UNIVERSITÀ DI PISA

DIPARTIMENTO DI INFORMATICA

Analisi e Caratterizzazioni del Traffico di Rete mediante Tecniche di DPI

Anno Accademico 2021/2022

Autore	**Tutore Accademico**	**Tutore Aziendale**
Alberto Martino	Luca Deri	Raffaele Borgese

Ringrazio il professor Luca Deri che, con la sua guida esperta e passione per la materia, è stato e sarà di grande ispirazione per questa esperienza e quelle a venire. Ringrazio altresì Raffaele Borgese, Alessandro Piazza, Giovanni Cozzi e Shabi Hashemi di IES Italia per l'assistenza, l'accoglienza e la loro disponibilità, oltre che per i loro utilissimi insegnamenti. Dedico infine questa tesi a mio padre che confido sarebbe stato fiero di me nel vedermi completare quel percorso di studi che, assieme a mia madre, ha sempre trovato il modo di supportare con orgoglio.

Indice

Introduzione

Per alcuni gestori è indispensabile poter caratterizzare il traffico sulle reti di propria competenza: un'analisi qualitativa fornisce infatti informazioni discriminanti che consentono di facilitare, limitare o impedire certe conversazioni di rete. A titolo di esempio, basti pensare alla telemedicina per i paesi in via di sviluppo [7] o anche alla copertura su veicoli aerei [34]: entrambi casi estremamente complessi che richiedono una valutazione approfondita del traffico che non può quindi essere limitata all'osservazione di indirizzi MAC (Media Access Control), IP (Internet Protocol), porte o anche ai soli header dei pacchetti [8, 1].

Al fine di ottenere risultati informativi dall'analisi del traffico di rete è possibile ricorrere alla DPI (Deep Packet Inspection) mediante la quale si processano i dati contenuti specialmente nel payload dei pacchetti osservati, talvolta caratterizzando il traffico anche da un punto di vista di tipologia dei servizi (e.g. streaming video, videogiochi, pubblicità). Tuttavia, nonostante molti applicativi che offrono DPI prevedano già una larga platea di regole per la profilazione del traffico, affinché una caratterizzazione personalizzata abbia luogo, il gestore deve definire delle regole e ciò richiede del tempo: delle risorse umane devono essere impiegate per studiare il traffico, quindi decidere se aggiungere o modificare alcune regole; inoltre, affinché tali definizioni non divengano obsolete, è preferibile aggiornarle frequentemente considerando anche solo che un qualsiasi utente potrebbe voler utilizzare un qualsiasi servizio di rete e che di servizi di rete e di utenti ve ne sono presumibilmente sempre di nuovi [14, 15].

Lo scopo di questo progetto è realizzare un sistema che agevoli la profilazione del traffico di rete, estraendo dati utili mediante DPI e fornendo regole di caratterizzazione in modo semi-automatico. In questa tesi viene esposto e documentato il progetto dalle fasi iniziali di comprensione del problema fino al sistema che lo risolve, trattando le difficoltà incontrate e approfondendo le soluzioni più peculiari.

Prima di procedere con il corpo centrale della relazione, si ritiene opportuno introdurre a tutti quegli elementi fondamentali per la comprensione del caso di studio, nonché ai dettagli riguardanti la modalità di tirocinio. La seguente non è da intendersi come trattazione esaustiva delle tecnologie in uso né come presentazione completa dei soggetti coinvolti: piuttosto si danno cenni utili a contestualizzare l'esposizione - ben più approfondita nel capitolo successivo - del lavoro svolto.

Soggetti e Tecnologie Coinvolti

L'attività è stata svolta contestualmente a IES-ITALIA, rispetto al servizio Seafy. Tecnologie utilizzate dall'azienda, soggetti coinvolti e relazioni d'interesse sono presentate in questo capitolo e schematizzate nella seguente immagine.

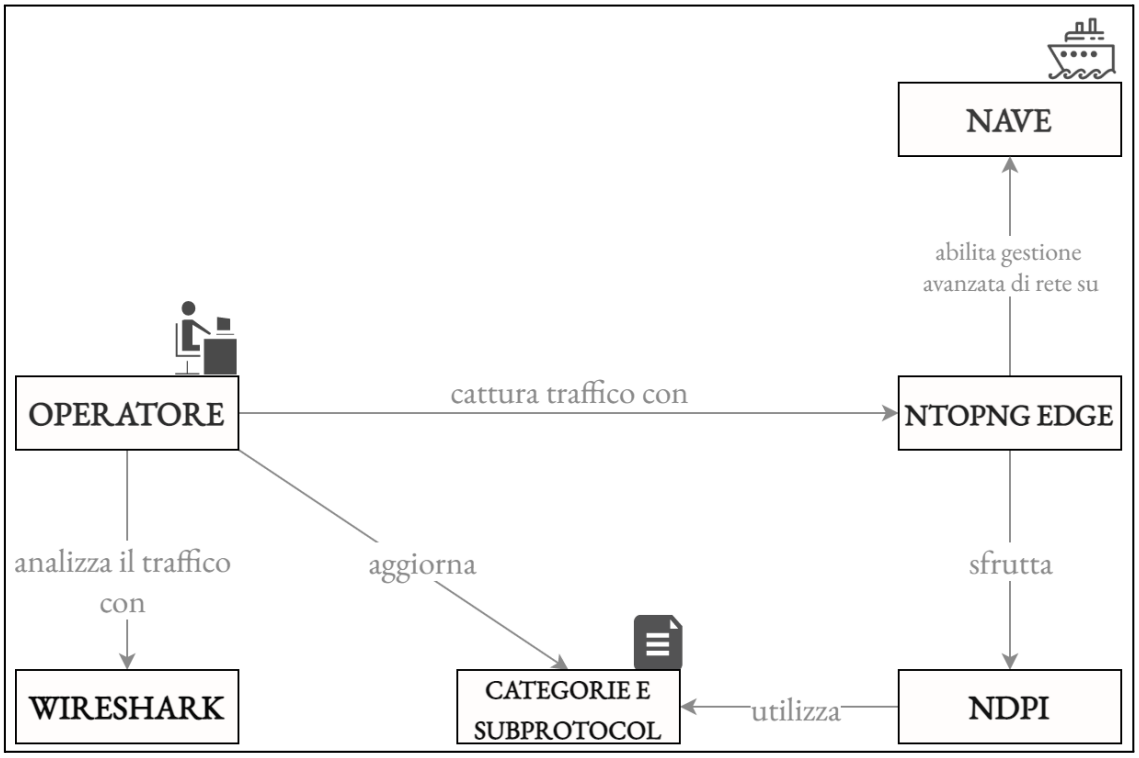


immagine 1: schematizzazione dei soggetti/tecnologie coinvolti (rappresentati da rettangoli etichettati) e relazioni d'interesse al progetto (rappresentate da frecce etichettate).

IES-Italia

IES è un'azienda italiana che fornisce servizi e prodotti informatici per privati ed altre aziende sfruttando tecnologie all'avanguardia; costituita da giovani talenti ed esperti nel settore, essa si concentra principalmente sullo streaming di contenuti audiovisivi e sulla gestione di reti private. Di IES non è stato coinvolto solo il tutore aziendale ma anche degli operatori, sia direttamente interessati al progetto, sia estranei ad esso al fine di fornire supporto e monitorare gli sviluppi.

Seafy

Seafy è un servizio grazie al quale compagnie di trasporto marittimo come Corsica Ferries e Grimaldi Lines forniscono connessione internet via satellitare a clienti ed equipaggio (previa sottoscrizione di abbonamenti) laddove in passato ciò non era possibile.

Ntopng Edge

A supporto del servizio Seafy, gioca un ruolo centrale ntopng Edge; su ogni nave è in esecuzione l'applicativo che consente in particolare di:

- imporre politiche aziendali per utente in base all'abbonamento sottoscritto abilitando, limitando o impedendo selettivamente certo traffico
- offrire monitoraggio avanzato di rete effettuando packet sniffing e ispezioni profonde dei pacchetti (DPI), raccogliendo statistiche accessibili anche tramite interfaccia grafica web
- effettuare il dump di una breve cattura di traffico applicando un filtro BPF (Berkeley Packet Filter; utile a catturare solo traffico d'interesse)

NDPI e Regole di Caratterizzazione

La libreria ndpi serve le funzionalità di DPI utilizzate da ntopng Edge e dal programma **ndpiReader** che, come sarà illustrato in seguito, fornirà risultati importanti sulle catture di traffico. ndpi, inoltre, consente di definire dei criteri personalizzati per la caratterizzazione del traffico da un punto di vista applicativo (i cosiddetti subprotocol) e di tipologia dei servizi (le categorie).

Un **subprotocol** designa le caratteristiche che identificano un servizio applicativo (e.g. YoutubeUpload, Facebook, InstagramVideo) mentre quando si parla di **categoria** ci si riferisce alla tipologia di servizio applicativo alla quale una certa conversazione di rete può essere associata (e.g. Streaming, Social Network, Videogiochi); in questo elaborato, subprotocol e categorie sono riferite genericamente come *regole di caratterizzazione*. È importante precisare che una regola di caratterizzazione può basarsi anche semplicemente sull'hostname

individuato (*hostname-based rules*): se, ad esempio, dall'analisi di una conversazione viene estratto un hostname allora tale informazione può essere utilizzata come discriminante. Come ultimo appunto, le categorie saranno talvolta riferite come *classi* quando trattate in riferimento al classificatore realizzato per questo progetto; si potrebbe discorrere su quale termine possa essere più appropriato, ma si esula da tale trattazione in quanto non rilevante per il caso di studio in essere, confidando di non creare ambiguità che minino la comprensione dell'elaborato.

Wireshark

Wireshark è un software per l'analisi di pacchetti di rete utilizzato dall'azienda grazie e al quale è possibile estrapolare dati visualizzabili anche da interfaccia grafica personalizzabile; consente la generazione di grafici, fornisce informazioni statistiche, permette di risolvere alcuni hostname da indirizzi IP e molto altro. In questo progetto sarà utilizzato per le analisi preliminari dei dati.

Il Caso di Studio

Impiegare delle risorse umane per svolgere determinati compiti richiede un investimento in tempo e quindi anche in denaro; si possono adottare le tecnologie migliori e farle utilizzare ad individui altamente qualificati ma se non si adotta una strategia efficace si finisce per ridurre l'efficienza della propria attività, fino addirittura a minarne le probabilità di sopravvivenza [11]. È proprio per questo motivo che IES continua a ricercare innovazione non solo nelle proprie risorse ma anche nei propri metodi e strategie; in questo capitolo si illustra il caso di studio presentando premesse ed obiettivi che lo caratterizzano.

Lo Stato Attuale

L'azienda (IES) necessita di caratterizzare il traffico in alcune tipologie di servizi per poter imporre le politiche previste dalle sottoscrizioni Seafy impedendo, ad esempio, lo streaming video. Per fare ciò, gli operatori analizzano manualmente dei campioni di traffico sfruttando Wireshark, quindi individuano gli hostname in chiaro e decidono se aggiungere o modificare regole di caratterizzazione. Per identificare la tipologia di un servizio associato ad un hostname non familiare all'operatore, quest'ultimo effettua delle verifiche servendosi di motori di ricerca (in particolare, di *Google*).

Interagendo con l'azienda emerge che vi sono alcune categorie specifiche che si vorrebbe poter riconoscere nel traffico (fra queste lo streaming video e i videogiochi); tuttavia, è posta molta attenzione riguardo il progresso e la diffusione di internet - sia in termini quantitativi che qualitativi - e quindi le due categorie sopracitate potrebbero non essere più le sole d'interesse in futuro.

L'Obiettivo

L'obiettivo da raggiungere è quindi quello di realizzare un sistema che snellisca i tempi impiegati per l'aggiornamento delle regole di caratterizzazione basate su hostname lasciando agli operatori la possibilità di affinare i risultati secondo le proprie preferenze; durante lo svolgimento del progetto sono emersi dei requisiti funzionali aggiuntivi che verranno documentati in itinere.

Modalità di Tirocinio

Il progetto è durato poco più di 300 ore ed è stato svolto in autonomia; sono stati indispensabili i riscontri e le indicazioni del tutor accademico, nonché decisamente utile il supporto fornito dall'azienda, sempre disponibile. Esclusa una settimana lavorativa passata nella sede aziendale (Milano), il lavoro è stato svolto in remoto con revisioni (da parte dell'azienda, del tutor o di ambo le parti) almeno bisettimanali.

Gli obiettivi prefissati nella proposta di tirocinio sono stati raggiunti con la realizzazione di un software per il quale si sono sfruttate conoscenze e competenze sia già acquisite in ambito universitario ed extrauniversitario, sia apprese in itinere.

Considerazioni Preliminari

Prima di dedicarsi allo sviluppo di una strategia, si è ritenuto opportuno svolgere delle ricerche e considerazioni preliminari al fine di delineare meglio alcune caratteristiche centrali. Comprendere più a fondo le relazioni di base fra categorie, servizi e subprotocol e localizzare le informazioni utili a definire regole di caratterizzazione si è rivelato essere di fondamentale importanza. Nei successivi capitoli verranno quindi introdotte riflessioni e caratteristiche fondamentali alla comprensione del caso; saranno presentate solo parzialmente, invece, alcune analisi preliminari utili soltanto ad esemplificare alcuni concetti chiave.

Categorie, Servizi Applicativi e Hostname

Utilizzando un computer, uno smartphone o, più in generale, un qualsiasi dispositivo che consente la navigazione in rete tramite browser dotato di interfaccia grafica, si è abituati a riconoscere servizi in base al loro nome (e.g. Google Documents) e all'hostname associato (e.g. docs.google.com) spesso individuato nell'indirizzo HTTP (*HyperText Transfer Protocol*) ma, come si può immaginare, ad un livello di astrazione più basso non è tutto così semplice come sembra. Inoltre, astrarre un sistema in modo non rigoroso, magari limitandosi all'intuito o ad un modello concettuale soggettivo comporta innumerevoli difficoltà perché facilmente soggetto ad interpretazione, ad ambiguità e all'impossibilità di stabilire degli standard. In questo capitolo si forniscono cenni utili e riflessioni circa le categorie, i servizi applicativi e gli hostname, mettendo l'accento sulla loro cardinalità e relazioni funzionali, sia in termini generali che più propriamente riguardo ai software per la DPI.

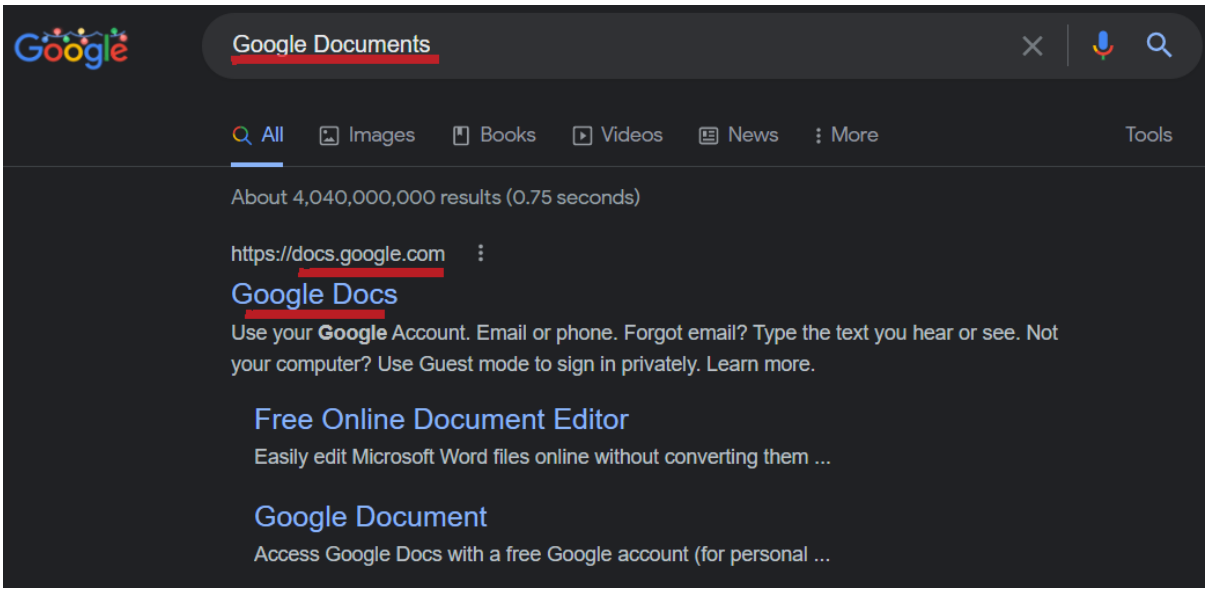


immagine 2: esempio di una ricerca utente del servizio Google Documents su browser Chrome da PC tramite motore di ricerca Google

Instaurare una Connessione con un Host Remoto

La connessione ad un host remoto non viene stabilita fra hostname o nomi generici ma fra interfacce di rete: un servizio è fornito da uno o più host ognuno dei quali è raggiungibile riferendosi ad un'etichetta numerica associata, ovvero il suo indirizzo IP pubblico (si veda *immagine 3* per un esempio). Quando si accede ad un servizio attraverso il suo hostname, questi viene risolto in un indirizzo IP pubblico che lo identifica; tale astrazione regala un modello concettuale semplice che, in particolare, rende dispensabile il ricordarsi decine di sequenze numeriche; a supporto di ciò entra in gioco il DNS (Domain Name System) che serve la risoluzione di identificativi in indirizzi IP. HTTP e DNS verranno approfonditi nel capitolo successivo.

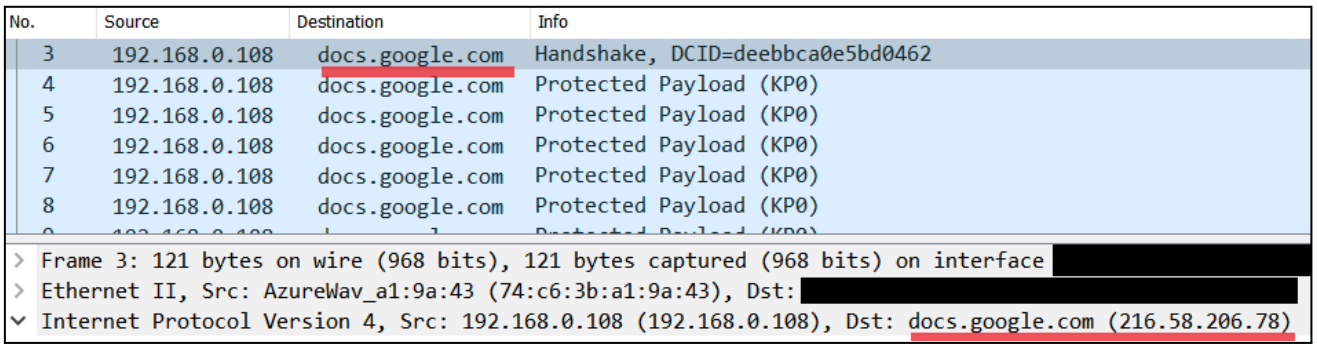


immagine 3: elenco dei primi pacchetti scambiati a seguito dell'accesso di un utente al servizio Google Documents; in evidenza l'hostname di destinazione e il relativo indirizzo IP pubblico risolto; infografica catturata su Wireshark

Cardinalità degli hostname

Di hostname ne esistono presumibilmente più di 1.5 miliardi [12] e ogni giorno potrebbero essercene di nuovi [15] ma, essendo unici, ognuno di lunghezza finita e costituiti da un alfabeto finito [17], in qualsiasi istante presente o futuro che sia, questi sono numerabili e costituiscono un insieme finito.

Cardinalità dei servizi di Rete

Un *servizio di rete* è un sistema software progettato per supportare l'interazione interoperabile da una macchina all'altra [39] e un *sistema software* consiste in un numero di programmi separati, file di configurazione e documentazione [37]. Tuttavia, non vi è un'autorità specializzata nel definire quale sia l'esatto sottoinsieme di programmi che definisce un servizio: per quanto la compagnia offerente possa concepire e magari proporre il proprio sistema software come un sottoinsieme specifico di programmi, un gestore di rete o anche più semplicemente un qualsiasi utente che utilizza il servizio potrebbe voler (o dover) concepire il sistema in modo diverso; in aggiunta, si potrebbe concepire un sistema software anche in base alle funzionalità che questi offre. Vista questa condizione e considerando il fatto che un servizio possa essere identificato con un nome arbitrario qualsiasi (e.g. "Google Docs" potrebbe anche essere riferito come "Documenti Google"), non è possibile, in un qualsiasi istante, stabilire il numero assoluto di servizi applicativi identificati in modo univoco e, in particolare, potremmo contare infiniti nominativi vista l'arbitrarietà.

Cardinalità delle Categorie

Similmente a quanto detto per i servizi applicativi, di categorie ne potremmo contare infinite: in questo caso l'arbitrarietà si fa ancora più invadente in quanto trattasi di categorizzare a rigor di logica propria (quindi senza necessità di formalità alcuna) servizi in tipologie di servizi.

Cardinalità e relazioni funzionali per la DPI

Affinché il software che offre DPI possa sfruttare delle regole di caratterizzazione basate su hostname, queste devono essere finite e devono garantire l'univocità delle associazioni <hostname, categoria> e <hostname, subprotocol>.
In particolare:

- ad un hostname dichiarato è associata una sola categoria ed un solo subprotocol
- ad una categoria dichiarata sono associati zero o più hostname
- ad un servizio applicativo dichiarato (subprotocol) è associato un solo hostname

Ne consegue che l'associazione fra hostname e servizio applicativo è biunivoca, mentre non è tale l'associazione fra hostname e categorie.

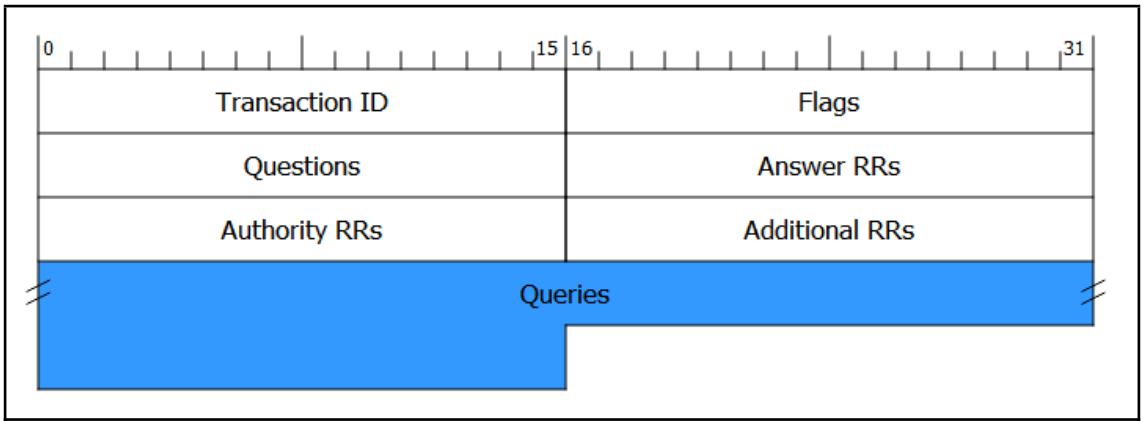
Visto che il sistema che si andrà a realizzare potrebbe definire nuove regole di caratterizzazione e che potrebbe verificarsi il caso in cui non si trovi una categoria opportuna (o d'interesse) per un certo hostname individuato, allora si definisce la categoria speciale nominata "UNCATEGORIZED" (dall'inglese "non categorizzato") alla quale saranno appunto associati tutti gli hostname non categorizzati.

Hostname: dove trovarli

Quando un operatore analizza traffico mediante Wireshark individua degli hostname all'interno di pacchetti DNS, HTTP e TLS (*Transport Layer Security*). In questo capitolo si danno cenni utili a localizzare i dati di interesse riguardo i protocolli sopracitati.

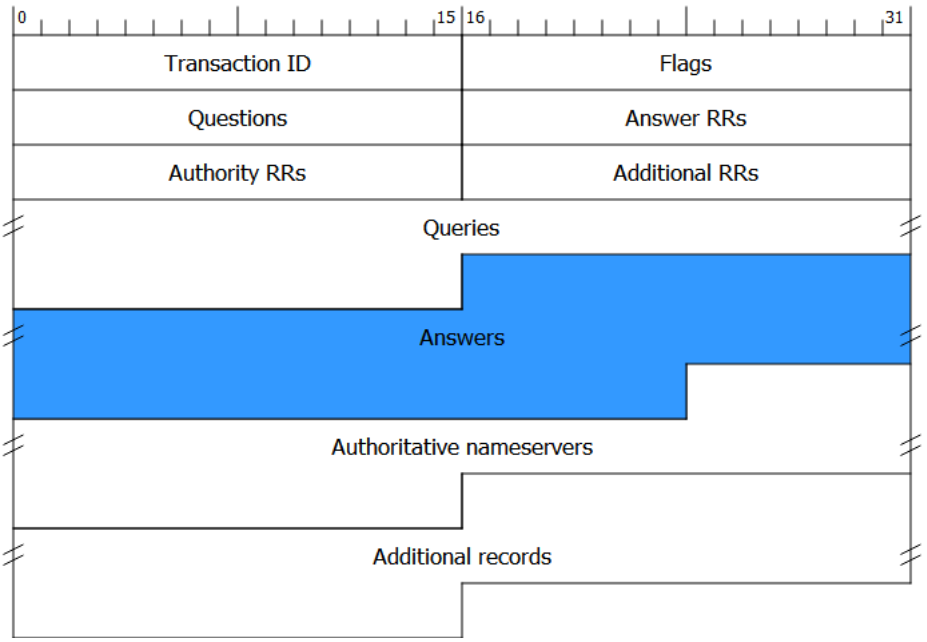
DNS

Il sistema DNS consente di tradurre un hostname in un indirizzo IP. Senza entrare troppo in merito, è sufficiente sapere che si tratta di un sistema gerarchico decentralizzato: a partire dal nome che si vuole risolvere, un'interrogazione può essere propagata a server DNS via, via più autorevoli fino a quando non si recupera la risposta cercata (o non si trova niente); non vi è infatti un unico server DNS che detiene tutte le associazioni utili, piuttosto queste sono distribuite su più server. Alcuni hostname non sono associati direttamente ad indirizzi IP ma ad altri hostname: in tal caso si parla di *hostname aliasing*; la risoluzione di un alias porta all'ottenimento di un altro hostname che (presumibilmente [12]) sia *canonico*, ovvero risolubile in un indirizzo IP. Al fine di agevolare tali ricerche, hostname risolti possono essere memorizzati temporaneamente dal proprio dispositivo e dal browser [4], di conseguenza potrebbe non essere necessario inoltrare una richiesta DNS in rete. In conclusione, nei pacchetti DNS può essere contenuto un hostname (vedi immagini 4, 5 per casi esemplificativi).



```
en.wikipedia.org: type A, class IN
    Name: en.wikipedia.org
    [Name Length: 16]
    [Label Count: 3]
    Type: A (Host Address) (1)
    Class: IN (0x0001)
```

immagine 4: diagramma di una query DNS e approfondimento delle queries contenute nella richiesta di risoluzione dell'hostname "en.wikipedia.org"; il tipo A sta ad indicare che si desidera risolvere un hostname in un indirizzo IP di versione 4; infografiche catturate su Wireshark




```
en.wikipedia.org: type CNAME, class IN, cname dyna.wikimedia.org
    Name: en.wikipedia.org
    Type: CNAME (Canonical NAME for an alias) (5)
    Class: IN (0x0001)
    Time to live: 12177 (3 hours, 22 minutes, 57 seconds)
    Data length: 17
    CNAME: dyna.wikimedia.org
dyna.wikimedia.org: type A, class IN, addr 91.198.174.192
    Name: dyna.wikimedia.org
    Type: A (Host Address) (1)
    Class: IN (0x0001)
    Time to live: 55 (55 seconds)
    Data length: 4
    Address: dyna.wikimedia.org (91.198.174.192)
```

immagine 5: diagramma di una risposta DNS e approfondimento delle risposte contenute nella risoluzione dell'hostname "en.wikipedia.org"; in questo caso l'hostname è stato risolto prima in un nome canonico (tipo CNAME) che, a sua volta, è stato risolto in un indirizzo IPv4; infografiche catturate su Wireshark

HTTP

Il protocollo applicativo client-server HTTP consente ad un client di ricostruire un documento *ipermediale* mediante una serie di richieste atte ad ottenere tutti i contenuti eterogenei che lo costituiscono, siano essi testi, immagini o, più in generale, contenuti multimediali; va precisato che tale protocollo abilita anche ad altre tipologie di azioni come il caricamento o l'eliminazione di una risorsa [30] ma, ai fini di questa introduzione, ci soffermeremo in particolare sul recupero di risorse. Una conversazione HTTP prevede delle richieste da parte del client e delle risposte da parte del server che, per inciso, potrebbe negare una qualsiasi richiesta per svariate ragioni; in ogni caso, il server risponde sempre includendo uno *status code* atto ad indicare l'esito della risposta [31].

Ogni risorsa recuperata per popolare un documento ipermediale viene raggiunta mediante un indirizzo che la identifica univocamente ed universalmente, tale è detto URI (*Uniform Resource Identifier*) e il suo prefisso è costituito dall'indirizzo dell'host che ospita la risorsa; in una richiesta, inoltre, è sempre indicato l'indirizzo del *referrer* (inglese per *referente*) ovvero l'host che ha promosso la richiesta. In ogni pacchetto di richiesta HTTP è quindi sempre presente l'host che si va a contattare.

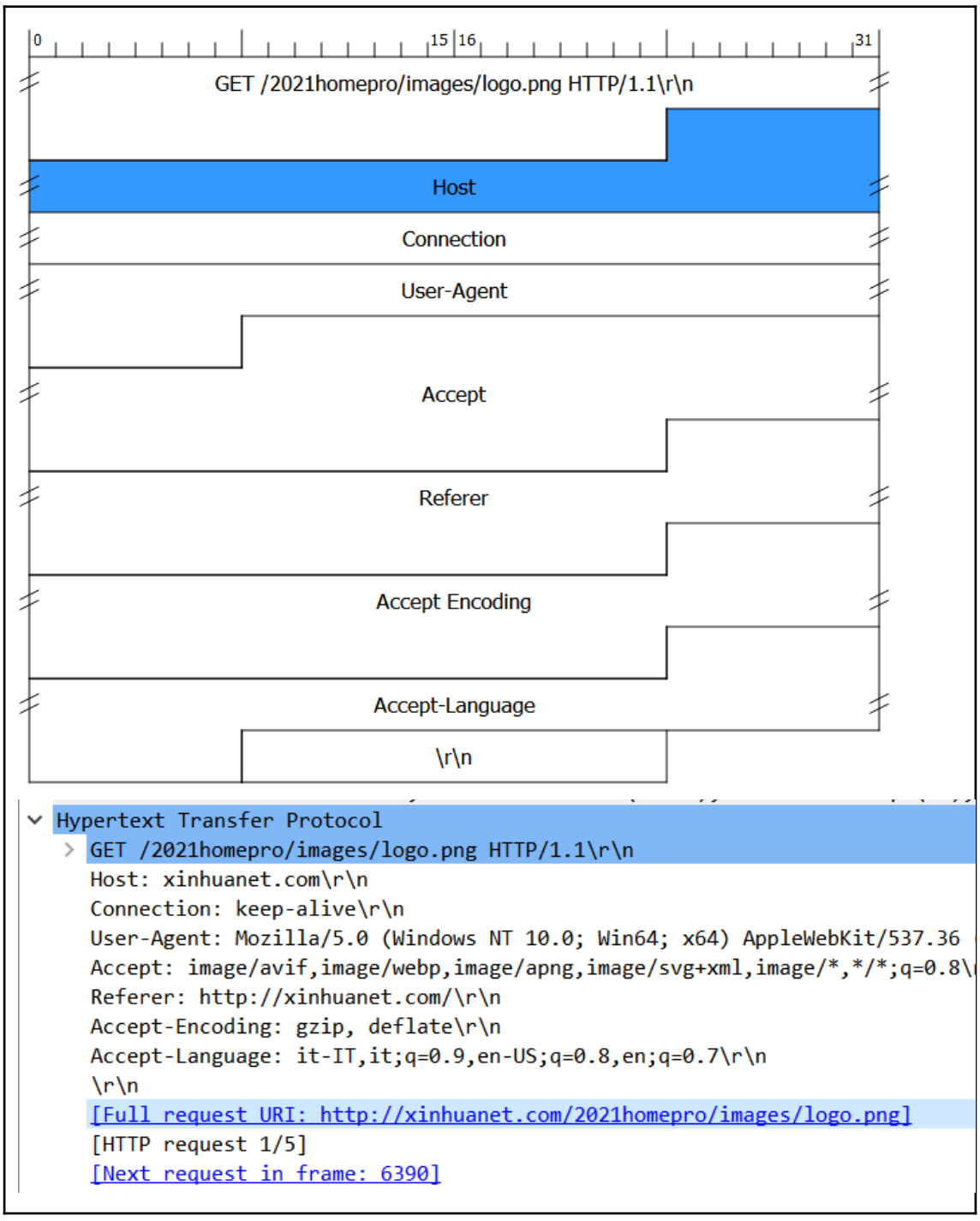


immagine 6: illustrazione di un pacchetto di richiesta HTTP ed esempio di dati contenuti in una richiesta HTTP (GET) dove si può osservare la risorsa richiesta e l'host alla quale si è richiesta; infografiche catturate su Wireshark

HTTPS, TLS, SSL

Com'è possibile immaginare leggendo i paragrafi relativi all'HTTP, i dati circolano in chiaro: ciò significa che tutta la conversazione effettuata con l'host

di un sito web per recuperarne risorse non è cifrata e, di conseguenza, se qualcuno la osservasse verrebbe a conoscenza di una svariata serie di informazioni riguardo agli host comunicanti e alle risorse trafficate ottenendo magari dati sensibili o che semplicemente sarebbe preferibile rimanessero privati. A tale proposito, negli anni, si sono adottate diverse misure per tutelare la sicurezza di chi naviga in rete; in particolare, la maggior parte dei siti web oggi offre un tipo di connessione considerata maggiormente sicura: HTTPS (*HyperText Transport Protocol Security*).

Tramite l'HTTPS, le conversazioni HTTP vengono cifrate e pertanto rimangono esoteriche a chiunque voglia osservarle senza essere in possesso delle opportune chiavi. Senza discorrere riguardo l'effettiva sicurezza di siti web via HTTPS [6, 18, 16], ci si limita a contestualizzare come avviene, quantomeno indicativamente, una conversazione con tale protocollo. Quando si accede ad una pagina web con protocollo HTTPS, si risolve prima l'hostname in un indirizzo IP (come accennato nel sottocapitolo relativo al DNS), quindi, invece che procedere con una conversazione HTTP in chiaro, si instaura una connessione sicura supportata da un protocollo crittografico come il SSL (*Secure Sockets Layer*) o il TLS (*Transport Layer Security*) che trasportano tale conversazione: tali sistemi crittografici si pongono fra il livello applicativo (e.g. HTTP) e quello di trasporto (e.g. *Transmission Control Protocol*). Affinché tale connessione possa avere luogo, si devono scambiare delle informazioni che consentono di stabilire le chiavi di cifratura: questa operazione è chiamata *handshake*.

Talvolta può verificarsi che più di un sito web sia fornito da un solo server condividendo un singolo indirizzo IP e che ognuno di questi siti utilizzi dei *certificati di sicurezza* (indispensabili per l'handshake) propri: con l'HTTP si specifica in chiaro quale sia il sito specifico da visitare ma nel caso di un HTTPS tale informazione non può essere passata allo stesso modo perché prima, appunto, si deve stabilire una connessione sicura e si ha quindi la necessità di recuperare il certificato di sicurezza opportuno. Al fine di risolvere tale problema è stata progettata un'estensione (n.b. facoltativa) per i protocolli crittografici

sopracitati che consente di specificare, durante l'handshake, quale sia il sito che si vuole contattare: tale estensione è detta SNI (*Server Name Indication*). Ne consegue che anche nel caso di connessioni HTTPS, come mostrato nell'immagine 7, è possibile ottenere l'hostname del sito contattato.

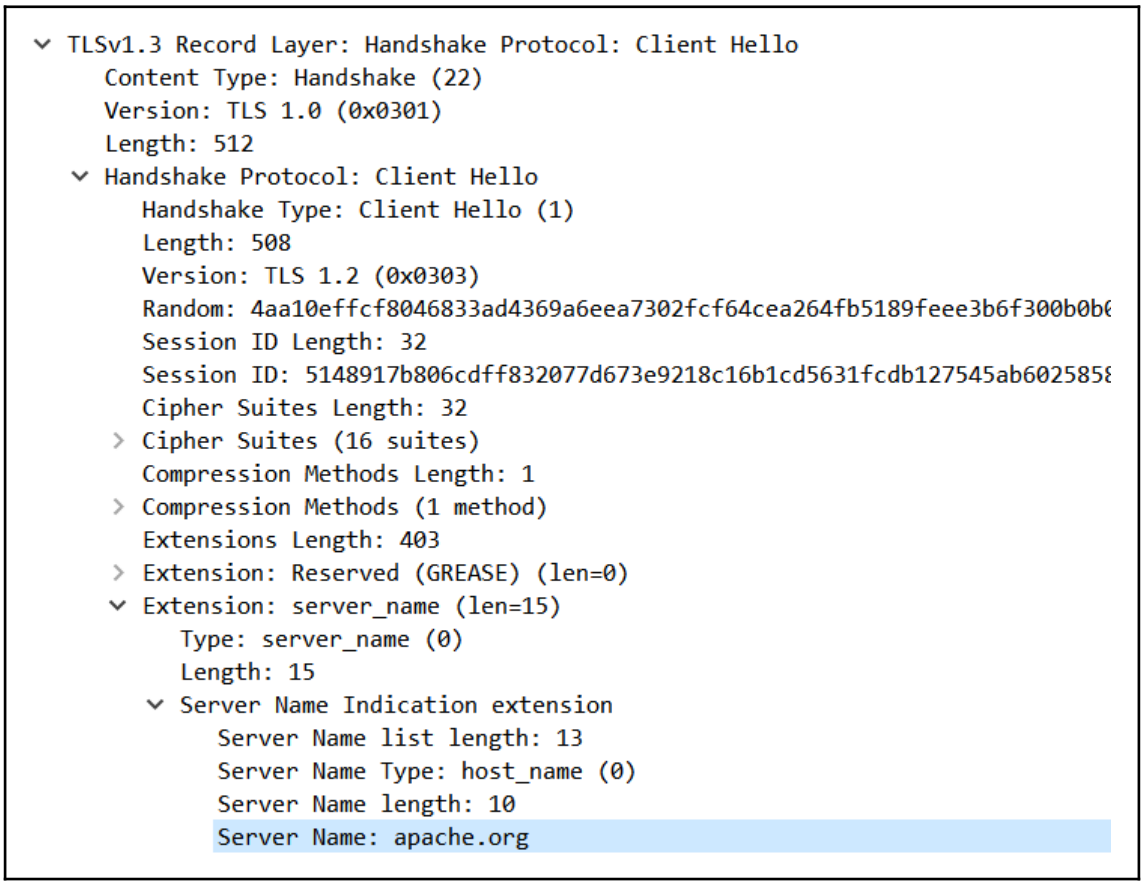


immagine 7: contenuto di un pacchetto TLS (versione 1.3) nella fase di handshake mettendo in evidenza il campo SNI; infografiche catturate su Wireshark

Altri Protocolli e Conclusioni

I protocolli trattati finora, ci consentono di individuare l'hostname contattato da un client all'interno della rete monitorata; va ribadito però che ciò non è sempre possibile e va aggiunto che esistono altre fonti dalle quali si potrebbero recuperare hostname (e.g.: SNI nei protocolli *QUIC*).

Non è sempre possibile individuare un hostname nei contesti sopra citati: nel caso del DNS, una richiesta potrebbe non essere proprio inviata (caching delle risoluzioni); nel caso dell'HTTP si ottiene sempre un hostname ma pochissimi

siti ad oggi servono ancora contenuti in HTTP [13, 10]; nel caso delle connessioni sicure e del QUIC, la SNI è facoltativa e comunque niente vieta di utilizzarla specificando un hostname che non sia quello da contattare.

Vi sono altri protocolli nei quali poter individuare hostname, fra questi abbiamo il GQUIC (*Google QUIC*) che, analogamente al QUIC offre il campo SNI; inoltre, non è da escludere la possibilità che all'interno di pacchetti, eventualmente aderenti a protocolli proprietari non noti, possano essere inclusi degli hostname.

In conclusione, si sono individuati dei contesti dove il sistema che verrà realizzato possa individuare hostname. Si potrebbero trattare le limitazioni di queste fonti, magari trattandone altre in aggiunta, ma ci si limiterà a quelle descritte finora perché più che sufficienti ad automatizzare il comportamento degli operatori e perché fare altrimenti avrebbe richiesto delle tempistiche troppo lunghe contestualmente all'esperienza di tirocinio e in concomitanza allo sviluppo delle *features* richieste; tuttavia, per non chiudere il sistema ad estensioni in tale ambito sono state prese delle scelte architetturali che consentiranno di adeguarlo e migliorarlo (documentate in seguito).

La Soluzione

Se è vero che trovare una soluzione ad un certo problema è difficile, a maggior ragione lo è trovarne una senza conoscerlo a fondo; del resto, non si può pretendere che la realtà si riduca ad un esercizio definito senza ambiguità, con requisiti e dati chiari, invariati nel tempo, per il quale si possono prevedere tutte e sole le competenze che dovranno essere impiegate per risolverlo. Certo, si hanno a disposizione degli strumenti metodici per affrontare tali situazioni ma l'idea di avere una panacea che consenta a noi informatici di portarci sempre in uno stato sicuro, nel quale abbiamo tutto sotto controllo è, per di più, un'illusione. Questo progetto non è stato un'eccezione in tal senso, vedendo sin da subito il susseguirsi di cambiamenti - anche importanti - a requisiti, il sorgere di complicazioni, imprevisti e difficoltà. Questa premessa è importante per comprendere che molte delle scelte (sia strategiche che più prettamente implementative) sono state fatte al fine di evitare il più possibile che un fallimento, un cambio di direzione, un bug o, più generalmente, un imprevisto indesiderato qualsiasi, potessero minare la riuscita del progetto e la soddisfazione delle aspettative dell'azienda.

Visto il dinamismo che ha tanto caratterizzato l'esperienza di tirocinio, non ci si è approcciati in modo totalmente rigoroso all'elaborazione del programma; questa scelta non è stata fatta meramente a discrezione personale: anche a causa della propria inesperienza, se si fosse anche solo tentato di affrontare il caso delineando correttamente requisiti, progettando l'architettura e via discorrendo, si sarebbe impiegato un tempo presumibilmente troppo lungo, sia in termini didattici, sia per l'azienda. Tuttavia, non si sono ignorati i rischi di tali mancanze [38] e si è sempre cercato di procedere, per quanto possibile, metodicamente: le competenze e conoscenze sviluppate specialmente in ambito universitario sono state quanto di più prezioso, ispirando e guidando scelte che, in ultima analisi, si sono dimostrate efficaci. Alcuni modelli concettuali ed algoritmi sono stati strutturati ad hoc, ispirandosi a principi (e.g. *loosely coupled*, *Single Responsibility*) e strategie (e.g. *Divide et Impera*) note in particolar modo nel

mondo informatico; tali risultati saranno opportunamente descritti e giustificati, cercando di non appesantire l'elaborato.

Si conclude questa introduzione anticipando che saranno incluse brevi riflessioni personali circa questioni reputate molto interessanti emerse durante l'elaborazione del progetto: dal concetto stesso di "categoria" a considerazioni più generali sull'analisi del traffico, si cercherà di dare contributi propri, decisamente umili contenutisticamente parlando, non approfonditi per ragioni di attinenza.

La Strategia Risolutiva

Una soluzione scelta per rispondere ad un'esigenza e la relativa implementazione hanno conseguenze volendo anche importanti non solo sull'efficienza ma anche riguardo altre caratteristiche fondamentali come la manutenibilità del codice. In questo capitolo si espongono la strategia risolutiva proposta, le scelte implementative principali e le tecnologie utilizzate accennando ad alternative e motivando scelte progettuali; saranno inoltre approfonditi alcuni aspetti legati all'efficienza computazionale delle scelte più peculiari per le quali si presenteranno inoltre paragrafi aggiuntivi dedicati e diagrammi a semplificarne la comprensione quando ritenuto opportuno.

Strategia Principale

Per raggiungere l'obiettivo proposto, si è deciso di automatizzare parte del lavoro che verrebbe svolto manualmente offrendo un software che, data una o più catture di traffico, esporti delle regole di caratterizzazione basandosi sugli hostname in modo che gli operatori:

- ☑ ottengano risultati analoghi ad una loro analisi manuale
- ☑ possano gestire i risultati ottenuti
- ☑ possano adattare il comportamento del sistema alle proprie esigenze

Automatizzare il processo manuale non è certamente l'unica via percorribile alla soluzione: si potrebbe partire dalle categorie di interesse ed individuare tutti i servizi ad esse associati oppure si potrebbe interrogare un database esterno che contenga già delle associazioni <hostname, categoria> che siano utili. Nel primo caso si individuerebbero i servizi associabili ad una certa categoria invece che la categoria nella quale poter inserire un certo servizio ma, anche supponendo di riuscire in questo intento, si dovrebbero comunque trovare tutti gli hostname associati e ciò potrebbe essere un'operazione molto difficoltosa sapendo che, almeno ad oggi, non sono noti servizi che offrono associazioni simili neppure consultando siti come ICANN che comunque potrebbero contenere dati volontariamente omessi, dati non attendibili (spesso, ma non necessariamente, ciò è eventualmente segnalato) oppure dati completi ed esaustivi ma con

identificativi non direttamente correlabili. Nel secondo caso invece, si sarebbe dovuto trovare un database adeguato ed eventualmente processare dati contenuti in esso; simili database potrebbero esistere in servizi di DPI ma, comunque sia, non sono omnicomprensivi ed è così anche per ragioni molto semplici: è irragionevole realizzare una caratterizzazione che sia esaustiva per tutti i casi d'uso perché, come già elaborato, di categorie se ne possono pensare a piacere, di servizi ve ne sono sempre di nuovi e facendo così si appesantirebbe troppo l'analisi del traffico che potrebbe dover riguardare tutte le regole di caratterizzazione definite. L'approccio proposto, tuttavia, non è stato scelto solo perché sembrava meno difficoltoso ma anche più appropriato; il sistema dovrà essere utilizzato da degli operatori abituati ad un certo procedimento: si suppone che una sua simulazione sia intrinsecamente più facile da adottare ed anche da estendere o modificare.

Fatta questa doverosa premessa, si presenta intanto un'astrazione ad alto livello del sistema che può essere concepito in tre **fasi principali**:

- **Preprocessing:** si preparano i dati per l'analisi nel minimo set di informazioni utili a produrre l'output
  - estrazione: data una o più catture, si acquisiscono gli hostname estraibili da esse mediante ndpiReader
  - preparazione: si aggregano gli hostname in un insieme di soli hostname distinti
- **Processing:** si elabora l'insieme di dati ottenuti in preprocessing producendo l'output richiesto
  - subprotocol: si raggruppano gli hostname per suffisso non pubblico condiviso più lungo in *gruppi* (o *servizi*), estraendo dei subprotocol
  - categorie: si tenta di classificare il traffico sfruttando una lista di <categoria, <keyword, score>[]> definita dagli operatori estraendo delle categorie
  - esportazione: si produce un report CSV (*Comma Separated Value*) e JSON (*Javascript Object Notation*) contenenti le informazioni estratte e i file testuali *subprotocols.txt* e *categories.txt* con le regole di caratterizzazione per protocolli e categorie ndpi-compatibili

- **Postprocessing:** si produce dell'output aggiuntivo utile a valutare alcuni risultati del processing
    - report: si fornisce un report aggiuntivo circa l'estrazione di categorie compiuta a partire dai risultati in csv forniti alla fase precedente

Come si può osservare, ogni fase principale designa *obiettivi* astratti (●) concretizzati da *compiti* specifici (■); se si deve estendere il sistema, allora si può individuare la fase principale riguardante l'estensione in questione e quindi aggiungere un compito; se invece si deve modificare il sistema, si modifica un compito già esistente.

Questa semplice concezione del sistema, anche se non presenta certamente i vantaggi di una progettazione architetturale rigorosa, ha guidato lo sviluppo dall'inizio alla fine, lasciando gli obiettivi sempre intatti e facilitando l'aggiunta/modifica di compiti, richiedendo tempi brevissimi di manutenzione.

Preprocessing

Come mostrato nel capitolo relativo agli hostname (*Hostname: dove trovarli*) è nota la localizzazione dei dati all'interno di traffico che presenti conversazioni che aderiscono a certi protocolli. L'approccio scelto per ottenere questi dati è proprio la DPI: visto che il sistema in uso da IES prevede già l'utilizzo delle tecnologie nTop, che queste sono state consigliate dai tutor e che durante il corso tenuto dal tutore accademico è stato introdotto nDPI - che fornisce agilmente l'estrazione di tali dati - si è scelto di usare quest'ultimo per l'ispezione. In particolare, si è utilizzato ndpiReader per l'estrazione in formato CSV di conversazioni di rete e loro caratteristiche sfruttando un filtro BPF in modo da ignorare ogni conversazione che non prevedesse almeno un indirizzo pubblico, quindi si è utilizzato lo strumento da linea di comando *Q* [3] per effettuare delle interrogazioni SQL (*Structured Query Language*) al fine di aggregare e filtrare i soli hostname distinti per i quali non esistessero già regole di caratterizzazione in un unico CSV finale.

Le istruzioni relative a questo procedimento sono programmate in uno script bash che permane alcuni dei dati prodotti in itinere per snellire, su conferma dell'utente, un eventuale prossimo preprocessing che sia sullo stesso set di input; questi dati sono trattati come file temporanei: vengono infatti cancellati o sovrascritti automaticamente all'esecuzione successiva in modo da non appesantire inutilmente lo spazio di archiviazione in uso ma consentendo all'utente di salvarli ed utilizzarli in seguito; è fornita inoltre una funzione eseguibile da linea di comando per cancellare tutti i temporanei suddetti.

Lo script consente di ottenere il riferimento ad ndpiReader sia come argomento durante la chiamata, sia sfruttando una variabile globale, sia utilizzando un apposito file di configurazione; il programma consente inoltre di ottenere il riferimento alle catture di traffico sia specificando un singolo file, sia specificando una cartella contenente più catture. Sono effettuati vari controlli sulla validità dei dati di input e sulla presenza degli strumenti necessari all'esecuzione: in caso venga rilevato qualche problema, il processo viene interrotto e viene generato un messaggio che ne indica la causa ed eventualmente un suggerimento su come risolverlo.

Gestire più opzioni di input e file temporanei sono scelte fatte per rendere la fase più efficiente per alcuni casi d'uso e per predisporre l'interfaccia ad essere adeguata agilmente in corso d'opera, aggiungendo o estendendo funzionalità fino alla configurazione attuale.

Processing

Per ogni hostname si individua la porzione di interesse escludendo un eventuale suffisso pubblico individuato, quindi si sfrutta l'etichetta più esterna per realizzare un nuovo subprotocol che consenta di identificare tale servizio in futuro. Nonostante la classificazione esulasse inizialmente dallo scopo del tirocinio, è naturalmente emersa essere una funzionalità centrale da automatizzare (per quanto possibile) a beneficio degli operatori: dopo aver valutato diverse opzioni si è optato per gettare le basi di un classificatore che potesse fornire risultati empiricamente interessanti - seppur su casi di studio

decisamente modesti - ma che, soprattutto, potesse essere migliorato sensibilmente a posteriori facendo del *learning*; il classificatore, a partire dai dati già elaborati per realizzare i subprotocol, tenta di classificare il servizio associandolo ad una categoria predefinita d'interesse.

Subprotocol

Per designare quali siano i suffissi pubblici si utilizza la lista offerta da publicsuffix.org [32]: dato che la lista è fornita in formato testuale, necessita di un certo grado di interpretazione (eccezioni e wildcards); non essendo presumibilmente aggiornata ad ogni esecuzione del programma questa viene processata e serializzata in una struttura dati che ne consente un utilizzo più efficiente.

La struttura dati in questione consiste in un albero per il quale ogni nodo è costituito da un dizionario contenente riferimenti ad altri nodi figli indicizzati da una stringa che può essere:

- un'etichetta
- un asterisco
- un'etichetta preceduta da un punto esclamativo

Come mostrato nell'immagine 8, per ottenere la posizione del suffisso in un hostname si cerca l'ultima etichetta nel dizionario del nodo corrente (a partire dalla radice) e, se è presente oppure se è presente un asterisco, si procede a cercare nel dizionario del nodo ottenuto l'etichetta immediatamente precedente dell'hostname. Se, però, il nodo ottenuto accedendo ad un elemento del dizionario è nullo oppure l'etichetta cercata non è trovata nel dizionario correntemente osservato, oppure se nel dizionario corrente è presente l'espressione composta dall'etichetta anticipata da un punto esclamativo o infine se non abbiamo più etichette da valutare allora siamo arrivati ad un'etichetta che non fa più parte del suffisso pubblico.

Data tale configurazione, il numero di passi per individuare il suffisso pubblico (se esiste) di un hostname è costante rispetto al numero delle etichette

dell'hostname cercato e, più precisamente è al più pari al numero di etichette della regola più lunga contenuta nel file di definizione delle regole che non supera le sei unità; per ogni passo è necessario l'hashing dell'etichetta e di conseguenza abbiamo un po' di overhead legato a tale funzione e, più generalmente, all'implementazione del dizionario; avendo utilizzato dizionari forniti dalla libreria .Net (Microsoft) si dovrebbe far riferimento a questa per la quale, come è facile aspettarsi (dato l'utilizzo di tabelle hash), è documentato un costo prossimo a O(1): la struttura offerta è in grado di ottimizzare il fattore di carico ridimensionando il dizionario quando necessario; nel caso in questione però non abbiamo alcun ridimensionamento a posteriori dato che è possibile inizializzare i dizionari specificando il numero di elementi che vi andremo ad inserire (valore noto a priori), numero utilizzato dalla classe per scegliere una capienza opportuna per la tabella hash.

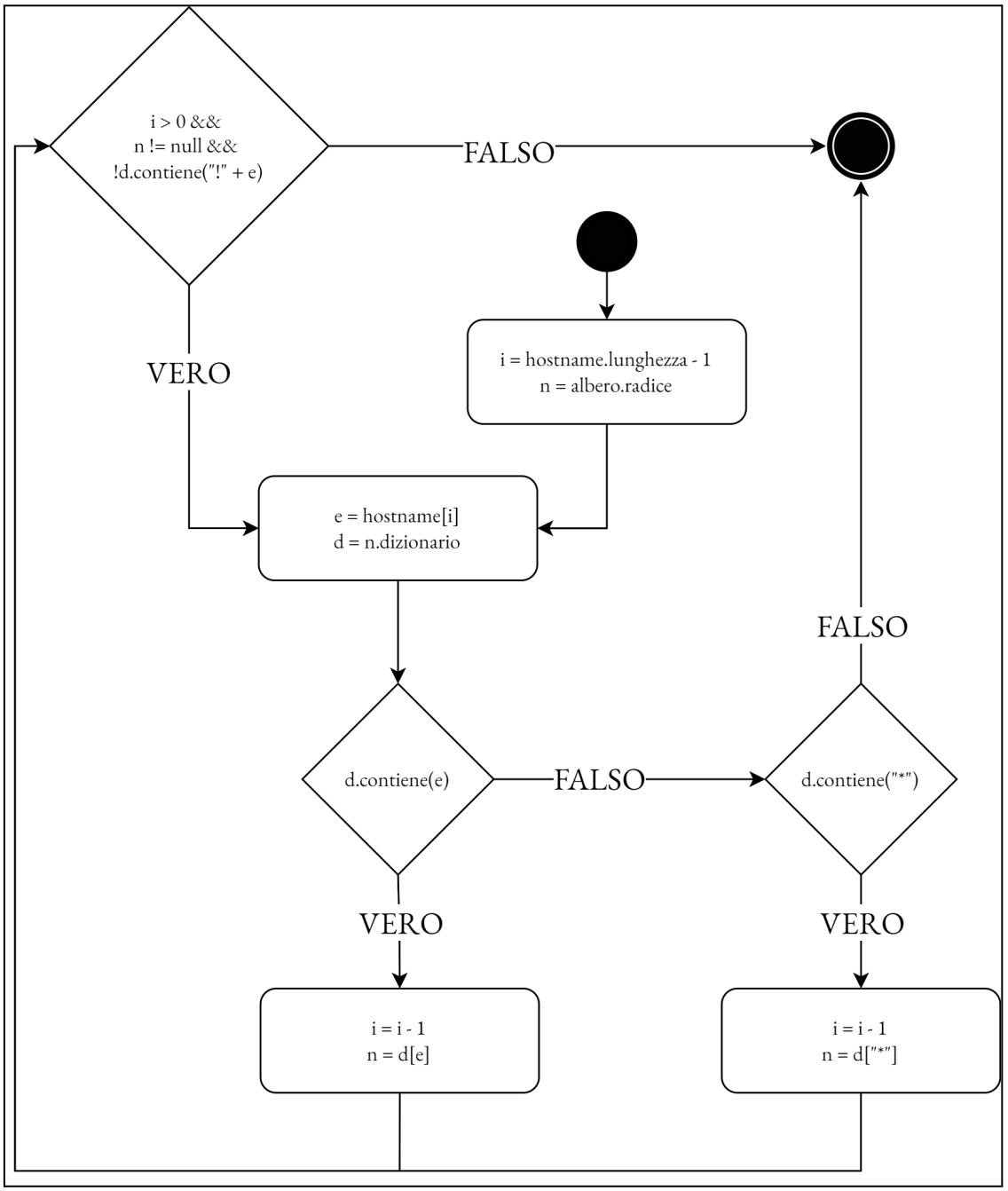


immagine 8: diagramma che astrae l'attività d'analisi del suffisso di un hostname; ogni nodo contiene dello pseudocodice esemplificativo delle operazioni associate; al termine, "i" indicizza la posizione della prima etichetta che non fa parte del suffisso pubblico nell'hostname; per semplicità, si suppone che albero, radice e dizionari non siano nulli, che l'hostname sia un array di etichette (stringhe alfanumeriche) non nullo di lunghezza positiva tale che almeno un'etichetta non faccia parte di alcun suffisso riconosciuto.

L'utente ha la possibilità di personalizzare la lista di suffissi da ignorare aggiungendo, modificando o eliminando regole in essa contenute sempre rispettando la sintassi documentata su publicsuffix.org; così facendo si possono ad esempio escludere hostname composti risultato di artefatti dell'infrastruttura (e.g. facebook.com.*seafy.me*).

In definitiva, il costo per individuare tutti i suffissi è al più O(N*L) con N il numero di hostname da analizzare ed L la lunghezza (in numero di etichette/wildcard) della regola specificata più lunga (L = 6 per i suffissi pubblici); ignorando la possibilità di personalizzare le regole d'individuazione del suffisso, allora il costo è lineare nel numero di hostname (O(N)).

Per quanto riguarda lo spazio, la capienza di un dizionario alla profondità *prof* è correlata al numero di regole che contengono almeno *prof* etichette (scelta essere il numero primo prossimo più grande o uguale) e il numero di dizionari in uso è proporzionale alla lunghezza (in termini di numero di etichette) della regola più lunga; questo comporta, soprattutto col crescere della cardinalità di etichette per livello (considerando la distribuzione dei primi, anche solo fin quando nota), a ritrovarsi con un alto numero di celle vacanti allocate per i dizionari. Al fine di mitigare tale spreco, i dizionari indicizzano istanze di classe e non delle *struct* (che andrebbero, se non trattate appositamente con complicazioni nel codice, allocate interamente sullo stack indipendentemente dal fatto che siano popolate da dati interessanti [20]). In aggiunta, si tratta di una mole di dati molto bassa per un computer moderno (<< 1Mb utilizzando la lista di publicsuffix.org) ed è per questo motivo che non si è ritenuto essere tanto rilevante da giustificare ulteriori approfondimenti, che avrebbero richiesto del tempo; se, in futuro, il sistema dovrà essere in esecuzione su di un server (magari in una formula completamente automatica) allora sarebbe opportuno ridurre l'occupazione in spazio, soprattutto se questi ha una disponibilità di memoria dedicata maggiormente limitata.

Una volta individuato il suffisso di ogni hostname si effettuano dei

raggruppamenti per il suffisso condiviso più lungo che sia d'interesse (i.e. non pubblico) e si realizza un subprotocol per ognuno di questi gruppi. Il subprotocol viene realizzato in modo che sia nella sintassi utilizzata da nDpi e il servizio prende come nome la prima etichetta del suffisso condiviso; giusto una precisazione minore: la sintassi in questione consente di specificare più suffissi/hostname per servizio in maniera compatta ma, realizzando che gli operatori elencano comunque tali dati in forma estesa, ho deciso di rispettare le loro linee guida.

Categorie

Dopo aver scartato soluzioni considerate troppo complesse da realizzare vista la mia inesperienza (come nel caso di soluzioni che coinvolgessero l'apprendimento automatico), per assegnare categorie ai servizi ho deciso di realizzare un classificatore che emulasse il comportamento di un operatore umano e che potesse essere migliorato in futuro anche tramite un processo di apprendimento.

Come ci si può facilmente immaginare, gli operatori non classificano un servizio analizzando manualmente i byte del traffico di rete ma interrogando approssimativamente motori di ricerca. Analogamente, il classificatore cerca il nome di ogni servizio utilizzando il motore di ricerca Google quindi osserva nei risultati l'occorrenza di alcune espressioni chiave associate a categorie d'interesse (i.e.: Videogiochi, Streaming, ...) associando un punteggio cumulativo opportuno alle relative categorie: la categoria che ha associato il punteggio più alto sarà quindi la categoria associata al servizio in questione a meno che tale punteggio non sia troppo basso per considerare la categorizzazione valida (*threshold*, configurabile), in tal caso il servizio sarà considerato non categorizzato.

Per le interrogazioni al motore di ricerca si è utilizzata la libreria *Custom Search JSON API* (Google) [11] sfruttando un search engine personalizzato configurato per ricerche non ristrette a specifici domini (*unrestricted search*); il provider limita a 100 il numero di interrogazioni effettuabili al minuto (*QPMu: query per minute per user*) e a 1000 le richieste al giorno (*QPD: query per day*) ma, dato che l'azienda ha richiesto limitazioni meno restrittive, ho modificato il classificatore in modo che le richieste effettuate all'API venissero fatte sfruttando uno *user-id* a rotazione: quando si effettua un'interrogazione, infatti, va specificato un id che tipicamente viene utilizzato per distinguere l'utente che utilizza il search engine personalizzato; inizialmente questi era sempre lo stesso ma, per consentire un numero di richieste maggiore, viene attualmente cambiato ad ogni interrogazione. Considerando che un id utente consiste in un massimo di 39 caratteri e che ogni carattere può essere una qualsiasi lettera dell'alfabeto inglese, una cifra, un trattino basso o alto, è possibile ottenere più

di 39^64 identificativi distinti e di conseguenza 39^64 * 100 richieste al minuto; ho contattato il team di Google che ha approvato 10'000 richieste giornaliere (ulteriormente estendibili sotto richiesta) e che non ha espresso contrarietà circa l'utilizzo di user id a rotazione; volendo si potrebbero utilizzare più caratteri di quelli elencati qui e si potrebbero anche considerare identificativi con meno di 39 caratteri estendendo ulteriormente il limite di richieste massimo al minuto, tuttavia si è ritenuto più che sufficiente il livello di libertà raggiunto, preferendo non introdurre complicazioni non indispensabili nel codice. Se si volesse fare di meglio, si dovrebbe richiedere un'estensione delle QPD piuttosto, limite attualmente dominante.

Ottenuti i risultati dal motore di ricerca per un certo servizio in analisi, si effettua la ricerca di espressioni chiave per individuare la categoria più opportuna da associare. L'utente definisce a priori una lista di categorie di interesse (ognuna associata ad un intero che identifica la categoria nel sistema ndpi) e, per ognuna di queste, delle coppie <espressione chiave, punteggio intrinseco>: ogni volta che viene trovata un'espressione chiave nei risultati di ricerca verrà associato cumulativamente il punteggio intrinseco alla categoria di appartenenza. Un'espressione chiave può essere una qualsiasi sequenza alfanumerica (non *case-sensitive*) contenente facoltativamente spazi ed eventualmente preceduta o chiusa da un punto. Affinché un'espressione chiave venga individuata in un testo, questa deve comparire preceduta da un carattere che non sia alfanumerico nel caso inizi con un punto (*condizione di apertura*) e chiusa da un carattere che non sia alfanumerico nel caso l'espressione termini con un punto (*condizione di chiusura*): queste due condizioni facoltative consentono di effettuare ricerche di espressioni "isolate", di prefissi e di suffissi (le cosiddette *Whole Word* in Visual Studio, o *Entire Word* nei software Microsoft Office). Come già anticipato, la categoria alla quale viene associato il punteggio più alto caratterizza il servizio che si sta tentando di classificare se il punteggio supera una certa soglia personalizzabile dall'utente.

Analogamente a quanto accadeva per la ricerca dei suffissi, anche per la ricerca delle espressioni chiave si è sfruttato una struttura dati apposita; in questo caso si utilizza un *Trie* per il quale ogni nodo è caratterizzato da un insieme di dati (la

struct *Match*) che contengono l'espressione chiave (senza punti), la categoria associata, due booleani per le condizioni di apertura e chiusura, il punteggio intrinseco e la categoria associati. Quando si effettua una ricerca si procede basilarmente con un adattamento dell'algoritmo Aho-Corasick: si passa il testo del risultato di ricerca ad una classe contenente l'albero insieme ad un evento delegato, si procede con la ricerca nell'albero e, non appena si verifica una corrispondenza con una delle espressioni chiave, si invoca il delegato che, dato il match inerente, il testo di ricerca e l'offset al quale si è trovata l'espressione chiave, controlla che siano rispettate eventuali condizioni di apertura e chiusura quindi aggiunge il punteggio intrinseco al punteggio totalizzato dalla categoria; l'algoritmo termina quando si è esaurito il testo nel quale ricercare le espressioni chiave.

Al fine di semplificare la comprensione di questa attività se ne astraggono le fasi principali in diagrammi mostrati nelle immagini seguenti (9, 10, 11, 12); questi non sono da intendersi in corrispondenza diretta con l'implementazione del codice, piuttosto come indicazioni riduttive ma esemplificative.

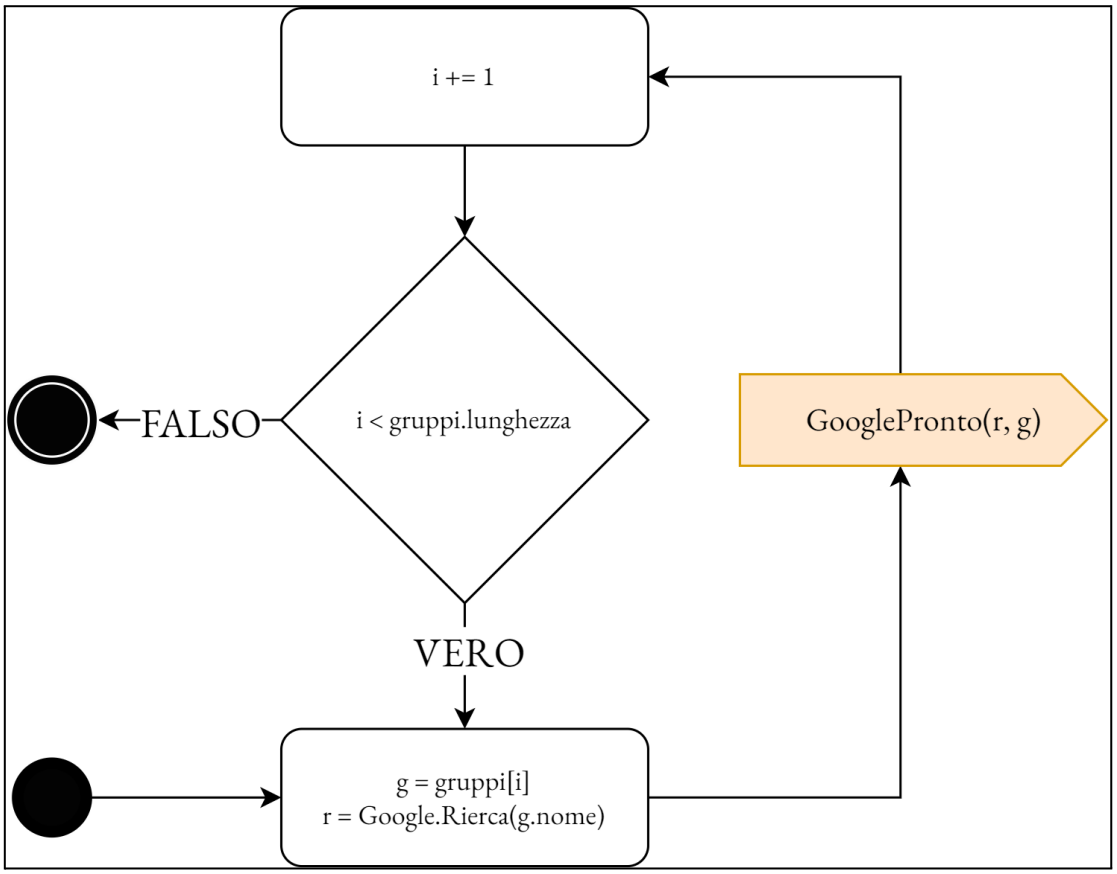


immagine 9: diagramma che astrae la fase di ricerca utilizzando il motore Google per l'attività di classificazione dei servizi; la ricerca restituisce un testo pre-elaborato "r" con sole informazioni rilevanti; per semplicità, si suppone che l'indice "i" sia inizializzato a zero, che le varie strutture siano opportunamente inizializzate, che vi sia almeno un gruppo e che i nomi dei gruppi siano validi; le segnalazioni di eventi sono implementate come invocazioni di eventi delegati

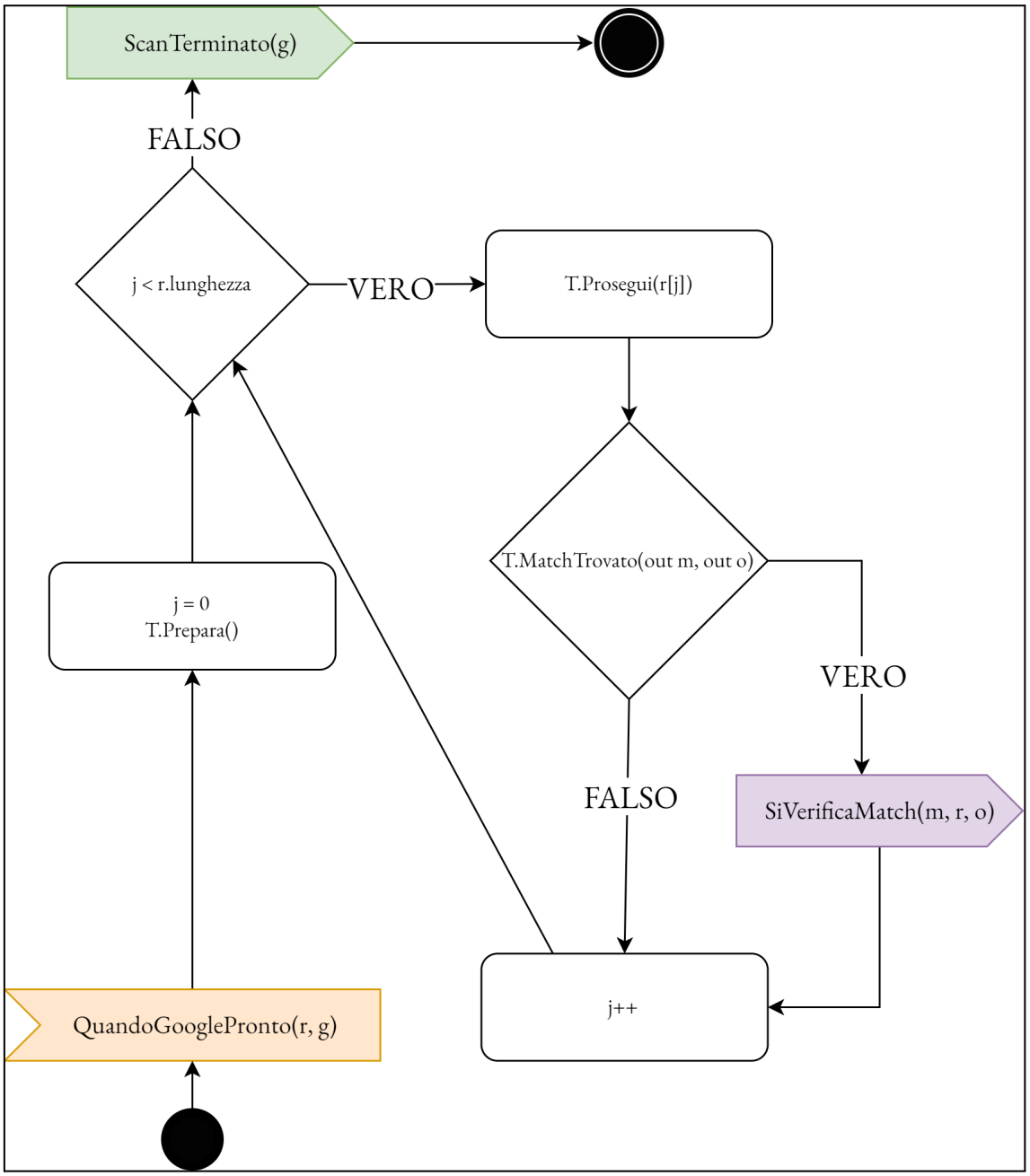


immagine 10: diagramma che esemplifica la fase di analisi di un testo di ricerca “r” per il gruppo “g” tramite l’albero “T” al solo scopo di evidenziare la chiamata dell’evento delegato per segnalare il match “m” rilevato ad un offset “o” nel testo in analisi; per semplicità, si suppone che l'albero sia opportunamente inizializzato in modo da operare con i match e le relative espressioni chiave predefinite, “r” e “g” validi; le segnalazioni di eventi sono implementate come invocazioni di eventi delegati

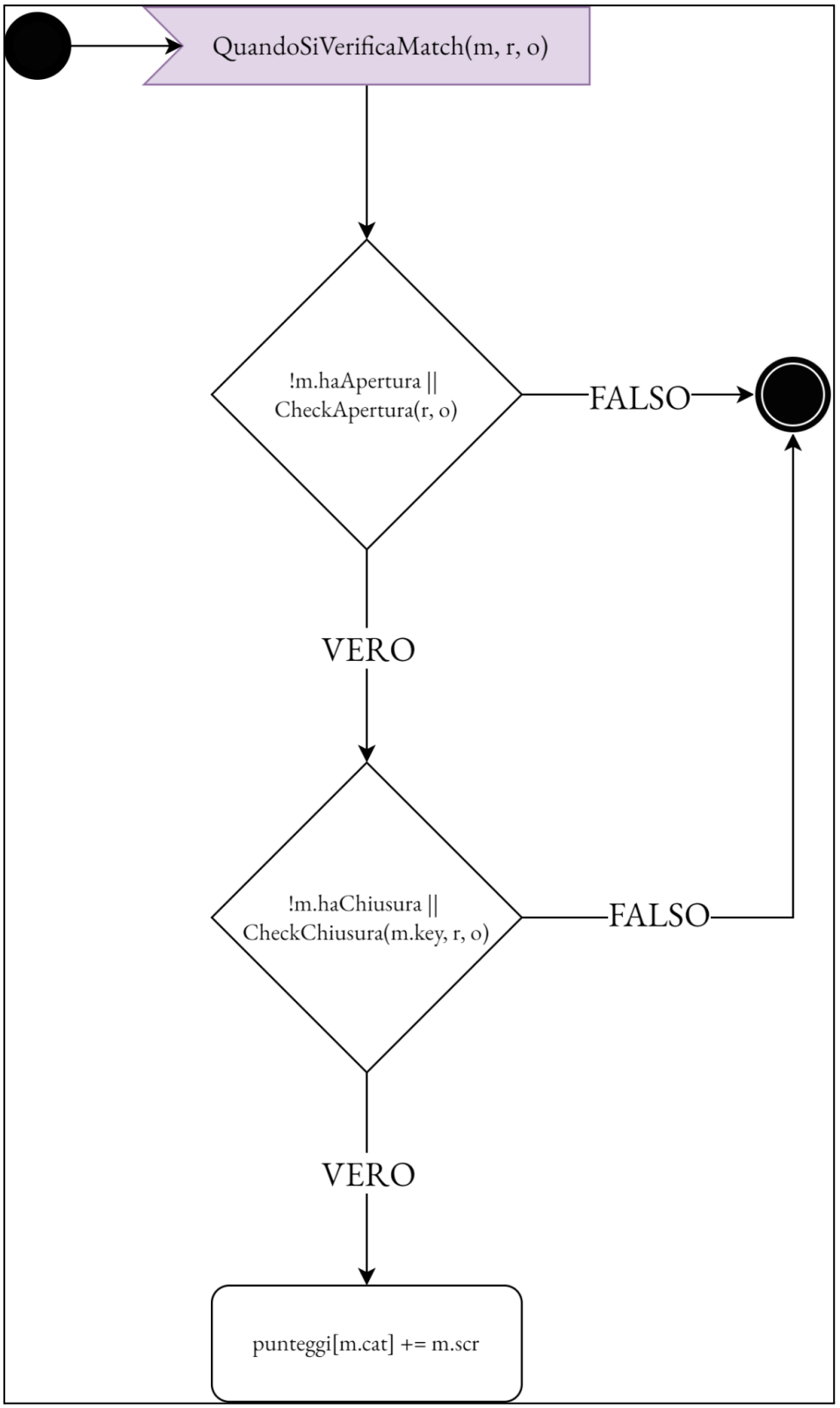


immagine 11: diagramma che astrae il controllo delle condizioni di apertura/chiusura per un match “m” in un testo “r” rilevato all’offset “o”; “m.key” indica l’espressione chiave, “m.cat” la categoria, “m.scr” il punteggio intrinseco associati al match; per semplicità si suppone che la struttura “punteggi”, indicizzata da categorie, sia opportunamente inizializzata; le segnalazioni di eventi sono implementate come invocazioni di eventi delegati

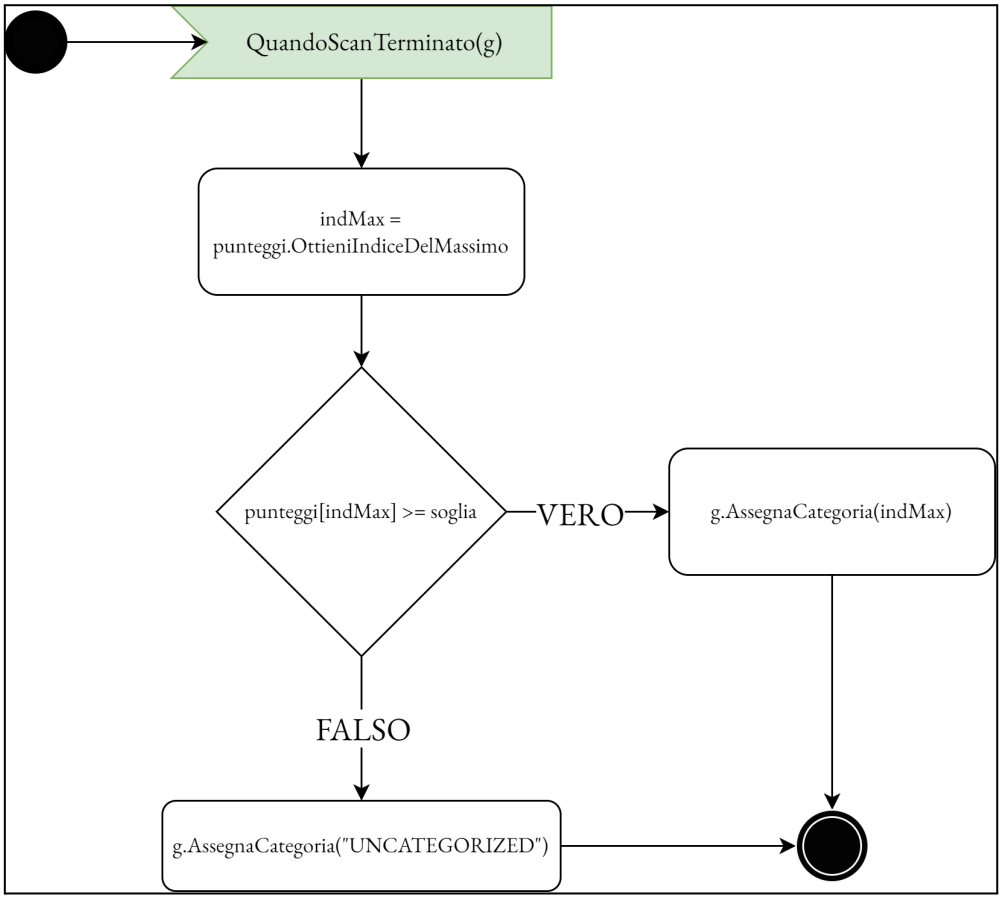


immagine 12: diagramma che astrae l'attività di assegnamento di una categoria ad un gruppo; per semplicità, si suppone che la struttura "punteggi" e il gruppo "g" siano opportunamente inizializzati; si fa notare che "punteggi" è indicizzata da categorie

Sia N la lunghezza di un testo nel quale effettuare la ricerca di espressioni chiave, L la somma delle lunghezze delle espressioni chiave e Z il numero totale di occorrenze riscontrate, ignorando per un momento il costo legato alla ricerca in rete, computazionalmente questa soluzione ci consente di ottenere un costo pari a O(N+L+Z) per ogni ricerca; si dovrebbe aggiungere il costo legato all'invocazione del delegato come moltipicando di Z ma questi è asintoticamente costante trattandosi di operazioni aritmetiche semplici (un accesso indicizzato costante e l'incremento di un contatore); altre operazioni che si possono osservare nei diagrammi di sopra sono compiute in itinere e hanno costo complessivamente costante. In definitiva il classificatore spenderebbe O((N+L+Z)*S + G) per classificare tutti gli S servizi individuati considerando anche il tempo per ottenere le risposte dall'API Google (G): quest'ultimo dipende da vari fattori sia legati alla connessione internet dell'host che esegue il

programma, sia al servizio stesso di Google; tale costo non è prevedibile a priori, è variabile e non sembra essere documentato, quindi non mi è stato possibile rilevare con precisione il suo impatto generale sul costo.

Dato che ogni interrogazione effettuata con l'API ha un costo (in denaro) e che richiede un tempo non prevedibile a priori allora si memorizzano le risposte dell'API in un dizionario, indicizzandole con il nome del servizio utilizzato per la ricerca; in questo modo, sapendo che la lettura dal dizionario è prossima ad O(1) si abbatte il costo delle interrogazioni che sono già state fatte in passato portando in tal caso il costo della classificazione di un servizio a O(N+L+Z).

I punteggi intrinseci associati alle parole chiave possono essere utilizzati come iperparametri costituenti un classificatore e quindi resi oggetto di un processo di *learning* senza apportare alcuna modifica al sistema. Questa vantaggiosa possibilità è dovuta al fatto che la dichiarazione dei punteggi e le categorie sono definiti - come già precisato - in un file testuale a parte: se si volesse integrare un processo di learning, basterebbe quindi operare su tale file senza dover aggiungere, rimuovere o modificare una singola linea di codice. Integrare del learning sarebbe utile non solo per migliorare la qualità dei risultati del classificatore ma anche per preservarla programmaticamente a seguito dell'aggiunta di nuove categorie d'interesse; sfruttando il *machine learning* e i dati forniti dal programma sviluppato, si potrebbe addirittura tentare un apprendimento automatico, sempre senza modificare alcuna linea di codice perché il programma consente di esportare anche i risultati del motore di ricerca.

Per concludere questo approfondimento riguardo al classificatore, se ne mostra - solamente a scopo indicativo - una matrice di confusione elaborata sulla classificazione di 450 servizi in 7 classi: pubblicità (ADS), trackers (TRK), social networks (SNT), streaming video (STR), notiziari (NWS), videogiochi (GMS), giochi scommesse (BTT). La classe associabile ad un servizio potrebbe non essere nota (l'operatore non sa classificare il servizio) oppure nota ma non inerente (l'operatore riconosce che un certo servizio potrebbe appartenere ad una classe specifica che, però, non è d'interesse); in entrambi i casi la classe reale viene attribuita essere "UNCATEGORIZED" (UNC). La classificazione reale è

stata tenuta dal sottoscritto operando in modo analogo a ciò che avrebbero fatto gli operatori ma con qualche accorgimento in più: i servizi non immediatamente classificati a seguito di una breve ricerca su Google sono stati ricercati anche su siti specifici (e.g. store per videogiochi; report di tracker/cookies) ed è capitato, seppur per poco più di 20 hostname, di aver compiuto delle ricerche più approfondite osservando direttamente il traffico tramite Wireshark. I servizi sono risultanti da un preprocessing di più catture di traffico effettuate nel periodo di tirocinio su più navi aderenti al servizio Seafy. Il sistema di iperparametri (espressioni chiave e relativi punteggi intrinseci) è stato realizzato iterativamente e manualmente cercando di migliorare la precisione del classificatore sulla base di intuizioni ed osservazioni personali. Va precisato però che i dati utilizzati per il training (1049 hostname, con un ratio approssimato di 7:3) non sono disgiunti a quelli utilizzati per realizzare la tabella: di fatti, poco meno del 12% degli hostname del training si sono ripresentati; a posteriori, sarebbe stato meglio rimuovere tali hostname da questo test per non viziare i risultati che, come già detto, vanno presi quindi con il giusto riguardo e solo a scopo indicativo.

	ADS	TRK	SNT	STR	NWS	GMS	BTT	UNC
ADS	71	0	0	0	0	0	0	8
TRK	0	46	0	0	0	0	0	22
SNT	0	0	2	0	0	0	0	0
STR	0	0	0	27	0	0	0	7
NWS	0	0	0	0	12	0	0	8
GMS	0	0	0	0	0	8	0	0
BTT	0	0	0	0	0	0	9	0
UNC	1	2	0	0	0	1	1	227

immagine 12: matrice di confusione per classificatore multiclasse delle categorie di servizi; 450 istanze, 8 classi; classi reali intestate su prima colonna, classi predette intestate su prima riga.

Dai dati riportati in tabella risulta che la percentuale di errore delle previsioni sul totale delle istanze è del 12% (il tasso di errore approssimato per difetto alla seconda cifra decimale è pari a 0.12) e di conseguenza abbiamo un'accuratezza dell'88%. Si evince inoltre che la maggior parte degli errori è dovuta alla mancata classificazione di alcuni servizi di tracking. A discapito di quanto immaginassi, le ricerche manuali più approfondite (21 ricerche) hanno consentito di migliorare la classificazione solo per 8 servizi (5 ADS, 3 TRK) che altrimenti sarebbero rimasti nella stessa classe predetta dal classificatore (*uncategorized*) e, pertanto, si suppone che con ricerche più approssimative, quindi maggiormente simili a quelle che condurrebbe un operatore, i risultati peggiorerebbero solo marginalmente. La maggior parte dei TRK non correttamente classificati (19/22; ~86%) sono stati classificati manualmente semplicemente ricercando l'hostname completo sul motore di ricerca; fra le altre cause di errore (62%) si sono presentati casi in cui i risultati di ricerca non erano in italiano né in inglese (7/31; ~22%; uniche due lingue utilizzate per specificare le espressioni chiave) e, meno raramente, casi per i quali si aveva a che fare con servizi classificabili concettualmente in più classi (e.g.: siti dedicati a news principalmente fornite tramite streaming video): in quest'ultimo caso è stato necessario nuovamente utilizzare l'intero host (o etichette precedenti a quella utilizzata per l'identificazione del servizio) per designare la classe più appropriata (7/24; ~29%) ma per la maggiore mi è stato impossibile determinare una classe reale vista l'ambiguità.

Ritengo che tali risultati siano sufficienti a giustificare un futuro approfondimento del classificatore, magari con un learning supervisionato e la triade *training-validation-test* con approccio e dati scelti opportunamente; quest'analisi, realizzata appositamente per questo elaborato, suggerisce in modo chiarissimo ulteriori accorgimenti che si potrebbero apportare al sistema come il fatto di sfruttare altre etichette dell'hostname per la classificazione o l'utilizzo di traduttori automatici per ovviare alla questione linguistica: non è detto che risolvendo alcuni casi d'errore con tale approccio non se ne verifichino di nuovi ma varrebbe la pena tentare visti i presupposti concettuali.

Si ritiene opportuno spendere un appunto per i servizi non classificabili a causa di ambiguità: in che classe si inserisce, ad esempio, un notiziario online che propone le proprie notizie via streaming video? E' chiaro che l'hostname non identifichi una risorsa (e.g. un video, una pagina, ...) di per sé, neppure se questa è offerta via CDN (*Content Delivery Network*), caso per il quale si potrebbero ottenere indicazioni utili per alcune classi; è altrettanto chiaro che non si può utilizzare come discriminante la quantità di dati (e.g. *goodput*) trafficati in una conversazione di rete perché non è detto che, in primis, questi consentano effettivamente di discriminare due classi (e.g. news vs blog) e, secondariamente, che questi siano rilevanti in generale perché la fruizione di una o più risorse potrebbero avvenire in una conversazione distinta a quella in analisi. Quest'interessante casistica necessiterebbe di un grande approfondimento se si volesse affrontare con la dovuta serietà, magari iniziando col definire rigorosamente che cosa sia una categoria (e se sia opportuno chiamarla così) e cosa ne distingua una da altre perché, altrimenti, mancherebbero delle basi solide sulle quali poter fare ipotesi rilevanti (si ricorda che, praticamente, si possono definire categorie senza alcun rigore logico); inoltre si potrebbe riflettere sul modo in cui ci si approccia all'analisi del traffico automatizzata che, per quanto possa essere approfondita *per pacchetto* (DPI) e magari per conversazione e fra pacchetti, non considera la relazione fra conversazioni: si potrebbero realizzare infatti degli "alberi genealogici delle conversazioni" in modo da determinare - ove possibile - quale di queste è scaturita da quale host, in una sorta di "analisi profonda *relazionale* del traffico di rete". Per evitare un volo pindarico su questo argomento, se ne conclude la trattazione, segnalando l'eventualità di concepire categorie di servizi come categorie "propriamente dette" [19], in particolare considerandone applicazioni su grafi di conoscenza [33, 36].

Effettuata la classificazione si possono finalmente esportare delle categorie ottenendo così delle regole di caratterizzazione, per così dire, complete.

Estrazione

Per concludere con il processing, si esporta un file CSV che contenga tutti i servizi analizzati per ognuno dei quali è associato un subprotocol, una categoria in sintassi ndpi e altre informazioni utili ad una valutazione dei risultati come, per citarne alcuni, il punteggio di classificazione (nel caso la categoria associata non sia UNCATEGORIZED), il numero di occorrenze delle parole chiave individuate nel relativo testo di ricerca con il relativo punteggio e un elenco degli hostname associati. A seguito di una richiesta dell'azienda, si forniscono gli stessi dati contenuti nel CSV in formato JSON, più precisamente: per ogni servizio analizzato si esporta la serializzazione dell'istanza di una classe che ne contiene tutti i dati d'interesse.

Postprocessing

Effettuando delle interrogazioni SQL tramite Q sul CSV prodotto in fase di processing, si produce un semplice report testuale che indica il numero di hostname categorizzati per ogni categoria, i file temporanei generati ed un'indicazione su come rimuoverli; l'applicazione principale di questa funzionalità è stata il dimostrare rapidamente un sommario dell'esecuzione del programma all'azienda.

Il Codice

Il codice scritto per implementare il sistema è suddivisibile in un *Makefile* utile a compilare il progetto e fornire alcuni comandi aggiuntivi utili, una *soluzione* C# (*protoexcursus.sln*) che implementa la fase di processing ed uno script bash (*px.sh*) utilizzato per la fase di pre e post processing oltre che per un'esecuzione semplificata di *protoexcursus.exe,* eseguibile ottenuto dalla compilazione del *progetto* C#. La parte centrale del programma, quella realizzata in C#, può essere compilata (sfruttando, ad esempio, il compilatore *mono*) ed eseguita sia su Windows che su sistemi Linux ; lo script bash e il Makefile sono invece solo per sistemi Linux ed utilizzando una *Virtual Box* o, ancor più semplicemente, il *Windows Subsystem For Linux* [21] è possibile eseguirli agilmente anche sul sistema operativo Microsoft.

Inizialmente si è lavorato su sistema Linux per accedere più facilmente alle varie tecnologie in uso, in particolare aziendali: caratteristica per la quale si è preservata la compatibilità con tale SO (*Sistema Operativo*). La scelta di adottare C#, il linguaggio di programmazione ad oggetti orientato ai componenti, multi-paradigma (imperativo/dichiarativo/funzionale) fortemente tipato, progettato da Anders Hejlsberg (Microsoft) nel 2000, è stata fatta principalmente per una questione di conoscenza personale del linguaggio, dell'IDE (*Integrated Development Environment*) che lo supporta (Visual Studio), e della sua libreria più nota: il *.Net* [22]; queste caratteristiche hanno consentito di sviluppare il sistema separando le responsabilità dei vari componenti che lo compongono ed estendendone le funzionalità mano a mano che i requisiti divenivano più chiari; inoltre, l'estrema facilità con la quale si possono integrare librerie esterne via *NuGet* [23] ha reso decisamente semplice l'aggiunta di funzionalità come l'esportazione di JSON e CSV. Così facendo, si è strutturato un progetto ragionevolmente facile da mantenere ed estendere, senza perdere tempo nella realizzazione di funzionalità non direttamente correlate alla caratterizzazione del traffico e già disponibili gratuitamente, concentrandosi così su ciò che caratterizza davvero il sistema.

Makefile

Il Makefile fornisce funzioni per compilare il codice sorgente, eliminare artefatti dell'esecuzione e della compilazione, installare mono e Q. Durante la compilazione si cancella (se applicabile) la serializzazione della struttura dati adibita all'individuazione dei suffissi in quanto quest'ultima è realizzata dal programma che, se cambia, potrebbe gestire in modo diverso tale struttura. Fornire delle funzioni per l'installazione di mono e Q semplifica la risoluzione di queste dipendenze.

Protoexcursus

Protoexcursus, combinazione di "proto" (da *protocollo*) ed "excursus" (inteso come "intervento che devia dal filo principale"), è realizzato da un insieme di componenti scritti in C#, ognuno dei quali svolge una funzione specifica all'interno del sistema; si elencano brevemente alcune componenti (incluse quelle principali), indicandone nome (in grassetto) ed una descrizione.

AssemblyInfo
Dichiara tutti i metadati che verranno associati all'eseguibile dall'IDE come il nome del prodotto, l'autore e, soprattutto, la versione. Per più informazioni, si rimanda alla documentazione Microsoft [24].

Group
Un gruppo modella l'insieme di dati che caratterizzano un servizio, dagli hostname alla categoria associata con il relativo punteggio di classificazione al subprotocol. I dati di un'istanza di questa classe verranno aggiornati dai componenti più opportuni tramite appositi metodi o setter esposti. Le istanze di questa classe verranno serializzate in file JSON e utilizzate per la realizzazione del CSV

Topic
Questa classe modella il concetto di categoria all'interno del programma; dato che le categorie non sono note a priori e che è l'utente a definirle non si può

realizzare un'enumerazione che non sia dinamica (dynamic type C# [25]); tuttavia, non ritenendo i tipi dinamici particolarmente semplici da gestire, ho optato per realizzare la classe in questione che, implementando operatori impliciti ed espliciti di cast da/a stringa [26] ed implementando *IEquatable* [27], consente una conversione diretta e trasparente delle categorie espresse come stringa (ottenute dal file utente di definizione di categorie, parole chiave e punteggi intrinseci) e una modellazione sufficiente a gestire le categorie all'interno del sistema. Per quanto riguarda la categoria dei servizi non categorizzati (UNCATEGORIZED), si istanzia - mediante costruttore statico [28] - un Topic identificato dalla stringa "UNCATEGORIZED" e si rigetta un'eventuale ridefinizione implicita dell'utente di tale Topic in fase d'istanza tramite un opportuno controllo nel costruttore.

StringExtensions

Fornisce estensioni per il tipo *string* utili per compiere operazioni su stringhe all'interno del sistema; sfruttare tali estensioni è risultato essere particolarmente comodo non solo per rendere il codice più compatto ma anche per separare ulteriormente la responsabilità dei componenti.

Triex

È la classe che modella il trie già descritto nel capitolo relativo alla categorizzazione dei servizi; va precisato che la classe fa utilizzo dei cosiddetti *generics* [29]: il tipo dell'elemento che caratterizza un nodo è infatti generico e deve essere specificato in fase di istanza. Ho introdotto questa astrazione perché ho voluto riutilizzare la stessa classe anche in altri programmi; l'astrazione è stata preservata perché, comunque sia, complicando marginalmente il codice lo rende decisamente più versatile: per cambiare strutturalmente il comportamento in caso si riscontri una parola chiave durante una ricerca (si ricorda che a tale scopo si invoca un evento delegato) infatti non è necessario cambiare la firma dei metodi di questa classe, né modificarla.

Program

È la classe che si occupa, dato l'input, di produrre l'output servendosi dei vari componenti del sistema. Grazie alla libreria *CommandLine* [35], gli argomenti con i quali eseguire il programma possono essere resi tali semplicemente associando degli attributi alle proprietà che modellano tali dati d'ingresso, specificando inoltre testi d'aiuto, versioni lunghe e brevi delle opzioni e molto altro che solitamente richiede diverse linee di codice per essere implementato. Gli argomenti di input consistono in:

- un percorso al file contenente gli hostname
- un percorso al file di output CSV
- un percorso alla cartella dove serializzare gli output JSON
- un percorso al file delle espressioni chiave per la categorizzazione
- l'opzione di abilitare o meno la classificazione (se disabilitata, tutti i servizi saranno UNCATEGORIZED)
- la chiave per la Google API: l'utilizzo del motore di ricerca necessita di una chiave
- l'ID del motore di ricerca

Logger

Si occupa di fornire metodi per astrarre la serializzazione dei dati e il log del programma; per quanto riguarda la manipolazione di CSV viene fatto uso della libreria *CsvHelper* [5] che, anche in questo caso, ha evitato di sviluppare funzionalità già esistenti e facilmente reperibili.

Serializer

Fornisce metodi per serializzare da/a binario e da/a JSON. La separazione fra Logger e Serializer è utile per dividere la logica (cosa si vuol fare) dall'implementazione (come viene fatto); si precisa che tale separazione è valida anche per la manipolazione dei CSV, ottenuta tramite l'utilizzo della libreria suddetta.

Fetcher
Si occupa di popolare una lista con dei gruppi realizzati a partire dal CSV di input fornendo un metodo che richiede, in particolare, una funzione per individuare i suffissi non interessanti (i.e. pubblici); passare la funzione via argomento consente di separare completamente questa fase di parsing dalle operazioni di analisi dell'hostname.

SuffixValidity
Fornisce metodi per il parsing degli hostname dai suffissi non interessanti

GoogleSearch
Fornisce le funzionalità legate all'interazione con il motore di ricerca inerente, gestendo interrogazioni, id utente a rotazione ed eventuale indisponibilità o rifiuti del motore di ricerca. Riguardo a quest'ultime circostanzialità, il comportamento del programma dipende dalla situazione: se il rifiuto è generico, si effettua un numero limitato di riprove e, in caso di continui dinieghi, si interrompe il processo restituendo un messaggio d'errore; se invece il rifiuto è specifico e, ad esempio, riguarda il superamento di una soglia allora si potrebbe interrompere subito il processo per evitare di violare le politiche d'uso della libreria Google e/o proseguire invano.

TopicMatcher
Fornisce funzionalità per analizzare le occorrenze delle espressioni chiave in un testo di ricerca individuando la categoria che totalizza più punti.

Classifier
Dati dei gruppi, tenta di classificarli servendosi dei componenti legati alla ricerca e al match dei topic; è in questa classe, o più precisamente in una sua istanza, che si gestisce il caching dei risultati del motore di ricerca, servendosi di altri componenti fra i quali, il Serializer.

L'approccio per componenti indipendenti abilitato in particolare dalla centralizzazione su un'unica classe (Program) che, in un certo senso, svolge il ruolo di parte di controllo dei vari componenti, ha facilitato notevolmente estensioni e modifiche del programma.

Una scelta particolare, sicuramente discutibile, è stata il fatto di non utilizzare interfacce per la comunicazione di *Program* con il resto dei componenti: in corso d'opera, non era solo l'implementazione di un componente a dover cambiare ma anche il contratto che lo caratterizzava e se senza interfaccia si modificava un componente e la sua gestione in Program, con interfaccia si sarebbe dovuta modificare anche quest'ultima; del resto, erano gli stessi concetti (i.e.: classificazione, parsing dell'input, ...) a rendersi più chiari mano a mano che si confrontavano i risultati soprattutto con l'azienda. Per lo sviluppo del programma allo stato attuale, non utilizzare interfacce nel contesto in questione è risultato vantaggioso ma ciò non toglie che queste possano essere decisamente utili in futuro: una volta che un componente è assodato essere adeguato allo scopo per cui è stato pensato, varrebbe la pena interfacciarlo per sfruttare a pieno i vantaggi dell'indipendenza inter-componente.

Analogamente a quanto detto per le interfacce inter-componente, va speso un appunto riguardo l'architettura: l'assenza di requisiti chiari in principio, le tempistiche e il bisogno di adattarsi a tecnologie e ambienti non familiari ha reso difficile progettare un'architettura a priori che potesse essere utile e funzionale. Si è scelto quindi di ignorare una progettazione più rigorosa prima di passare a scrivere codice, errore che sarebbe potuto costare caro ma che, probabilmente grazie alla bassa complessità architetturale intrinseca al caso di studio e un approccio agile, non ha avuto alcuna implicazione sui risultati dell'operato; va precisato però, che forse si sarebbe potuta manutenere una visione architetturale così come si è fatto per il codice, prima di operare modifiche a quest'ultimo. Ciò non avrebbe soltanto reso più chiaro e sicuro lo sviluppo ma avrebbe probabilmente semplificato anche l'esposizione dell'operato e dei suoi aggiornamenti ai tutor e all'azienda. In conclusione, nonostante non si siano presentate complicazioni, sarebbe stato più opportuno realizzare e mantenere una progettazione architetturale ad alto livello fin dall'inizio.

Considerazioni Finali

Lo strumento realizzato rispetta le aspettative dell'azienda e può essere migliorato sotto vari aspetti. Per quanto riguarda il **preprocessing** si potrebbe sfruttare direttamente la libreria nDPI, senza dover utilizzare ndpiReader: diverse feature presenti in tale programma non sono infatti utili al sistema ed avere un'implementazione fatta appositamente per estrarre i dati di interesse dalle catture di traffico potrebbe essere quindi conveniente quantomeno in termini di manutenzione e, presumibilmente, anche in termini di efficienza: se, infatti, allo stato attuale si estraggono degli hostname via DPI e solo successivamente si processano in un insieme di soli elementi distinti, con un'implementazione ad hoc si potrebbe evitare quest'ultima elaborazione evitando di aggiungere hostname già presenti. Per quanto riguarda il **processing**, si potrebbe migliorare il classificatore sotto vari aspetti che sono già stati descritti nel capitolo inerente; oltre a ciò che è già stato detto, si potrebbe migliorare il sistema di espressioni chiave: allo stato attuale, se si hanno due espressioni chiave K1, K2 tali che K1 è una sottosequenza in K2, allora tutte le volte che si trova (in un testo di ricerca) l'espressione K2, si troverà necessariamente anche K1 contando entrambi i punteggi associati; questo comportamento non è detto che sia indesiderato ma potrebbe essere forse reso più "pulito" evitando il conteggio di espressioni chiave nel caso in cui queste siano individuate come sottosequenze di altre espressioni anch'esse individuate. Più **in generale**, oltre a ciò che è già stato indicato nei capitoli precedenti (sia riguardo ad implementazioni, che al progetto architetturale), si potrebbe concepire una parallelizzazione del sistema: una volta ottenuti gli hostname, si potrebbero processare parallelamente; le uniche accortezze che si dovrebbero tenere in tal caso sono, principalmente, il gestire un accesso sincronizzato alle strutture condivise, il propagare l'interruzione della comunicazione con la Google API (in caso risulti impossibile usufruire del servizio) e l'attendere i risultati finali per produrre un eventuale CSV di report e passare al postprocessing; sempre riguardo la parallelizzazione, varrebbe la pena considerare una versione di Aho-Corasick parallela [2]. Inoltre, si potrebbe scegliere definitivamente quali sono gli OS target: se si volesse mantenere il

supporto per Windows e Linux, allora sarebbe opportuno far sì che le funzionalità del sistema accessibili direttamente da Linux lo siano anche da Windows, senza dover utilizzare macchine virtuali o WSL; una possibile soluzione che, inoltre, potrebbe semplificare il progetto e consentire una gestione più granulare delle allocazioni di memoria potrebbe essere quella di adottare un linguaggio alternativo, come il C++ che presenta vantaggi analoghi a quelli per i quali si è scelto il C# ma che ne offrirebbe, appunto, ulteriori: si potrebbe infatti integrare agilmente la libreria nDpi, compilarlo più facilmente su entrambi gli OS e gestire meglio la memoria (possibile anche su C# ma più complesso), solo per citarne alcuni.

Come anticipato, requisiti e obiettivi sono variati anche sensibilmente durante tutta la durata del progetto: inizialmente, era noto il fatto che si volesse sviluppare un qualche tipo di strumento per discriminare del traffico mediante DPI ma non cosa si desiderasse esattamente. Ciò ha comportato un impegno di tempo nella prima settimana, spesa per installare, configurare e comprendere applicativi e servizi dei quali, per quanto utili nel proprio bagaglio di conoscenze, non c'è stato bisogno (fra questi *Kibana* ed *Elastic Search*). Fortunatamente però, dopo qualche interazione con i tutor, il caso è andato a definirsi più chiaramente e si è iniziata una fase di analisi mirata a comprendere cosa si potesse fare per agevolare la caratterizzazione del traffico con uno strumento da fornire all'azienda; tuttavia, non aver discusso esaurientemente i requisiti ha lasciato spazio a qualche ambiguità, risolta solo in seguito. Di fatti, lo strumento avrebbe dovuto riguardare l'identificazione di subprotocol e solo secondariamente la categorizzazione dei servizi; inoltre, esso era inizialmente pensato per essere utilizzato dagli operatori in una formula "semi-automatica" ma successivamente l'azienda ha espresso il desiderio di renderlo completamente automatico, integrandolo nell'infrastruttura; questo fatto, emerso nella settimana in presenza a Milano, è stato estremamente interessante ma purtroppo solo parzialmente compiuto: le tempistiche per adattare non solo lo strumento ma anche l'infrastruttura si sono rivelate più lunghe del previsto con la conseguenza di dover rinunciare a completare tale conversione. Ci tengo ad esprimere tuttavia quanto questo tentativo è stato tutt'altro che infruttuoso: mi

ha consentito di - per così dire - "immergermi" nel vivo di un'operazione aziendale interna che ha visto il coinvolgimento di più settori e si è concluso con il predisporre, da una parte, il programma ad essere modificato più agevolmente, dall'altra il sistema di GRE Tunnel (*Generic Routing Encapsulation Tunnel*) dell'infrastruttura per convogliare il traffico generatosi su tutte le reti che coinvolgessero il servizio Seafy. Si potrebbe quindi proseguire con la conversione dello strumento in versione completamente automatica, continuando ad interagire con l'azienda per trovare il modo e la maniera più opportuni. In conclusione, un approccio più deciso e consapevole nel trattare requisiti e richieste avrebbe reso più gestibile e sotto controllo lo sviluppo del sistema.

Infine, viste le funzionalità offerte dal programma realizzato, viste le possibilità di configurazione dello stesso, tale progetto potrebbe essere reso open source consentendo anche ad altre aziende di poterne usufruire e, soprattutto, consentendo a chiunque di collaborare al fine di migliorarlo anche sotto aspetti che sono sfuggiti all'analisi del sottoscritto o di chiunque altro sia stato coinvolto. In particolare per questo scenario, sarebbe bene fare quantomeno del refactoring per portare il codice ad essere il più accessibile possibile anche a chi non conosce ancora il sistema, nonostante questo sia già commentato per esplicare classi e metodi; in tal caso, sarebbe altrettanto opportuno scriverne una documentazione esauriente e concisa. Questa direzione però presenta alcune caratteristiche delicate: renderlo open source significherebbe renderlo utile per altri utenti (non solo IES) e, di conseguenza, potrebbe (n.b. non necessariamente) complicare lo sviluppo di una soluzione ad hoc (come ipotizzato per la conversione completamente automatica).

Ho trovato l'esperienza di tirocinio estremamente dinamica ed altamente formativa: ha richiesto approfondimenti di vario genere, sforzi tecnici quanto intellettuali e mi ha dato la possibilità di apprendere da chi, di esperienza, ne ha maturata molta, in decine di anni di lavoro, studio e ricerca. Nonostante la formula a distanza, è stato importantissimo il supporto del tutor che mi ha assistito molto pazientemente ed in modo efficace, insegnandomi tantissimo e non solo da un punto di vista tecnico ma anche, per così dire, in termini di

savoir-faire nel relazionarsi con un'azienda, condividendo la sua grande dedizione e passione per la materia; anche gli operatori aziendali non sono stati da meno, fornendo indicazioni tecniche per l'interazione con l'infrastruttura aziendale quando ciò si è rivelato necessario, nonostante fossero molto impegnati con altre mansioni improcrastinabili e prioritarie per l'azienda, facendomi sempre sentire benvenuto e a mio agio; da ultimo, ma decisamente non per importanza, è stato veramente d'ispirazione il direttore di IES che è sempre riuscito ad individuare tempestivamente potenzialità ed opportunità con un'audacia formidabile, dimostrando anch'esso un'esperienza dalla quale non si può che imparare moltissimo.

Bibliografia

[1] Apple, randomizzazione degli indirizzi MAC,
https://support.apple.com/en-us/HT211227,
https://support.apple.com/it-it/guide/security/secb9cb3140c/web

[2] Arudchutha S., Nishanthy T., Ragel R.G. , String Matching with Multicore CPUs: Performing Better with the Aho-Corasick Algorithm,
https://arxiv.org/ftp/arxiv/papers/1403/1403.1305.pdf

[3] Ben-Attia H., q - Run SQL directly on CSV or TSV files,
http://harelba.github.io/q/

[4] Chromium Organization, DNS Prefetching,
https://www.chromium.org/developers/design-documents/dns-prefetching

[5] Close J. e contributori, CsvHelper, https://joshclose.github.io/CsvHelper/

[6] Cloudfare, What is mixed content? | HTTP vs. HTTPS,
https://www.cloudflare.com/it-it/learning/ssl/what-is-mixed-content/

[7] Combi C., Pozzani G., Pozzi G., Telemedicine for Developing Countries: A Survey and Some Design Issues,
https://www.ncbi.nlm.nih.gov/pmc/articles/PMC5228142/

[8] Deri L., Martinelli M., Cardigliano A., nDPI: Open-Source High-Speed Deep Packet Inspection, https://luca.ntop.org/nDPI.pdf

[9] Emam K. E., Koru A.G., A Replicated Survey of IT Software Project Failures,
https://ruor.uottawa.ca/bitstream/10393/12988/1/El_Emam_Khaled_2008_A_replicated_survey_of_IT_software.pdf

[10] Felt A. P., Barnes R., King A., Palmer C., Bentzel C., Tabriz P., Google, Cisco, Mozilla, Measuring HTTPS Adoption on the Web, https://static.googleusercontent.com/media/research.google.com/it//pubs/archive/46197.pdf

[11] Google, Custom Search JSON API: Introduction, https://developers.google.com/custom-search/v1/introduction

[12] Hamilton M., Wright R., Use of DNS Aliases for Network Services, https://datatracker.ietf.org/doc/html/rfc2219

[13] Hunt T. , The World's Most Popular Websites Loaded Insecurely, https://whynohttps.com/

[14] International Telecommunication Union, Individuals using the Internet, https://www.itu.int/en/ITU-D/Statistics/Pages/stat/default.aspx

[15] Internet Corporation for Assigned Names and Numbers, Registrations in IDN TLDs as Compared to the Total Number of Registrations in New gTLDs, https://www.icann.org/resources/pages/cct-metrics-domain-name-registration-2016-06-27-en

[16] Kaspersky, HTTPS doesn’t mean safe, https://www.kaspersky.com/blog/https-does-not-mean-safe/20725/

[17] Kerrisk M., Hostname(7) - Linux Manual Page, https://man7.org/linux/man-pages/man7/hostname.7.html

[18] Kumar A. , Fake SSL Certificates: How Can They Be a Problem?, https://medium.com/globant/fake-ssl-certificates-how-can-they-be-a-problem-901cfe0b34f7

[19] Marquis JP, Category Theory (Stanford Encyclopedia of Philosofy, pubblicazione 2019), https://plato.stanford.edu/entries/category-theory/

[20] Microsoft, Structure types (C# reference),
https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/builtin-types/struct

[21] Microsoft, What is the Windows Subsystem for Linux?,
https://docs.microsoft.com/en-us/windows/wsl/about

[22] Microsoft, .NET class library overview,
https://docs.microsoft.com/en-us/dotnet/standard/class-library-overview

[23] Microsoft, NuGet for Visual Studio,
https://docs.microsoft.com/en-us/nuget/quickstart/install-and-use-a-package-in-visual-studio

[24] Microsoft, AssemblyInfo Class,
https://docs.microsoft.com/en-us/dotnet/api/microsoft.visualbasic.applicationservices.assemblyinfo

[25] Microsoft, Using type dynamic,
https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/types/using-type-dynamic

[26] Microsoft, User-defined conversion operators,
https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/operators/user-defined-conversion-operators

[27] Microsoft, IEquatable<T> Interface,
https://docs.microsoft.com/en-us/dotnet/api/system.iequatable-1?view=net-6.0

[28] Microsoft, Static Constructors,
https://docs.microsoft.com/en-us/dotnet/csharp/programming-guide/classes-and-structs/static-constructors

[29] Microsoft, Generic classes and methods,
https://docs.microsoft.com/en-us/dotnet/csharp/fundamentals/types/generics

[30] Mozilla e contributori, HTTP request methods,
https://developer.mozilla.org/en-US/docs/Web/HTTP/Methods

[31] Mozilla e contributori, HTTP response status codes,
https://developer.mozilla.org/en-US/docs/Web/HTTP/Status

[32] Mozilla Foundation, Public Suffix List, https://publicsuffix.org/list/

[33] Reformat M., D'Aniello G., Gaeta M., Knowledge Graphs, Category Theory and Signatures,
https://www.researchgate.net/publication/330393976_Knowledge_Graphs_Category_Theory_and_Signatures

[34] Sampson H., Airplane WiFi is getting better. But why is it still so bad? (Washington Post, 2019),
https://www.washingtonpost.com/travel/2019/06/20/why-is-airplane-wifi-still-so-bad/

[35] Scala G. S. e contributori , Command Line Parser Library for CLR and NetStandard, https://github.com/commandlineparser/commandline

[36] Singhal A., Introducing the Knowledge Graph: things, not strings,
https://blog.google/products/search/introducing-knowledge-graph-things-not/

[37] Sommerville I. F., What is software? (ottava edizione). Addison-Wesley, 2007,

[38] Standish Group International, Chaos Report (2015),
https://www.standishgroup.com/sample_research_files/CHAOSReport2015-Final.pdf

[39] W3C Working Group, Web Services Architecture,
https://www.w3.org/TR/ws-arch/