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.

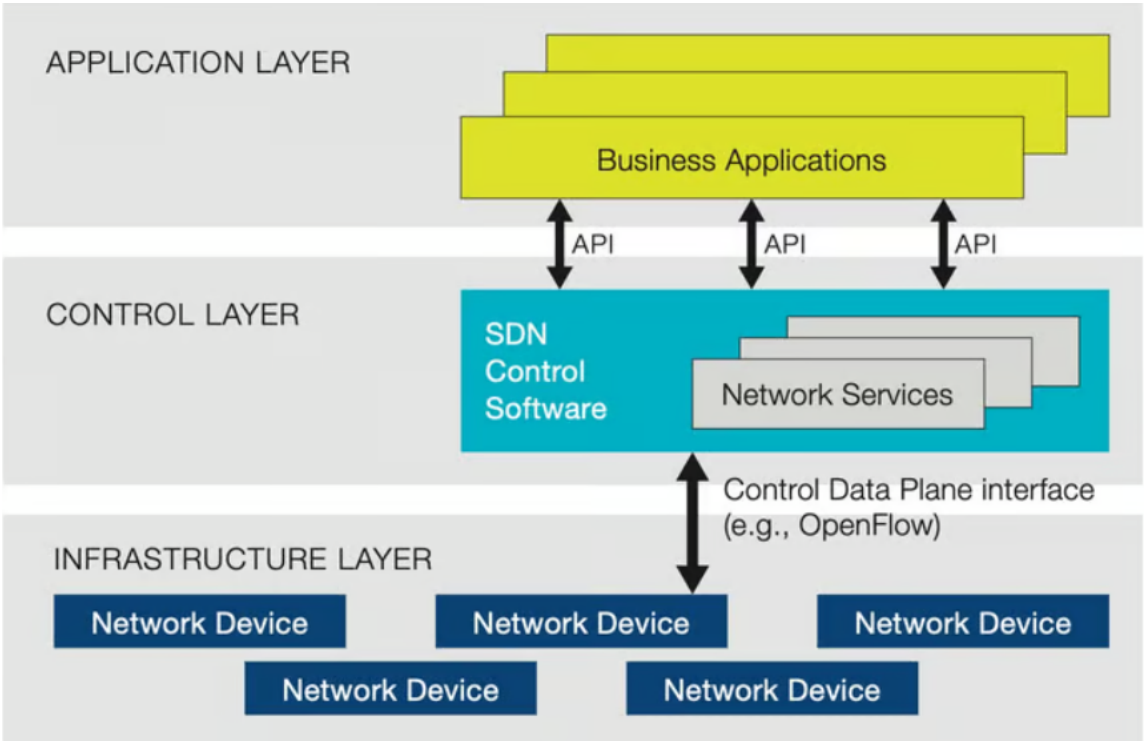

Figura 2.1: struttura di una rete SDN

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

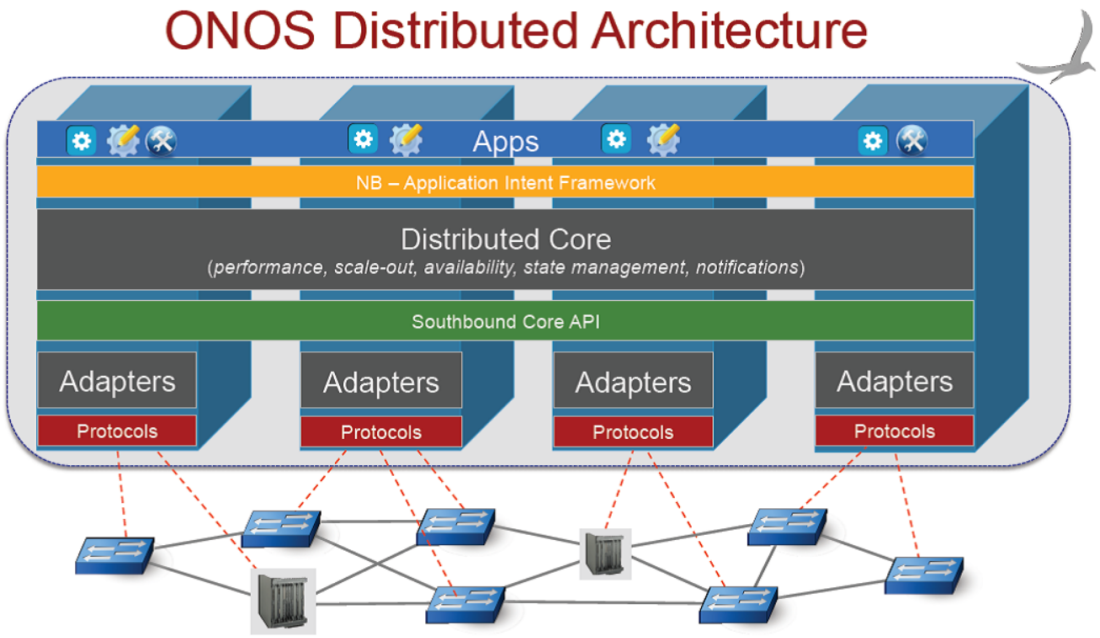

Figura 2.2: Architettura di ONOS

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 ne-

cessarie 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.

2.3.2 ODL

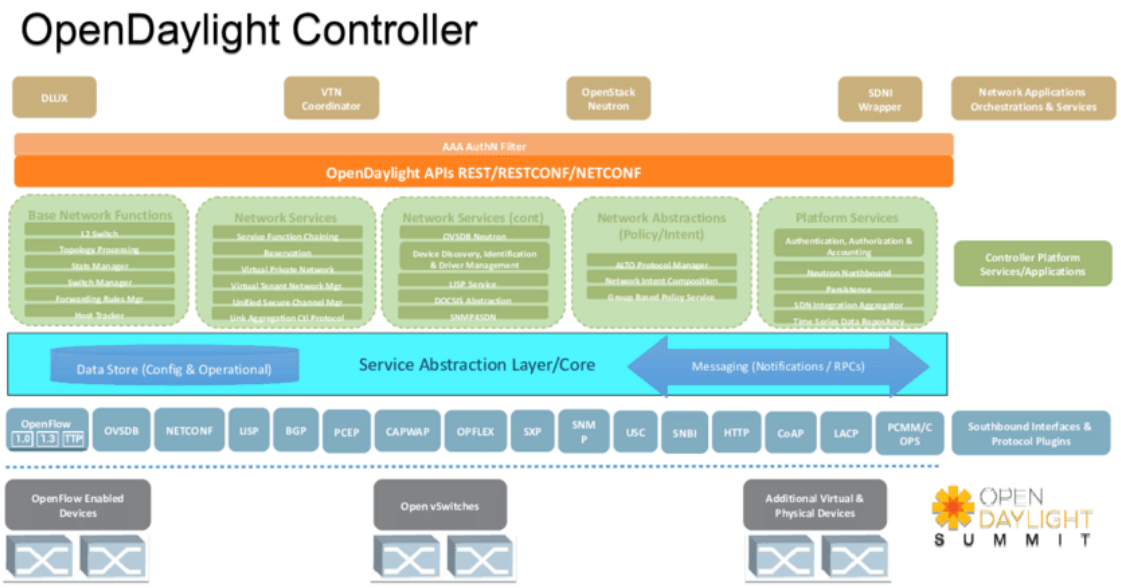

Figura 2.3: Architettura del controller OpenDayLight

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) [7] 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

comportamento 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

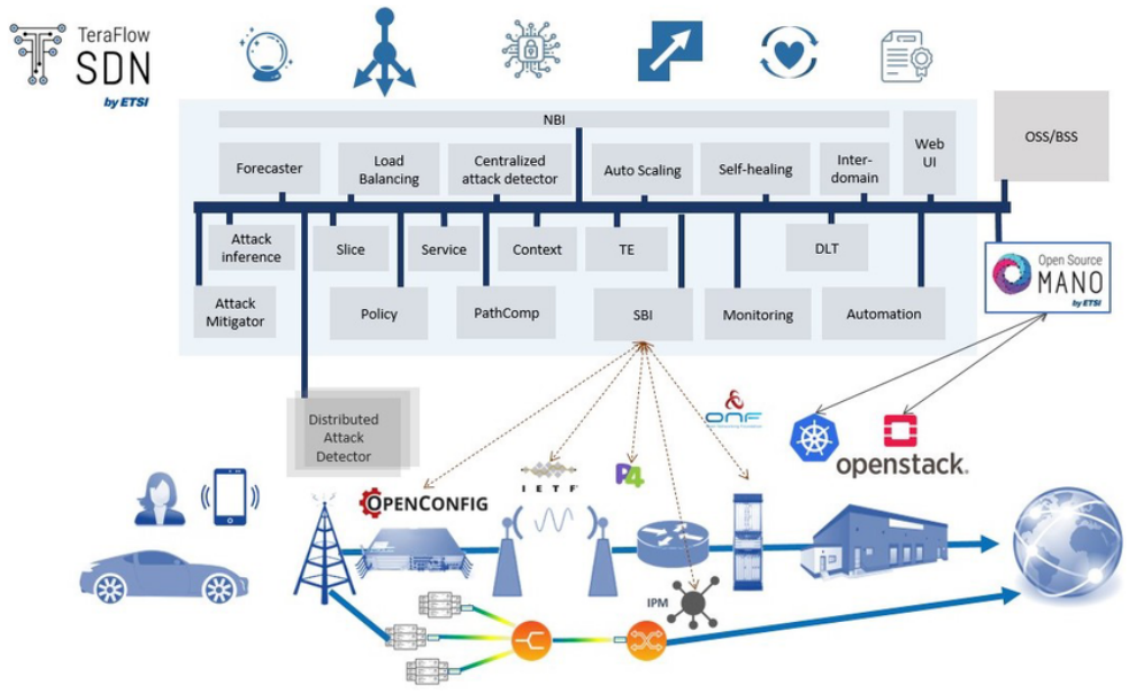

Figura 3.1: Architettura di TeraFlow

Il controller SDN su cui ci focalizzeremo è TeraFlow [8].

TeraFlow è stato finanziato dall'unione europea per il programma di ricerca e innovazione Horizon 2020 [9]. Nonostante la quantità di controller SDN i fondatori di TeraFlow hanno riscontrato come problema comune il declino delle contribuzioni negli ultimi anni. Di conseguenza, i nuovi bisogni e requisiti delle reti moderne che si stanno sviluppando in questo momento non saranno più supportati.

L'obiettivo di TeraFlow è implementare un controller che soddisfi i requisiti attuali ed eventualmente futuri, sia architetturali che infrastrutturali, per le reti. Il controller mira a migliorare le capacità di elaborazione dei flussi consentendo la gestione di oltre un Tera di servizi di connettività, questo rappresenta una capacità cruciale per supportare le esigenze delle reti B5G. Per raggiungere questo scopo è necessaria una contribuzione attiva da parte

degli utenti. A tal fine TeraFlow è un progetto OpenSource al quale tutti i membri della comunità ETSI (European Telecommunications Standard Institute) [10] possono contribuire.

Un ulteriore 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. E' inoltre in grado di interfacciarsi con altri controller come ONOS, ma anche altre istanze di TeraFlow che gestiscono diversi domini di rete, per poter sfruttare alcune funzionalità.

Segue un'architettura nativa cloud, basata su container (tecniche di virtualizzazione leggera) sviluppati sopra un ambiente basato su Kubernetes presso la sede del CTTC a Barcellona. Kubernetes presenta diverse funzionalità che garantiscono al controller dinamicità, autoriparazione, integrità e bilanciamento del carico. Ogni container o componente ha delle responsabilità specifiche 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. Per ottimizzare l'accesso concorrente e gestire la configurazione di rete, TeraFlow utilizza una componente chiamata Context. Questa componente memorizza la configurazione di rete, inclusi topologie, dispositivi, collegamenti e servizi, in un database No-SQL. La componente ottimizza l'accesso concorrente e garantisce la coerenza dei dati gestiti dai vari componenti del controller SDN. La componente di Monitoring invece gestisce le diverse metriche configurate per le apparecchiature e i servizi di rete, memorizzando i dati di monitoraggio relativi alle Key Performance Indicators (KPI) selezionate. 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.

Altri elementi fondamentali del controller sono rappresentati da:

- **usabilità**: realizzata grazie l'interfaccia utente webui che consente la configurazione di servizi predefiniti e la possibilità di visualizzare le metriche in modo personalizzabile in base alle esigenze.

- **scalabilità**: è intrinseca nel design del controller con la replicazione automatica di miscroservizi per gestire elevati volumi di richieste in ingresso.
- **affidabilità**: garantita attraverso robusti meccanismi di monitoraggio che supervisionano lo stato dei miscroservizi e dei flussi attivando automaticamente dei processi di ripristino se necessari.

TeraFlow si propone di affrontare le sfide delle reti moderne attraverso un'architettura innovativa e la collaborazione attiva della comunità. Il suo focus è sviluppare un controller SDN Carrier Grade (reti o infrastrutture ben testate con livelli estremamente elevati di affidabilità, ridondanza e sicurezza) per le reti B5G.

3.1 Componenti

TeraFlow utilizza un bus gRPC (Google Remote Procedure Call) come protocollo interno per la comunicazione tra le sue componenti. Questo protocollo è stato preferito a REST per la sua capacità più concreta di descrivere le interazioni tra entità in modo efficiente e preciso. gRPC sfrutta i Protocol Buffers per definire in modo preciso gli schemi dei messaggi condivisi e scambiati tra le componenti, consentendo ai vari servizi di implementare le adeguate funzioni di comunicazione. I Protocol Buffers sono un meccanismo indipendente dalla piattaforma e dal linguaggio per la serializzazione delle strutture dati. Ogni componente di TeraFlow implementa i propri Protocol Buffers, dettagliati nella pagina dedicata del sito [11].

3.1.1 Device Level Abstraction

Il Device Level Abstraction permette l'interazione con i dispositivi presenti all'interno della rete e corrisponde all'Infrastructure layer descritto nell'architettura SDN 2.1. Una componente fondamentale 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 stabilire una connessione con i dispositivi per integrarli nell'ecosistema del controller e essere in grado

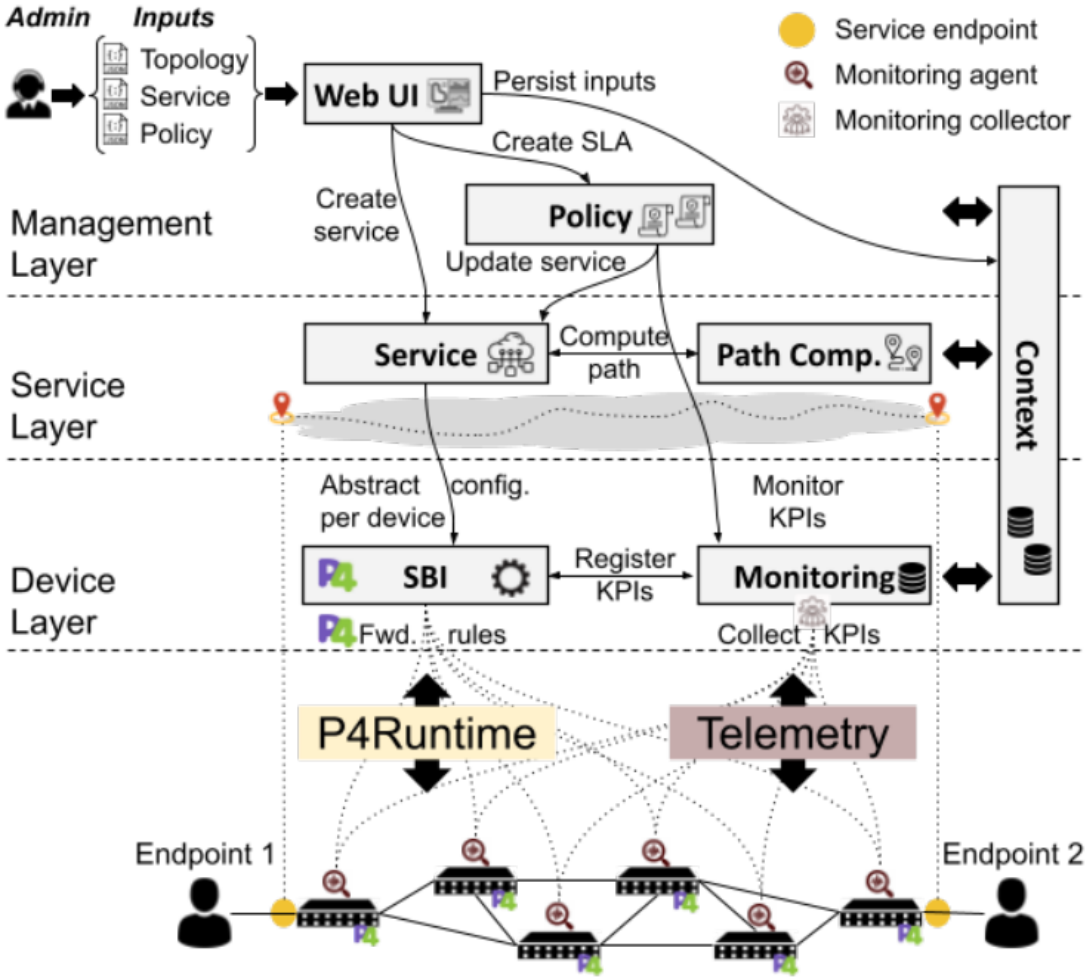

Figura 3.2: Interazione tra alcune componenti in TeraFlow

di configurarli dinamicamente a tempo di esecuzione. Dopo aver stabilito la connessione con un determinato dispositivo e aver verificato la disponibilità, la componente offre una API che permette di mandare in input le configurazioni scelte dal controller TeraFlow tramite un file JSON. Il controller provvederà a inserire le giuste entry nelle tabelle degli switch o router pertinenti.

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 in tempo reale sulla base di eventi. Questa componente interagisce con i dispositivi per catturare lo stato della rete attraverso delle KPI (Key Performance

Indicators) persistendo le informazioni all'interno di database. Quando un valore KPI registrato supera determinate soglie predefinite, la componente di Monitoring utilizza il canale API degli eventi del controller per notificare i componenti interessati. Ogni evento è composto da una KPI che identifica la regola a cui si riferisce, un timestamp e un KPIValue, che rappresenta il valore della metrica monitorata in tempo reale richiesta dalla KPI al momento specificato dal timestamp. Per il corretto funzionamento la componente di Monitoring deve essere in grado di recuperare le metriche da tutti i diversi dispositivi monitorati. Questi implementano spesso protocolli diversi per notificare le KPI, per questo motivo sono stati inclusi una serie di sottomoduli che si connettono agli elementi monitorati utilizzando i protocolli SBI necessari. I dati estratti vengono memorizzati nel Metrics Database così da poter fornire dati dimensionali con serie temporali visualizzabili su Grafana [12]. Tuttavia quanto si tratta di gestire topologie più complesse che coinvolgono molti dispositivi è necessario un livello di astrazione superiore per specificare la connessione tra vari end-points. A tal fine si introduce il Service-level.

3.1.2 Service Level Abstraction

Il Service Level è responsabile della creazione e dell'aggiornamento dei servizi di rete. Questo livello di astrazione permette agli utenti di definire intenti specifici per la connessione tra gli end-points attraverso la componente di Service. La componente di Service supporta diversi tipi di servizi ed è in grado di utilizzare vari protocolli per configurare i dispositivi di rete. Inoltre implementa una Service Handler API che consente agli operatori di rete di definire i comportamenti necessari per ciascun tipo di servizio.

- **L2-VPN**: servizio per dispositivi OpenConfig
- **L3-VPN**: servizio per dispositivi emulati o OpenConfig con supporto per ACLs
- **Connectivity**: servizio per dispositivi TAPI
- **L2 service Handler**: servizio per dispositivi P4
- **Microwave service Handler**

La componente ha al suo interno un blocco gRPC che espone una NBI al resto delle componenti del controller. Gestisce i vari servizi inviando richieste di calcolo del percorso alla PathComp e successivamente, grazie a uno scheduler, si occupa di configurare i dispositivi in base alle connessioni restituite. Questo permette di astrarre la complessità del livello sottostante all'utente in quanto la componente è in grado di tradurre l'intento in un insieme di regole che vengono propagate all'SBI attraverso dei file JSON.
La componente PathComp (path computation) si occupa di gestire la selezione del percorso tra gli end-points per i servizi di connettività di rete. Riceve richieste dalla componente di Service e, interagendo con la componente di Context, recupera le informazioni sulle topologie sottostanti al fine di creare percorsi che soddisfino i requisiti del servizio di rete richiesto. La PathComp rappresenta un'entità singola e specializzata dove possono essere ospitati diversi algoritmi. Questo permette che qualsiasi nuovo algoritmo utilizzato non impatti su altre componenti del controller. Per confrontare i percorsi viene utilizzato inizialmente un algoritmo regolare K-SP dove i k percorsi sono ordinati per numero di passi (hop), ritardo end-to-end e larghezza di banda disponibile sul link più congestionato.

Per automatizzare e rendere reattivo questo processo agli eventi che si verificano nello stato della rete, è essenziale introdurre un ulteriore livello di astrazione.

3.1.3 Management Level Abstraction

Questo livello di astrazione è stato introdotto per consentire l'interazione dinamica con la componente di Service per la creazione, l'aggiornamento o l'eliminazione di un servizio in risposta agli eventi notificati. Una delle componenti fondamentali che ne fanno parte è la componente di Policy. Si occupa di definire condizioni di politica che possono essere aplicate sia a livello di singoli dispositivi che a livello di dominio della rete. Le politiche possono includere più regole collegate tra loro tramite condizioni AND/OR.
Ogni regola è composta da una KPI che la identifica, un operatore numerico di paragone e un KPIValue, che rappresenta, insieme all'operatore numerico, l'insieme dei valori accettabili o non per quella determinata metrica. Quando le condizioni definite dalla politica sono verificate (ad esempio, il KPIValue identificato per la latenza supera il valore massimo indicato), la componente di Monitoring genera degli allarmi. Questi allarmi vengono

inviati al componente di origine per poi eseguire l'azione predefinita specificata nella politica (ad esempio ricalcolo del percorso). Questo meccanismo consente al controller di reagire agli eventi e di ripristinare un set di dispositivi ad uno stato desiderato.
Quando nella policy è specificato l'id del servizio, la componente interroga il database di contesto per recuperare l'insieme di dispositivi attraverso i quali transita il servizio. In caso contrario la componente itera su una lista di dispositivi per identificare quali devono rispettare la politica di input. Una regola di politica può avere vari stati:

- **inserted** (inserita)
- **validated** (convalidata)
- **provisioned** (provvista)
- **actively enforced** (attivamente applicata)
- **failed** (fallita)

L'integrazione della componente di Policy con quella di Monitoring ha reso quindi possibile l'associazione delle condizioni di politica con gli allarmi del sistema di Monitoring.

3.2 gRPC

Google Remote Procedure Calls (gRPC) è un framework OpenSource sviluppato da Google per facilitare la comunicazione tra applicazioni distribuite. Permette di connettere, invocare, operare e fare debug di programmi eterogenei in modo semplice. E' basato sul protocollo di trasporto HTTP/2, che supporta la comunicazione bidirezionale. Consente di definire i servizi, i loro metodi di comunicazione e trasportare messaggi attraverso dei file di descrizione dell'interfaccia detti Protocol Buffer, o più semplicemente file proto. I file proto sono un meccanismo indipendente dal linguaggio e dalla piattaforma per la serializzazione delle strutture dati. Questo sistema di serializzazione è più efficiente rispetto a formati come JSON o XML in termini sia di dimensione dei messaggi ma anche di velocità di serializzazione e deserializzazione.

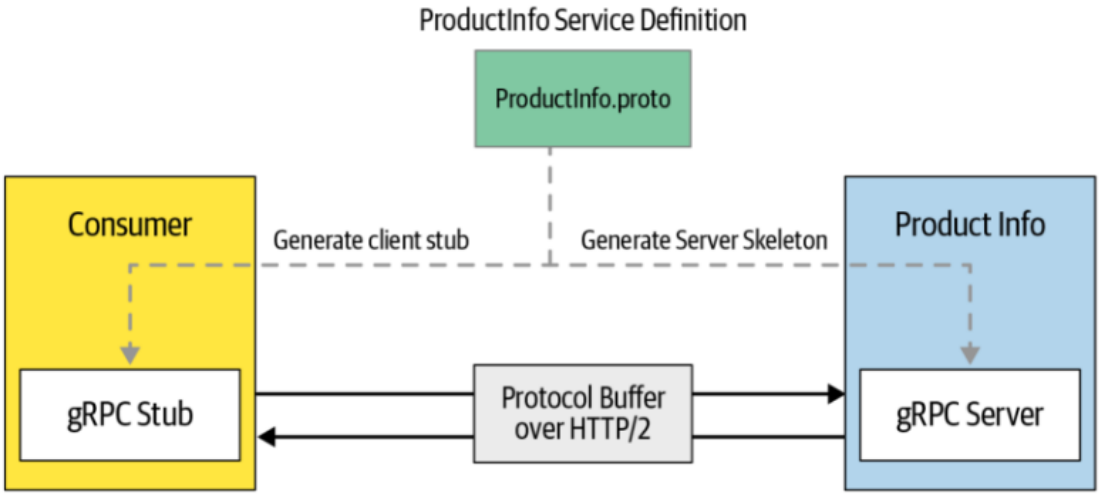

Figura 3.3: Funzionamento del protocollo gRPC

Per sviluppare un'applicazione gRPC è necessario definire un'interfaccia dei servizi. Questa contiene informazioni su come il servizio deve essere usato e quali sono i metodi, i parametri e il formato da utilizzare per i messaggi. Tutti i metodi specificati nella definizione di questa interfaccia possono essere invocati dal client da remoto. Utilizzando il compilatore Protobuf, si può generare il codice lato server (server skeleton) e il codice lato client (client stub) nel linguaggio desiderato.

Quando il gRPC client invoca un servizio gRPC, il lato client utilizza i protocol buffer per serializzare la chiamata di procedura remota nel formato appropriato. Successivamente, la richiesta viene mandata tramite HTTP/2. Sul lato server, viene deserializzata e viene invocata la relativa procedura usando i protocol buffers. La risposta segue il flusso inverso da server a client.

Il framework gRPC sottostante gestisce tutta le complessità che normalmente sono associate all'imposizione di vincoli di servizio, serializzazione dati, comunicazioni di rete, autentificazione e molto altro. gRPC è progettato per trasportare messaggi peer-to-peer in modo distribuito e non durevole, consentendo a più servizi di scambiarsi informazioni attraverso un bus condiviso. Grazie all'uso di HTTP/2 e alla codifica orientata ai byte, gRPC riesce a introdurre bassa latenza, rendendolo adatto a sistemi altamente distribuiti e scalabili. Un'altra caratteristica importante è la sicurezza. Supporta nativamente il proto-

collo TLS (Transport Layer Security), garantendo che le comunicazioni tra i servizi siano cifrate e sicure.

3.3 P4

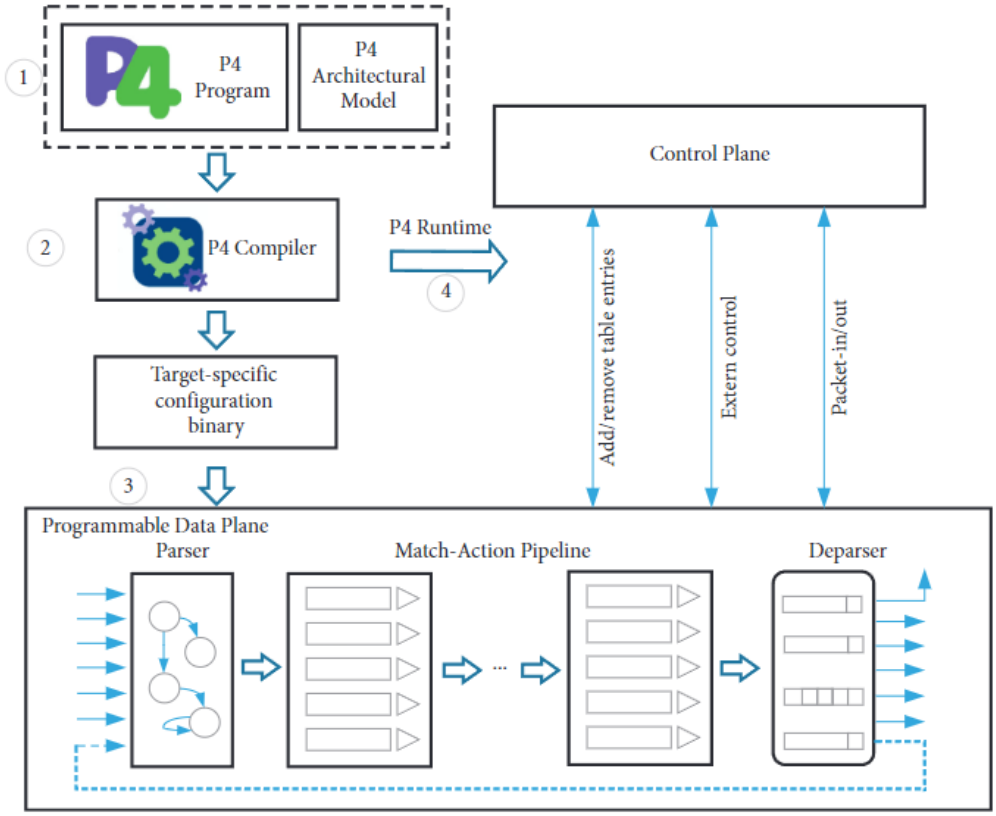

Figura 3.4: Workflow di P4 sul piano dati

P4 [13], che sta per ”Programming Protocol-indipendent Packet Processor”, è un linguaggio di programmazione ad alto livello che consente di descrivere il comportamento di un elemento di rete e del piano dati in modo flessibile. Il linguaggio permette di personalizzare il modo in cui i dispositivi di rete elaborano i pacchetti, senza essere vincolati a un set predefinito di protocolli.

P4 è stato introdotto per l'esigenza di superare le limitazioni con cui si stava interfacciando OpenFlow. Per quest'ultimo protocollo l'hardware e il software non erano più sincronizza-

ti. In alcuni casi gli switch non riuscivano a supportare tutte le funzionalità che OpenFlow poteva offrire a causa delle limitazioni hardware, in quanto progettati per supportare un insieme fisso di protocolli e funzionalità. Per introdurre un cambiamento dell'hardware sono richiesti vari sforzi per lo sviluppo e il finanziamento, inoltre richiederebbe mesi se non anni per essere portato a termine.

P4 è nato quindi con l'obiettivo di definire nuove astrazioni per programmare il piano dati senza dover andare incontro alle limitazioni riscontrate. Offre un linguaggio di programmazione che permette di definire regole per cui i pacchetti vengono processati direttamente nei dispositivi di rete. Di seguito vengono indicati i vari passi che si eseguono durante un flusso di lavoro di un programma P4 che si possono ritrovare anche nell'immagine 3.4.

Un programma P4 è costituito da diverse sezioni, ognuna delle quali descrive un aspetto specifico del trattamento dei pacchetti. La dichiarazione degli header permette di dichiarare gli header dei protocolli che si vogliono analizzare e riconoscere o inventare nuovi headers per scopi di ricerca o esperimenti. Questo include campi come indirizzi IP, numeri di porta e tutti i dati di protocollo. Il Parser specifica come estrarre e interpretare i vari header dai pacchetti. Esso definisce uno stato macchina che determina, attraverso gli header in arrivo, come passare da uno stato all'altro. La Pipeline di Elaborazione comprende tabelle e azioni che definiscono come i pacchetti vengono processati dopo aver effettuato il parsing. I Controlli del flusso coordinano il parser, le tabelle di elaborazione e le azioni. Definiscono quindi i controlli necessari per il flusso corretto di un pacchetto.

Dopo aver definito il programma esso viene mandato al compilatore P4 che genera due tipi di output. Il primo è un file binario P4 ed è ciò che viene installato all'interno del dispositivo target. Questo file binario è specifico per un target e quindi per uno specifico hardware (ad esempio Asics, fpga..). Il secondo output è indipendente dal target ed è diretto verso la parte NorthBound del controller. Questo è chiamato P4 info e genera i metadati necessari per consentire al piano di controllo e al piano dati di comunicare attraverso P4 Runtime.

P4Runtime è un API messa a disposizione da P4 che permette al controller di connettersi ai dispositivi, vedere cosa c'è attualmente nel pipeline e poter mandare le configurazioni rilevanti nella relativa tabella. Permette inoltre di definire il piano dati in modo dinamico collegandolo al piano di controllo. Per il piano di controllo, P4Runtime protegge i dettagli

hardware del piano dati ed è indipendente dalle funzionalità e dal protocollo supportati. P4Runtime riesce quindi a raggiunge l'indipendenza dal target, dal pipeline e dal protocollo.
I nodi programmabili che possono essere ottenuti tramite software o hardware sono definiti P4 target. Essi hanno una pipeline di elaborazione dei pacchetti la cui struttura è specifica per il target ed è descritta da un determinato modello di architettura.

I principali obiettivi quindi includono:

- **Protocollo-indipendenza**: P4 non è vincolato a nessun protocollo specifico consentendo addirittura di definire nuovi protocolli o modificare quelli esistenti
- **Target-indipendenza**: Il codice può essere compilato per una varità di target sia hardware che software
- **Riprogrammabilità**: permette di aggiornare e modificare il comportamento del dataplane in tempo reale rispondendo ai cambiamenti nei requisiti di rete.

P4 rappresenta un passo significativo verso reti più flessibili e programmabili, consentendo agli sviluppatori di adattare e innovare rapidamente in risposta ai cambiamenti nei requisiti di rete. Si propone come una soluzione innovativa e versatile per superare le limitazioni degli attuali protocolli e dispositivi di rete offrendo un linguaggio dinamico e indipendente dall'hardware

3.4 Mininet

Mininet [14] è un sistema open source di orchestrazione per l'emulazione di reti su un unico ambiente Linux, permettendo di simulare l'intera rete su un singolo computer. E' ampiamente utilizzato per la creazione, il test e la sperimentazione di reti virtuali realistiche. Rispetto ad altri emulatori presenti in circolazione, che emulano ogni dispositivo su una macchina virtuale, Mininet offre una serie di vantaggi. Inanzitutto, permette l'avviamento rapido di una rete, la capacità di eseguire test e programmi con tipologie ampie e personalizzate. Inoltre, consente di personalizzare l'inoltro dei pacchetti per testare diverse funzionalità con la possibilità di condividere e replicare il codice. A tal proposito

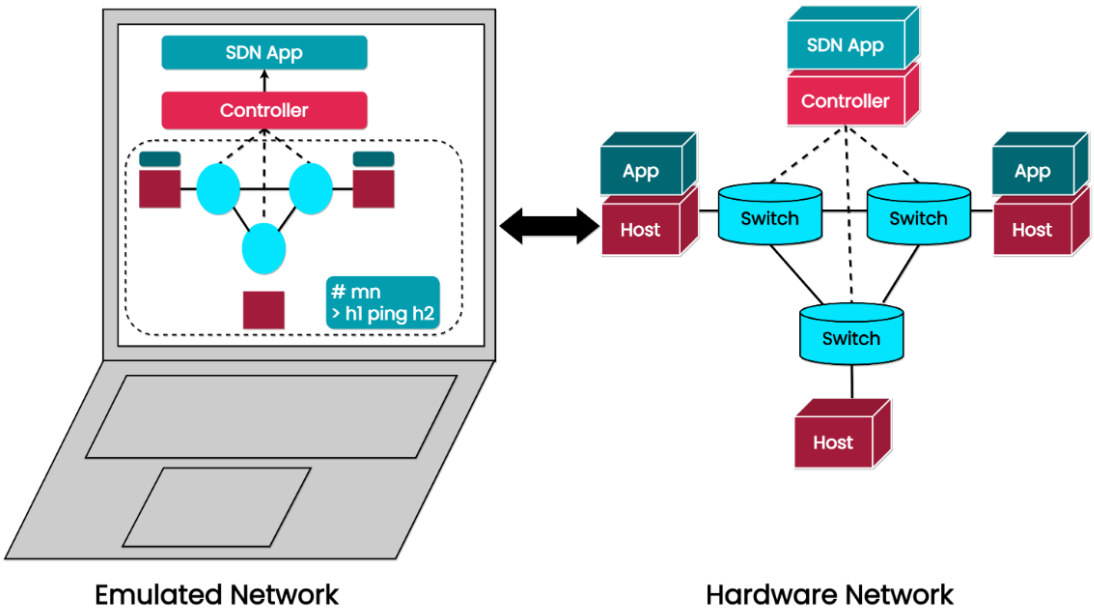

Figura 3.5: Rete mininet

Mininet presenta delle API e un interprete Python che consentono di definire e gestire facilmente delle topologie di rete. Ciò è possibile anche tramite interfaccia a riga di comando (CLI). In entambi i casi possono essere sia predefinite che personalizzate con la possibilità di aggiungere e rimuovere switch, router, host, controller e link, tutti eseguiti su un unico computer. Mininet utilizza infatti dei container di virtualizzazione leggera per creare nodi di rete, ognuno con la propria pila di rete in modo tale che possano comunicare con gli altri nodi nella rete virtuale come farebbero in una rete fisica. Ciò consente di testare nuove applicazioni, protocolli e algoritmi in un ambiente controllato e modificabile prima di implementarli su reti reali.
Mininet mette a disposizione tre livelli differenti di API:

- **Low-level**: consiste nelle classi dei nodi e dei link istanziati individualmente e usati per creare una rete.
- **Mid-level**: aggiunge un containter per nodi e link, l'oggetto Mininet, e fornisce metodi per la configurazione di rete.

- **High-level**: aggiunge l'astrazione della topologia di rete, la classe Topo. Offre la possibilità di creare modelli di topologia riusabili passandoli al comando mn da linea di comando.

Per rendere la rete più realistica e adatta a esperimenti di test si possono configurare i link come up o down e inserire metriche specifiche come possono essere quelle di banda, ritardo, perdita o massima lunghezza della coda di recezione. Gli host su Mininet condividono il filesystem root del server sottostante. Ciò significa che non è necessario trasferire file tra gli host virtuali perché tutti accedono agli stessi file direttamente. Tuttavia, questa condivisione del filesystem può creare problemi se un programma ha bisogno di file di configurazione specifici per ogni host. In tal caso, è necessario creare un file di configurazione separato per ogni host e specificare quale file utilizzare quando si avvia il programma. Un'altra limitazione riguarda la condivisione delle risorse del sistema su cui è in esecuzione che dovranno essere bilanciate tra gli host della rete.
Mininet è stato progettato per essere facilmente integrabile con altri software e sistemi di rete. Consente anche di connettere un controller SDN remoto, quindi esterno alla rete, agli switch, indipendentemente dal PC su cui è installato, in modo da fornire un ambiente adatto allo sviluppo e al test.

3.4.1 Alcuni comandi fondamentali

Linea di comando

Inanzitutto è fondamentale creare una topologia di rete con il seguente comando:
$ sudo mn
Di default viene inizializzata la topologia minimale (–topo=minimal) che consiste in uno switch connesso a due host e un controller OpenFlow. All'interno di Mininet si possono trovare altre topologie disponibili e visualizzabili con il comando
$sudo mn -h
che si possono specificare tramite l'opzione $--topo$.
Per avviare la topologia esistono diverse opzioni da poter applicare.Ad esempio, l'opzione $--controller$ seguito dall'indirizzo IP specifica il controller al quale gli switch dovranno

collegarsi al posto del predefinito offerto da Mininet.
Una volta creata la topologia per avere informazioni su di essa esistono diversi comandi:

- *nodes*: per visualizzare i nodi presenti.
- *net*: per visualizzare i nodi e i link presenti.
- *dump*: per visualizzare tutte le informazioni di dump dei nodi.
- *h1 ifconfig*: per visualizzare le interfacce del nodo h1.

Alcuni comandi per interagire con la rete e fare dei test minimali sono:

- *h1 ping -c 1 h2* : verifica il corretto funzionamento del percorso tra h1 e h2.
- *pingall*: esegue il ping tra tutti gli host connessi alla rete.
- *iperf*: esegue un test di banda tra 2 degli host della rete.
- *xterm h1*: permette di avviare il terminale relativo al nodo h1.
- *exit*: esce dalla rete.

Per manipolare le metriche relative ai link invece vengono messi a disposizione i seguenti comandi:

- *link s1 h1 down*: disabilita un link, in questo caso quello tra lo switch s1 e l'host h1.
- *link s1 h1 up*: attiva un link, in questo caso quello tra lo switch s1 e l'host h1.
- *s2 tc qdisc add dev s2-eth2 root netem loss 50%* : aggiunge una packet loss del 50% sulla porta eth2 dello switch s2.
- *s2 tc qdisc add dev s2-eth2 root netem delay 200ms*: aggiunge un ritardo di 200ms sulla porta eth2 dello switch s2.
- *s2 tc qdisc del dev s2-eth2 root netem loss 50%* : elimina una packet loss del 50% sulla porta eth2 dello switch s2.

- *s2 tc qdisc del dev s2-eth2 root netem delay 200ms*: elimina un ritardo di 200ms sulla porta eth2 dello switch s2.

API Python

Le API Python di Mininet permettono di creare e gestire topologie di rete in modo più flessibile e programmabile. Di seguito esponiamo alcune classi e comandi della Mid-level API:

- *Mininet*: classe per creare e gestire la rete. Il costruttore prende in input diversi parametri la topologia, gli host, gli switch,i controller, i link e ritorna un oggetto di rete.
- *addSwitch()*: aggiunge uno switch alla topologia.
- *addHost()*: aggiunge un host alla topologia.
- *addLink()*: aggiunge un link alla topologia. Si possono specificare paramentri come la banda espressa in Mbit (bw=10), il ritardo (delay='5ms'), massima dimensione della coda espressa in numero di pacchetti (max_queue_size=1000), la loss espressa in percentuale (loss=10)
- *start*: avvia la rete
- *stop*: esce dalla rete
- *pingall*: esegue il ping tra tutti gli host connessi alla rete
- *h1.cmd('comando da eseguire')*: esegue un comando su h1 da CLI e prende l'output

Con le API in Python si può anche estendere il comando *mn* usando l'opzione *–custom* per invocare la topologia ricreata nello script.
sudo mn –your_script.py –topo your_topo

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 [15]. 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.

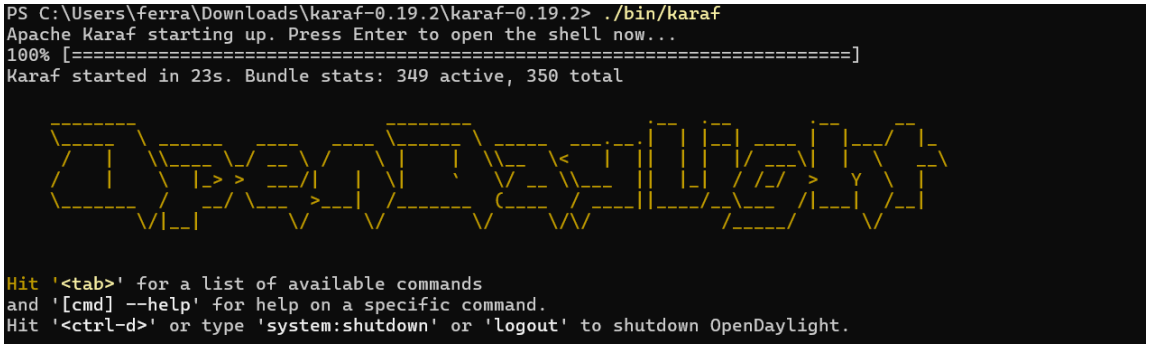

Figura 6.1: Installazione ODL

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

```
feature:list
```

Per installarle invece

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

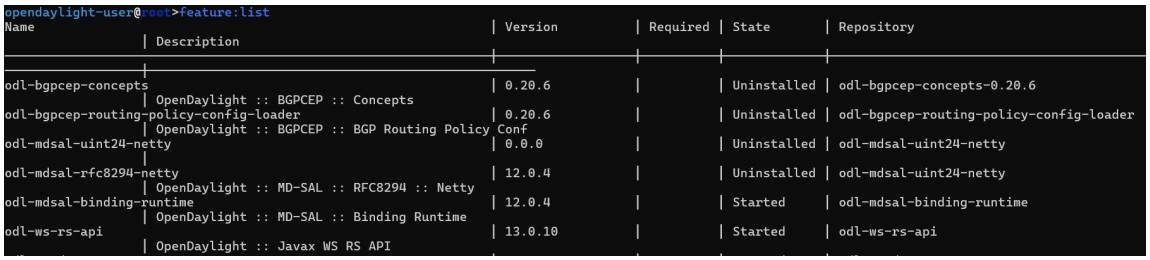

Figura 6.2: Alcune features disponibili

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] Modulo nic di odl. https://test-odl-docs.readthedocs.io/en/stable-boron/user-guide/network-intent-composition-(nic)-user-guide.html#.

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

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

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

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

[12] Grafana. https://grafana.com/.

[13] P4. https://p4.org/.

[14] Mininet. https://mininet.org/.

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