1. Introduzione

2. Contesto

2.1 Software Defined Networking

L'architettura tradizionale di rete si basa su dispositivi fisici di interconnessione situati al livello rete dello stack TCP/IP, switch e router, che collegano più host a livello locale e permettono di scambiarsi informazioni. Questi dispositivi implementano al loro interno diverse funzioni. Le funzioni del piano dati (data plane) si occupano della ricezione, del processamento e dell'inoltro dei pacchetti, ossia ciò che serve per l'instradamento. La decisione dell'inoltro del dato viene presa in base alla tabella di routing, nel caso dei router, o alla MAC address table nel caso degli switch.
Le tabelle vengono inizializzate e modificate grazie al piano di controllo (control plane) che si occupa di decidere i percorsi per l'instradamento sulla base della destinazione e comunicarli al data plane aggiornando le tabelle di inoltro dei dispositivi. Nei protocolli di rete tradizionali questi due piani all'interno dei dispositivi sono separati tra loro e svolgono i loro compiti indipendentemente. Per decidere i percorsi il control plane può scegliere tra due tipi di algoritmi di instradamento con caratteristiche differenti:

- decentralizzati: nessun nodo conosce la topologia di tutta la rete ma ha informazioni solo dai nodi vicini.
- globali: si basano sulla conoscenza della topologia di tutta la rete. Questa soluzione porta allo scambio di molti messaggi di broadcast e non è efficiente in quanto ogni nodo si occupa di calcolare soltanto la propria tabella.

Entrambi gli approcci necessitano di un algoritmo di routing in esecuzione su ciascun router che, attraverso un apposito protocollo, scambia messaggi con gli altri componenti della rete per prendere decisioni. Ciò introduce ritardi che non sono necessari non rendendo così la rete adatta alle nuove esigenze delle applicazioni moderne che necessitano di alta dinamicità.
Il Software Defined Networking (SDN) è un'architettura di rete proposta negli ultimi anni

dalla Open Networking Foundation (ONF) [1] per rimediare ai problemi di scalabilità e affidabilità dei controller decentralizzati. SDN rappresenta un nuovo paradigma dinamico, gestibile e facilmente adattabile grazie alla separazione del piano di controllo dal piano dati, che risulta direttamente programmabile. Di conseguenza è possibile un disaccoppiamento tra hardware e software per la gestione di device con API diverse.

Per poter funzionare, i dispositivi devono essere in grado di comunicare con il controller centrale e riconoscere cambiamenti significativi degni di notifica per una gestione della rete adattabile ai cambiamenti in tempo reale. Questo è possibile tramite l'installazione al loro interno di componenti software con le caratteristiche necessarie detti Control Agents. La base di questo paradigma è quindi un controller remoto che, interagendo con i Control Agents locali, riceve informazioni sui collegamenti e sul traffico in tempo reale ed è in grado di configurare autonomamente i dispositivi collegati sulla base degli eventi notificati. Lo scopo principale è quindi ridurre e semplificare il carico di amministrazione per i singoli dispositivi.

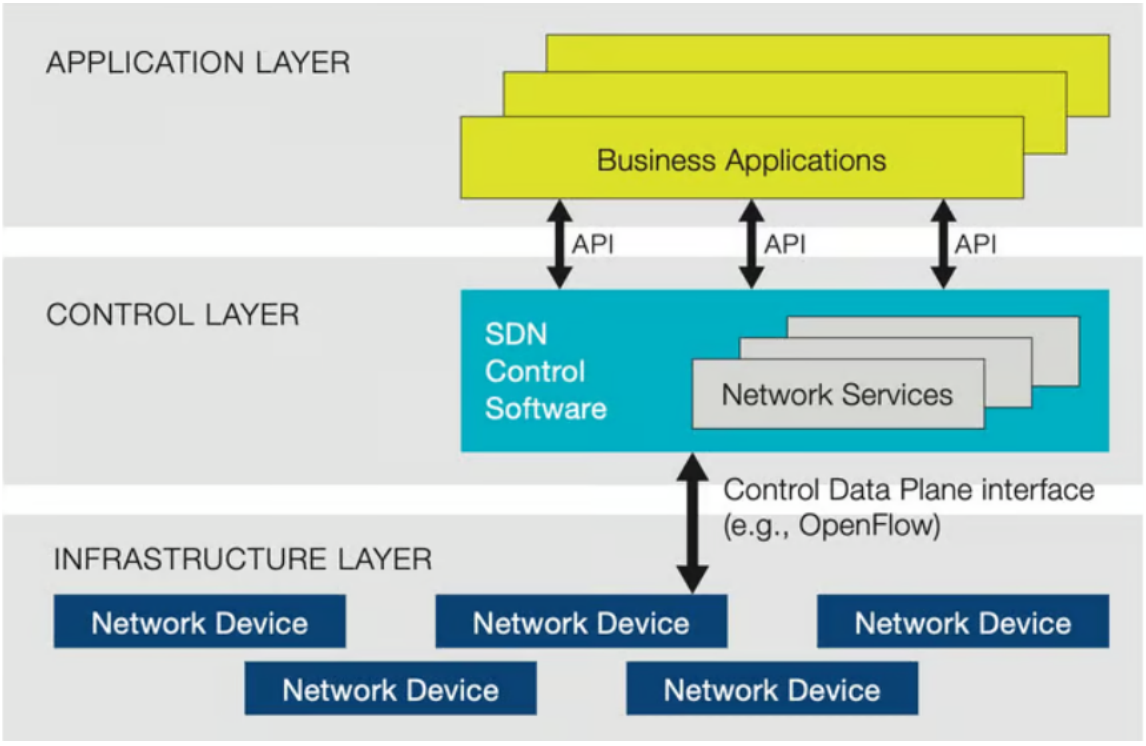

Figura 2.1: struttura di una rete SDN

Come si può notare dalla figura 2.1 la rete viene suddivisa in tre livelli: Infrastructure layer, Control layer e Application layer. Partendo dal livello più basso troviamo l'infrastruttura di rete il cui unico compito è implementare il piano dati, ossia la parte che supporta un protocollo condiviso per comunicare con il controller e gestire i paccchetti sulla base delle configurazioni imposte da quest'ultimo. Questa divisione consente di evitare algoritmi di routing per il forwarding dei pacchetti all'interno dei dispositivi di rete visto che sarà gestito direttamente dai livelli sovrastanti.

Nel control layer si trova il controller SDN che, tramite API northbound (NBI) e southbound (SBI), permette di comunicare con gli altri due livelli. Le API NBI consentono al controller di interfacciarsi con le applicazioni e i servizi situati nel livello superiore, mentre le API SBI, tipicamente implementate tramite OpenFLow, permettono al controller di comunicare con i dispositivi di rete nel livello inferiore.

Questo livello consente il monitoraggio e l'implementazione del piano dati. E' in grado di configurare e di gestire rapidamente le risorse di rete tramite programmi dinamici e automatizzati. Consente infatti di imporre regole di inoltro ai dispositivi sottostanti tramite la manipolazione delle tabelle di routing mediante le API SBI dopo aver calcolato, o aggiornato sulla base di eventi, il percorso migliore. Il controller inoltre è logicamente centralizzato, anche se fisicamente può essere distribuito su più dispositivi. In questo modo il piano di gestione (management plane) situato sopra di esso interagisce con un unico punto di accesso.

L'application layer comprende le applicazioni e i servizi che sfruttano le capacità della rete SDN per la realizzazione del piano di gestione. Grazie a questo livello si possono definire politiche o intenti da implementare all'interno della rete. Queste regole sono comunicate al controller tramite le API NBI e quest'ultimo si occuperà di farle rispettare mediante il costante monitoraggio delle risorse del data plane.

Questo disaccoppiamento dei vari livelli consente alla rete di diventare direttamente programmabile da un'unica unità centralizzata riuscendo a mantenere una visione globale e permettendo l'astrazione dell'infrastruttura sottostante per affrontare le sfide di gestione incontrate nelle reti moderne.

2.2 Intent-based Networking

2.3 Controller allo stato dell'arte

2.3.1 ONOS

Open Newtwork Operating System (ONOS) [2] è uno dei controller SDN più noti. E' un progetto nato dalla Open Networking Foundation (ONF) [1] al fine di soddisfare le esigenze degli operatori per poter costruire reali soluzioni SDN/NFV. I principali obiettivi sono quelli di introdurre modularità del codice, configurabilità, separazione di interessi e agnosticismo dei protocolli.
Per adattarsi alle esigenze degli utenti è necessario poter sviluppare una piattaforma applicativa modulare ed estendibile. Per questo motivo la base dell'architettura di ONOS è costituita da una piattaforma di applicazioni distributite collocata sopra OSGi [3] e Apache Karaf [4]. Queste applicazioni offrono delle funzionalità di base e sostegno al livello superiore il quale fornisce una serie di controlli di rete e astrazioni di configurazione necessarie per il corretto funzionamento del controller.
Per estendere le funzionalità a seconda delle esigenze sono invece necessarie delle applicazioni ONOS aggiuntive che si comportano come una estenzione di quelle già presenti. Ognuna di esse è gestita da un singolo sottosistema che all'interno del controller è rappresentato da un modulo. I moduli attualmente installabili che si possono incorporare a quelli inizialmente offerti dal sistema sono più di 100. Tutti i servizi principali sono scritti in Java come bundles all'interno del Karaf OSGi container così da permettere l'installazione e l'esecuzione dinamicamente.
ONOS supporta diverse API northbound tra cui:

- GUI: offre un'interfaccia grafica per interagire con l'utente
- REST API: facilita l'integrazione con sistemi di orchestrazione e altri controller
- gRPC: per un'interazione ad alte prestazioni tra applicazioni e altre entità o protocolli della piattaforma

Per quanto riguarda le API southbound supportante fornisce diversi adattatori che rendono il sistema indipendente dai vari protocolli.

ONOS è sviluppato come un sistema simmetrico distribuito in cui ogni entità, dal punto di vista software, è indentica alle altre. In caso di guasto di una componente le altre sono in grado di sostenere mantenere la continuità del servizio, assicurando la disponibilità del sistema. Inoltre, per far fronte ai cambiamenti del carico di lavoro o dell'ambiente, ONOS è dinamicamente scalabile, consentendo una replica virtualmente illimitata della capacità del piano di controllo.

Pur essendo fisicamente disaggregato offre una visione logicamente centralizzata al fine di fornire l'accesso di ogni informazione alle applicazioni in maniera uniforme.

ONOS offre numerosi vantaggi agli operatori di rete, tra cui la capacità di implementare soluzioni altamente modulari e configurabili. Grazie alla sua architettura distribuita e al supporto per diverse API consente di gestire reti in modo efficiente e scalabile. Inoltre è facilmente integrabile con sistemi di orchestrazione esistenti, migliorando la flessibilità e la reattività.

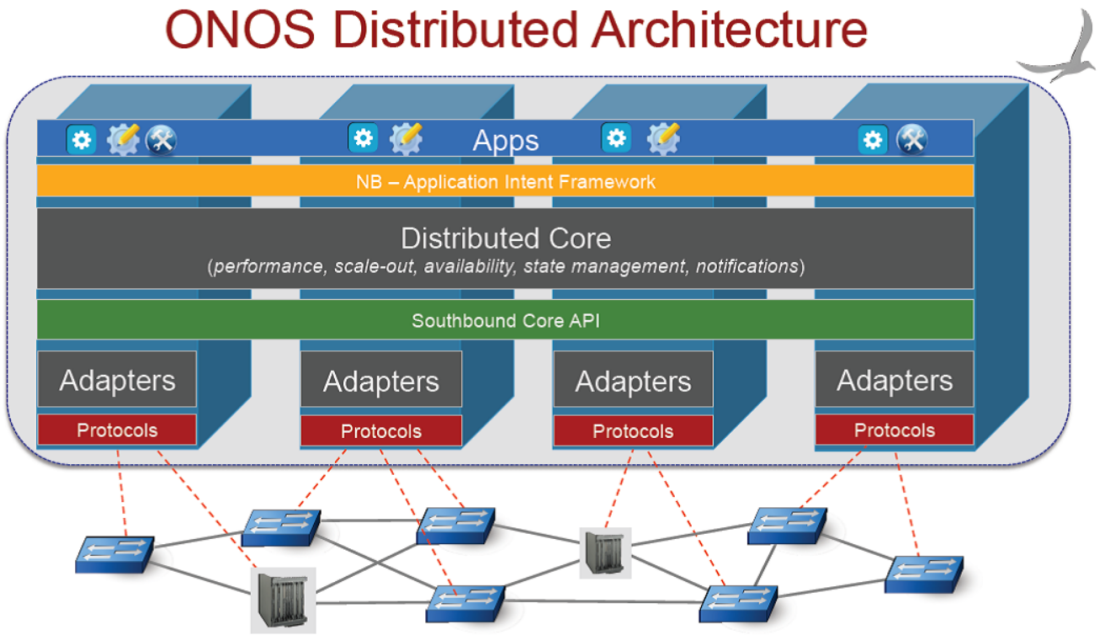

Figura 2.2: Architettura di ONOS

2.3.2 ODL

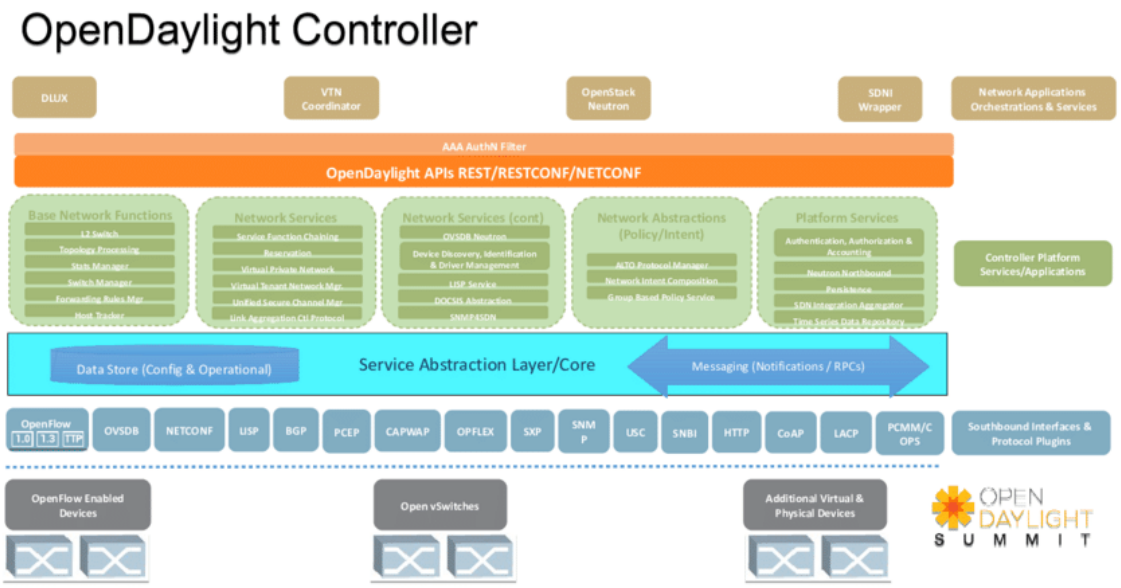

Figura 2.3: Architettura del controller OpenDayLight

OpenDaylight [5] è un progetto open source che utilizza protocolli aperti al fine di fornire controlli centralizzati e gestire il monitoring della rete. Fa parte della fondazione LF Networking [6] che si occupa di stanziare e coordinare il supporto a progetti open source coinvolti nello spostamento o nella comunicazione di dati su una rete. Ciò semplifica il coinvolgimento dei membri e aumenta la collaborazione tra i progetti e gli organismi di standardizzazione.

ODL è un framework scritto in Java che consente di soddisfare esigenze specifiche dell'utente al fine di fornire alta flessibilità. Agisce come un software che può essere eseguito su un qualsiasi sistema operativo che supporti java, come una JVM.

L'architettura di OpenDayLight, come mostrato in figura 2.3, è su più livelli. Il livello principale è costituito dal Controller Platform in quanto al suo interno risiede il controller stesso il quale si occupa di gestire il flusso di traffico andando a modificare le tabelle di inoltro dei dispositivi fisici o virtuali.

Il Service Abstraction Layer (SAL) è il livello inferiore che si occupa di offrire suppor-

to ai vari protocolli SBI come OpenFlow o NETCONF. All'interno di questo livello il collegamento dei moduli tra il controller e i dispositivi avviene dinamicamente al fine di soddisfare il servizio richiesto indipendentemente dal protocollo utilizzato.
Una caratteristica molto impostante è l'architettura costituita da microservizi che un utente può decidere se abilitare o meno. Di default sono tutti disabilitati così da permettere una totale personalizzazione. Questi microservizi sono rappresentati da moduli, contenuti all'interno del controller, collegabili tra loro, che si occupano dell'esecuzione delle varie attività di rete. I moduli sono collegati al SAL dinamicamente. Per la gestione dei moduli a runtime e l'installazione di funzionalità Karaf da implementare nel software di ODL viene utilizzato Apache Karaf [4]. Attraverso il framework MD-SAL (Model-Driven Service Abstraction Layer) gli sviluppatori sono in grado di creare nuove features sotto forma di servizi e protocolli collegati tra loro. Il controller espone delle API NBI di supporto alle applicazioni. Alcune delle API supportate sono il framework OSGi [3], per le applicazioni in eseguzione all'interno del controller, e REST per comunicare con le applicazioni esterne al controller.
Per far fronte ai problemi di scalabilità, disponibilità e persistenza dei dati si possono avere più instanze di ODL distribuite su macchine differenti che cooperano tra loro attraverso il meccanismo dei cluster.

ODL per gestire gli intenti aveva messo a disposizione una NorthBound Interface che successivamente è stata abbandonata nelle release successive a Boron.
Network Intent Composition (NIC) è l'interfaccia che permette all'utente di esprimere uno stato desiderato in una forma neutrale rispetto all'implementazione, detto intento. Quest'ultimo verrà applicato tramite la modifica delle risorse disponibili grazie alla gestione dei servizi da parte del controller sulla base delle specifiche. Gli intenti sono descritti al controller tramite l'interfaccia NBI che mette a disposizione la semantica necessaria per la generalizzazione e l'astrazione delle policy, invece di specificare i comandi di configurazione dei dispositivi come il resto delle interfacce NBI. E' responsabilità dell'implementazione della NIC trasformare l'intento nelle regole di configurazione delle risorse. Questa feature permette di avere a dismosizione un modo descrittivo per richiedere il compor-

tamento desiderato della rete. NIC è stato progettato per essere un'interfaccia indipendente dal controller in modo che gli intenti siano trasferibili tra varie implementazioni in quanto una specifica di intento non dovrebbe contenere specifiche di implementazione e tecnologia.

3. Teraflow

Il controller SDN su cui ci focalizzeremo è TeraFlow [7].

TeraFlow è stato finanziato dall'unione europea per il programma di ricerca e innovazione Horizon 2020 [8]. Nonostante la quantità di controller SDN i fondatori di TeraFlow hanno riscontrato come problema comune il basso numero di contribuzioni che negli ultimi anni o mesi si stanno sempre di più diradando. Ciò ha come conseguenza che i nuovi bisogni e requisiti delle reti moderne che si stanno sviluppando in questo momento non saranno supportati.

L'obiettivo di TeraFlow è implementare un controller che soddisfi i requisiti attuali ed eventualmente futuri, sia architetturali che infrastrutturali, per le reti. Per raggiungere questo scopo è necessaria una contribuzione attiva da parte degli utenti, per questo TeraFlow è un progetto OpenSource al quale tutti i membri della comunità ETSI (European Telecommunications Standard Institute) [9] possono contribuire. Un altro proposito è di riuscire a diminuire il gap tra ciò che le industrie richiedono e ciò che si può ricavare dagli standard SDN. Questo controller, che è ancora in fase di sviluppo, sarà in grado di integrarsi con gli attuali framework NFV e MEC e fornire l'integrazione delle apparecchiature di rete ottica e a microonde.

TeraFlow segue un'architettura nativa cloud, è basato su container sviluppati sopra un ambiente basato su Kubernetes presso la sede del CTTC a Barcellona che garantiscono scalabilità, dinamicità e integrità. Ogni container, o componente, ha delle responsabilità e definisce un microservizio che interagisce con gli altri attraverso la connessione di rete rendendo il controller disaggregato. Le componenti principali sono implementate in Java (solo quelle di Automation e Policy) e Python. I servizi sono semplici e dettagliati rendendo possibile l'uso di protocolli leggeri. Dal punto di vista della sicurezza utilizza un sistema di Machine Learning per la prevenzione e la mitigazione di attacchi.

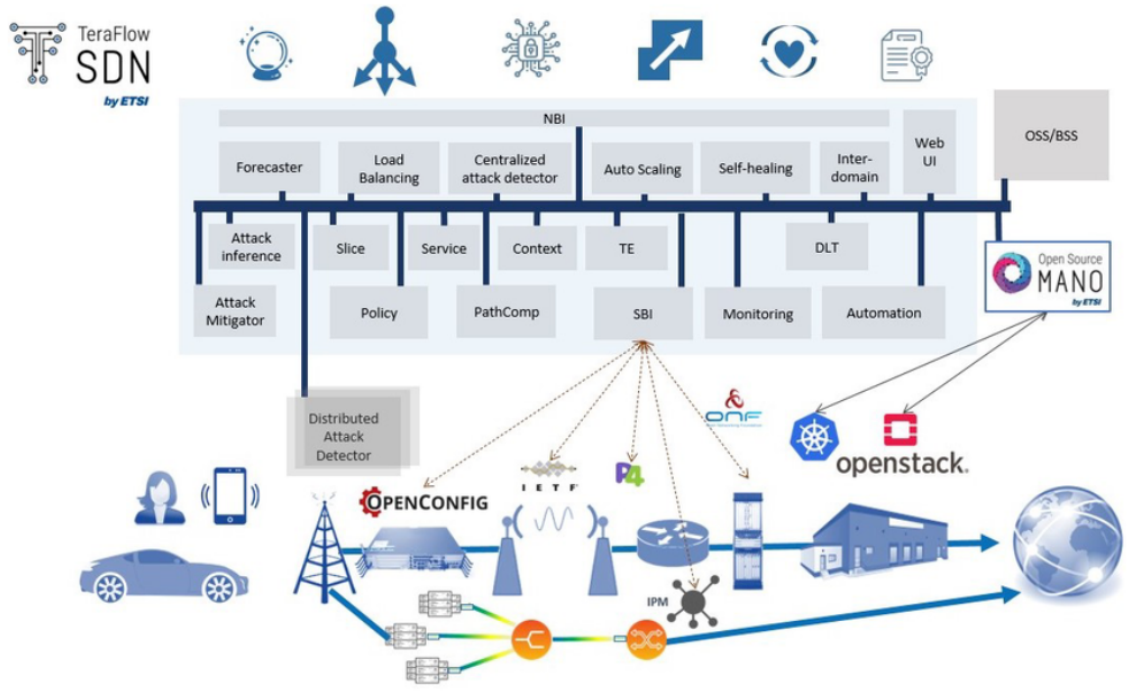

Figura 3.1: Architettura di TeraFlow

3.1 Componenti

TeraFlow utilizza un bus gRPC(Google Remote Procedure Call) per comunicare tra le componenti. E' stato scelto come protocollo interno di TeraFlow, al posto di REST, per i modi più concreti di descrivere l'interazione tra due entità. La descrizione di messaggi condivisi è specificata usando dei Protocol Buffers nei quali è descritto lo schema dei messaggi che le componenti si scambiano, quindi i vari servizi che implementano con le adeguate funzioni da usare. I Protocol Buffer sono un meccanismo indipendente sia dalla piattaforma che dal linguaggio per la serializzazione di strutture dati. Ogni componente ne possiede uno e sono descritti nella pagina del sito [10].

3.2 Device Level Abstraction

Il Device Level Abstraction fornisce la capacità di interagire con i dispositivi presenti all'interno della rete e corrisponde all'Infrastructure layer descritto nell'architettura SDN 2.1. Una componente necessaria per questo livello di astrazione è la componente SBI. Per far comunicare più tipi di device possibili con il controller la componente presenta eterogeneità offrendo supporto a differenti protocolli. Il suo compito principale è effettuare l'handshake con i device in modo da introdurli nell'ecosistema del controller e essere in grado di configurarli a runtime. Dopo aver effettuato la connessione con un determinato device e essersi assicurata la disponibilità, la componente presenta una API che permette di mandare in input le configurazioni scelte attraverso un file JSON al controller TeraFlow. Il controller provvederà a inserire le giuste entry nella tabella del relativo switch o router. Un'altra componente che fa parte di questo livello è quella di monitoring il cui compito è offrire supporto al Management-level. E' essenziale per l'automazione dei servizi e per prendere le decisioni a run time sulla base di eventi. La componente interagisce con i device per catturare lo stato della rete sttraverso delle kpi e persiste le informazioni all'interno di database necessari per esporle e renderle utilizzabili ad altre componenti. Quando si tratta di topologie più ampie, dovendo gestire molti device, serve però un livello di astrazione superiore per specificare la connessione tra vari end-points, a tal fine si introduce il Service-level.

3.3 Service Level Abstraction

Il compito principale del Service Level sono i servizi. Questo nuovo livello di astrazione permette a un utente di definire intenti per la connessione tra gli end-points grazie alla componente di Service. Un utente può selezionare un path scelto o chiedere al controller di determinarne uno che rispetti determinati requisiti. La componente che si occupa di calcolare una serie di path tra gli end-points è la PathComp. Per astrarre la complessità del livello sottostante all'utente la componente di service è in grado di tradurre l'intento in un insieme di regole che propaga all'SBI (sempre attraverso file JSON) cosicchè venga

configurato ogni device all'interno del percorso per stabilire la connessione.

3.4 Management Level Abstraction

Questo livello di astrazione è stato introdotto per poter interagire con la componente di service dinamicamente per creare, aggiorare o eliminare un servizio sulla base degli eventi che derivano dallo stato della rete. Una delle componenti che ne fanno parte è la componente di Policy. Si occupa di creare un service level agreement (SLA) per uno specifico servizio identificato tramite un id. Specifica delle condizioni che devono essere rispettate chiedendo alla componente di Monitoring di notificare con un allarme quando non saranno più soddisfatte. A questo punto verrà creato un evento e si eseguiranno delle azioni prestabilite dal controller fino ai dispositivi per gestire i problemi o cambiare il servizio. Solitamente le azioni contemplano l'interazione con altre componenti al livello inferiore. Un'esempio può essere una notifica alla componente di Service che si occuperà di modificare il percorso identificato precedentemente per far rispettare nuovamente i requisiti.

3.5 gRPC

GRPC è un protocollo basato sul protocollo di trasporto HTTP/2 che utilizza la codifica orientata ai byte introducendo così bassa latenza. E' progettato per trasportare messaggi peer-to-peer in modo distribuito e non durevole consentendo a più servizi lo scambio di informazioni attraverso un bus condiviso.

4. Studio e sperimentazione della gestione di policy in Teraflow

5. Conclusioni

6. Appendice

6.1 Installazione

Il controller viene eseguito all'interno di una Java Virtual Machine (JVM), quindi è necessario verificare quali versioni di Java supporta la distribuzione che si decide di installare. La versione stabile più recente di ODL è compatibile con le versioni di Java superiori alla 17, mentre per le versioni più datate quest'ultime non vanno bene. Per l'installazione si deve scaricare la distribuzione desiderata che si trova sul loro sito ufficiale nella pagina di download del software [11]. Successivamente è necessario fare l'unzip del file, navigare nella cartella e eseguire il seguente comando da terminale per avviare il controller.

```
./bin/karaf
```

La versione di ODL dell'immagine 6.1 è Potassium.

Figura 6.1: Installazione ODL

Successivamente, si può trovare una lista completa delle feature disponibili eseguendo il seguente comando.

```
feature:list
```

Per installarle invece

```
feature:install <feature1> <feature2>..
```

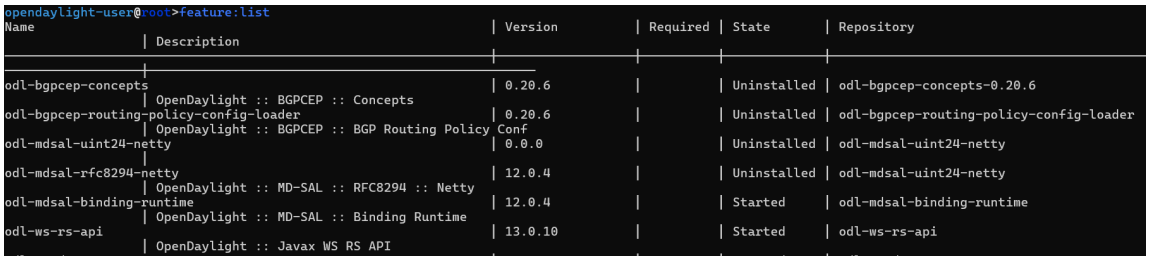

Figura 6.2: Alcune features disponibili

Bibliografia

[1] Open networking foundation. https://opennetworking.org/.

[2] Onos. https://opennetworking.org/onos/.

[3] Osgi. https://www.osgi.org/.

[4] Apache karaf. https://kafka.apache.org/.

[5] Opendaylight. https://www.opendaylight.org/.

[6] Lfnetworking. https://lfnetworking.org/.

[7] Teraflow. https://www.teraflow-h2020.eu/%.

[8] Progetto horizon 2020. https://research-and-innovation.ec.europa.eu/funding/funding-opportunities/funding-programmes-and-open-calls/horizon-2020_en.

[9] Etsi. https://www.etsi.org/.

[10] Protocol buffer delle componenti in teraflow. https://gitlab.com/teraflow-h2020/controller/-/tree/develop/proto.

[11] Installazione opendaylight. https://docs.opendaylight.org/en/latest/downloads.html.