12/2018

[8] Suranga Seneviratne, Harini Kolamunna, Aruna Seneviratne, *A measurement study of tracking in paid mobile applications*, 2015

[9] Amazon, *aws.amazon.com/it/solutions/case-studies/duolingo-case-study-dynamodb/*, acceduto l'ultima volta il 08/12/2018

[10] Bohmer, Hecht, Shoning, Kruger, Bauer, *Falling Asleep with Angry Birds, Facebook and Kindle – A Large Scale Study on Mobile Application Usage*, 2011

[11] Mauro Conti, Luigi Vincenzo Mancini, Riccardo Spolaor, Nino Vincenzo Verde, *Analyzing Android Encrypted Network Traffic to Identify User Actions*, IEEE Transactions On Information Forensics and Security 11.1, 2016