POLITECNICO DI TORINO

Corso di laurea Magistrale in Ingegneria Informatica

Tesi di Laurea Magistrale

sviluppo di un software per il monitoring di annunci BGP

Relatori

Prof. Luca ARDITO

Prof. Claudio CASETTI

Candidato

Luca ORLANDO

tutor Inrete

Giorgio BERNARDI

Ottobre 2022

Ringraziamenti

Mi è doveroso ringraziare coloro che hanno contribuito alla realizzazione dell'elaborato. In particolare, un ringraziamento al tutor aziendale Giorgio Bernardi per il supporto fornito nella stesura della tesi e ai relatori.

Ringrazio, inoltre la mia famiglia per il sostegno morale fornito durante tutto il percorso universitario e a tutti gli amici e colleghi con cui ho condiviso i momenti felici e difficili durante gli anni di studio

Indice

Elenco delle tabelle

Elenco delle figure

Capitolo 1

Introduzione

Il protocollo BGP (Border Gateway Protocol) è alla base del sistema globale di routing di internet, che gestisce come i pacchetti vengono instradati di rete in rete attraverso lo scambio di informazioni riguardo al routing e alla raggiungibilità tra i routers di frontiera. Questo protocollo dirige i pacchetti tra gli Autonomous Systems (AS), insiemi di reti gestite da una singola azienda o service provider e crea stabilità nella rete garantendo che i routers possano adattarsi ai guasti dei percorsi di rete: quando un certo percorso non risulta percorribile, ne viene trovato rapidamente un altro. Il BGP è stato descritto inizialmente nell'RFC 1105 del 1989 ed è in uso in internet dal 1994. L'attuale versione del protocollo BGP è la versione 4 (BGP4), che è stata annunciata nell'RFC 4271 nel 2006, usata dagli Internet service providers (ISPs) e dai transit carriers per stabilire il routing tra di loro. Oltre agli ISPs e ai Transit Carriers, gli ultimi anni hanno visto espandere la domanda per il multihoming da parte delle reti degli utenti finali, in particolare per filiali di imprese e governi. Le routing tables, gestite da una certa implementazione del BGP, vengono modificate continuamente per riflettere i cambiamenti effettivi nella rete, come ad esempio quando i collegamenti vengono interrotti o quando i routers non risultano raggiungibili per un certo periodo di tempo. Nella rete nel suo insieme è normale che questi cambiamenti avvengano quasi continuamente, ma in un certo router o in un dato link, questi cambiamenti dovrebbero essere relativamente rari. I routers in cui gira il BGP, di default accettano rotte annunciate dagli altri routers BGP; questo consente un automatico e decentralizzato instradamento del traffico in internet, ma lascia anche internet potenzialmente vulnerabile ad accidentali o volute interruzioni del traffico di rete. Uno dei principali problemi riscontrati nel BGP e nell'infrastruttura internet nel suo insieme, è la crescita delle tabelle di routing e della loro complessità. Sebbene esistano diversi strumenti per esaminare il comportamento delle tabelle di routing BGP (ad esempio i Looking Glasses), i grandi Internet providers li hanno sviluppati per soddisfare le loro esigenze di

gestione e non rispondono all'esigenza di una network multihomed per utenti finali, ovvero una rete che non è "di transito", ma puo' essere descritta come una "foglia".

L'obiettivo della tesi è la realizzazione un software di monitoring degli annunci BGP, che risponda alle esigenze di una rete di utenti finali multihomed "a foglia", che non solo mostri lo stato attuale, ma permetta anche di ispezionare gli eventi che riguardano la raggiungibilità avvenuti giorni o addirittura mesi diversi, andando ad esplorare diverse possibilità per quanto concerne il reperimento e il processamento dei dati delle sessioni BGP. Il lavoro è stato sviluppato in collaborazione con la ditta Inrete, che opera con grandi aziende ed enti nella gestione dei peering multicarrier (AS/BGP) per l'ottimizzazione delle continuità di servizio, tramite cui è stato possibile disporre di dati provenienti da sessioni BGP reali e verificare alcune situazioni di anomalia.

Esistono diversi tools per analizzare il traffico di rete e catturare i dati delle sessioni BGP, come ad esempio Tshark e Tcpdump. Tuttavia, questi strumenti forniscono l'output in formato testuale, pertanto, nel caso in cui venga catturato molto traffico, risulta difficile ottenere facilmente le informazioni relative alle rotte BGP. Una possibile soluzione a questo problema potrebbe effettuare la cattura del traffico di rete, salvarlo in un file pcap e analizzarlo tramite il programma *Wireshark*, che consente di visualizzare in maniera piu' agevole, rispetto all'output testuale, i vari pacchetti di rete e consente di effettuare facilmente ricerche mediante il supporto delle regular expressions. Tale strumento, però non risulta utile per processare traffico in tempo reale, inoltre richiede vari passaggi da effettuare per settare opportunamente i tipi di dati che si desiderano visualizzare. Per questo motivo, risulta essere adeguato disporre di un software che consenta di fornire informazioni in merito alla raggiungibilità delle varie network, che possa essere usato in maniera semplice e rapida. In particolare, risulta utile analizzare il percorso che viene effettuato dai pacchetti di rete, a partire da diversi carrier, per raggiungere le network di un dato "AS leaf", ossia non di transito, come gli IXC (Internet Exchange Point) o gli ISP (Internet Service Provider). Ottenere queste informazioni, non risulta semplice da effettuare manualmente, perchè occorre connettersi a router appartenenti ai vari carrier e lanciare comandi linux specifici (come ad esempio il traceroute) per conoscere le rotte BGP attive per raggiungere le network di interesse.

Inoltre, nella realizzazione del software, obiettivo della tesi, occorre analizzare quantità significative di dati, dato che, considerando una comunicazione tra due peer BGP, in funzione del momento della giornata, vengono scambiati mediamente 200-500 messaggi di update al minuto. Tali messaggi saranno trattati e trasformati in un formato comprensibile all'utente, ma per fare ciò, è necessario disporre di uno

strumento che consenta di effettuare la cattura dei dati con tempi di calcolo adeguati, in maniera da non dover attendere eccessivamente il termine del processamento dei dati. Pertanto, una buona parte del tempo è stata impiegata nella ricerca di librerie e tools esistenti che consentano di catturare e processare i messaggi BGP in maniera piu' rapida possibile, in particolare per quanto concerne il processamento dei dati catturati dai router Linux (di cui si parlerà in maniera piu' approfondita nel capitolo 2).

Un altro elemento importante nella navigazione in internet è la sicurezza, dato che nei siti web vengono spesso inserite informazioni riservate, ad esempio per quanto concerne i siti di shopping online, vengono informazioni circa carte di credito, bancomat o altri metodi di pagamento. I malitenzionati, tuttavia potrebbero rediregere il traffico internet verso siti falsi, dall'aspetto simile ai siti originali, in modo da rubare informazioni private dagli utenti. Per effettuare questo tipo di attacco informatico, puo' essere usato il BGP Hijacking, di cui se ne tratterà nel capitolo 2, che puo' usato dai malintenzionati per dirottare il traffico BGP verso AS non corrispondenti a quelli reali della destinazione da raggiungere. Per rilevare questo tipo di attacco, gli strumenti di networking attualmente disponibili come pacchetti Linux, come ad esempio il traceroute, non risultano essere molto efficaci, dato che non forniscono una panoramica sulla rotte BGP attive per raggiungere le varie network di un dato AS "leaf" da parte dei vari carrier, in modo da individuare facilmente eventuali "AS path" anomali. Pertanto, il software di monitoring che è stato sviluppato risulta essere utile anche per rilevare attacchi hacker, oltre per ragioni legate all'efficienza nella navigazione in internet.

Capitolo 2

Funzionamento del BGP

Il BGP (Border Gateway Protocol) è un protocollo che viene usato nell'ambito degli AS (Autonomous Systems), che sono dei sistemi contenenti un insieme di networks sotto il controllo di una singola entità di amministrazione, identificato tramite un ASN (Autonomous System Number). Il BGP puo' essere di tipo IBGP (Internal BGP) che viene usato nelle comunicazioni intra-AS, cioè all'interno di un Autonomous System o EBGP (External BGP) che viene usato nelle comunicazioni inter-AS. Una sessione BGP viene stabilita tra due diversi router, chiamati peer, usando il tcp come protocollo di trasporto, in cui il router richiedente la sessione manda una richiesta all'altro peer mettendo come destination port la porta 179 e come source port una porta random. La seguente figura mostra un esempio di connessioni BGP tra diversi router sia all'interno dello stesso AS, sia tra AS diversi:

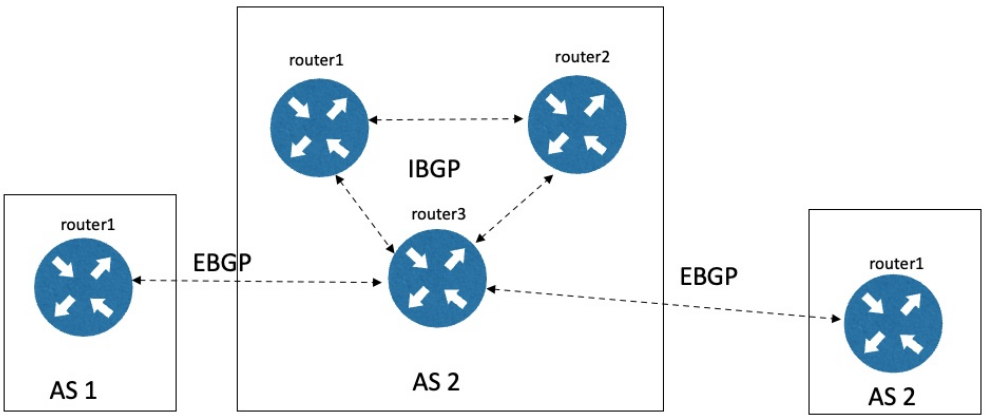


Figura 2.0.1: esempi di comunicazioni mediante il BGP

I messaggi BGP vengono scambiati tra i router e hanno lo scopo di notificare

aggiornamenti sui percorsi da seguire per raggiungere le altre networks a partire dall'AS a cui appartiene il router che invia un certo annuncio BGP. I messaggi BGP possono essere di tipo open, keepalive, notification o update; i primi tre sono usati durante lo stabilimento di una sessione BGP, mentre i messaggi di update contengono informazioni sulla raggiungibilità delle diverse networks. Il formato dei messaggi di update, come scritto nell'RFC 4271 del BGP [1], è il seguente:

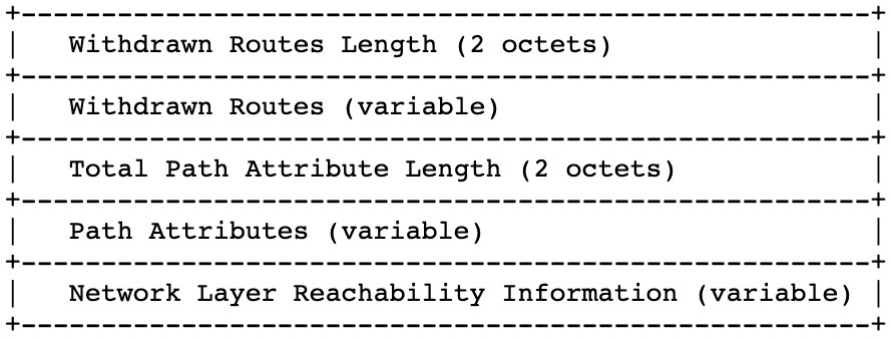


Figura 2.0.2: formato dei messaggi BGP

I campi dei messaggi di update sono:

- **Withdrawn Routes Length**: indica la dimensione in byte del campo *Withdrawn Routes*;
- **Withdrawn Routes**: contiene la lista delle networks di withdrawn;
- **Total Path Attribute Length**: indica la dimensione in bytes del campo *Path Attributes*
- **Path Attributes**: contiene diversi campi di cui alcuni sono opzionali, tra cui l'AS path, ossia gli AS da percorrere per raggiungere le network presenti nel campo *Network Layer Reachability Information* partendo dall'Autonomous System a cui appartiene il router che ha mandato il messaggio BGP di update.

I campi *Withdrawn Routes* e *Path Attributes* sono campi rilevanti del protocollo BGP, in quanto i primi contengono le network che sono state "ritirate", ossia non sono raggiungibili dall'AS del router che ha mandato l'annuncio, mentre i Path Attributes contengono l'AS path da seguire per raggiungere una data network, ossia gli AS da percorrere. La seguente immagine, presa da una cattura del traffico di rete mediante il software Wireshark, mostra come esempio, un messaggio di update che è stato mandato da un router di TIM verso un router di Inrete:

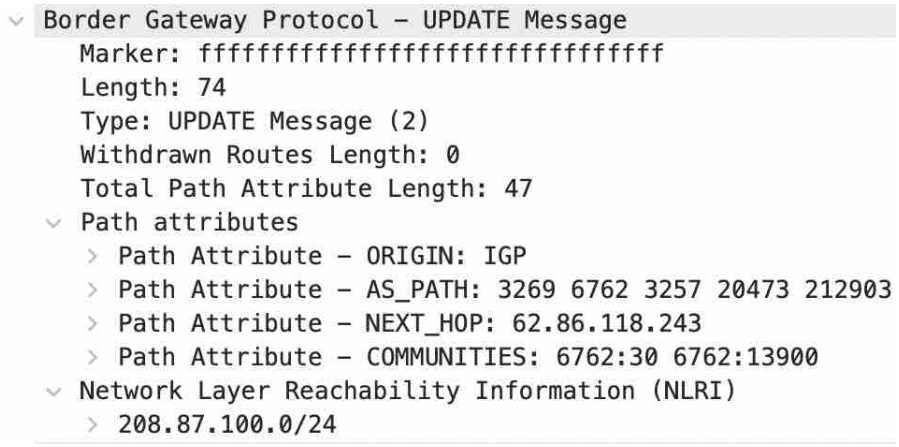


Figura 2.0.3: campi di un messaggio BGP di update

L'annuncio BGP indica che per raggiungere la network 208.87.100.0/24 partendo dal router Tim, occorrerà seguire l'AS path presente nell'omonimo campo del messaggio, nel quale ovviamente l'AS di partenza sarà l'AS di Tim, mentre l'AS di destinazione sarà quello a cui appartiene la network annunciata. Prendendo in considerazione una data network, quando si riceve un certo AS path da un messaggio di update, tale percorso sarà il percorso effettivamente seguito che andrà a sostituire i precedenti percorsi eventualmente ricevuti: pertanto se ad esempio si è ricevuto un messaggio con un AS path "x" e dopo un po' di tempo si riceve un messaggio BGP con un AS path "y", quest'ultimo sarà il percorso effettivamente seguito dai pacchetti. Nel caso in cui in una certa sessione BGP vengano propagati annunci di update di una data network e annunci di una sua sottorete, se si vuole raggiungere ad esempio un host il cui indirizzo ip ha overlap con entrambe le network, i pacchetti seguiranno la rotta BGP della network piu' interna. Pertanto se ad esempio si ricevono un annuncio di update della network 85.89.0.0/16 con AS path "1768 1278 6578 25156" e un annuncio della network 85.89.128.0/24 con AS path "1768 3278 25156", nel caso in cui si voglia raggiungere l'host 85.89.128.137, verrà seguito l'AS path "1768 3278 25156" presente come percorso per raggiungere la network 85.89.128.0/24. In maniera simile, per quanto concerne i withdrawn, se si riceve un annuncio di update di una certa network e un annuncio contenente withdrawn di una sua subnetwork, per raggiungere un host che ha overlap con entrambe le network verrà seguito il percorso propagato dalla network, essendo l'unico percorso fattibile. Pertanto, considerando l'esempio di prima, se si riceve un messaggio BGP contenente la rete 85.89.128.0/24 come withdrawn, per raggiungere l'host 85.89.128.137, verrà seguito l'AS path "1768 1278 6578 25156" presente come percorso per raggiungere la network 85.89.0.0/16

2.1 BGP routing table

La BGP routing table, detta anche RIB (Routing Information Base), è una tabella contenuta nei router che contiene le rotte da seguire per raggiungere delle date destinazioni. Per quanto concerne il BGP, esistono diversi tipi di routing tables:

- Adj-RIB-In: contiene i percorsi BGP ricevuti dai router BGP con cui si è stabilita una sessione BGP, che serviranno al router per calcolare le "best routes" per raggiungere le varie destinazioni a partire da esso;

- Loc-RIB: contiene le rotte presenti nell'Adj-RIB-In che vengono scelte come "best routes" per raggiungere le varie network e ogni router contiene una Loc-RIB;

- Adj-RIB-Out: contiene, per ogni router con cui è stabilita la sessione BGP, i percorsi che sono stati annunciati.

2.2 Routers BGP

Le sessioni BGP che vengono monitorate sono presenti in diverse tipologie di router (Cisco, Juniper e router Linux); per quanto concerne i router in cui è presente Linux come sistema operativo, viene usato Quagga come software per la gestione del protocollo BGP e di altri protocolli di routing, come ad esempio l'OSPF (Open Shortest Path First), il RIP (Routing Information Protocol) e il BGP. Per quanto concerne l'architettura di Quagga è presente Zebra, un demone che funge da livello di astrazione del kernel Unix e fornisce delle "Zserv API", ossia delle interfacce di comunicazione tra diversi componenti software, tramite Unix Domain Socket o TCP sockets rivolte ai client Quagga che implementano dei protocolli di routing e comunicano gli aggiornamenti a Zebra. Alcuni esempi di Zserv client sono ospfd per il protocollo OSPF, bgpd per il BGP che consente di usare la versione piu' recente BGPv4 e ripd per il RIP. Il seguente schema mostra le connessioni tra i vari moduli di Quagga:

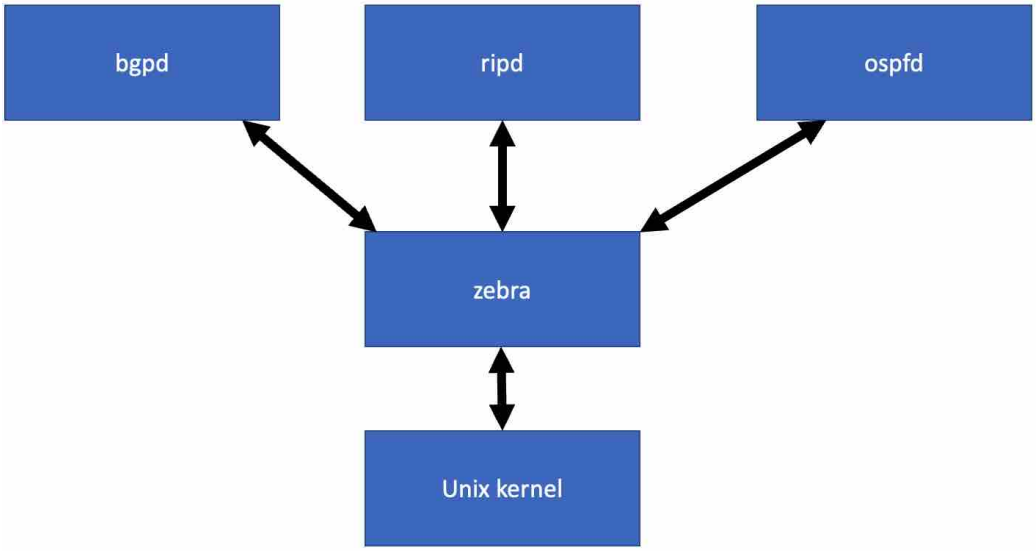


Figura 2.2.1: funzionamento di Quagga

2.3 BGP Hijacking

Il protocollo BGP puo' essere esposto all'attacco di BGP hijacking, che prevede l'annuncio di messaggi di update che prevedono percorsi che in realtà non sono dei "best path". L'attacco puo' avvenire di proposito o accidentalmente per uno dei seguenti motivi:

- un AS annuncia erroneamente una network come originata al suo interno;
- un AS annuncia una network piu' specifica di quella che potrebbe essere annunciata da un altro AS;
- un AS annuncia che esiste un percorso verso un determinato AS tramite una via piu' breve, anche se il percorso non esiste realmente.

A causa di questo attacco, i pacchetti vengono instradati verso destinazioni sbagliate e quindi entrano in un loop infinito, oppure diventano "prede" dell'AS che ha effettuato l'attacco. Pertanto, il BGP hijacking puo' essere usato per redirigere verso siti web falsi per scopi malevoli, oppure per fare in modo che i pacchetti non seguano lo "shortest path"; in questo caso, gli utenti notano rallentamente durante la navigazione internet quando vengono seguiti questi percorsi. Negli ultimi anni ci sono stati diversi casi di BGP hijacking, come ad esempio nell'aprile del 2020 in cui un attacco ha coinvolto circa 8800 prefissi di rete annunciati e ha causato danni a diverse aziende, tra cui Akamai, Amazon e Alibaba.

2.4 Analisi del traffico BGP

All'interno di Autonomous Systems che vengono usati prevalentemente come "transito" negli AS path, come ad esempio gli Internet Service Provider e gli Internet Exchange Point, sono presenti i Looking Glass servers, tramite cui è possibile lanciare comandi di monitoraggio di rete in uno dei router appartenenti all'AS. Alcuni comandi che è possibile lanciare sono il *ping* per assicurarsi che un host sia raggiungibile e vedere il tempo di risposta, oppure *traceroute* per vedere il percorso che viene seguito dai pacchetti per raggiungere una data network come mostrato nella seguente figura:

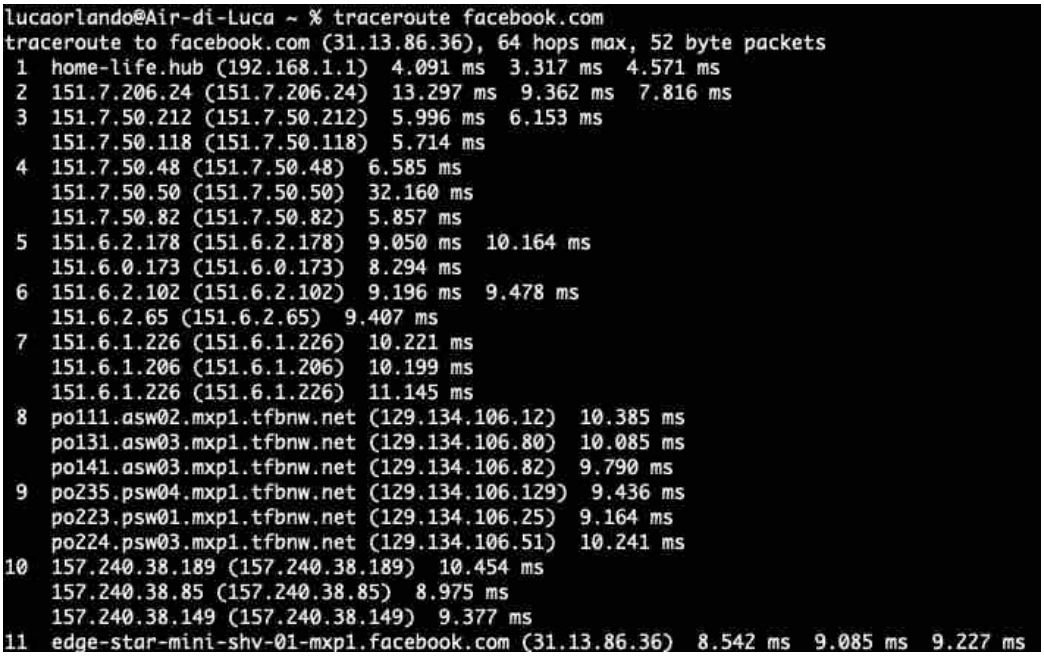


Figura 2.4.1: esempio di output di traceroute

Tuttavia, i looking Glass non risultano adatti per rilevare attacchi di BGP Hijacking, dato che in essi occorre fornire un settaggio preliminare per ottenere informazioni sulla raggiungibilità di una determinata destinazione e per individuare tali attacchi in maniera semplice, senza dover effettuare diversi passaggi, conviene avere uno strumento che fornisca una panoramica delle rotte BGP attive e ritirate (ossia di cui si sono ricevuti messaggi di update con delle "withdrawn routes") per raggiungere le varie network di un dato AS. Tale motivo è una delle ragioni che ha portato a sviluppare il software di monitoring oggetto della tesi, di cui si forniranno informazioni piu' dettagliate dopo i capitoli di introduzione, ossia dal capitolo 5 in poi.

Nello sviluppo del software di monitoring, un aspetto alla base di tutto è l'analisi del traffico di rete, che consente di raccogliere informazioni in tempo reale e puo' essere usata come strumento di troubleshooting nell'individuazione di problemi di connettività e di lentezza nella navigazione in rete. Tale attività è particolarmente importante nell'ambito di Observability di sistemi informativi che consiste nell'individuazione degli stati interni di un sistema informativo dagli input che arrivano dall'esterno. Esistono diversi tool che consentono di analizzare il traffico di rete; tra di essi alcuni possono essere usati soltanto tramite command line come ad esempio tcpdump e tshark, mentre altri sono provvisti di interfaccia grafica come Wireshark. Per quanto concerne l'analisi del traffico BGP è stato usato il protocollo BMP, dato che risulta essere uno strumento adatto a catturare i messaggi BGP, ma è supportato soltanto nei router Juniper e Cisco, pertanto nei router con Linux, in cui è presente Quagga, si è usato tcpdump per catturare i dati delle sessioni BGP.

2.5 BMP

BMP (BGP Monitoring Protocol) è un protocollo, definito nell'RFC 7854 [8] nel 2016, che viene usato per monitorare le sessioni BGP. Un BMP client (il router che viene monitorato) stabilisce diverse sessioni BGP con altri BGP speaker (detti anche BGP peer). Il BMP Client incapsula i messaggi relativi alle sessioni BGP con i BGP speakers in uno stream tcp che verrà mandato a uno o piu' BMP Collectors (ossia gli host che prelevano i messaggi BMP per analizzarli). La seguente figura riassume le connessioni BMP che vengono stabilite:

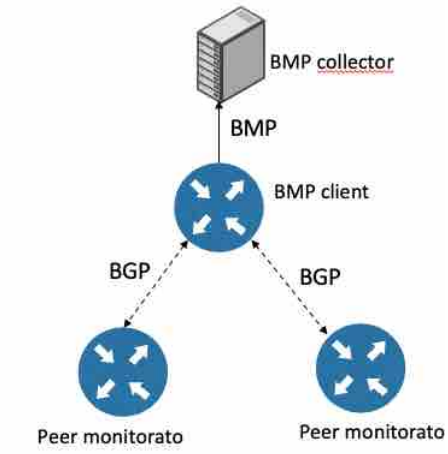


Figura 2.5.1: schema delle connessioni BMP

Per quanto concerne il collector BMP, è stato usato il software OpenBMP per prelevare i messaggi BGP di update dalle sessioni BGP, che puo' interfacciarsi con un Apache Kafka ed inviare in tempo reale i messaggi letti, in maniera che possano essere processati da diversi consumer (Apache Kafka verrà trattato in maniera piu' approfondita nel capitolo 3). Pertanto, OpenBMP ha un funzionamento simile a Firehose che permette di prelevare uno stream di dati real-time e mandarlo a diversi service.

Capitolo 3

Big Data e Data Science

Negli ultimi anni si è diffuso il termine *Big Data*, che consiste nell'analisi di grandi quantità di dati eterogenei per ottenere informazioni circa patterns, correlazioni, trends per ottenere benefici dal punto di vista del marketing, ma non solo. L'analisi di grandi moli di dati risulta inoltre utile per produrre dei dati elaborati, che siano facilmente comprensibili agli utenti finali; questo è stato l'obiettivo della realizzazione del software di monitoring, in cui è possibile avere una panoramica dello stato degli annunci BGP ottenuta elaborando diverse fonti contenenti grandi quantità di dati. La necessità di elaborare grandi volumi di dati è dettata anche dall'incremento negli anni dei prefissi di network annunciati in rete, come si vede nella seguente figura:

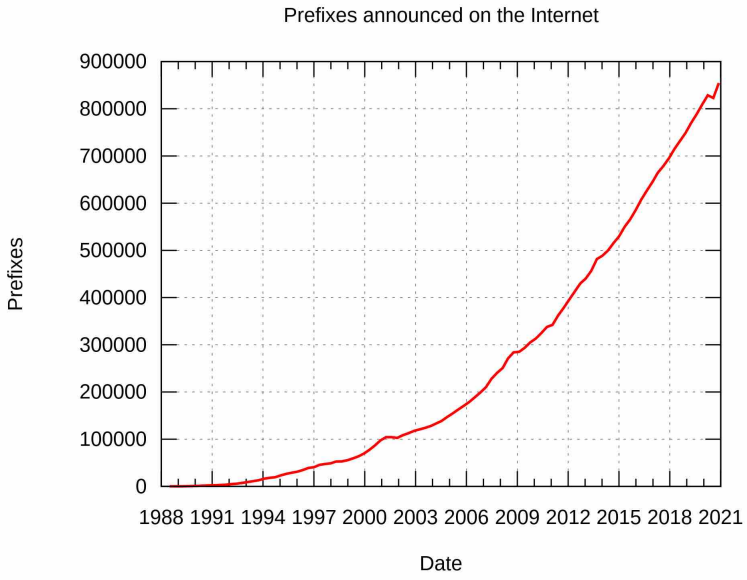


Figura 3.0.1: numero di prefissi di network annunciati negli anni [2]

In aggiunta, per quanto concerne il BGP, in una comunicazione tra due peer BGP, in funzione del momento della giornata, vengono scambiati mediamente 200-500 messaggi di update al minuto; pertanto, vista la numerosità, tale flusso non risulta essere elaborabile da un umano e occorre avere strumenti in grado di processare il flusso di dati in modo da estrarne informazioni in un formato comprensibile.

In seguito vengono riportati alcuni tool che sono stati usati nella realizzazione del software come strumenti per analizzare grandi moli di dati.

3.1 Pandas

Quando si ha a che fare con l'analisi di grandi quantità di dati, spesso vengono effettuate operazioni di filtraggio, raggruppamento o riduzione e l'uso esclusivo delle liste e dei dictionary python non risulta essere convenevole nè dal punto di vista delle performance, nè dal punto di vista della comprensione del codice. Per questi motivi, nel contesto dell'analisi di grandi quantità di dati con Python viene usata la libreria open source Pandas, che consente di adoperare il paradigma della programmazione funzionale e di gestire i dati in forma tabulare, in modo tale da rendere piu' semplice la comprensione dei risultati e permettere l'eventuale visualizzazione dei dati mediante l'uso di appropriate librerie python.

3.2 JupyterHub

Per quanto concerne la Data Science e l'analisi di grandi moli di dati, per fare in modo che i *Data Scientists* si concentrino sull'analisi dei dati e non sulla scrittura del codice, negli ultimi anni si sono diffusi i Notebook JupyterLab. Questi Notebook sono un'efficace interfaccia tra l'utente che scrive il codice e gli utilizzatori, dato che la peculiarità principale sono le celle, all'interno delle quali puo' essere eseguito del codice Python oppure puo' essere definito del codice Markdown per spiegare il funzionamento di blocchi di codice Python definiti in celle successive. La seguente figura mostra un esempio di notebook JupyterLab in cui è presente una prima cella di Markdown contente la descrizione del contenuto presente nella cella di codice ad essa sottostante:

Figura 3.2.1: esempio di notebook JupyterLab

Tramite JupyterLab, un utente puo' definire una variabile in una cella, far eseguire il codice delle celle seguenti e vedere il risultato generato in funzione del contenuto della variabile, rendendo così piu' interattiva l'esecuzione di blocchi di istruzioni Python. Tuttavia JupyterLab di per sè non supporta l'autenticazione multi-utente e pertanto, per rendere disponibile una piattaforma di sviluppo in cui ogni utente abbia una propria area personale riservata, è necessario usare JupyterHub, che supporta il login di utenti presenti in una macchina Linux ed è possibile rendere disponibili agli utenti i file presenti nelle loro home directory Linux, in tal modo gli utenti possono vedere nell'interfaccia tali file senza doverli spostare o copiare manualmente in una sezione apposita.

3.3 Voilà

I notebook JupyterLab, come detto in precedenza, risultano dei validi strumenti per fornire parametri di configurazione in date porzioni di codice python e visualizzare in maniera interattiva il risultato dell'esecuzione. Tuttavia, i notebook JupyterLab di per sè non forniscono la possibilità di scrivere codice python ed eseguire in automatico tutte le celle, visualizzando in output soltanto il risultato dell'esecuzione

senza vedere il contenuto delle varie celle. Per sopperire a questo problema è possibile usare Voilà, che consente di trasformare notebook JupyterLab in applicazioni web, che possono essere raggiunte via web come se fossero dei normali siti internet.

3.4 Apache Kafka

Apache Kafka è una piattaforma di message broker, che consente di scambiarsi messaggi in maniera asincrona. In Kafka sono presenti i concetti di producer e consumer, dove il producer pubblica i messaggi sotto un certo topic e il consumer puo' sottoscrivere un topic e ricevere i messaggi di tale argomento. Questo meccanismo di message broker consente di scambiare grandi quantità di dati tra piu' processi software e permette a un producer di pubblicare messaggi che saranno letti da piu' consumer. In aggiunta, quando un messaggio viene letto da un consumer non viene subito eliminato, in modo tale da mantenere i messaggi persistenti per un certo periodo di tempo. Pertanto, è possibile far sì che un certo messaggio venga processato da due diversi consumer, in modo tale da consentire l'elaborazione in parallelo e la possibilità di implementare diverse logiche di gestione di un messaggio per diversi consumer. Inoltre, dato che Kafka è in ascolto su una determinata porta di un dato ip, puo' essere usato come metodo per scambiare dati in real-time tra host o container diversi.

Capitolo 4

L'ecosistema Linux

In questo capitolo vengono spiegate brevemente alcune funzionalità di Linux usate nella realizzazione del software di monitoring.

4.1 Linux Containers

Nell'ambito della virtualizzazione, i container sono un metodo piu' "leggero" rispetto alle macchine virtuali, in quanto consentono di creare un ambiente isolato in cui è possibile installare in maniera pulita un miscroservizio componente un'applicazione software. Pertanto è possibile creare un container per ogni servizio presente in modo da installare le varie librerie e dipendenze necessarie per farlo funzionare correttamente, evitando che tali installazioni possano interferire negativamente con gli altri servizi presenti.

4.2 systemd

Systemd consente di gestire l'avvio di processi in background e gestirne la terminazione e il restart mediante la definizione di files di configurazione, chiamati service. Nel file di configurazione, sotto *ExecStart* viene settato il comando che lancerà il processo che verrà istanziato quando il service verrà avviato, che rimarrà in esecuzione finchè il service systemd non riceverà un segnale Linux che ne indicherà la terminazione. Tramite systemd, è possibile schedulare un service in maniera tale che venga fatto partire al boot della macchina o del container (nel caso in cui un service systemd venga settato dentro un container) ed è inoltre possibile settare delle dipendenze di avvio dei vari service; ad esempio, considerando due service A e B, è possibile fare in modo che il service A venga fatto partire prima del service B, oppure stoppare il service A nel caso in cui B venga stoppato.

4.3 Crond

Crond è un demone con il quale è possibile schedulare periodicamente l'esecuzione di un dato comando, ad esempio una volta alla settimana, una volta al giorno o ogni ora. Questo comando risulta particolarmente utile, ad esempio, nel caso in cui si vogliano schedulare l'esecuzione di script di monitoraggio dello stato del sistema o di pulizia di file di dati su disco vecchi.

Capitolo 5

Architettura hardware e software

5.1 Hardware usato

Il traffico BGP processato nel software di monitoring viene raccolto da sessioni BGP presenti in diversi carrier, pertanto il volume di dati da processare risulta essere significativo. Inizialmente si è messo in piedi il software su una macchina Raspberry Pi 4 Model B, in modo tale da vedere quanto ci si poteva spingere in basso, in termini di potenza hardware, per processare i dati delle varie sessioni BGP, cercando di ottimizzare il software in termini di tempi di procesamento. Tuttavia, si sono notati rallentamenti nell'uso della macchina mentre erano in corso le elaborazioni dei dati, pertanto si è spostato il software in una macchina Odroid-N2 in cui è presente un cluster di processori *ARM Cortex-A73 quad core* e un cluster *dual core Cortex-A53*, che risulta essere piu' performate rispetto al processore *Quad core Cortex-A72* della Raspberry. A parità di quantità di memoria RAM presente nella Raspberry Pi e nella macchina Odroid-N2, in quest'ultima si sono notati risultati decisamente migliori in termini di tempi di processamento, pertanto si deciso di adottare in maniera definitiva la macchina Odroid-N2.

5.2 architettura software

Per quanto riguarda l'architettura software, si sono creati i seguenti container lxc in modo tale da separare i vari moduli:

- **BGPstate**: contiene i vari script responsabili per la raccolta e il processamento dei messagi BGP catturati;

- **WEBint**: contiene il web server che gestisce le varie richieste provenienti dall'esterno;
- **Jvoilà**: contiene voilà, che si occupa di leggere i notebook JupyterLab corrispondenti alle pagine del software e mostrarne l'output;
- **Jhub**: contiene un server JupyterHub che consente l'accesso mullti-utente a una piattaforma Jupyter, in cui è presente un accesso comune per effettuare l'inizializzazione degli interest, di cui si parlerà nel capitolo 7.

Per gestire le richieste provenienti dall'esterno si è usato nftables, che funziona da firewall per redirigere il traffico verso il container WEBint o verso il container BGPstate nel caso in cui il traffico sia relativo a openBMP. Il web server presente all'interno di WEBint si occupa di effettuare vari controlli di sicurezza e di redirigere eventualmente il traffico verso i server presenti nei container Jhub o Jvoilà nel caso in cui le richieste siano relative rispettivamente a JupyterHub o alle pagine del software di monitoring. La seguente figura mostra la gestione delle richieste dall'esterno nella macchina Odroid:

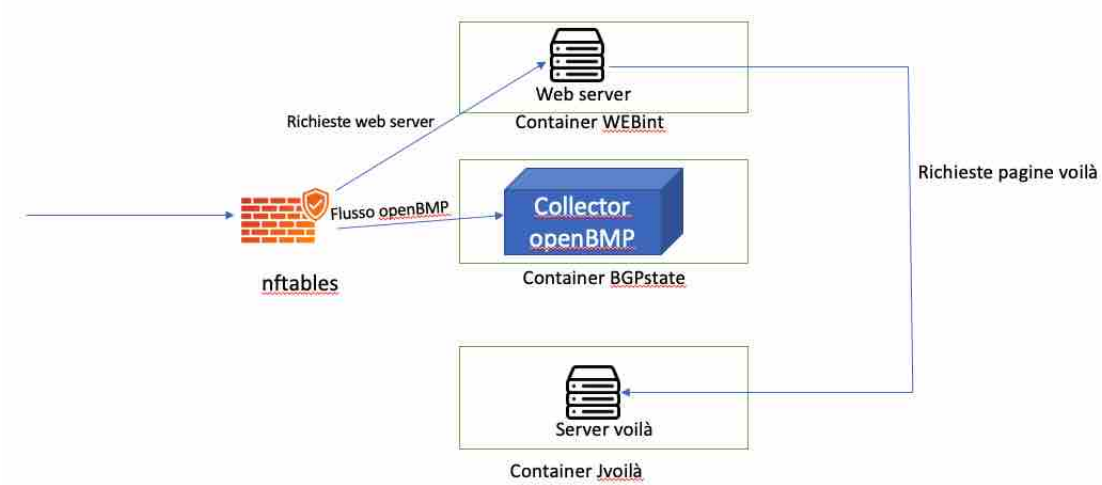


Figura 5.2.1: gestione richieste dall'esterno nella Odroid

Capitolo 6

Raccolta dei messaggi BGP

In questo capitolo si descriverà come vengono raccolti ed elaborati i messaggi BGP, che saranno elaborati in una macchina Odroid-N2 e mostrati nell'interfaccia del software di monitoring. Per quanto riguarda il reperimento dei dati BGP, vengono catturati i messaggi di update, che contengono informazioni circa gli AS che i pacchetti di rete devono percorrere per raggiungere le varie networks. Per gestire la raccolta e il processamento dei messaggi BGP, si è creato un apposito container lxc *BGPstate*, dove sono presenti script che consentono di processare i dati provenienti da tcpdump e da openbmp, di cui si parlerà in seguito.

Nella directory */usr/BGPstate/Data* sono presenti le directory con i nomi dei carrier, al cui interno sono contenuti i files ".csv" e ".cnt" di ogni peer BGP, per ogni giorno. I file csv contengono gli annunci BGP catturati e hanno un formato simile: < *carrier* > _<source>_<peer_ip>_<day>.csv, dove:

- <carrier> corrisponde al nome del carrier (Fastweb, Retelit, Tim, Windtre o Colt);
- <source> è il source ip, cioè l'indirizzo del router da cui vengono ricevuti i messaggi BGP;
- <peer_ip> è il peer ip, cioè l'indirizzo IP del router che ha stabilito la connessione BGP con il router "source". Nel processamento vengono considerati i messaggi BGP che hanno come sorgente il peer IP e come destinazione il source IP (e non viceversa);
- <day> è la data corrispondente al file csv.

I files file cnt, invece contengono il numero di messaggi di update e di withdrawn ricevuti ogni minuto che servono per creare gli allarmi di notifica che verranno spiegati in seguito. Di seguito viene riportato un esempio di organizzazione delle

directory contenenti i file csv/cnt, dove *ip1* e *ip2* sono gli indirizzi ip dei due peer BGP:

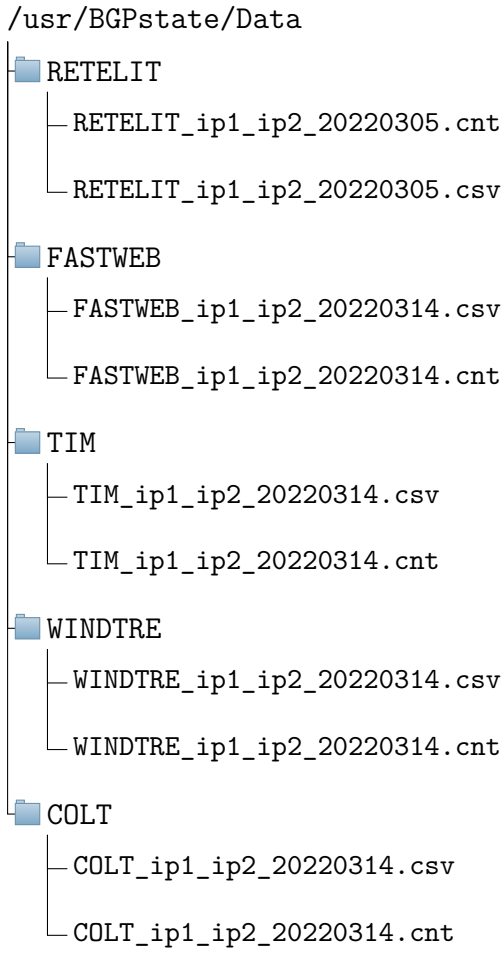


Figura 6.0.1: esempio di files giornalieri relativi agli annunci BGP

Gli script responsabili del reperimento e del processamento dei messaggi BGP, di cui si discuterà in seguito, sono presenti nella directory */usr/BGPstate/bin*.

6.1 OpenBMP

Per catturare i messaggi BGP dalle sessioni dei peer, viene usato OpenBMP, che come accennato nel capitolo 2, implementa un collector BMP che riceve dai BMP client i dati BGP. Ad esempio, prendendo come riferimento la figura sottostante, il router appartenente a *Inrete*, con IP *ipA*, cattura i messaggi BGP relativi alla

comunicazione con *ipB* e li manda alla macchina Odroid dove è in ascolto il collector openBMP

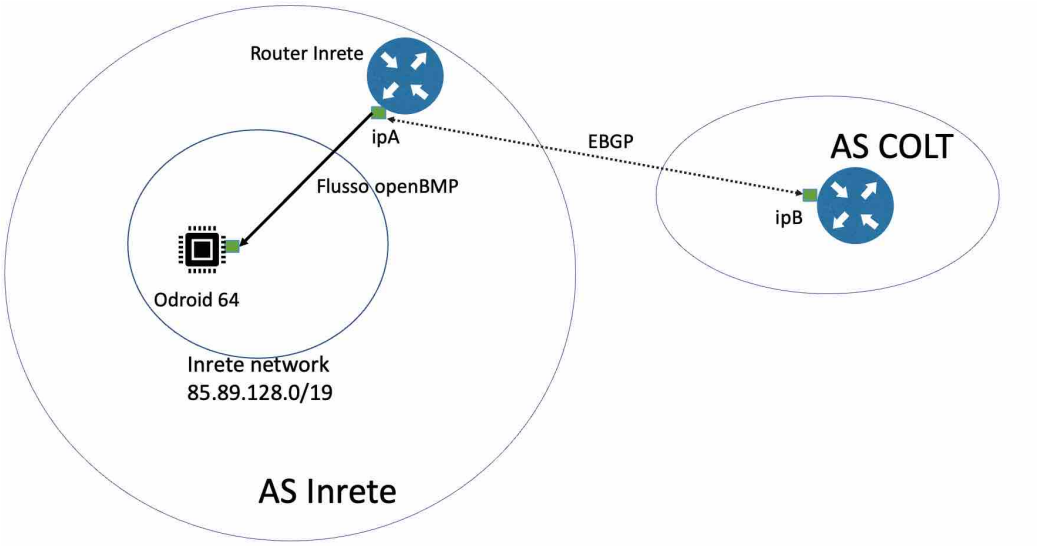


Figura 6.1.1: connessione con i router OpenBMP

I messaggi catturati dal collector openBMP vengono pubblicati in kafka sotto il topic *openbmp.parsed.unicast_prefix* e per leggere tali messaggi, si è adoperato lo script python ConsumerOpenBMP.py che internamente usa la libreria Python *OpenBMP API Message* per decodificare oppurtamente i messaggi kafka in maniera da poterli processare opportunamente. Tuttavia, ci è accorti che tale libreria ha un bug nella decodifica del timestamp dei messaggi Kafka di openBMP in quanto lo setta alla precisione in secondi sebbene debba essere supportato il timestamp fino ai microsecondi. Facendo debug, si è riusciti ad individuare il punto nel codice in cui è presente il problema e a correggerlo, facendo in modo che la precisione del timestamp sia in millisecondi, dato che si è deciso che è sufficiente tale precisione per poter stabilire l'ordine di arrivo dei messaggi BGP. Lo script *ConsumerOpenBMP.py*, dopo aver letto i messaggi tramite la libreria *OpenBMP API Message* e averli formattati opportunamente, svolge le seguenti operazioni:

- scrive i messaggi ricevuti in file .csv;
- scrive il numero di messaggi di update e di withdrawn ricevuti ogni minuti all'interno di file con estensione.cnt;
- pubblica i messaggi ricevuti sotto il topic kafka *BGPcsv*, che saranno processati da un consumer kafka "globale" che scriverà i messaggi all'interno dei csv degli "interest".

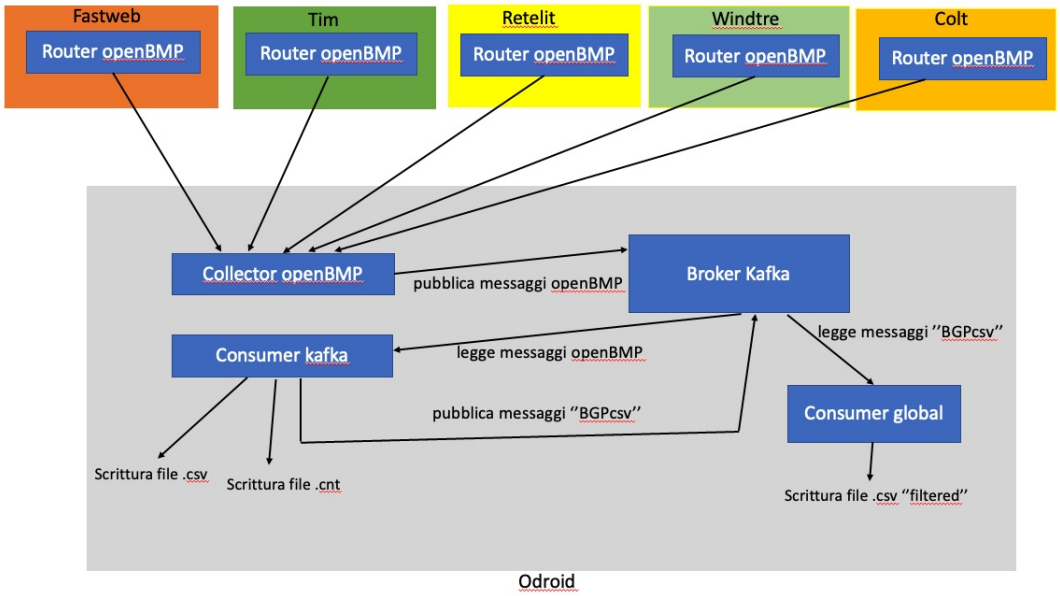


Figura 6.1.2: funzionamento di OpenBMP

6.2 Tcpdump

OpenBMP risulta essere un adeguato strumento per catturare i messaggi di update dalle sessioni BGP; tuttavia, il protocollo BMP è supportato soltanto nei router Cisco e Juniper, pertanto, se si vuole monitorare il traffico BGP per altri router, occorre usare altri mezzi. Nello specifico, si è adoperato Tcpdump per catturare il traffico BGP nei router linux, in maniera da avere la stessa tipologia di contenuto informativo di OpenBMP, ossia i messaggi di update. Per quanto concerne i dispostivi usati in questo contesto, due router dei quali uno appartiene all'AS Inrete, mentre l'altro appartiene a un AS esterno, hanno stabilito una comunicazione EBGP. Il dispositivo di Inrete a cui ci si vuole connettere per catturare i dati, ha tcpdump in ascolto in modalità promiscua (cioè cattura tutto il traffico che si puo' osservare e non solo il traffico diretto all'interfaccia stessa) e stampa il traffico catturato in formato binario. Il comando tcpdump che viene lanciato è il seguente:

```
1 /usr/sbin/tcpdump -f -s 0 -i eth0 -nn -N -S -vvv -tttt port bgp and
    src host <srcHost>
2
```

Nel comando soprastante, "*port bgp and src host <srcHost>*" indica di catturare i messaggi BGP in cui l'ip sorgente è <srcHost>, in modo tale da prelevare soltanto i messaggi che vengono mandati dal router con cui si è stabilita la sessione BGP

verso il router in cui si è lanciato il comando tcpdump e non viceversa, indicando così la direzione del flusso. Per reperire i messaggi BGP da processare, dalla macchina Odroid64 ci si connette via sshpass, in modo tale da usare il protocollo ssh (Secure Shell) che permette di entrare in una macchina da remoto, tramite una shell, cifrando il traffico che viene scambiato. Pertanto, prendendo ad esempio la figura sottostante, verrà processato il traffico BGP che ha come source IP *ipC* e come destination IP *ipB*

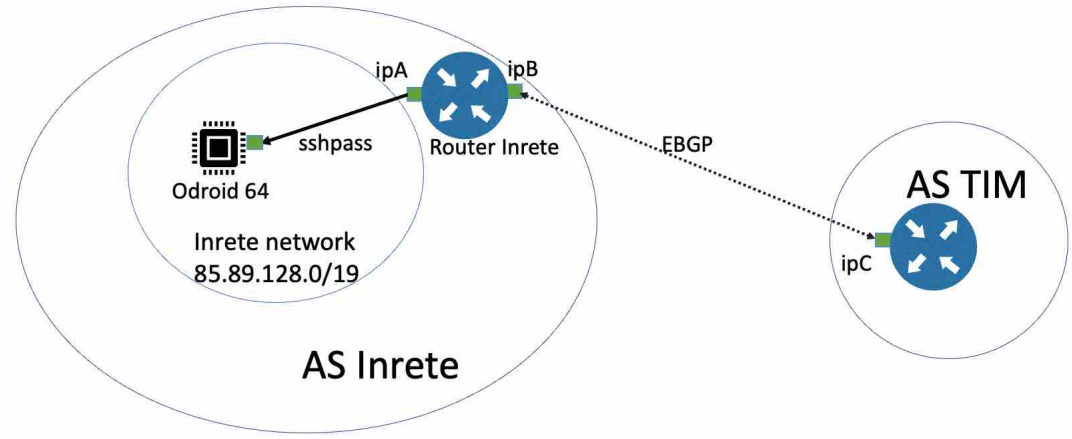


Figura 6.2.1: raccolta dei messaggi BGP dai router

Inizialmente, per processare il flusso binario tcpdump si è adoperata la libreria pbgpp [12], che consente di visualizzare il flusso di dati in formato testuale o in formato json. Si è modificato parte del codice della libreria per fare in modo che i messaggi BGP vengano presi in formato "json" memorizzando ogni messaggio letto dentro un dictionary python e scritti dentro file csv, anzichè stamparli in stdout, come mostrato nel seguente schema:

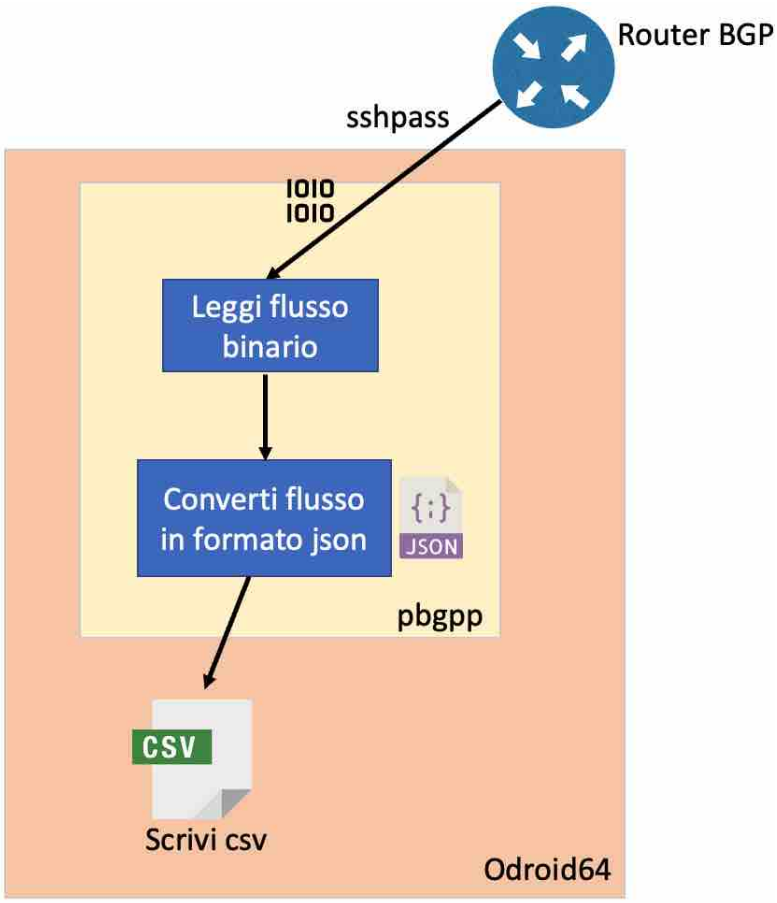


Figura 6.2.2: processamento dei messaggi BGP tramite pbgpp

Adoperando pbgpp, tuttavia in seguito ci si è accorti che non venivano catturati tutti i messaggi BGP; facendo un po' di debug e ricerche nella documentazione, ci si è accorti che c'erano problemi con il riassemblaggio dei pacchetti. Pertanto, si è adoperato lato Odroid64 uno script python per parsificare il flusso binario, che internamente usa tcpdump, che fornisce l'output in formato testuale; facendo del pattern matching mediante le regular expressions, vengono individuati i vari campi dei messaggi BGP da inserire nei file csv. Tuttavia, ci si è accorti che tcpdump ha dei problemi con il riassemblaggio dei pacchetti tcp, pertanto alcuni messaggi venivano persi. Pertanto, dato che si è usato Wireshark per fare il debug, con cui non si sono avuti problemi di riassemblaggio, si è deciso di adoperare Tshark (che è la versione command-line di Wireshark) per processare il flusso binario anzichè tcpdump. Tuttavia, facendo dei test processando dei file pcap contenenti l'output binario del tcpdump per un certo istante di tempo, ci si è resi conto che lo script python non è molto performante e facendo dei test con un time profiler (ossia

tramite una libreria python che fornisce statistiche sul tempo di esecuzione e in particolare indica le tempistiche delle singole istruzioni nel codice) si è notato che il "collo di bottiglia" è il processamento dei dati mediante le regular expressions. Pertanto, si è provato a creare uno script che facesse le stesse operazioni del codice python, ma scritto in perl e facendo il confronto dei tempi di esecuzione dei due script con diversi files pcap binari contenenti il traffico catturato, si sono ottenuti i seguenti risultati:

	consumer python	consumer perl
pcap1	32s	32s
pcap2	52s	23s
pcap3	2min 10s	52s
pcap4	2min 34s	57s

Tabella 6.1: confronto dei tempi di esecuzione tra lo script python e lo script perl per il processamento dei dati tcpdump

Lo script *Consumer_tshark.pl* che si è messo in piedi processa i messaggi BGP ricevuti e successivamente:

- scrive i messaggi ricevuti in file .csv;
- scrive il numero di messaggi di update e di withdrawn ricevuti ogni minuto all'interno di file con estensione.cnt;
- pubblica i messaggi ricevuti sotto il topic kafka *BGPcsv*, che saranno processati da un consumer kafka "globale" che scriverà i messaggi all'interno dei csv degli "interest" (di cui si parlerà nel capitolo 7).

Lo schema seguente sintetizza il funzionamento del processamento dei messaggi BGP tramite tcpdump:

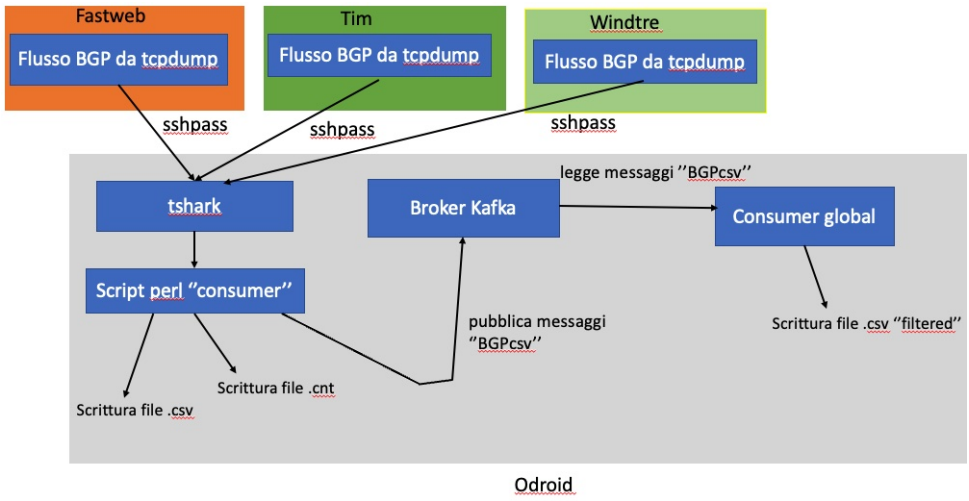


Figura 6.2.3: funzionamento di Tcpdump

Nel container "BGPstate" sono presenti i vari script per il reperimento e processamento dei messaggi BGP. Si è creato inizialmente uno script per per il processamento dei dati tramite tcpdump, ma ci si è resi conto che tcpdump ha dei problemi con il riassemblaggio dei pacchetti, perciò in seguito si è deciso di adoperare Tshark, un analizzatore alternativo del traffico di rete tramite command line. Per ogni router da cui si riceve il flusso binario di dati, è stato creato uno script bash avente nome startConsumer_<sourceIP>_<carrier>.sh che effettua la connessione via sshpass al router. Di seguito viene riportato uno degli script (l'indirizzo ip del router è stato sostiuito con *<ipRouter>* dato che è una informazione riservata):

```
1 #! /bin/sh
2 /usr/bin/sshpass -p $password /usr/bin/ssh -T luca@<ipRouter> |
3 /usr/bin/buffer -s 16224 -b 2048 | \
4 /usr/BGPstate/bin/Consumer_tshark.pl - --router=<ipRouter>\
5 - --carrier=TIM
6
```

Ogni script che preleva i dati dai router tramite sshpass è messo in pipe allo script *Consumer_tshark.pl* che si occupa di scrivere i messaggi BGP letti dentro la directory */usr/BGPstate/Data/<nome_carrier>/* che contiene i files .csv e files .cnt per i peer BGP appartenenti al carrier monitorato. vengono memorizzati dentro dei file csv i messaggi BGP visti per ogni carrier. Internamente, lo script *Consumer_tshark.pl* invoca il comando:

```
/usr/bin/tshark -r - -V -Y '(tcp.srcport == 179 or tcp.dstport ==
    179)\
and bgp.type == 2'
```

che esegue il programma "tshark" che andrà a leggere in stdin (indicato con l'opzione "-r -") i messaggi tcp con source o destination port uguale a 179, che è la porta con cui viene aperta una sessione BGP. Con l'opzione "bgp.type=2", cioè soltanto i messaggi di update, scartando quindi i messaggi di open, keepalive e notifications.

6.2.1 gestione dell'esecuzione dei demoni tcpdump

Per garantire che gli script relativi al processamento e reperimento del flusso BGP fornito dai router che hanno in esecuzione tcpdump vengano eseguiti in background e che la loro esecuzione sia riavviata in automatico nel caso in cui vengano stoppati per qualche causa esterna (ad esempio se viene persa la connessione sshpass con il router BGP), si è deciso di creare dei services systemd per gestire la loro esecuzione. I services systemd sono presenti nella directory */etc/systemd/system* e si occupano di far partire lo script sh relativo a uno specifico sourceIP, di uno specifico carrier, di seguito viene riportato un esempio di service (viene riportato "<ipRouter>" al posto dell'ip del router, dato che sono informazioni riservate):

```
[Unit]
Description=Avvio del consumer tshark per <ipRouter> con WINDTRE
PartOf=kafka.service

[Service]
Type=simple
ExecStart=/usr/BGPstate/bin/startConsumer_<ipRouter>_WINDTRE.sh
ExecStop=/usr/BGPstate/bin/kill_tcpdump_sshpass.pl\
- -router=<ipRouter>
RestartSec=2s
Restart=always

[Install]
WantedBy=multi-user.target
```

Nel service, *partOf=kafka.service* indica che se il service relativo ad Apache Kafka viene stoppato o riavviato, tale azione viene effettuata anche dal service che si sta considerando, dato che lo script *Consumer_tshark.pl* richiede che il broker di Apache Kafka sia in esecuzione. Inoltre, sono stati settati 2 secondi come *RestartSec*

dato che, in caso di restart, il broker di Apache Kafka ci impiega qualche secondo per completare il riavvio; pertanto il tempo di restart di 100 ms di default non è sufficiente. Dato che tshark aumenta progressivamente il consumo di memoria con il passare del tempo e non è presente alcuna opzione settabile nell'esecuzione per indicare di limitare il consumo di memoria, si è deciso di eseguire l'avvio di una nuova istanza dei service systemd *startConsumer_<sourceIP>_<carrier>@.service* ogni giorno alle due di notte. Per gestire l'avvio di una nuova istanza del service systemd, è stato creato lo script */usr/BGPstate/bin/start_new_tshark.pl*, il cui codice è il seguente:

```
#!/usr/bin/perl
use strict;
use warnings;
use Getopt::Long;

my $source;
my $carrier;
GetOptions ('source=s' => \$source, 'carrier=s' => \$carrier);
if ((!defined($source)) || ("$source" eq "")) {
  print STDERR "Invalid --source\n";
  exit;
}

if ((!defined($carrier)) || ("$carrier" eq "")) {
  print STDERR "Invalid --carrier\n";
  exit;
}

sub is_running {
  my ($service) = @_;
  my $running = 0;
  open(CMD,"/bin/systemctl status $service|");
  while(<CMD>) {
    if(/active \(running\)/) {
      $running=1;
      last;
    }
  }
  return $running;
}

my $prefix_service = "startConsumer_" . "$source" . "_" . "$carrier"\
. "\@";
my $service_first = "$prefix_service" . "1.service";
my $service_second = "$prefix_service" . "2.service";
```

```
my $service_not_running = 1;
my $service_start = "";
my $command_start;
if (is_running("$service_first")) {
  $service_start = $service_second;
  $service_not_running = 0;
}
elsif (is_running("$service_second")) {
  $service_start = $service_first;
  $service_not_running = 0;
}

if ($service_not_running) {
  $command_start = "/bin/systemctl start $service_first";
} else {
  $command_start = "/bin/systemctl start $service_start";
}
`$command_start`;
```

Lo script riceve come parametri "–source" e "–carrier", che corrispondono rispettivamente al sourceIP e al carrier, che serviranno per identificare il nome dell'istanza del service da avviare. Le istanze del service hanno il formato *startConsumer_<sourceIP>_<carrier>@1.service* e *startConsumer_<sourceIP>_<carrier>@2.service*.

Nel codice, la funzione *is_running* riceve come parametro il nome dell'istanza di un service e restituisce 1 se l'istanza è in esecuzione, 0 altrimenti. Per fare questa verifica, viene invocato il comando */bin/systemctl status <nomeService>* il cui output, nel caso in cui un service sia in esecuzione, conterrà la stringa *active (running)*. Tramite la funzione *is_running*, viene dedotta quale sia l'istanza del service in esecuzione e viene invocato il comando */bin/systemctl start <nomeService>* per l'istanza del service che non è in esecuzione. Nel *Consumer_tshark.pl*, quando viene letto il primo messaggio da processare, viene eseguito il seguente codice:

```
if ($first) {
    my $service_stop = get_service_to_stop();
    my $command_stop = "/bin/systemctl stop $service_stop &";
    system("$command_stop");
    $first = 0;
}
```

Tramite la funzione *get_service_to_stop*, viene ricavato il nome dell'istanza del service da stoppare e, successivamente, viene invocato il comando *systemctl stop* su tale service; tale comando viene eseguito in background in modo tale da continuare l'esecuzione delle istruzioni seguenti senza dover attendere la terminazione del comando di stop. La funzione *get_service_to_stop* ha il seguente codice:

```
sub get_service_to_stop {
  my $prefix_service = "startConsumer_" . "$Par_router" . "_"\
  . "$Par_carrier@";
  my $service_first = "$prefix_service" . "1.service";
  my $time_start_first = get_time_start("$service_first");
  my $service_second = "$prefix_service" . "2.service";
  my $time_start_second = get_time_start("$service_second");
  if ($time_start_first <  $time_start_second) {
    return $service_first;
  } else {
    return $service_second;
  }
}
```

Nella funzione, tramite la funzione *get_time_start* viene reperito il timestamp epoch di avvio delle istanze dei service *startConsumer_<sourceIP>_<carrier>@1.service* e *startConsumer_<sourceIP>_<carrier>@2.service* in cui il sourceIP viene specificato nella variabile *$Par_router*, mentre il carrier corrisponde alla variabile *$Par_carrier*. Successivamente, viene restituita l'istanza del service con il timestamp di start piu' basso e viene fatto il confronto tra il timestamp di start dell'istanza "1" del service e dell'istanza "2".
Lo script *kill_tcpdump_sshpass.pl*, che viene invocato quando viene stoppata l'istanza di un service (dato che questo script è indicato nella parte *ExecStop* del service) esegue il seguente codice:

```
#!/usr/bin/perl
use strict;
use warnings;
use Getopt::Long;

my $router;
GetOptions ('router=s' => \$router);
if ((!defined($router)) || ("$router" eq "")) {
  print STDERR "Invalid --router\n";
  exit;
}
my $password = "*****";
my $command = "/usr/bin/sshpass -p '$password' /usr/bin/ssh -T
    lucakill@" . "$router";
my $res = `$command`;
print "$res\n";
```

Nello script ci si connette via sshpass all'indirizzo IP del router con l'account *lucakill*. Tramite questa connessione, nel router viene svuotato il buffer tcpdump dai messaggi rimasti da processare e successivamente viene stoppato tcpudmp.

Questa operazione aggiuntiva serve per fare in modo che vengono processati dallo script *Consumer_tshark.pl* gli eventuali messaggi BGP residui nel buffer.

6.3 allarmi Monin "BGP Updates"

La raccolta dei messaggi BGP potrebbe interrompersi a causa di qualche evento imprevisto, ad esempio per un guasto nel router BGP che invia i messaggi alla macchina Odroid, oppure per interruzione temporanea della corrente elettrica. Per assicurarsi che si stiano raccogliendo messaggi BGP da tutti i router BGP, si è deciso di mettere in piedi delle notifiche sul sistema di allarmistica "Monin" di Inrete in cui loggandosi si possono vedere notifiche di eventi di varia natura. Nel sistema è stato creato un allarme per ogni peer BGP da cui vengono raccolti i dati BGP, che sarà di colore verde se non ci sono problemi, ossia la macchina Odroid 64 continua a ricevere i messaggi BGP della comunicazione del peer BGP, come mostrato nella figura seguente:

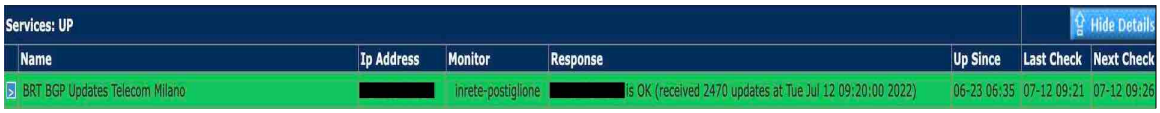


Figura 6.3.1: Monin situazione senza problemi

Nel caso in cui non si ricevano messaggi da un dato peer BGP, l'allarme Monin sarà di colore rosso, come nel seguente caso:

Figura 6.3.2: allarme Monin rosso

Per generare gli allarmi su Monin, nel container *BGPstate* della macchina Odroid64 è stata creata un'API su un server python, che consente di conoscere, per un dato *<carrier>_<source>_<peer_ip>*, l'ultimo minuto in cui si sono ricevuti messaggi BGP e il numero di messaggi di update e di withdrawn ricevuti in tale minuto. Nel sistema Monin sono stati configurati dei controlli periodici relativi ai peer BGP usati per raccogliere i dati in cui viene richiamata l'API e nel caso in cui la differenza in tempo tra il timestamp attuale e l'ultimo minuto in cui si sono ricevuti dei messaggi BGP supera una certa soglia di secondi, viene emesso un allarme Monin di colore rosso di notifica dell'anomalia. La figura seguente mostra il flusso di esecuzione delle richieste all'API del BGP per gli allarmi di ogni *<Carrier>_<SourceIP>_<PeerIP>*.

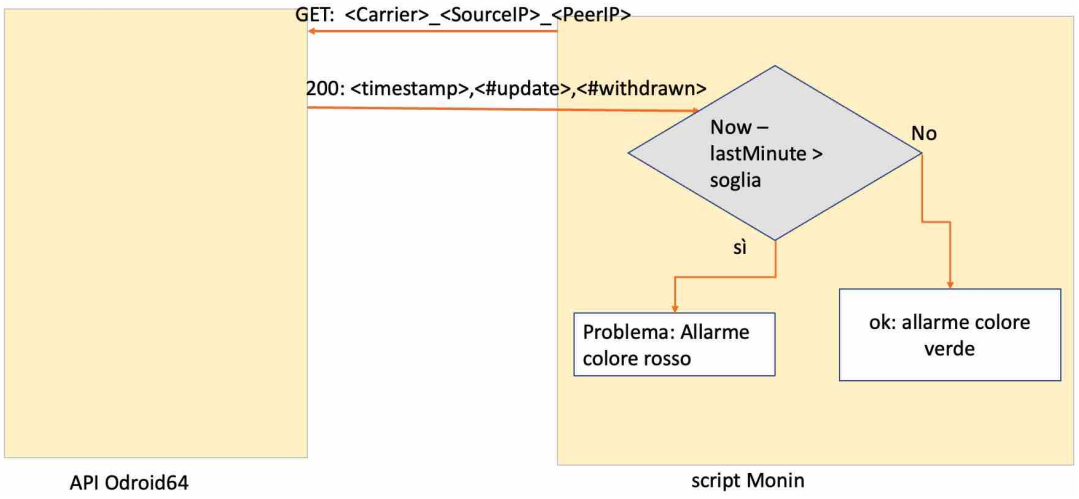


Figura 6.3.3: funzionamento delle richieste con Monin

Nella condizione della figura, viene verificato se la differenza tra il timestamp epoch attuale e il timestamp dell'ultimo minuto in cui sono stati catturati annunci BGP è maggiore di una determinata soglia di tempo e, in tal caso, viene emesso un allarme Monin di colore rosso.

6.4 Archiviazione dei messaggi BGP

Dato che ogni file csv presente nelle directory dei carrier in */usr/BGPstate/data* occupa circa 20 MB, a lungo andare i file csv potrebbero causare problemi relativi all'occupazione dello spazio su disco. Pertanto si è deciso di creare, per ogni carrier, dei files zip annuali con il formato: <carrier>_<sourceIP>_<PeerIP>_<anno>.zip contenenti i files csv relativi a piu' di 6 mesi fa. Ad esempio, per il carrier Fastweb, i csv del 2021 piu' vecchi di 6 mesi relativi al router con sourceIP <ipSource> e al PeerIP <ipPeer> saranno memorizzati nello zip:
FASTWEB_<ipSource>_<ipPeer>_2021.zip. Di seguito viene riportato il codice dello script di cui è stata schedulata l'esecuzione tramite crond ogni giorno alle 3 di notte:

```
1 #!/usr/bin/env python3
2 #
3
4 import os
5 import glob
6 import time
7 import datetime
```

```
import re
from zipfile import ZIP_DEFLATED
from zipfile import ZipFile

data_dir = "/usr/BGPstate/Data/"
min_days_csv_old = 180
now = time.time()
now = int(str(now).split(".")[0])
seconds_int_6_months = now - min_days_csv_old  * 86400
day_6_months = datetime.datetime.utcfromtimestamp(
    seconds_int_6_months)\
 .strftime('%Y%m%d')
for carrier in os.listdir(data_dir):
  regex_file = re.compile("^" + carrier +
  "_[0-9]+\.[0-9]+\.[0-9]+\.[0-9]+_[0-9]+\.[0-9]+\.[0-9]+\.[0-9]+
  _[0-9]+\.(csv|cnt)")
  for path_file_old in glob.glob(data_dir + carrier + "/*"):
    filename_no_path = os.path.basename(path_file_old)
    if regex_file.search(filename_no_path) is not None:
      split_values = filename_no_path.split("_")
      day_file = split_values[3]
      if day_file < day_6_months:
        filename_zip = "_".join(split_values[0:3])
        year = day_file[0:4]
        filename_zip += "_" + year + ".zip"
        path_filename_zip = os.path.dirname(path_file_old) + "/" + \
        filename_zip
        if os.path.isfile(path_filename_zip):
          zipObj = ZipFile(path_filename_zip, 'a',\
          compression=ZIP_DEFLATED, compresslevel=6)
        else:
          zipObj = ZipFile(path_filename_zip, 'w',\
          compression=ZIP_DEFLATED, compresslevel=6)
        zipObj.write(path_file_old, os.path.basename(path_file_old))
        zipObj.close()
        os.remove(path_file_old)
```

Nello script, tramite la variabile *min_days_csv_old*, viene definito il numero minimo di giorni di differenza del timestamp corrente rispetto al timestamp indicato nel file .csv o .cnt (dato che questi files hanno formato <carrier>_<sourceIP>_<PeerIP>_<giorno>) che si andrà a considerare nel loop successivo. Successivamente, viene definita la variabile *day_6_months*, corrispondente al giorno, espresso nel formato "%Y%m%d" (cioè con una stringa <anno><mese><giorno>, ad esempio "20220124") di 6 mesi fa. Viene definita l'espressione regolare regex_file che servirà a controllare se il file, della directory */usr/BGPstate/Data/<Carrier>* che si sta considerando è un file .csv o .cnt, dato che si andrà a eseguire un loop su tutti i files della directory. Per ogni file che matcha la regex e che ha la data antecendente alla data della

variabile "day_6_months", viene controllato se esiste il file ".zip" relativo alla terna (sourceIP, peerIP, anno) che si sta considerando; nel caso in cui esista, viene aperto in modalità "append", altrimenti viene aperto in modalità "write" in modo da creare l'archivio zip. In seguito, tramite l'istruzione *zipObj.write(path_file_old,os.path.basename(path_file_old))*, verrà inserito il file dentro l'archivio zip e, tramite l'istruzione *os.remove(path_file_old)* viene rimosso il file

Capitolo 7

raggruppamento dei messaggi BGP

7.1 scrittura dei csv "filtered"

Inizialmente, facendo dei test tramite un Notebook JupyterLab, si è provato a filtrare tutto il traffico relativo a un certo AS leaf andando a leggere tutti i csv di un determinato range temporale; tuttavia ci si è resi conto che il processamento non era molto performante dal punto di vista computazionale, per cui si è deciso di creare un file csv per ogni "interest", ossia per un certo gruppo di network appartenenti a determinati AS e; tali file csv si è deciso di chiamarli "filtered" dato che per scriverli viene eseguito un filtraggio sugli annunci BGP. Nello specifico, in un file json vengono definiti, per un dato interest, le network e i suoi AS e, nel file csv filtered verranno scritti in real time gli annunci BGP la cui network appartiene a tale interest o il cui AS Path contiene uno degli AS definiti. Nell'interfaccia del "BGP Sentinel", di cui si parlerà nel capitolo 8, un utente sarà abilitato a vedere lo stato delle rotte BGP di un dato interest, che potrà essere scelto mediante un menù a tendina che conterrà la lista degli interest di cui l'utente è abilitato a vedere le rotte BGP e, per visualizzarne i dati, verrà processato il file csv relativo all'interest selezionato. Per popolare i file csv "filtered", si è deciso di convertire i messaggi letti tramite OpenBMP e Tcpdump in un formato comune e inviarli al Broker kafka sotto il topic "BGPcsv". Questi messaggi kafka, verranno letti dallo script python *Consumer_global.py* in cui è instanziato un consumer Kafka che fungerà da "consumer globale", con lo scopo di leggere tutto il flusso di messaggi e scrivere nel csv dell'interest i messaggi relativi. Le network e gli AS corrispondenti ai vari interest sono state definite dentro il file json "Consumer_global.json", in cui degli esempi di entry sono:

```
{
  "Interests": {
    "Inrete": {
      "Nets": [
        "85.89.128.0/19"
      ],
      "ASes": [
        25156
      ],
      "tz": "Europe/Rome"
    },
    "Polito": {
      "Nets": [
        "130.192.55.240/32"
      ],
      "tz": "Europe/Rome"
    },
  }
}
```

Figura 7.1.1: esempio di json contenente la definizione degli "interest"

Nel json soprastante:

- i nomi "Inrete" e "Polito" sono i nomi degli interest;
- nel campo "Nets" sono definiti le network dell'interest;
- nel campo ASes sono definiti gli AS dell'interest (questo campo è opzionale, dato che per certi interest si vuole monitorare solo lo stato degli annunci BGP relativi ad alcune network);
- il campo "tz" contiene il timezone di cui si visualizzerà il datetime dell'interest, in modo tale da supportare la visualizzazione con fusi orari predefiniti (in un capitolo dedicato si spiegherà meglio l'utilità di questo campo)

Prendendo l'esempio visto nella figura 6.2.1 del capitolo 6, che mostra una sessione BGP tra un router appartenente all'AS Inrete e un router appartenente all'AS TIM, gli annunci BGP mandati dall'ip *ipC* verso *ipB* saranno relativi a percorsi a di network conosciuti da *ipC*. Per quanto riguarda la network 85.89.128.0/19, dato che è la network a cui appartiene un ip dell'interfaccia del router dell'AS di Inrete, gli annunci di tale network non saranno presi in condiderazione, pertanto se

si vogliono sapere gli annunci BGP relativi alla network 85.89.128.0/19 da TIM, occorrerà prelevarli da un'altra sessione BGP in cui nessuno dei due peer abbia interfacce con ip che appartengono a tale network. Per gestire questi casi particolari, è stato creato il file di configurazione *ReferenceForCarrier.json*, il cui contenuto è il seguente (gli ip del source ip e del peer ip sono stati sostituiti rispettivamente dalle stringhe <ipSource> e <ipPeer>, dato che sono informazioni riservate):

```
{
  "ReferenceForCarrier" : {
    "FASTWEB_<ipSource>_<ipPeer>": {
      "carrier": "FASTWEB",
      "excludeAS": 25156,
      "excludeNet": "85.89.128.0/19"
    },
    "FASTWEB_<ipSource>_<ipPeer>": {
      "carrier": "FASTWEB",
      "includeAS": 25156,
      "includeNet": "85.89.128.0/19"
    },
    "TIM_<ipSource>_<ipPeer>": {
      "carrier": "TIM",
      "excludeAS": 25156,
      "excludeNet": "85.89.128.0/19"
    },
    "TIM_<ipSource>_<ipPeer>": {
      "carrier": "TIM",
      "includeAS": 25156,
      "includeNet": "85.89.128.0/19"
    },
    "RETELIT_<ipSource>_<ipPeer>": {
      "carrier": "RETELIT",
      "excludeAS": 28767,
      "excludeNet": "31.193.56.0/21"
    },
    "WINDTRE_<ipSource>_<ipPeer>": {
      "carrier": "WINDTRE"
    },
    "COLT_<ipSource>_<ipPeer>": {
      "carrier": "COLT",
      "excludeAS": 28767,
      "excludeNet": "31.193.56.0/21"
    }
  }
}
```

Figura 7.1.2: json "ReferenceForCarrier"

Il json contiene come chiavi delle stringhe <carrier>_<source_ip>_<peer_ip> e come valore contiene un json con i seguenti campi (non è obbligatorio che ci siano

tutti i campi):

- **carrier**: il carrier a cui appartiene il peer ip
- **includeAS**: i messaggi BGP relativi alla terna (carrier, source_ip, peer_ip) presenti nella chiave del json vengono scritti nel csv dell'interest se contengono l'AS specificato come valore nell'AS path e verranno processati
- **excludeAS**: i messaggi BGP relativi alla terna (carrier, source_ip, peer_ip) presenti nella chiave del json vengono scritti nel csv dell'interest se contengono l'AS specificato come valore nell'AS path non verranno processati
- **includeNet**: i messaggi BGP relativi alla terna (carrier, source_ip, peer_ip) presenti nella chiave del json vengono processati se hanno la network che coincide o è una sottorete della network specificata nel campo del json
- **excludeNet**: i messaggi BGP relativi alla terna (carrier, source_ip, peer_ip) presenti nella chiave del json se hanno la network che coincide o è una sottorete della network specificata nel campo del json non vengono processati

Il file json viene letto dallo script *Consumer_global.py* che quando riceve i messaggi BGP da processare corrispondenti a un <carrier>_<source_ip>_<peer_ip> presenti come chiave, se l'AS path contiene l'AS specificato in *includeAS* o la network appartiene alla network specificata in *includeNet*, il messaggio verrà processato. Viceversa, se il messaggio contiene l'AS specificato in *excludeAS* o la network appartiene alla network specificata in *excludeNet*, tale messaggio non viene processato. Il processamento successivo dei messaggi BGP consiste nel controllare se tale messaggio è relativo a qualche interest e, in caso affermativo, tale messaggio verrà scritto nel file csv di tale interest

7.2 inizializzazione degli "interest"

Per quanto concerne la definizione di un nuovo interest, essa viene effettuata internamente da Inrete, usando il notebook JupyterLab "inizializzazione_interest.ipynb" in cui è possibile settare il nome dell'interest, le network, gli AS (opzionalmente) e il timezone di appartenenza (che servirà per visualizzare il datetime degli annunci BGP con il fuso orario corretto) ed eseguendo tutte le celle del notebook, si leggono i csv giornalieri relativi fino a un massimo di giorni predefinito a partire da oggi. Di seguito viene riportata la cella del notebook in cui l'utente definisce l'interest, in cui come esempio è stato settato "Inrete":

1.0 - Definizione degli ip e AS di interesse

Nel codice seguente, vengono definiti gli ip dei quali si vogliono cercare annunci con network con intersezione con le "Nets" indicate e gli AS dei quali si vogliono cercare annunci che li contengano (in AS_path o in AS_Aggr)

[1]:
```
MaxDays = 150

Interests = {
  "Inrete": {
    "Nets": [
      "85.89.128.0/19"
    ],
    "ASes": [
      25156
    ],
    "tz": "Europe/Rome"
  }
}
```

Figura 7.2.1: inizializzazione dell'interest

Al termine dell'inizializzazione, verrà aggiornato il json *Consumer_global.json* con le informazioni del nuovo interest e gli annunci BGP ad esso relativi potranno essere visualizzati nelle pagine del software di monitoring di cui si parlerà in maniera piu' approfondita nel capitolo *BGP Sentinel* e, inoltre, verrà aggiunto l'interest appena creato alla lista degli interest visualizzabili dall'utente "Inrete", che è abilitato a visualizzare tutti gli interest. Il seguente schema riassume la procedura di inizializzazione dell'interest:

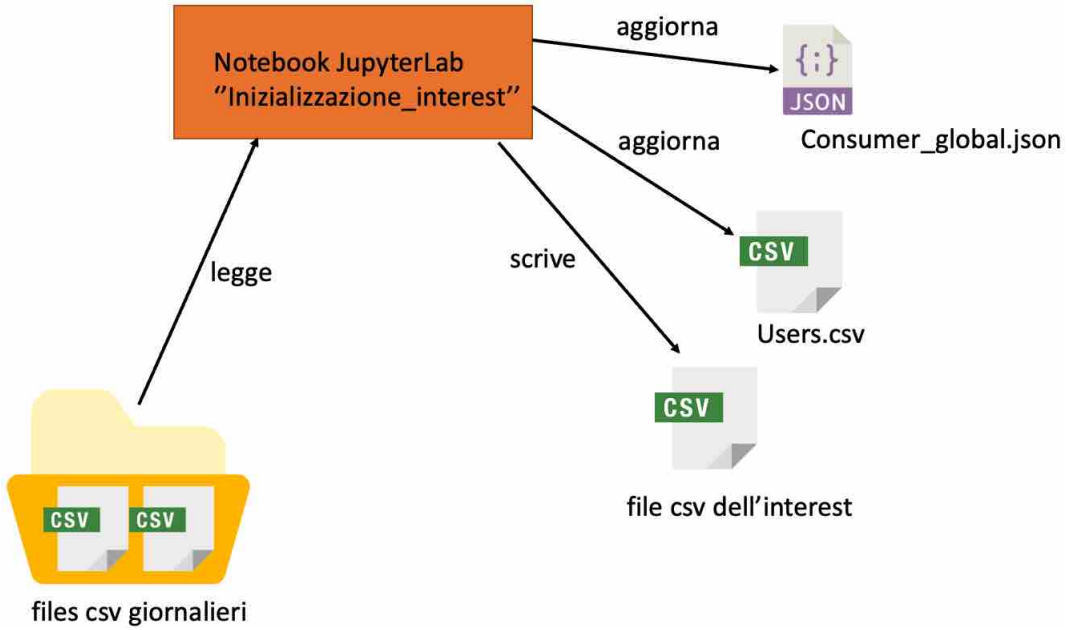


Figura 7.2.2: inizializzazione dell'interest

Il notebook JupyterLab è presente all'interno dell'ambiente JupyterHub che è stato creato in maniera tale da consentire ai dipendenti di Inrete di avere delle proprie aree personali in cui possano creare liberamente dei notebook JupyterLab. I notebook JupyterLab di uso comune a piu' utenti, tra cui il notebook *inizializzazione_interest.ipynb*, sono stati posti sotto l'utente bgpinfo, in modo tale da avere un'area condivisa in cui sia possibile avere in futuro, in caso di necessità, vari notebook che possano essere usati per fare analisi sui dati BGP.

Capitolo 8

BGP Sentinel

Tramite l'applicativo "voilà", si è realizzata un'interfaccia grafica, che potrà essere usata per uso interno aziendale per vedere lo stato degli annunci BGP degli "interest" che l'utente loggato è abilitato a visualizzare. Il seguente schema mostra le varie pagine del software in cui puo' navigare un utente loggato:

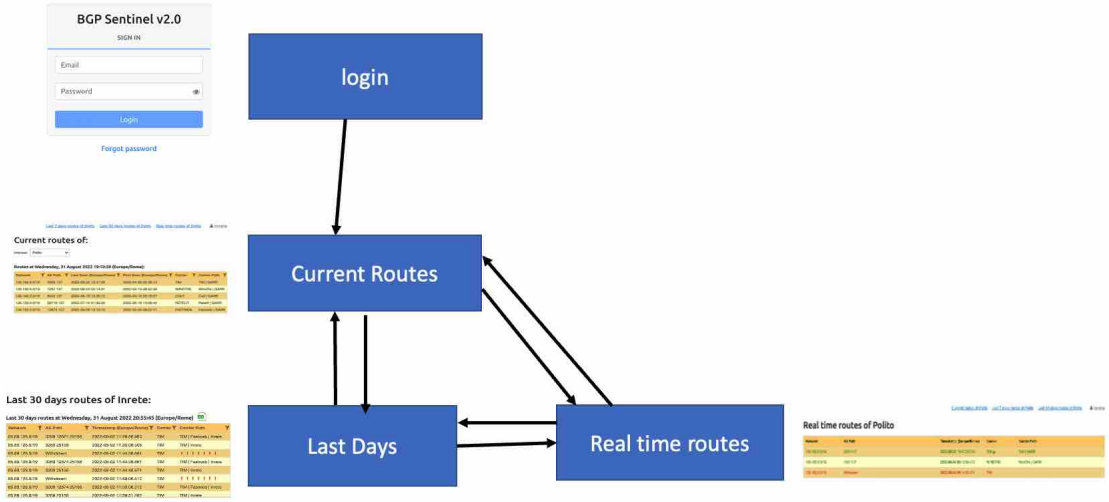


Figura 8.0.1: pagine del "BGP Sentinel"

Dato che voilà consente di eseguire le celle di un notebook JupyterLab, si sono creati i notebook JupyterLab:

- **Current_routes.ipynb**: nel caso in cui un utente loggato sia abilitato a vedere piu' interest, consente di selezionare l'interest di cui visualizzarne i dati e per ogni network dell'interest visualizza le rotte BGP attive, se presenti;

- **Last_days.ipynb**: visualizza tutti gli annunci BGP degli ultimi 7 o 30 giorni per le network dell'interest selezionato nella pagina delle Current Routes;

- **Real_time_routes.ipynb** visualizza in tempo reali tutti gli annunci BGP che vengono visti per l'interest selezionato nella pagina delle Current Routes;

Per ospitare l'installazione di voilà e JupyterLab, è stato creato il container lxc *Jvoilà*, che ha quindi lo scopo di isolare il codice i moduli software relativi all'interfaccia grafica dal resto.

8.1 autenticazione

Per vedere le funzionalità offerte dal software "BGP Sentinel 2.0", occorre effetuare il login nella seguente pagina:

Figura 8.1.1: pagine del login di "BGP Sentinel 2.0"

Dopo aver effettuato il login, viene creato un token JWT (JSON Web Token) corrispondente all'utente che si è autenticato, che contiene varie informazioni tra cui il nome utente, la email e il timestamp fino a cui il token è valido. Un esempio di token JWT decodificato è il seguente:

```
{
  "eml": "esempio@staff.inrete.it",
  "acc": "Inrete",
  "srv": "BS",
  "sub": "BS",
  "url": "bso.inrete.it/routes",
  "last_pw": 1604334965846,
  "grant": "ALL",
  "iat": 1647283851,
  "exp": 1647284751
}
```

Figura 8.1.2: esempio di token jwt decodificato

8.2 Current Routes

Dopo aver effettuato il login, viene visualizzata la pagina delle *Current Routes*, che contiene informazioni sullo stato attuale delle rotte BGP riguardanti le network degli interest da parte dei vari carrier. Viene reperito il nome dell'utente che si è loggato dal campo "acc" del token JWT e, leggendo il file csv *users.csv* viene estratta la lista degli interest che l'utente è abilitato a visualizzare. Tale file csv, contiene ogni riga con il formato:
<nomeAccount>,{"interest": <listaInterest>}
dove <nomeAccount> è il nome dell'account dell'utente, mentre <listaInterest> contiene la lista degli interest che l'utente è abilitato a visualizzare. Ad esempio, è stato creato un account Inrete contente la lista di tutti gli interest attualmente presenti, la cui entry è la seguente (non sono stati riportati gli interest reali dato che sono informazioni riservate):

```
Inrete, {"interest": ["interest1", "inretest2", "interest3"]}
```

Nella pagina delle *Current Routes*, se si è abilitati a visualizzare piu' di un interest, tramite in menù di dropdown di potrà selezionare l'interest di cui vedere le routes BGP attive. Nella pagina, per ogni carrier e per ogni network vengono visualizzati l'AS path, il timestamp in cui l'annuncio è apparso l'ultima volta, il timestamp in cui è apparso la prima volta e il carrier path, ossia le organizations a cui appartengono gli AS presenti nell'AS path. Il carrier path è stato inserito in modo tale che l'utente non debba ricordare a memoria chi sono gli AS visualizzati nell'AS path, in modo tale da rendere piu' comprensibili i dati che vengono visualizzati e,

per reperire i nomi delle organizations, vengono eseguite delle chiamate a un'api pubblica.

8.2.1 esempio di lettura dei dati delle Current Routes attive

Nella figura seguente viene visualizzato l'output per un esempio di interest *Polito*, che è stato inizializzando mettendo 130.192.55.240/32 (ossia l'ip di didattica.polito.it) come "network" dell'interest:

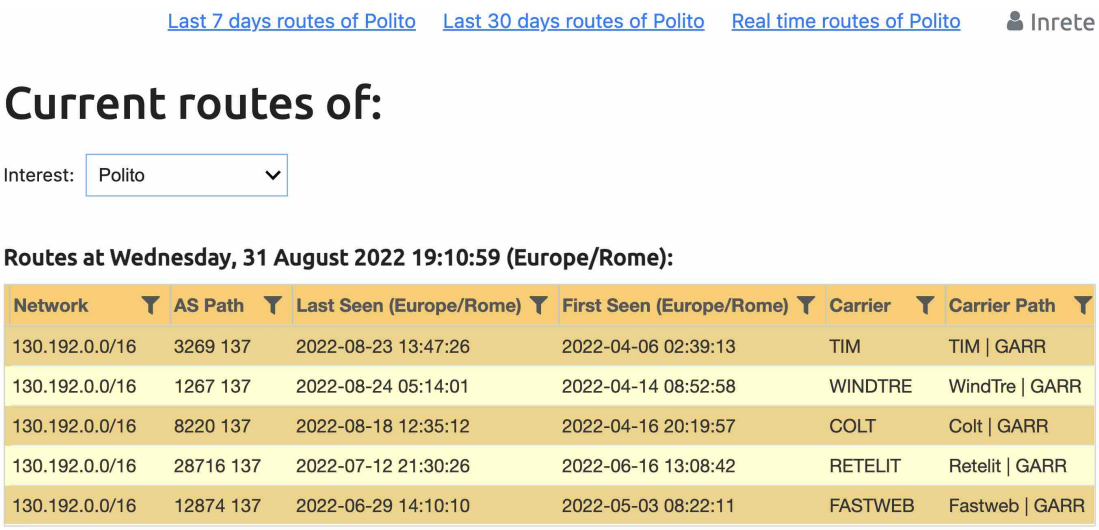
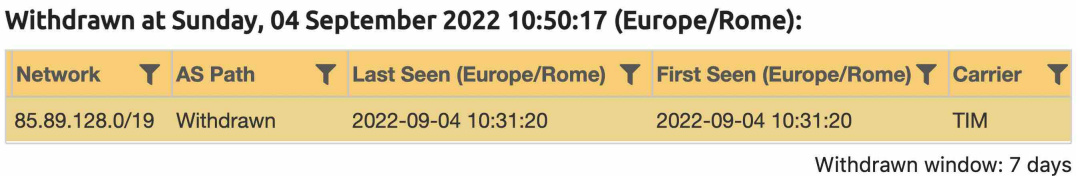


Figura 8.2.1: current routes di Polito

prendendo ad esempio la seconda riga della tabella, dal carrier WINDTRE, per raggiungere la network 130.192.0.0/16 occorre passare per gli AS 1267 137 che corrispondono rispettivamente alle organizations WINDTRE, e GARR. Questo annuncio è stato visto per la prima volta il 14/04/2022 alle 8:52:58 e l'ultima volta è stato visto il 24/08/2022 alle 5:14:01

8.2.2 Withdrawn routes

Nel caso in cui per un certo carrier, l'ultimo messaggio di una data network sia Withdrawn, non comparirà la riga corrispondente alla network/carrier nella tabella delle routes attive vista precedentemente, ma comparirà nella tabella dei *withdrawn*. Tuttavia, si è deciso di mostrare nella tabella dei withdrawn soltanto le entry in cui il timestamp del *Last Seen* non sia precedente a una settimana dal timestamp corrente. Di seguito viene mostrato un esempio di tabella contenente i Withdrawn (il withdrawn presente è stato inserito come esempio e non è da considerarsi veritiero):



Withdrawn at Sunday, 04 September 2022 10:50:17 (Europe/Rome):

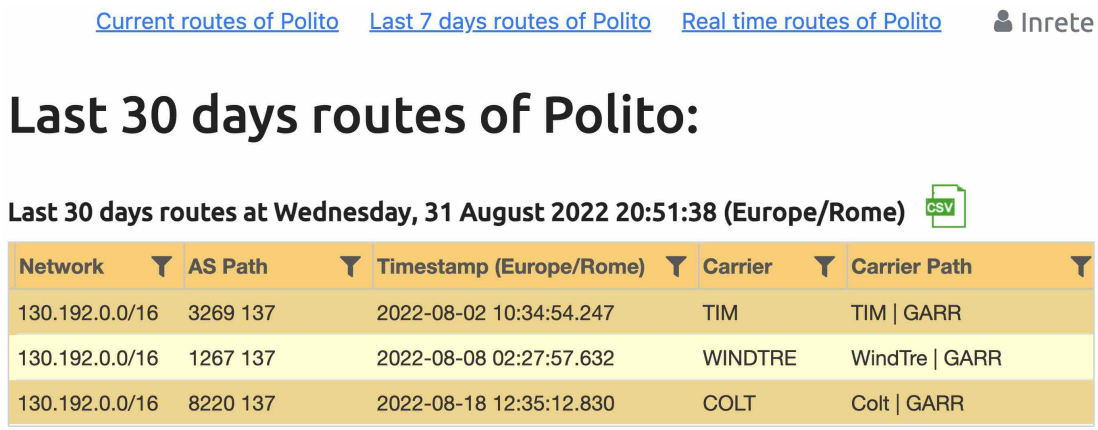
Network	AS Path	Last Seen (Europe/Rome)	First Seen (Europe/Rome)	Carrier
85.89.128.0/19	Withdrawn	2022-09-04 10:31:20	2022-09-04 10:31:20	TIM

Withdrawn window: 7 days

Figura 8.2.2: withdrawn routes di Inrete

8.3 Last Days

cliccando sui link "Last 7 days" o "Last 30 days", è possibile aprire la pagina che consente di visualizzare tutti gli annunci BGP dell'interest visti negli ultimi 7 giorni o 30 giorni. Un esempio di output è il seguente:



Current routes of Polito Last 7 days routes of Polito Real time routes of Polito Inrete

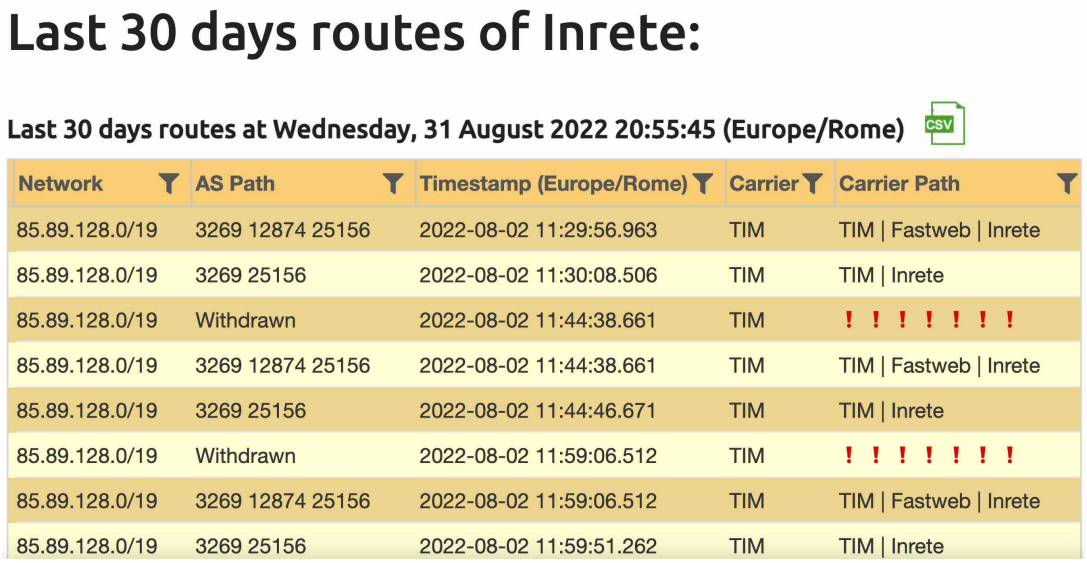
Last 30 days routes of Polito:

Last 30 days routes at Wednesday, 31 August 2022 20:51:38 (Europe/Rome) CSV

Network	AS Path	Timestamp (Europe/Rome)	Carrier	Carrier Path
130.192.0.0/16	3269 137	2022-08-02 10:34:54.247	TIM	TIM | GARR
130.192.0.0/16	1267 137	2022-08-08 02:27:57.632	WINDTRE	WindTre | GARR
130.192.0.0/16	8220 137	2022-08-18 12:35:12.830	COLT	Colt | GARR

Figura 8.3.1: last days

Nella tabella viene mostrato, per ogni network dell'interest annunciata, il carrier da cui parte il percorso del campo *AS Path*, il timestamp in cui è stato visto l'annuncio e il carrier path che, come nella pagina delle Current Routes, indica le organizations a cui appartengono gli AS dell'AS path. La tabella mostrata nella pagina dei "Last Days" puo' essere scaricata in formato csv cliccando sul'icona verde "csv" presente vicino al titolo della tabella. Gli annunci BGP, mostrati nella figura soprastante, sono relativi all'interest "Polito" e si puo' notare che la situazione risulta essere stabile, dato che non sono presenti messaggi BGP contenenti dei Withdrawn. La figura seguente, invece mostra gli annunci degli ultimi 30 giorni di Inrete:



Last 30 days routes of Inrete:

Last 30 days routes at Wednesday, 31 August 2022 20:55:45 (Europe/Rome)

Network	AS Path	Timestamp (Europe/Rome)	Carrier	Carrier Path
85.89.128.0/19	3269 12874 25156	2022-08-02 11:29:56.963	TIM	TIM | Fastweb | Inrete
85.89.128.0/19	3269 25156	2022-08-02 11:30:08.506	TIM	TIM | Inrete
85.89.128.0/19	Withdrawn	2022-08-02 11:44:38.661	TIM	! ! ! ! ! ! !
85.89.128.0/19	3269 12874 25156	2022-08-02 11:44:38.661	TIM	TIM | Fastweb | Inrete
85.89.128.0/19	3269 25156	2022-08-02 11:44:46.671	TIM	TIM | Inrete
85.89.128.0/19	Withdrawn	2022-08-02 11:59:06.512	TIM	! ! ! ! ! ! !
85.89.128.0/19	3269 12874 25156	2022-08-02 11:59:06.512	TIM	TIM | Fastweb | Inrete
85.89.128.0/19	3269 25156	2022-08-02 11:59:51.262	TIM	TIM | Inrete

Figura 8.3.2: annunci degli ultimi 30 giorni di Inrete

Nella tabella si notano, per una certa network, messaggi di update seguiti da messaggi contenenti withdrawn; ciò significa che si sono verificate delle interruzioni del traffico di rete che possono essere dovute a diverse cause (ad esempio per manutenzione, configurazione sbagliata dei router BGP o malfunzionamenti). Pertanto, in questo caso, contrariamente all'interest Polito, la situazione delle rotte BGP non è risultata stabile.

8.4 Real time

Nella pagina degli annunci in real time, vengono mostrati tutti gli annunci in tempo reale di un dato interest. Nella visualizzazione vengono mostrati gli stessi campi mostrati nella visualizzazione delle routes dei "last days" e gli annunci di update vengono mostrati in verde, mentre gli annunci di withdrawn vengono mostrati in rosso (le tabelle contenti gli annunci real time mostrate in seguito contengono dati di esempio e non sono da considerarsi veritiere):

Current routes of Inrete Last 7 days routes of Inrete Last 30 days routes of Inrete Inrete

Real time routes of Inrete

Network	AS Path	Timestamp (Europe/Rome)	Carrier	Carrier Path
85.89.128.0/19	Withdrawn	2022-09-04 10:31:20.345	TIM	
85.89.128.0/19	3269 25156	2022-09-04 10:33:00.315	TIM	TIM | Inrete
85.89.128.0/19	28716 12874 25156	2022-09-04 10:34:40.245	RETELIT	Retelit | Fastweb | Inrete

Figura 8.4.1: annunci in real time

passando il cursore su una riga della tabella, inoltre, viene mostrato un tooltip con informazioni sul router da cui si ottengono i messaggi BGP e il secondo peer con cui esso ha stabilito la sessione BGP.

Figura 8.4.2: tooltip negli annunci in real time

Per identificare la presenza del campo "aggregator", nella tabella dei messaggi real time visualizzati viene inserito un simbolo copto vicino al nome del carrier. Il campo aggregator contiene, per un dato BGP speaker che ha formato una route aggregata, l'indirizzo IP e l'AS number a cui appartiene. Nella seguente figura è mostrato un esempio di tabella degli annunci real time contenente un annuncio BGP con l'aggregator:

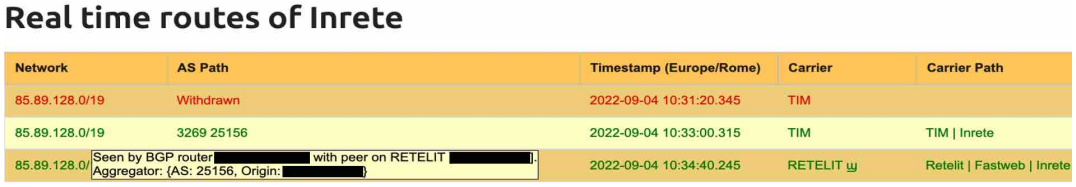


Figura 8.4.3: annunci in real time con aggregator

Capitolo 9

Conclusioni

Si è riusciti a realizzare un prototipo di un software di monitoring che per un dato AS leaf consenta di visualizzare lo stato corrente della raggiungibilità delle varie network, gli annunci BGP degli ultimi 7 o 30 giorni e gli annunci BGP che vengono ricevuti in tempo reale. In particolare, mediante l'uso dell'interfaccia del "BGP Sentinel", si evita di dover digitare svariati comandi linux per conoscere lo stato degli annunci BGP, riducendo in questo modo i tempi e gli step necessari ad ottenere tali informazioni. Una possibile implementazione futura è la realizzazione di allarmi sul sistema Monin di Inrete in merito agli stati "anomali" del BGP, ossia le situazioni in cui, prendendo una data network di un "interest", lo stato corrente delle rotte BGP è withdrawn per qualche carrier o withdrawn per tutti i carrier. Per rappresentare gli stati del BGP, si provato a usare le *Reti di Petri*, che sono un modello matematico che puo' essere usato per descrivere un sistema distribuito, ma successivamente si è abbandonata questa strada perchè ci si è accorti che, tramite le librerie python e javascript attualmente presenti, che consentono di creare reti di Petri, l'elaborazione dei dati risulta essere troppo laboriosa.

Bibliografia

[1] *BGP rfc4271*. URL: https://datatracker.ietf.org/doc/html/rfc4271 (cit. a p. 5).

[2] *BGP*. URL: https://en.wikipedia.org/wiki/Border_Gateway_Protocol (cit. a p. 13).

[3] *AS*. URL: https://en.wikipedia.org/wiki/Autonomous_system_(Internet).

[4] *quagga*. URL: https://www.quagga.net/.

[5] *quagga*. URL: https://en.wikipedia.org/wiki/Quagga_(software).

[6] *BGP hijacking*. URL: https://www.cloudflare.com/it-it/learning/security/glossary/bgp-hijacking/.

[7] *BGP hijacking*. URL: https://www.anapaya.net/blog/border-gateway-protocol-hijacking-examples-and-solutions.

[8] *BMP rfc*. URL: https://datatracker.ietf.org/doc/html/rfc7854 (cit. a p. 10).

[9] *openBMP API Message library*. URL: https://github.com/SNAS/openbmp-python-api-message.

[10] *looking glass*. URL: https://en.wikipedia.org/wiki/Looking_Glass_server.

[11] *tcpdump man page*. URL: https://www.tcpdump.org/manpages/tcpdump.1.html.

[12] *pbgpp*. URL: https://github.com/DE-CIX/pbgp-parser (cit. a p. 24).

[13] *tshark*. URL: https://www.wireshark.org/docs/man-pages/tshark.html.

[14] *linux container*. URL: https://linuxcontainers.org/.

[15] *systemd*. URL: https://wiki.debian.org/it/systemd.

[16] *cron manual page*. URL: https://www.linux.org/docs/man8/cron.html.

[17] *voilà*. URL: https://voila.readthedocs.io/en/stable/index.html#.

[18] *jupyterhub*. URL: `https://jupyter.org/`.