Indice

1. Introduzione

L'evoluzione delle reti moderne, accompagnata dall'incremento delle esigenze di prestazioni, affidabilità e dall'aumento della quantità del traffico, ha portato alla necessità di sviluppare nuovi paradigmi di gestione e controllo delle reti.
In questo contesto, il Software Defined Networking (SDN) ha rivoluzionato la gestione delle reti attraverso un approccio logicamente centralizzato e programmabile, separando il piano di controllo dal piano dati. Questa separazione permette agli amministratori di rete di semplificare configurazioni complesse e di introdurre politiche di rete dinamiche e affidabili. Inoltre, SDN fornisce il supporto per interfacce nordbound e southbound che facilitano rispettivamente l'integrazione con varie applicazioni di terze parti e con i dispositivi nella rete, contribuendo a un ecosistema più flessibile e reattivo [1].
L'SDN è oggi largamente impiegato in diversi contesti. Nei cloud data center, ad esempio, consente una gestione semplificata e ottimizzata dell'infrastruttura di rete, migliorando la distribuzione delle risorse e l'efficienza delle attività amministrative riducendo allo stesso tempo i costi [2]. Nelle reti di trasporto, SDN viene utilizzato per gestire il traffico in modo più dinamico e flessibile, permettendo un'ottimizzazione della larghezza di banda su lunghe distanze. Allo stesso modo, nelle reti mobili, SDN offre un controllo più efficiente dell'infrastruttura di comunicazione tra le applicazioni, i servizi in cloud e l'utente finale. In particolare, con l'avvento del 5G, l'SDN assume un ruolo fondamentale nel regolare dinamicamente la larghezza di banda per ciascun punto di accesso radio fornendo una gestione flessibile dei router e delle risorse di rete[3] [4].
Tuttavia, nonostante l'SDN astragga il controllo di gestione dai dispositivi, le sue funzioni rimangono principalmente focalizzate sulla configurazione dei singoli componenti della rete. Le attuali interfacce SDN, infatti, si concentrano sulla definizione dettagliata dei percorsi per la trasmissione dei dati piuttosto che su un modello più astratto che permetta agli amministratori di esprimere in modo semplice cosa desiderano ottenere dalla rete [1]. Questo approccio, benché efficace nella gestione delle risorse di rete, limita parzialmente la flessibilità e l'automazione in contesti fortemente dinamici, come quelli caratterizzati

da architetture di Edge Computing o reti altamente virtualizzate.

Per questo motivo viene introdotto l'Intent-Based Networking (IBN), o Networking basato su intenti, un paradigma che mira a separare ulteriormente la complessità di implementazione dal livello di gestione. IBN affronta le sfide emergenti fornendo un'astrazione di alto livello che consente di esprimere gli obiettivi di rete in modo chiaro e intuitivo. Ad esempio, un intento potrebbe essere: *"consentire alle applicazioni contabili di accedere al server XYZ, ma non consentire l'accesso alle applicazioni di produzione"* oppure *"mantenere sempre un'elevata qualità del servizio e un'elevata larghezza di banda per gli utenti di un certo livello"* [5].

Il framework IBN si occupa di tradurre gli intenti in configurazioni di rete specifiche adattando dinamicamente la rete per rispettarli. Grazie al paradigma SDN, IBN non richiede l'inserimento manuale dei comandi di policy: una volta ricevute le richieste di servizio, le converte in Key Performance Indicators (KPI), che rappresentano le metriche rilevanti da monitorare. Questo permette una verifica continua dello stato della rete, assicurandosi che soddisfi le richieste attraverso un monitoraggio in tempo reale delle KPI, garantendo così un'elevata qualità dell'esperienza (QoE) [6]. Questo approccio consente quindi alla rete di adattarsi dinamicamente ai cambiamenti, gestendo automaticamente le complessità sottostanti, migliorando l'efficienza e la reattività.

Alcuni controller SDN implementano già il concetto di Intent-Based Networking.

ONOS, ad esempio, integra un componente chiamato Intent Framework, che consente alle applicazioni di esprimere le loro esigenze di rete tramite direttive basate su policy. In modo diverso, OpenDayLight ha sviluppato l'Intent Northbound Interface (NBI), un progetto attivo fino alla release "Oxygen" del 2018 ma successivamente abbandonato per favorire approcci più pratici e scalabili. Cisco, con il suo DNA Center, ha introdotto soluzioni IBN che automatizzano la configurazione e l'implementazione della rete grazie all'intelligenza artificiale e all'apprendimento automatico concentrandosi principalmente sulla sicurezza. Ad esempio, quando un nuovo dispositivo viene aggiunto alla rete, il sistema assegna automaticamente policy basate sulla sua identità, proponendosi di risolvere i problemi di funzionamento della rete [6].

Recentemente nell'ambito del progetto europeo TeraFlow H2020, è stato sviluppato in

controller SDN omonimo. L'obiettivo di questa tesi è lo studio del controller open source Teraflow, sviluppato da ETSI [7] a partire dal 2020. In particolare il lavoro si è concentrato su come in TeraFlow vengano modellati e gestiti gli "intenti" attraverso le componenti di policy e service. Nella prima fase, è stato necessario comprendere l'architettura del controller per capire come le diverse componenti interagiscano tra loro e come queste supportino la gestione dei servizi e delle politiche di rete. L'analisi teorica è stata accompagnata dallo studio del codice e da test basati su scenari forniti dagli sviluppatori del progetto. Questi test sono stati cruciali per comprendere il comportamento del sistema in condizioni applicative specifiche vista la scarsità della documentazione aggiornata. Successivamente, si è passati alla fase di sperimentazione.

Non avendo a disposizione una rete reale, è stato utilizzato l'emulatore di reti Mininet come ambiente di test.

Partendo da file preesistenti, è stato configurato un servizio tra due end-points, permettendo la comunicazione tra di essi. Per garantire l'elevata qualità del servizio (QoS), è stata implementata una politica di gestione della rete basata su tre KPI principali: il rapporto di perdita dei pacchetti (loss ratio), la latenza e la capacità. Questi parametri dovevano rispettare soglie predefinite per assicurare prestazioni di alto livello. Un elemento chiave dell'esperimento è stato il meccanismo di reazione automatica implementato tramite il controller quando una delle metriche superava i limiti stabiliti. In questo esempio specifico, il sistema interveniva automaticamente attivando il ricalcolo del percorso del traffico tra gli end-points per ristabilire i livelli di performance richiesti.

Per garantire un monitoraggio continuo della qualità del servizio, è stato implementato un probe, un programma di utilità che cattura informazioni che, utilizzando protocolli come ping e iperf, misurava costantemente le metriche critiche (latenza, perdita di pacchetti, capacità). Queste informazioni venivano inviate al controller che le utilizzava per prendere le decisioni in tempo reale riducendo al minimo l'impatto negativo sull'esperienza degli utenti.

Il resto del documento sarà organizzato come segue.

Nel Capitolo 2 verrà esposto il contesto. All'interno ci sarà un'analisi approfondita del Software Defined Networking (SDN) e dell'Intent-Based Networking (IBN) e verranno

illustrati alcuni dei principali controller SDN presenti sul mercato che implementano soluzioni differenti per l'IBN. Alla fine del capitolo saranno introdotti Kubernetes e i microservizi.

Nel Capitolo 3 verrà descritto in modo approfondito il controller Teraflow su cui si incentra la tesi. Verrà delineata l'architettura e l'interazione necessaria tra le componenti con attenzione particolare alle componenti coinvolte nella gestione dei servizi e delle policy. Sarà inoltre introdotto gRPC, un framework grazie al quale le componenti comunicano tra di loro. Nel Capitolo 4 vengono presentati gli esperimenti svolti che costituiscono la parte pratica del lavoro. Dopo aver introdotto la teoria e gli strumenti necessari, viene definita la configurazione di un servizio di rete tra due end-point. Gli esperimenti includono la gestione automatica delle politiche di rete e il monitoraggio dei parametri con l'adozione di misure correttive nel caso essi non siano più rispettati. Infine, nell'ultimo Capitolo, saranno presentate le conclusioni.

2. Background

Nel seguente capitolo verranno presentate le principali tecnologie alla base di questo lavoro, ovvero il Software Defined Networking (SDN) e l'Intent-Based Networking (IBN). Verranno descritti due dei controller SDN più rilevanti allo stato dell'arte, ONOS [8] e OpenDaylight (ODL) [9] e infine Kubernetes e i microservizi, tecniche alla base del controller TeraFlow.
Questa panoramica fornirà il contesto necessario per comprendere meglio i successivi sviluppi trattati nel documento.

2.1 Software Defined Networking

L'architettura tradizionale di rete si basa su dispositivi fisici di interconnessione che facilitano la comunicazione tra più host a livello locale e consentono lo scambio informazioni. Nell'architettura tradizionale, con un controller decentralizzato, ciascun dispositivo integra al suo interno sia le funzioni del piano dati (data plane) che del piano di controllo (control plane).
Il piano dati è responsabile della ricezione, del processamento e dell'inoltro dei pacchetti in base alle tabelle di routing che associano un indirizzo a una data porta d'uscita [10]. Queste tabelle vengono gestite dal piano di controllo che calcola i percorsi per l'instradamento in base alla destinazione dei paccchetti e aggiorna le tabelle dei dispositivi. Nei protocolli di rete tradizionali, questi due piani operano separatamente all'interno dei dispositivi svolgendo i loro compiti in maniera indipendente.
Per determinare i percorsi di rete esistono diversi protocolli di routing che adottano un approccio decentralizzato.
RIP (Routing Information Protocol), ad esempio, utilizza un algoritmo di distance-vector in cui ogni nodo conosce solo le informazioni dai suoi vicini, quindi non la conformazione globale, e aggiorna la propria tabella sulla base dei messaggi di routing scambiati con loro [11]. Il protocollo OSPF (Open Shortest Path First) adotta invece un approccio globale.

In questo caso, ogni router all'interno dell'area conosce la topologia completa della rete e calcola i percorsi in modo indipendente utilizzando l'algoritmo di Dijkstra [12].
Questo approccio richiede in ogni caso l'esecuzione di un algoritmo di routing che, tramite un protocollo dedicato, scambia messaggi con altri dispositivi della rete per prendere decisioni.
Tuttavia, ciò introduce ritardi non necessari, rendendo così la rete meno adatta alle nuove esigenze delle applicazioni moderne che necessitano di un'elevata dinamicità.
La complessità e la staticità dell'architettura di rete tradizionale, progettata intorno a una serie di protocolli indipendenti ciascuno dei quali è focalizzato su una parte specifica delle esigenze di rete accentua ulteriormente questi problemi, specialmente con tecnologie emergenti come cloud computing, big data, streaming in tempo reale e l'Internet of Things (IoT).
Aggiungere o spostare dispositivi nella rete diventa particolarmente complicato: ogni volta che avviene una modifica, il personale di rete deve aggiornare manualmente le configurazioni di numerosi dispositivi il che introduce un significativo problema di scalabilità. Per far fronte a limitazioni di capacità e ai picchi di traffico imprevisti, invece di aggiungere collegamenti, molte aziende sovradimensionano quelli già presenti nella rete sulla base di previsioni di traffico che però risultano spesso inadeguate. Un ulteriore ostacolo è rappresentato dalla mancanza di interfacce aperte e standardizzate per le funzioni di rete. Questa dipendenza dai fornitori di apparecchiature con protocolli proprietari riduce la flessibilità e rallenta l'introduzione di nuove funzionalità [13].
Le reti tradizionali, con il loro approccio distribuito e decentralizzato al controllo e all'instradamento, si sono dimostrate inefficaci nel rispondere rapidamente a cambiamenti dinamici.
Il Software Defined Networking (SDN) nasce per rimediare ai limiti delle attuali infrastrutture di rete [14]. Proposto negli ultimi anni dalla Open Networking Foundation (ONF) [15], SDN introduce un'architettura che separa il piano di controllo dal piano dati, rendendo quest'ultimo programmabile e semplificando la gestione della rete. A differenza delle reti tradizionali, dove il controllo è distribuito su ogni dispositivo, SDN centralizza il controllo logico tramite un controller che gestisce e coordina le politiche di rete, la confi-

gurazione e l'instradamento. Questa separazione permette ai dispositivi di rete di operare come semplici apparati di inoltro, mentre il controller centrale gestisce le decisioni più complesse.Anche se la logica è centralizzata, il controller è spesso distribuito fisicamente, soprattutto nei processi di produzione, per garantire scalabilità e affidabilità [16].

Per poter funzionare, i dispositivi devono essere in grado di comunicare con il controller e riconoscere cambiamenti significativi degni di notifica per una gestione ottimizzata della rete. Questo è possibile tramite l'installazione al loro interno di componenti software con le caratteristiche necessarie detti agent [17].

La base di questo paradigma è quindi un controller remoto che, interagendo con gli agent 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 o delle richieste da parte degli utenti. Lo scopo principale è quindi ridurre e semplificare il carico di amministrazione per i singoli dispositivi.

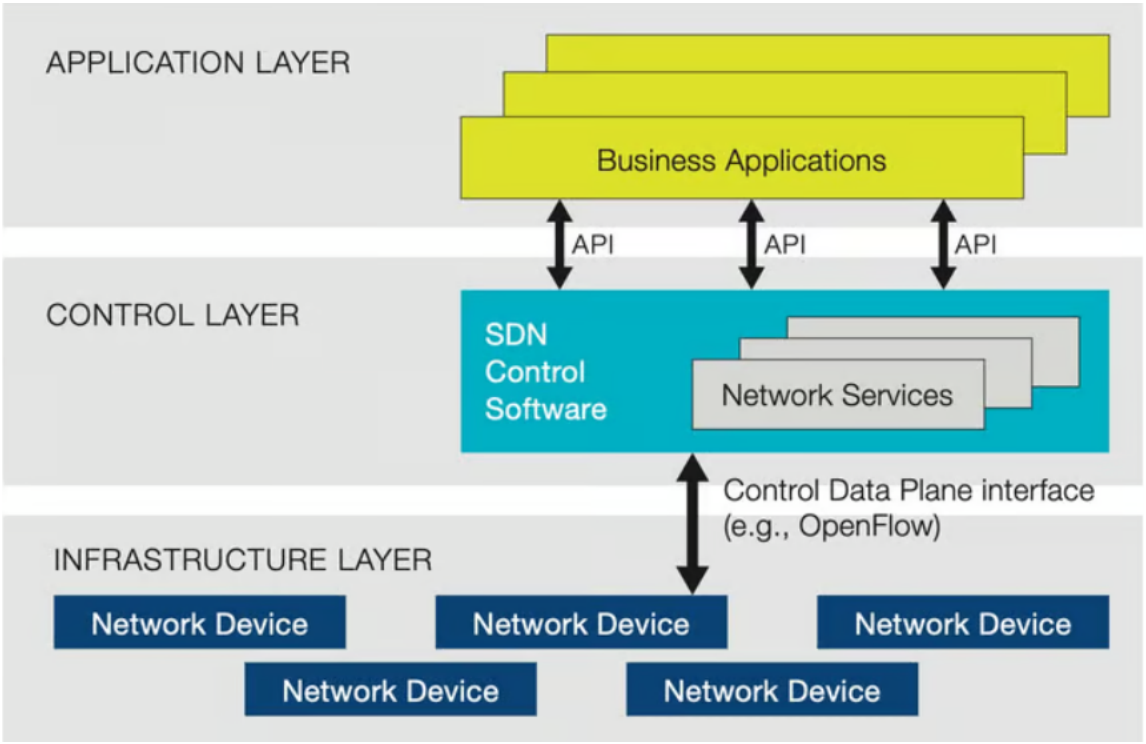

Figura 2.1: struttura di una rete SDN [18]

Come si può notare dalla Figura 2.1 la rete viene suddivisa in tre livelli: Infrastructure

layer, Control layer e Application layer [16].

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 installare le regole di inoltro per 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 [17].

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.

Il control layer consente l'implementazione del piano di controllo, che gestisce la configurazione e l'ottimizzazione delle risorse. Attraverso programmi dinamici e automatizzati, il controller calcola i percorsi ottimali per il traffico di rete, non più sulla base della destinazione ma in modo generalizzato sui vari header del pacchetto e, tramite le API SBI, impone regole di inoltro ai dispositivi sottostanti. Questo processo include la manipolazione delle tabelle di routing e l'aggiornamento delle stesse in risposta a eventi in tempo reale.

Il controller inoltre è logicamente centralizzato, in questo modo il piano di gestione (management plane) situato sopra di esso interagisce con un unico punto di accesso [10].

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 tramite interfacce grafiche e strumenti dedicati all'utente finale. Queste regole, tramite le API NBI, sono poi comunicate al controller che si occuperà di farle rispettare mediante il costante monitoraggio delle risorse del piano dati. Per esempio, nel contesto dell'application layer, le applicazioni possono includere strumenti tradizionali come firewall che definiscono politiche di sicurezza per bloccare il traffico da indirizzi IP sospetti [19]. La gestione delle risorse può anche essere gestita tramite bilanciatori di carico che distribuiscono il traffico tra diversi server per evi-

tare sovraccarichi. Le politiche di bilanciamento sono definite per ottimizzare l'uso delle risorse e per migliorare le prestazioni della rete.
Il 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

L'Intent-Based Networking (IBN) è un paradigma innovativo per la gestione delle reti che permette di separare la complessità di implementazione dal livello di gestione. Esso è nato per rispondere alla crescente ampiezza e dinamicità delle reti moderne, dove la gestione tradizionale basata su comandi manuali e configurazioni dettagliate non è più sostenibile. Negli ultimi anni, grazie a nuove tecnologie come il 5G o l'Internet of Things (IoT), diverse applicazioni stanno emergendo in differenti campi industriali.
In questo contesto, le implementazioni cloud si sono estese ed è diventato essenziale aumentare le capacità di elaborazione, eseguire servizi distribuiti e garantire il comportamento interattivo che queste nuove applicazioni richiedono.
Sono state concepite diverse tecnologie emergenti, tra cui l'IBN, per far fronte a queste necessità. Ognuna ha differenti obiettivi e spesso si integrano tra loro [20].
Il Multi-Access Edge Computing[21] (MEC) fornisce funzionalità cloud alla rete per migliorare la qualità dei servizi offerti in tempo reale portando della capacità di calcolo ai punti di accesso.
Il Network Function Virtualization[22] (NFV) permette di distribuire le funzioni di rete (firewall, NAT, DPI) come apparecchi virtuali. Questi vengono forniti in modo flessibile al cloud, consentendo così modelli innovativi di fornitura di servizi che migliorano la flessibilità e l'agilità della rete.
Nonostante queste innovazioni, rimane un divario semantico tra le esigenze delle aziende e gli obiettivi dei service provider che devono continuamente adattare e proteggere la rete in un panorama orientato ai servizi.

L'IBN nasce come un approccio nuovo, concepito dall'IETF [23]. Si occupa della gestione della rete per astrarne la complessità permettendo agli utenti finali di concentrarsi sugli obiettivi di performance senza preoccuparsi dei dettagli tecnici.
L'IBN può essere visto come un'evoluzione dell'SDN, poiché incorpora le sue principali caratteristiche superandone alcuni limiti. Mentre l'SDN fornisce delle northbound APIs che solitamente sono complesse e richiedono la conoscenza di dettagli tecninci di rete [24], IBN, invece, adotta un approccio più astratto in cui gli utenti possono esprimere le proprie esigenze definendo degli intenti, ovvero una serie di obiettivi di alto livello.
Gli intenti sono espressi in un linguaggio naturale che descrive i risultati desiderati lasciando al sistema IBN il compito di tradurli sollevando gli utenti dalla gestione diretta dei dettagli di configurazione. Un intento di rete si riferisce infatti a un livello di astrazione in cui la logica dell'applicazione è espressa in termini di cosa deve essere fatto, utilizzando regole di semantica, piuttosto che di come deve essere implementato [25]. L'idea centrale dell'IBN è di non specificare i dettagli di implementazione della rete; piuttosto, è la rete stessa che deve eseguire le azioni necessarie per soddisfare gli intenti espressi. In questo modo le applicazioni non devono gestire le direttive di rete di basso livello specifiche della tecnologia. Infatti i livelli applicativi possono interagire con l'Intent Layer evitando di apprendere il linguaggio tecnico-specifico del sistema sottostante.
L'approccio IBN è reso possibile grazie la mediazione di un Intent Orchestration Layer, che gestisce e regola il ciclo di vita delle richieste di intenti provenienti dalle applicazioni attraverso operazioni di adempimento e garanzia in un flusso di lavoro a ciclo chiuso. [20]. Queste operazioni, oltre a includere la traduzione e l'eventuale orchestrazione di configurazione per la realizzazione dei singoli intenti, mirano a garantire che la rete rispetti effettivamente l'intento desiderato sulla base della raccolta, aggregazione e valutazione in tempo reale dei dati di monitoraggio. L'IBN fa uso di un Intent Repository, un database in grado di interagire con i moduli di gestione e traduzione degli intenti per fornire la mappatura tra l'intento e la sua configurazione [26]. Questo paradigma offre vantaggi anche ai fornitori di rete, infatti permette di migliorare l'agilità, la disponibilità e la gestione delle reti a un livello di astrazione più elevato e verificare continuamente che gli obiettivi siano raggiunti.

2.3 Controller allo stato dell'arte

Prima di analizzare nel dettaglio TeraFlow si introducono due controller che hanno già raggiunto lo stato dell'arte: ONOS e OpenDayLight. Questi controller rappresentano soluzioni già consolidate nel campo del Software Defined Networking e sono ampiamente utilizzati e studiati sia in ambito accademico che industriale. La loro descrizione ci consentirà di mettere in evidenza le differenti caratteristiche per poter analizzare meglio le innovazioni introdotte da TeraFlow.

2.3.1 ONOS

Open Network Operating System (ONOS) [8] è uno dei controller SDN più noti. E' un progetto nato dalla Open Networking Foundation (ONF) [15] 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 stato necessario poter sviluppare una piattaforma applicativa modulare ed estendibile. Per questo motivo la base dell'architettura di ONOS, come si può vedere dalla Figura 2.2, è costituita da una piattaforma di applicazioni distributite, che utilizzano Java come linguaggio di programmazione. Quest'ultima è collocata sopra OSGi [28] e Apache Karaf [29] così da permettere l'installazione e l'esecuzione dinamicamente. 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.

ONOS è stato progettato come un sistema distribuito in cui tutti i nodi del cluster sono equivalenti in termini di funzionalità e capacità software. Ogni nodo può quindi svolge-

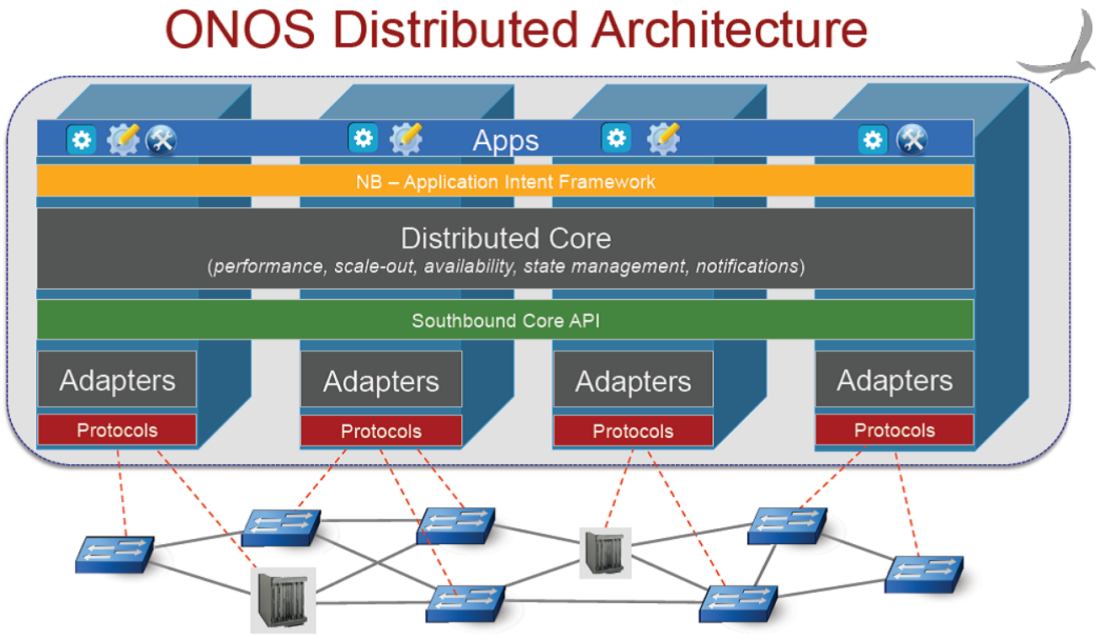

Figura 2.2: Architettura di ONOS [27]

re le stesse operazioni e contribuire in maniera simmetrica al funzionamento del sistema. In caso di guasto di una componente le altre sono in grado di sostenere e 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 supporta un'architettura modulare che permette agli operatori di configurare e adattare facilmente la rete alle loro esigenze specifiche. Grazie alla sua natura distribuita, consente una gestione scalabile e offre il supporto per API che facilitano l'integrazione con altri sistemi di gestione e orchestrazione. Questo permette una gestione efficiente delle risorse di rete, garantendo flessibilità e reattività alle variazioni delle condizioni operative.
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.
Abilitando il Transport Layer Security (TLS) per l'interfaccia SBI e l'Hypertext Transfer Protocol Secure (HTTPS) per l'interfaccia NBI, ONOS garantisce una buona sicurezza monitorando e bloccando l'accesso non autorizzato alle risorse in fase di esecuzione [30].

Per quando riguarda gli intenti ONOS utilizza il framework Intent Monitor and Reroute (IMR) per offrire un sistema dinamico e ottimizzato di gestione del traffico in rete. Questo framework permette non solo di reindirizzare il traffico in base a necessità specifiche, ma anche di ottimizzare l'uso delle risorse di rete in base a obiettivi e scopi definiti dagli utenti. IMR monitora continuamente le statistiche di flusso, adattando in tempo reale i percorsi per sfruttare al meglio i collegamenti disponibili.
IMR filtra le statistiche di raccolta dei dati sugli intenti e permette agli utenti di monitorare la rete tramite APIs come CLI o REST [31].
Questa capacità di monitoraggio si basa sulla raccolta di dati di flusso di basso livello generati da ONOS.
Attualmente, IMR supporta due tipologie di obiettivi: Point-to-point, che stabiliscono una connessione diretta tra nodi, e link collection, che si riferiscono a un insieme di collegamenti monitorati per ottimizzare il traffico e la distribuzione delle risorse. Inoltre, IMR interagisce con l'Intent Manager e il Flow Rule Manager di ONOS per tracciare i flussi e le destinazioni [32], garantendo una gestione efficiente delle risorse di rete. Questa integrazione permette non solo il monitoraggio continuo, ma anche la riconfigurazione dinamica della rete, massimizzando l'uso di ciascun collegamento durante la trasmissione dei dati. Il sistema riduce così le interruzioni di servizio e migliora la gestione del traffico.
Un'intento di rete è considerato un sottoinsieme del traffico con valori specifici assegnati a ciascun pacchetto. Gli utenti possono definire percorsi che passino attraverso un numero specifico di nodi o che garantiscano una determinata quantità di larghezza di banda. Questo permette di monitorare e modificare facilmente i percorsi in modo flessibile [31].

2.3.2 ODL

OpenDaylight[9] 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 [33] che si occupa di coordinare il supporto a progetti open source volti a migliorare la comunicazione e la gestione dei dati su una rete.
ODL è un framework scritto in Java progettato per soddisfare esigenze specifiche dell'utente e offrire alta flessibilità.

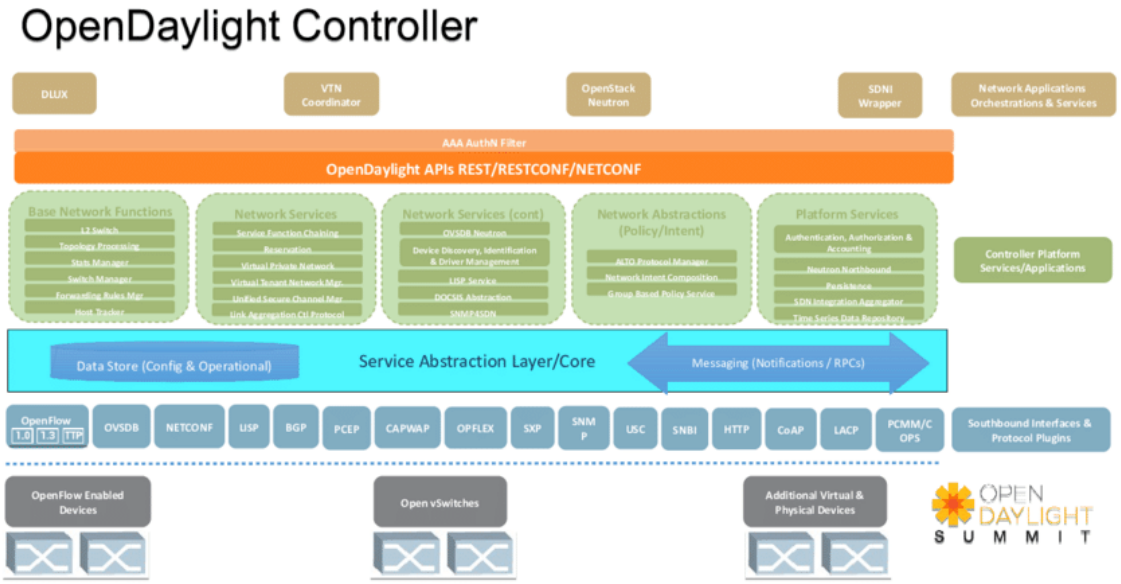

Figura 2.3: Architettura del controller OpenDayLight [34]

L'architettura di OpenDayLight, come mostrato in Figura 2.3, è su più livelli [35]. 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 supporto 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.

Un aspetto rilevante dell'architettura è la presenza di microservizi, che l'utente può attivare o disattivare in base alle proprie esigenze. Di default, questi microservizi sono disabilitati, offrendo un alto livello di personalizzazione.
I microservizi sono implementati come moduli all'interno del controller, e possono essere collegati tra loro per eseguire diverse attività di rete. Questi moduli si connettono dinamicamente al Service Abstraction Layer (SAL), che funge da interfaccia tra il piano di controllo e il piano dati.
Per la gestione dei moduli a runtime e l'installazione di nuove funzionalità da implementare nel software di ODL viene utilizzato Apache Karaf [29]. Karaf fornisce un ambiente modulare in cui è possibile implementare e aggiornare le funzionalità senza interrompere il funzionamento del controller. Grazie al framework Model-Driven Service Abstraction Layer (MD-SAL), gli sviluppatori possono creare nuove funzionalità sotto forma di servizi e protocolli interconnessi. Il controller espone delle API NBI di supporto alle applicazioni. Alcune delle API supportate includono il framework OSGi [28], per gestire le applicazioni che girano all'interno del controller, e REST per la comunicazione con le applicazioni esterne [36].
Per risolvere i problemi legati alla scalabilità, disponibilità e persistenza dei dati, ODL può essere distribuito in più istanze su macchine diverse, le quali cooperano tra loro tramite un meccanismo di clustering.

ODL per gestire gli intenti aveva messo a disposizione una NorthBound Interface (NIC) che è stata abbandonata dal 2018 nelle release successive a Oxygen.
Network Intent Composition (NIC) [37] è un'interfaccia progettata per consentire agli utenti di esprimere uno stato desiderato in una forma indipendente dall'implementazione sottostante, detto intento.
Un intento è una direttiva di alto livello che consente agli utenti di definire il comportamento richiesto della rete senza dover specificare i dettagli tecnici della configurazione delle risorse. Attraverso l'interfaccia Northbound (NBI), gli intenti vengono espressi in forma astratta e gestiti dal controller.
Una delle funzioni centrali è l'Intent Compilation Engine che ha il compito di convertire

e tradurre questi intenti in regole di configurazione concrete per il protocollo di controllo sottostante (come OpenFLow, SNMP, Netconf..), utilizzando le risorse di rete disponibili. Questo garantisce che le richieste siano trasformate in comandi eseguibili sui dispositivi di rete, sia fisici che virtuali.

Inoltre, NIC si avvale di un linguaggio di programmazione degli intenti che permette di utilizzare una varietà di linguaggi di policy e di programmazione SDN. Questa feature consente di descrivere in modo flessibile il comportamento desiderato della rete offrendo un approccio descrittivo.

NIC impiega una funzione di composizione che consente di combinare più richieste di policy provenienti da varie applicazioni SDN in un insieme coerente di azioni. La composizione degli intenti gestisce i conflitti tra le diverse richieste e li risolve per garantire che le azioni intraprese siano consistenti con le politiche della rete.

Per raccogliere e gestire le politiche NIC impiega diversi database logici di informazioni tra cui: il Network Service Intent DB per le politiche relative ai servizi di rete, l'End Point Intent DB per politiche sugli endpoint, il Network Security Intent DB per le politiche riguardandi la sicurezza della rete[38]. Oltre a questi, esistono altri database che si occupano di differenti tipi di politiche, garantendo così una gestione completa delle diverse esigenze.

NIC è stato progettato per essere un'interfaccia indipendente dal controller e dal protocollo di rete. Questo significa che gli intenti possono essere trasferiti tra diverse implementazioni di controller SDN, garantendo interoperabilità e flessibilità nell'uso di protocolli di controllo differenti. ATTUALMENTE ODL METTE A DISPOSIZIONE PER LA GESTIONE DEGLI INTENTI

2.4 Kubernetes

Kubernetes [39], noto anche come K8s, è una piattaforma open-source per l'orchestrazione dei container. Creato originariamente da Google nel 2014, è stato poi donato alla Cloud Native Computing Foundation (CNCF [40]), è progettato per automatizzare la gestione, lo scaling e il deployment di applicazioni containerizzate. Quest'ultime vengono eseguite

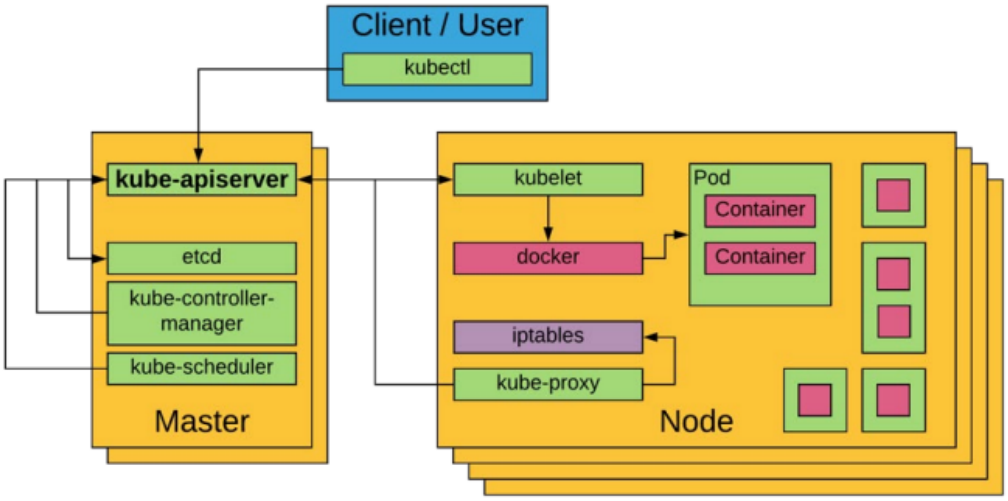

Figura 2.4: Architettura di Kubernetes [41]

all'interno dei cluster Kubernetes costituiti da macchine fisiche o virtuali chiamate nodi. Questi cluster comprendono due tipologie principali di macchine: il Master, che gestisce l'intera orchestrazione, e i Nodi, che eseguono i container all'interno di un pod. I pod rappresentano l'unità fondamentale in Kubernetes e contengono uno o più container che condividono risorse come CPU e memoria. Ogni nodo esegue due servizi principali: il kubelet, che riceve e gestisce i comandi per l'esecuzione dei container, e il kube-proxy, che si occupa della configurazione delle regole di rete, garantendo che le connessioni siano correttamente instradate verso i pod. Il Master è responsabile del coordinamento del cluster, e ciò avviene tramite alcuni componenti essenziali, tra cui [41]:

- **etcd**: un database distribuito che memorizza lo stato del cluster.
- **API server**: fornisce l'interfaccia di comunicazione tra i componenti interni del sistema e gli utenti esterni.
- **kube-controller-manager**: monitora lo stato delle risorse del cluster e applica modifiche necessarie.
- **kube-scheduler**: decide su quali nodi eseguire i pod in base alle risorse disponibili.

In Kubernetes è possibile avere più di un Master per migliorare la disponibilità del sistema, con un Master principale e dei nodi replica che garantiscono la continuità del servizio. Le risorse dei cluster vengono specificate attraverso file di configurazione, tipicamente utilizzando YAML (Yet Another Markup Language), consentendo agli utenti di specificare deployment, servizi e configurazioni necessarie all'esecuzione delle applicazioni.
Kubernetes segue un'architettura basata sul paradigma client-server (raffigurata in Figura 2.4), con i pod come unità di base e il Master che agisce da server centrale e i Nodi da client. Un aspetto fondamentale di Kubernetes è il suo sistema di self-healing: se un processo si arresta o un pod fallisce, Kubernetes è in grado di riavviarlo automaticamente. Questo sistema permette alle applicazioni di rimanere sempre in uno stato ottimale garantendo la continuità del servizio. Inoltre, durante gli aggiornamenti delle applicazioni, Kubernetes utilizza il meccanismo di rolling update (aggiornamenti in sequenza): i pod vengono aggiornati uno alla volta, senza provocare interruzioni del servizio. Questo garantisce che il numero richiesto di pod rimanga sempre attivo e in esecuzione [41]. Oltre alle funzionalità di base, Kubernetes offre concreti vantaggi operativi, soprattutto in termini di ottimizzazione delle risorse. Grazie alla capacità di distribuire le applicazioni containerizzate in modo efficiente su un numero limitato di macchine è possibile ridurre i costi legati all'infrastruttura. Questo porta a un utilizzo più efficiente delle risorse hardware, limitando il tempo in cui le macchine restano inattive e riducendo il consumo energetico [42]. Anche TeraFlow adotta Kubernetes per orchestrare i suoi componenti, sfruttando le capacità di scalabilità e automazione della piattaforma. Questo permette a TeraFlow di gestire dinamicamente i suoi servizi e adattarsi rapidamente alle esigenze di rete, garantendo così una maggiore flessibilità e resilienza.

2.5 Miscroservizi

Alcune indagini statistiche hanno dimostrato che nel 2021 il 71% delle aziende utilizzava almeno parzialmente i microservizi [43] e questo trend è in continua crescita.
I microservizi rappresentano un nuovo paradigma di progettazione software che si basa sulla scomposizione di applicazioni in una serie di servizi autonomi, chiamati microser-

vizi, ciascuno con uno scopo definito e gestibile in modo indipendente in grado di comunicare tra loro. Questa architettura consente una maggiore flessibilità nella manutenzione e nell'evoluzione del software, permettendo una trasformazione graduale di sistemi complessi senza dover ricorrere a una riscrittura completa. Un importante vantaggio dei microservizi è la loro capacità di semplificare la fase di realizzazione e testing: ogni servizio può essere sviluppato e testato separatamente, riducendo la complessità del progetto generale. Tuttavia, l'integrazione di diversi microservizi richiede una gestione attenta, per una maggiore complessità nella gestione della rete e per una possibile riduzione delle prestazioni a causa della distribuzione su più nodi.

Sistemi monolitici complessi possono essere gradualmente trasformati in microservizi, riducendo il rischio di un rifacimento completo e consentendo l'incremento della manutenzione e dell'agilità del software. L'esempio classico è quello di Amazon che è passato da un database unico a un'architettura orientata ai microservizi, ma la modularità dei microservizi può facilitare anche la modernizzazione delle applicazioni legacy, riducendo i rischi legati ai cambiamenti strutturali più radicali [44]. Altri esempi di grandi aziende che hanno scelto di utilizzare questa architettura grazie alla capacità di raggiungere un time-to-market migliore sono Netflix, Uber e Linkedin.

Nel contesto di applicazioni cloud-native, come TeraFlow, i microservizi hanno un ruolo fondamentale nel loro sviluppo insieme con Kubernetes. In un'applicazione basata su microservizi orchestrata da Kubernetes, ogni microservizio è isolato in un cointainer e può avere più istanze, ognuna rappresentata da un pod, consentendo una gestione più efficiente delle risorse rispetto alle tradizionali macchine virtuali (VM), dove ogni istanza necessita di un sistema operativo completo. I container invece condividono lo stesso sistema operativo del nodo host, permettendo un riavvio più rapido durante l'aggiornamento o il ripristino [45] e un minor consumo di risorse. I container all'interno dei pod possono condividere librerie e risorse necessarie

3. TeraFlow: architettura e gestione degli intenti

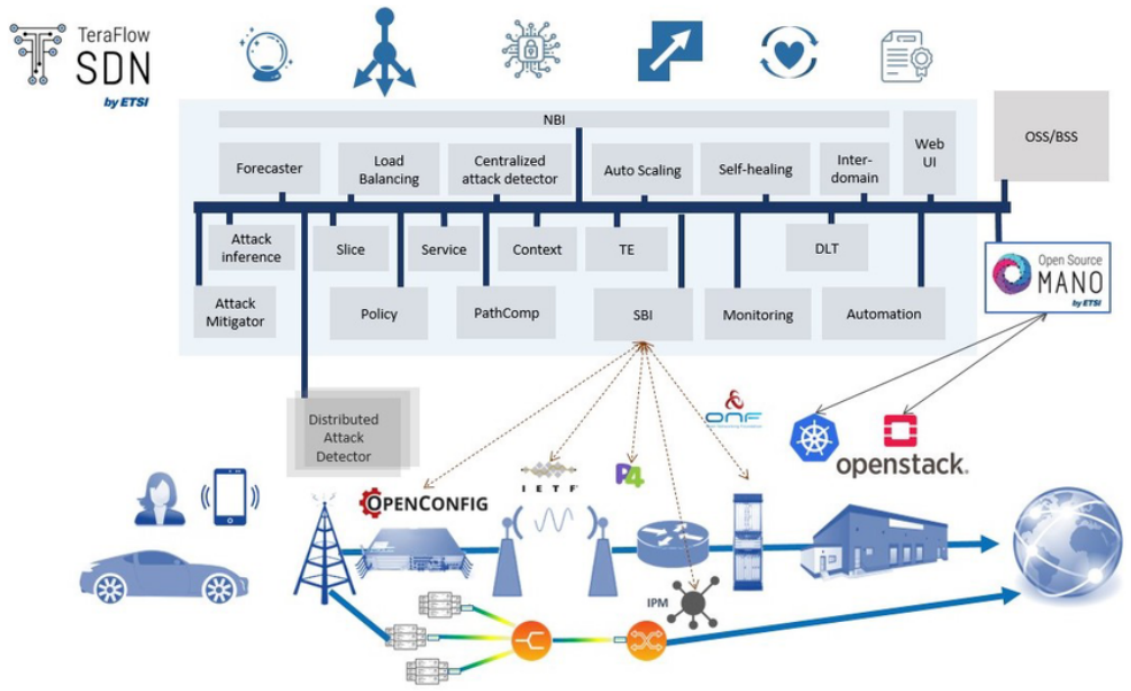

Figura 3.1: Architettura di TeraFlow [46]

Il controller SDN su cui ci focalizzeremo è TeraFlow [47], una piattaforma innovativa e recentemente proposta, sviluppata da un'ampia comunità open source nell'ambito di un progetto europeo.

Infatti, è stato finanziato dall'unione europea per il programma di ricerca e innovazione Horizon 2020 [48] e supportato dal 5G PPP [49], un'iniziativa congiunta tra la Commissione Europea e l'industria europea delle telecomunicazioni. Nonostante la quantità di controller SDN i fondatori di TeraFlow hanno riscontrato come problema comune il calo delle contribuzioni ai progetti negli ultimi anni. Questo declino mette a rischio il continuo sviluppo e supporto per i nuovi bisogni e requisiti delle reti moderne lasciando molte soluzioni esistenti inadeguate per affrontare le sfide emergenti.

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 permettendo di gestire un volume di traffico equivalente a un terabit al secondo. Questa capacità è cruciale per supportare le elevate esigenze di connettività delle reti B5G.

Un ulteriore obiettivo è ridurre il divario tra le esigenze delle industrie e le capacità offerte dagli standard SDN. Questo controller, attualmente in fase di sviluppo, sarà progettato per integrarsi con gli attuali framework NFV e MEC. Inoltre, si prevede che supporti l'integrazione delle apparecchiature di rete ottica e a microonde, e che sarà compatibile con altri controller come ONOS, ma anche con istanze multiple di TeraFlow che gestiscono diversi domini, al fine di sfruttare funzionalità avanzate e facilitare l'interoperabilità con altre reti.

A differenza dei controller presentati precedentemente TeraFlow è stato progettato con un'architettura cloud-native, pensata per sfruttare appieno gli ambienti cloud. Questo approccio consente una maggiore flessibilità e scalabilità rispetto ai sistemi tradizionali grazie alla suddivisione delle applicazioni in microservizi che possono essere gestiti, distribuiti e aggiornati in modo indipendente. Ciò differenzia TeraFlow dai controller modulari tradizionali che non sono stati progettati con la stessa capacità di adattamento alle moderne richieste delle reti.

L'uso di container, una tecninca di virtualizzazione leggera, permette di isolare ogni microservizio utilizzando una minore quantità di risorse rispetto alle macchine virtuali (VM). Ciascun microservizio interagisce con gli altri attraverso la connessione di rete rendendo il controller disaggregato. L'ambiente creato da un container include sia il codice di esecuzione che le sue dipendenze. Questi container sono gestiti tramite Kubernetes, un orchestratore responsabile dell'allocazione delle risorse in termini di capacità di calcolo, memoria e archiviazione[50]. Kubernetes offre diverse funzionalità che garantiscono dinamicità, autoriparazione, integrità e bilanciamento del carico [51] [52]. Le componenti principali del sistema sono implementate in Java (solo quelle di Automation e Policy) e Python e l'ambiente è sviluppato presso la sede del CTTC a Barcellona.

L'obiettivo di TeraFlow 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 che automatizzi la gestione della rete e sia in grado di scalare per gestire miliardi di dispositivi.
TeraFlow mira a ottimizzare l'uso delle risorse di rete per migliorare l'efficienza energetica e ridurre i costi operativi.
Per gestire la configurazione di rete, TeraFlow utilizza una componente chiamata Context che memorizza la configurazione di rete, inclusi topologie, dispositivi, collegamenti e servizi, in un database No-SQL[53]. Questa componente garantisce la coerenza dei dati gestiti dai vari componenti del controller SDN.
La componente di Monitoring 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 progettati per essere semplici e dettagliati, ciò è reso possibile grazie all'uso di protocolli leggeri. Dal punto di vista della sicurezza il sistema utilizza un approccio basato sul Machine Learning per la prevenzione e la mitigazione degli attacchi.
Alcuni dei requisiti funzionali del controller sono rappresentati da [54]:

- **usabilità**: realizzata grazie a un'interfaccia utente web (web UI) che consente la configurazione di servizi predefiniti e visualizzazione personalizzabile delle metriche.

- **scalabilità**: è intrinseca nel design del controller con la replicazione automatica dei 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.

Questo progetto riveste un ruolo chiave nel panorama delle tecnologie 5G, contribuendo a unire diverse università e istituti di ricerca per sviluppare soluzioni all'avanguardia lavorando con organismi di standardizzazione per garantire l'adozione su scala globale.

3.1 Componenti architetturali

Le componenti di TeraFlow sono classificate in due categorie; le componenti principali del sistema operativo e le netapp sovrapposte [55]. Le componenti del sistema operativo di rete (Network Operating System - NOS) formano la base dell'infrastruttura di TeraFlow, offrendo servizi di connettività per infrastrutture di rete programmabili avanzate [55]. Queste infrastrutture sono basate su tecnologie come P4 (Programming Protocol-independent Packet Processors), OpenConfig e TAPI (Transport API), che forniscono modalità flessibili e programmabili per gestire il traffico di rete e le configurazioni.
Le netapp sovrapposte sono applicazioni che operano sopra il livello del NOS, sfruttando le interfacce di comunicazione, indipendentemente dal linguaggio di programmazione, per interagire con le infrastrutture di rete. Queste applicazioni possono includere strumenti per il monitoraggio, la gestione della qualità del servizio, e altre funzionalità.
Per garantire l'interoperabilità tra le diverse componenti, TeraFlow utilizza un bus gRPC (Google Remote Procedure Call) come protocollo interno, un framework che consente la comunicazione tra servizi 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. I Protocol Buffers utilizzati sono dettagliati nella pagina dedicata del sito [56].
TeraFlow fornisce, inoltre, funzionalità di automazione SDN e IBN avanzate basate su policy e parti interessate. Nello specifico, TeraFlow Automation sfrutta importanti eventi di sistema per realizzare la (ri)configurazione di servizi e dispositivi in modo zero-touch, ossia minimizzando l'intervento umano attraverso l'automazione completa della configurazione e della gestione dei servizi e dei dispositivi. Questo approccio contribuisce significativamente alla riduzione delle spese operative, poiché riduce la necessità di interventi manuali e accelera i processi di configurazione e manutenzione.[55].
Le componenti di TeraFlow, come si può vedere dalla figura 3.2, sono suddivise in diversi livelli di astrazione per facilitare la gestione della rete:

- Device Level Abstraction: fornisce un'astrazione dei dispositivi fisici presenti, consentendo al controller di interagire con diverse tipologie di hardware.

- Service Level Abstraction: focalizzato sulla gestione e configurazione dei servizi, questo livello gestisce le interazioni con l'SBI (Southbound Interface) per fornire un'interfaccia unificata.

- Management Level Abstraction: realizza la gestione complessiva della rete, che include il monitoraggio, il controllo e la manutenzione dei servizi e dei dispositivi gestiti tramite SBI, garantendo un controllo centralizzato dell'infrastruttura.

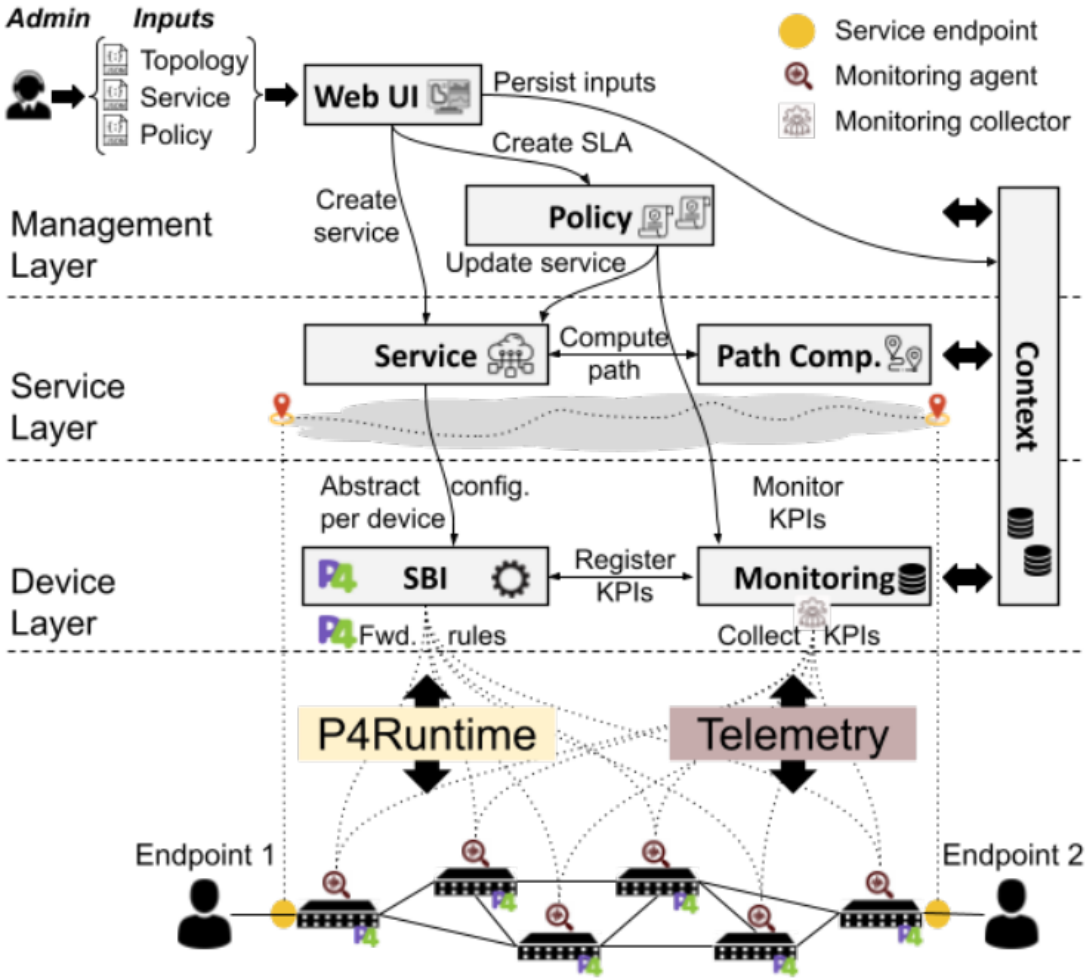

Figura 3.2: Interazione tra alcune componenti in TeraFlow

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.
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 di configurarli dinamicamente a tempo di esecuzione [57]. Dopo aver stabilito la connessione con un determinato dispositivo e aver verificato la disponibilità, la componente offre una API che consente di inviare le configurazioni scelte dal controller al dispositivo sotto forma di file JSON. In questo modo, il dispositivo riceve istruzioni per configurare i suoi parametri operativi in linea con le politiche definite a livello di rete.
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 un database (Metrics Database) così da poter fornire dati dimensionali con serie temporali visualizzabili su Grafana [58]. Quando un valore KPI registrato supera determinate soglie predefinite, la componente di Monitoring utilizza il sistema di gestione degli eventi del controller, un'API dedicata alla gestione e distribuzione di notifiche riguardanti eventi di rete per generare e notificare un allarme. Questo allarme viene inviato alla componente responsabile della gestione di quel servizio specifico, ossia quella che ha originariamente richiesto il monitoraggio della specifica metrica all'interno del controller. Questa API permette al controller di notificare in tempo reale le componenti del sistema (ad esempio, quelle di gestione o sicurezza) sugli eventi critici, consentendo interventi tempestivi e automatizzati [57].
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. Supponendo che la componen-

te di Monitoring stia monitorando la latenza di un collegamento di rete tramite una KPI denominata ”Latency” e che la latenza massima accettabile sia 100ms, se la latenza supera questa soglia, ad esempio 120ms, viene generato un evento così composto:

- KPI: ”Latency”
- Timestamp: ”06-09-2024 14:35:20” (la data e l’ora a cui si è verificato l’evento)
- KPIValue: 120 ms

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. 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. Per svolgere questo compito, la componente si avvale di diverse funzionalità offerte dalle altre parti del sistema. In particolare, per il calcolo dei percorsi di rete, si affida alla componente PathComp (Path Computation). Ad esempio, se un intento richiede la creazione di una connessione con una latenza minima tra due nodi specifici, la componente invia una richiesta a PathComp, che calcola il percorso ottimale basato su parametri come la larghezza di banda o la latenza. Una volta ricevuta la risposta, la componente di Service utilizza uno scheduler per configurare i dispositivi di rete lungo il percorso selezionato, utilizzando le connessioni restituite dalla PathComp [57]. Queste regole vengono poi propagate all’interfaccia Southbound (SBI) attraverso dei file JSON, consentendo una configurazione automatizzata della rete e permettendo di astrarre la complessità del livello sottostante all’utente.

La componente di Service supporta diversi tipi di servizi ed è in grado di utilizzare vari protocolli per configurare i dispositivi di rete. Implementa inoltre una Service Handler API che consente agli operatori di rete di definire i comportamenti necessari per ciascun tipo di servizio [57].

- **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**

Un'altra componente che fa parte di questo livello è la PathComp. Questa, come accennato in precedenza, 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 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 [51].
Il service layer permette al controller di tradurre gli intenti in regole concrete che vengono poi propagate al livello sottostante tramite l'SBI. Tuttavia, da solo non ha la capacità di rispondere dinamicamente agli eventi che si verificano nella rete in tempo reale, come il cambiamento dello stato di un collegamento o di una risorsa di rete. Per essere in grado di creare, aggiornare o cancellare i servizi di rete in base a tali eventi è necessario introdurre un ulteriore livello di astrazione che automatizzi queste operazioni: il Management Level Abstraction.

3.1.3 Management Level Abstraction

Questo livello di astrazione è stato introdotto per consentire l'interazione dinamica con la componente di Service permettendo la creazione, l'aggiornamento o l'eliminazione automatica di un servizio in risposta agli eventi provenienti dalla rete. In altre parole, il sistema non si limita a configurare i dispositivi solo su input manuali o su richiesta esplicita, ma è in grado di adattarsi dinamicamente ai cambiamenti dello stato della rete in tempo reale. Questo livello consente dunque al controller di reagire rapidamente a eventi come congestioni, guasti o modifiche nella topologia, garantendo che la rete mantenga un alto livello di efficienza e affidabilità.

Una delle componenti fondamentali è la componente di Policy, che interagisce strettamente con la componente Monitoring descritta precedentemente.

Si occupa di definire condizioni di politica che possono essere applicate sia a livello di singoli dispositivi che a livello di dominio della rete. Le politiche possono includere più regole collegate tra loro tramite condizioni logiche di AND/OR [57], ognuna delle quali genera una KPI specifica da far monitorare alla componente di Monitoring.

Ogni regola è composta da una KPI che la identifica, un operatore numerico di confronto, e un KPIValue. Quest'ultimo, in combinazione con l'operatore numerico, definisce l'intervallo di valori accettabili o non accettabili per quella specifica metrica.

Se le regole definite nella politica vengono soddisfatte la componente di Monitoring genera un allarme che viene inviato, in questo caso, alla componente di Policy, che reagisce attivando le azioni predefinite. Ad esempio, se si verifica un eccesso di latenza, la componente di Policy può attivare il ricalcolo del percorso tramite la componente di Service o altre azioni correttive.

Questo meccanismo consente al controller di reagire in modo efficace agli eventi e di ripristinare uno stato desiderato per i dispositivi.

La componente Policy utilizza il Context Database per identificare quali dispositivi o servizi sono coinvolti nella politica. Se viene fornito l'ID del servizio, la componente recupera i dispositivi associati a esso, in caso contrario la componente scansiona una lista di dispositivi per individuare quelli che devono rispettare le regole impostate.

Una regola di politica può avere vari stati:

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

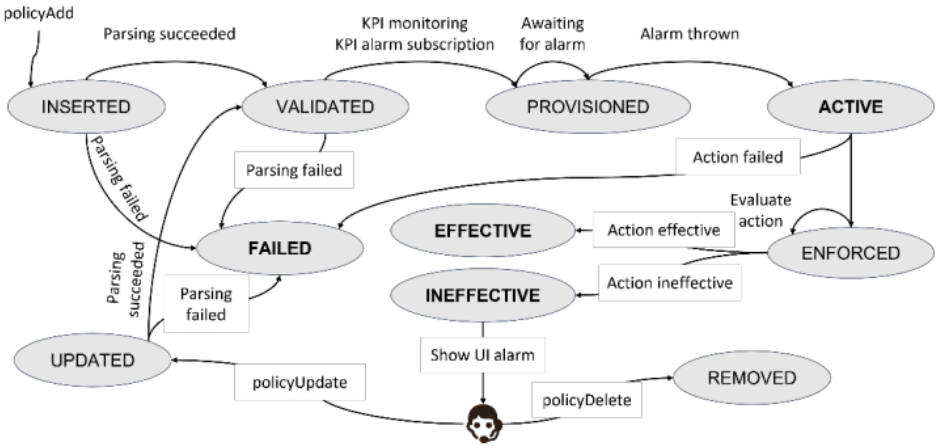

Figura 3.3: Macchina a stati interna alla componente di policy [57]

Grazie all'integrazione tra la componente di Policy e Monitoring, TeraFlow ha reso possibile l'associazione delle condizioni di politica con gli allarmi del sistema di Monitoring. In questo modo può implementare una gestione dinamica della rete, in grado di rispondere in tempo reale agli eventi che possono degradare la qualità del servizio, come l'aumento della latenza o la perdita di pacchetti. Questa reattività permette di mantenere gli SLA (Service Level Agreements) senza necessitare di interventi manuali continui da parte degli

operatori aumentando l'efficienza delle varie operazioni e riducendo la complessità operativa. In ambienti su larga scala, come le reti distribuite composte da microservizi, questa capacità di adattamento consente non solo di reagire agli eventi ma anche di prevenire potenziali criticità, ottimizzando continuamente le risorse di rete.

3.2 gRPC

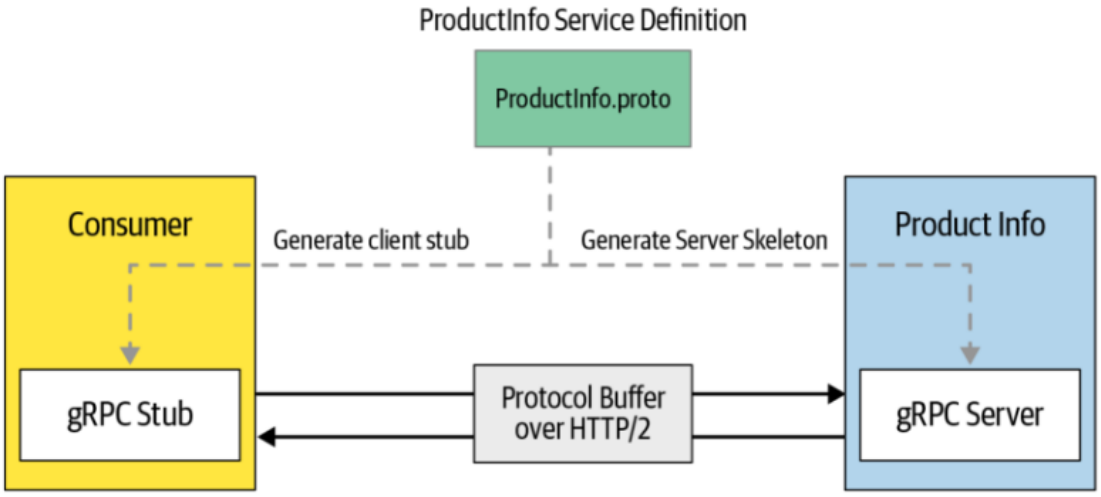

Figura 3.4: Funzionamento del protocollo gRPC [59]

RPC (Remote Procedure Call) era un popolare protocollo inter-processo[59]. Esso utilizza uno scambio di messaggi con le caratteristiche di basso sovraccarico, semplicità e trasparenza. Per questi motivi è stato ampiamente usato per diverso tempo in sistemi applicativi distribuiti. Con RPC un client può invocare da remoto una funzione di un metodo. Un utente di RPC non può distinguere dove viene eseguita la procedura chiamata, infatti viene chiamata come una procedura locale [60]. Tuttavia, la maggior parte delle convenzionali implementazioni (un esempio è RMI) sono complesse dato che richiedono una gestione diretta dei protocolli di trasporto, come TCP, che ostacolano l'interoperabilità e richiedono specifiche eccessive [59].

Google Remote Procedure Calls (gRPC[61]) 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.
gRPC è basato sul protocollo di trasporto HTTP/2 che supporta la comunicazione bidirezionale. Sebbene anche gRPC utilizzi TCP la differenza con i tradizionali protocolli RPC risiede nelle caratteristiche avanzate di HTTP/2, che includono funzionalità come il multiplexing delle connessioni, il flusso bidirezionale e una migliore gestione delle prestazioni. gRPC include supporto per il bilanciamento del carico, il tracciamento, il controllo dello stato e l'autentificazione[62] [63]. Inoltre, ha la capacità di supportare alte performance di streaming in modalità push da parte del server, consentendo l'indipendenza basata sul set di dati e definizioni, grazie ai metodi YANG, e la compressione di dati binari [62].
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 che di velocità di serializzazione e deserializzazione.
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. Essa viene definita in un file proto, che a prima vista può sembrare un file di testo ordinario, come si può notare in figura 3.5, in realtà specifica i metodi con i parametri di input e i valori di ritorno. Tutti i metodi nella definizione di questa interfaccia possono essere invocati dal client da remoto. Una volta definita si può generare il codice lato server (server skeleton) e il codice lato client (client stub) nel linguaggio desiderato usando il compilatore Protobuf *protoc*[59].
Negli esperimenti svolti in [64] si è dimostrato che gRPC, grazie ai file proto, in scenari che coinvolgono ambienti più complessi con molte componenti (come server, DNS, firewall..) può portare a una riduzione di quasi il 27% nel tempo di creazione del server rispetto a REST (Representational state transfer), uno tra i più moderni e utilizzati sistemi di trasmissione dati, riducendo il tempo di esecuzione complessivo. Ciò che è stato provato rende gRPC più adatto per architetture composte da microservizi, come TeraFlow, in

```
service ForecasterService {
  rpc GetForecastOfTopology (context.TopologyId) returns (Forecast) {}
  rpc GetForecastOfLink(context.LinkId) returns (Forecast) {}
  rpc CheckService (context.ServiceId) returns (ForecastPrediction) {}
}

message SingleForecast {
  context.Timestamp timestamp= 1;
  double value = 2;
}

message Forecast {
    oneof uuid {
      context.TopologyId topologyId= 1;
      context.LinkId linkId = 2;
    }
    repeated SingleForecast forecast = 3;
}

enum AvailabilityPredictionEnum {
    FORECASTED_AVAILABILITY = 0;
    FORECASTED_UNAVAILABILITY = 1;
}
```

Figura 3.5: Esempio di Protocol Buffer file in TeraFlow [56]

cui le prestazioni e la scalabilità del sistema sono elementi critici.

Quando il 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[59].

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, infatti supporta nativamente il

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

4. Studio e sperimentazione della gestione di policy in Teraflow

In questo Capitolo verranno esposti i vari esperimenti eseguiti utilizzando il controller SDN TeraFlow per quanto riguarda la gestione degli intenti. Nella fase iniziale si è seguito l'Hackfest 3 [65] dove è stato fornito un ambiente pre-configurato per testare TeraFlow SDN. Il lavoro è iniziato con l'installazione della macchina virtuale [66] creata appositamente per il congresso.
Di seguito vengono riportate le specifiche della VM utilizzata:

- IP Address: 10.0.2.X/24 (DHCP)
- Gateway: 10.0.2.1
- DNS: 8.8.8.8, 8.8.4.4
- Creata con VirtualBox 6.1 ma compatibile con versioni successive
- Requisiti minimi: 4 vCPU, 6 GB di RAM, 50 GB di spazio sul disco, Virtual Disk Image (VDI)
- Connessione di rete: NAT Network con porte 22 per SSH e 80 per HTTP esposte
- Sistema operativo senza interfaccia grafica per ridurre il consumo di risorse.

La VM ha al suo interno preinstallati MicroK8s con le componenti richieste e Mininet in formato docker. La versione di TeraFlow utilizzata è la 2.1 con adattamenti specifici per l'Hackfest.
Successivamente è stata installata la VM con la versione aggiornata di TeraFlow (3.0) a causa di un'incompleta implementazione della componente di Policy che non era in grado di gestire e riconoscere le KPI. Le caratteristiche richieste sono sostanzialmente simili, con un aumento di memoria RAM a 8 GB e dello spazio su disco a 60 GB. Il sistema operativo utilizzato è Ubuntu Server 22.04.4 LTS, compatibile anche con la versione successiva

22.04.6 LTS. Inoltre è richiesta l'installazione di MicroK8 v1.24.17 cone le componenti necessarie, Docker e la versione 3.9.18 di Python.
I vari esperimenti si sono svolti seguendo il seguente schema: attraverso il Service Level è stato introdotto un servizio nella rete sotto forma di intento per stabilire la connessione e il percorso. Successivamente, tramite il Management Level, è stata inserita una politica basata sugli eventi che consente di associare un Service Level Agreement (SLA) a un servizio specifico. Questo SLA include condizioni che devono essere monitorate e rispettate durante l'esecuzione del servizio.
Per iniziare il lavoro si è partiti da una demo già preesistente, apportando in seguito le modifiche necessarie.

4.1 Strumenti per la sperimentazione

INTRO
li descrivo perche verranno usati switch p4 emulati in mininet

4.1.1 Mininet

Mininet [67] è un sistema open source di orchestrazione per l'emulazione di reti su un unico ambiente Linux che permette di emulare un'intera rete su un singolo computer. E' ampiamente utilizzato in ambiti di ricerca e sviluppo per creare e testare reti virtuali in modo realistico. A differenza di altri emulatori che utilizzano macchine virtuali per ogni dispositivo, si distingue per la sua capacità di avviare rapidamente reti virtuali complesse e di eseguire test su scenari vari e personalizzati. Consente anche di configurare l'inoltro dei pacchetti per testare diverse funzionalità, facilitando la condivisione e la replica del codice.
Mininet offre delle API e un interprete Python che consentono di definire e gestire facilmente delle topologie di rete. È inoltre possibile utilizzare un'interfaccia a riga di comando (CLI) per la stessa funzione. In entrambi i casi, si possono configurare topologie predefinite o personalizzate, aggiungendo e rimuovendo switch, router, host, controller e link, tutti eseguiti su un unico computer.

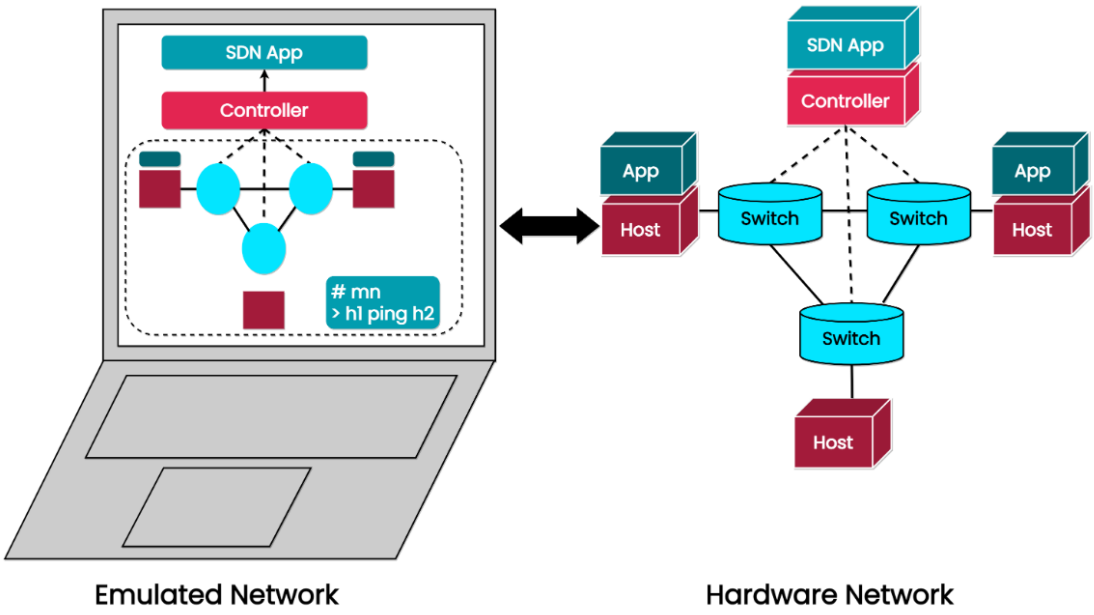

Figura 4.1: Rete mininet

Mininet è in grado di gestire un insieme di terminali di rete (host), utilizzando la virtualizzazione leggera attraverso tecnologie implementate nel kernel Linux, come i network namespaces. Questi permettono di creare istanze separate di interfacce di rete, tabelle di routing e tabelle ARP, che operano in modo indipendente [68]. Questo approccio consente di avviare numerosi host e switch (fino a 4096) su un singolo kernel del sistema operativo, simulando una rete completa su un'unica macchina [69]. 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 [70]:

- **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.

Si possono configurare i link come up o down e inserire metriche specifiche come quelle di banda, ritardo, perdita o massima lunghezza della coda di recezione per rendere la rete più realistica e adatta a esperimenti di test.
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 tutti 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.

Alcuni comandi fondamentali

Linea di comando

Inanzitutto è fondamentale creare una topologia di rete con il seguente comando[71]:
$ 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

4.1.2 P4

P4 [73], 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

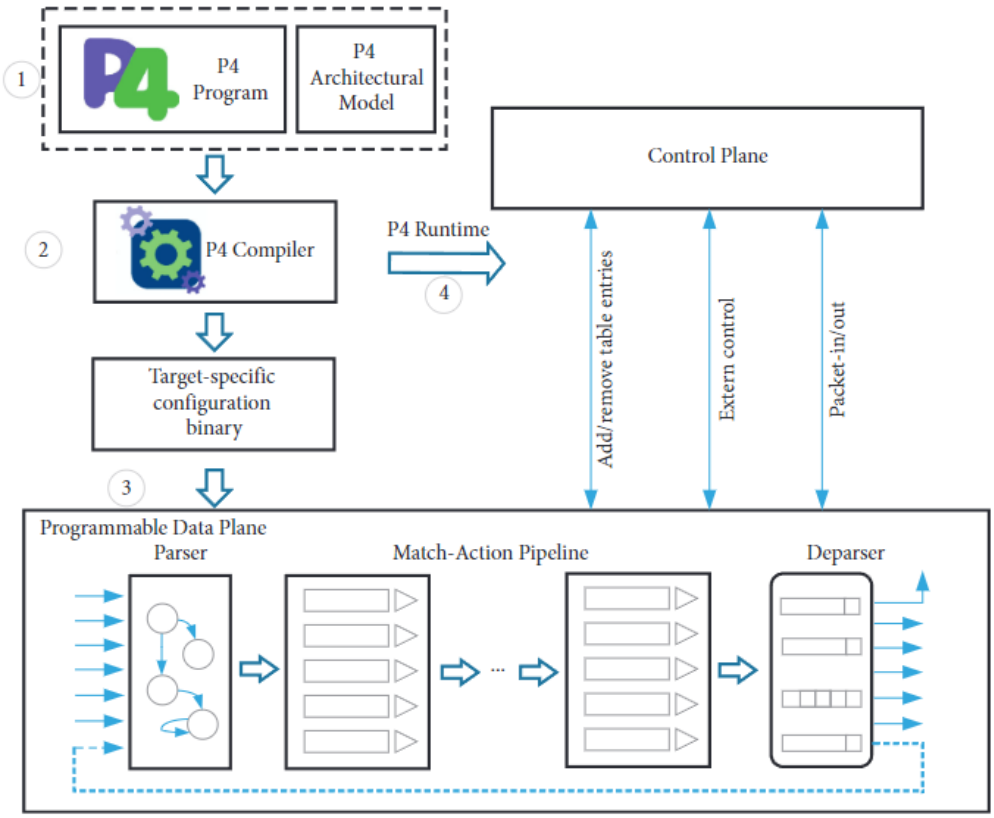

Figura 4.2: Workflow di P4 sul piano dati [72]

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 sincronizzati. 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 4.2.
Un programma P4 è costituito da diverse sezioni [72], ognuna delle quali descrive un aspetto specifico del trattamento dei pacchetti. La dichiarazione degli header permette di specificare i protocolli che si vogliono analizzare e riconoscere o inventarne di nuovi 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 le informazioni 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 con-

sentendo addirittura di definire nuovi protocolli o modificare quelli esistenti

- **Target-indipendenza**: Il codice può essere compilato per una varietà 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.

Servizio end-to-end

Come già detto in precedenza il Device Level sfrutta una South-Bound Interface (SBI) per interagire con i device tramite l'API P4Runtime. Inizialmente, il codice p4 compilato, ossia i vari artefatti, viene copiato nel pod SBI per poter inserire le giuste configurazioni nelle tabelle dei dispositivi. Il passo successivo è registrare i dispositivi e i link al controller SDN per permettere una corretta comunicazione tra di essi. A questo punto, siamo in grado di richiedere una connessione tra due end points specificando solamente i dispositivi finali tramite un servizio.

La creazione del servizio è realizzata in due passi. Inizialmente, tramite la funzione *CreateService*, si crea una servizio di connettività vuoto nel quale viene specificato solo il tipo associato per poi ritornare l'identificativo a cui è stato correlato. Successivamente viene aggiornato il servizio popolando i campi richiesti come endpoints, vincoli e configurazioni di servizio. Di questo si occupa la funzione *UpdateService* [57]. La componente di Service a questo punto si rivolge prima alla componente di Context per recuperare la versione più aggiornata del servizio e settare lo stato a Planned (pianificato), poi alla PathComp per calcolare un percorso.

Per eseguire questa operazione la PathComp utilizza delle informazioni della rete che risiedono nel database logicamente centralizzato della componente di Context. La PathComp è in grado di soddisfare anche richieste di servizio che attraversano più livelli, in questo

caso, la risposta includerà uno o più sottoservizi con le relative sottoconnessioni che li supportano.

Una volta ricevute le connessioni ed eventuali sottoservizi la componente Service istanzia un *Task Scheduler* con le relative funzioni, quest'ultimo è responsabile dell'esecuzione delle attività di installazione e smantellamento dei servizi e collegamenti nell'ordine appropriato. Infine viene eseguito il metodo *Execute* del *Task Scheduler* per realizzare tutte le operazioni di configurazione richieste per i dispositivi lungo il percorso tramite l'SBI; inoltre viene modificato lo stato del servizio. Al termine del processo, la componente Context viene aggiornata con le nuove informazioni e l'identificatore del servizio viene restituito all'entità chiamante.

Per mantenere questo processo agnostico rispetto ai dettagli della tecnologia, la componente di Service sfrutta una definizione minima permettendo agli utenti di esprimere cosa vogliono connettere, lasciando che sia il sistema sottostante a decidere come realizzare la connessione reale. La componente traduce automaticamente questa definizione minimale del servizio in modelli di configurazioni astratte dei dispositivi. Queste vengono a loro volta tradotte in regole P4 dal driver del dispositivo P4 della SBI.

Vincoli e configurazioni di servizio

È possibile richiedere azioni supplementari, come l'aggiunta di vincoli o configurazioni di servizio specifiche. Queste vengono specificate inizialmente alla creazione di un servizio e devono essere rispettate finchè quest'ultimo non verrà eliminato. Ciò semplifica la gestione dei servizi, in quanto la dichiarazione di queste informazioni aggiuntive può sostituire l'associazione di una politica.

Nella demo presa in considerazione i vincoli non erano specificati ma sono stati aggiunti per quanto riguarda la latenza e la capacità del percorso 4.3. Questa aggiunta non ha prodotto cambiamenti nella definizione del percorso poiché le funzioni relative all'effettivo funzionamento di queste funzionalità non sono ancora state implementate nel controller ma sono solo pianificate per release future. Altri esempi di vincoli possono essere rappresentati dalla posizione di un dispositivo, che può essere sia terminale che non, dal tempo o dal numero di passi massimo di un determinato percorso.

Politica

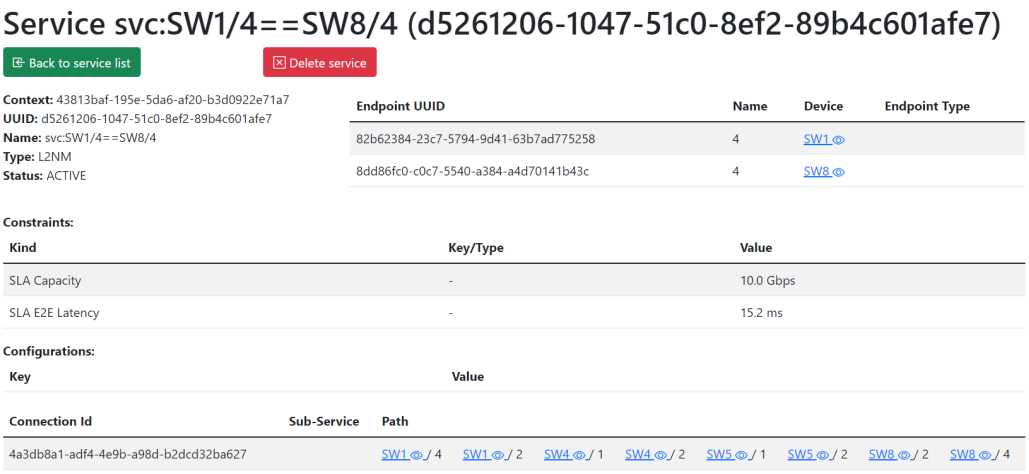

Figura 4.3: Servizio creato in fase di sperimentazione

Una parte fondamentale nei sistemi moderni è la gestione a run-time del servizio stabilito[74]. A tale scopo si sfrutta la componente di Monitoring che permette di associare il monitoraggio delle metriche nel proprio database con condizioni che devono essere rispettate. Quando questi requisiti non vengono soddisfatti, la componente di Monitoring solleva un allarme che fa scattare l'azione prestabilita.

Nella sperimentazione abbiamo introdotto una politica per il servizio stabilito con tre condizioni: una per la latenza (che non deve superare i 100ms), una per la loss (che non deve superare il 5%) e infine una per la capacità4.4. Queste tre condizioni sono legate tra loro tramite un "OR", quindi appena una di esse non è più rispettata viene invocata l'azione che in questo caso consiste nel ricalcolo del percorso.

4.2 Demo

Per verificare il comportamento del controller e assicurarsi che le condizioni di latenza, di loss e di capacità vengano rispettate, non avendo a disposizione una rete reale, abbiamo utilizzato una topologia di rete su Mininet di switch P4 basati su bmv2[75]. La topologia iniziale era composta da 4 switch e due possibili percorsi ed è stata poi ampliata a 8 switch con 5 percorsi per renderla più complessa e poter fare una sperimentazione più realistica.

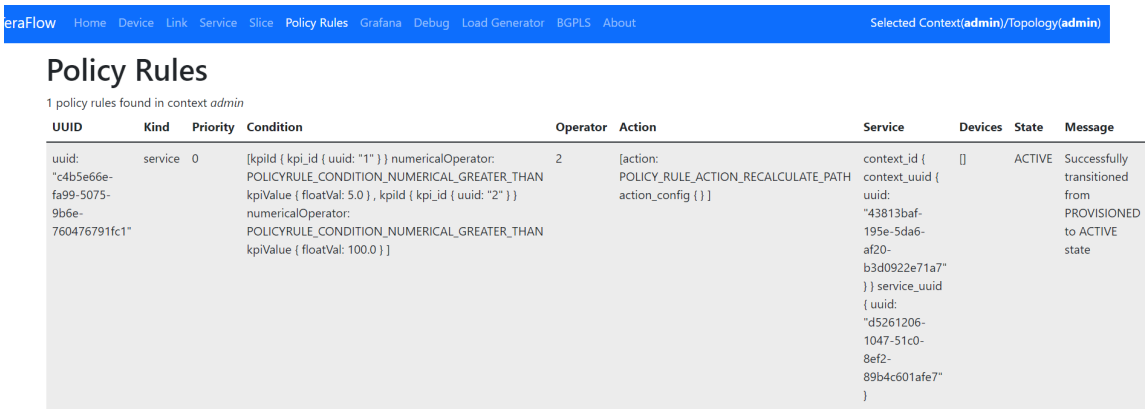

Figura 4.4: Politica creata in fase di sperimentazione

L'ultima topologia utilizzata è stata Abilene [76]; una rete di trasporto creata da Internet2. I dispositivi delle varie topologie sono stati connessi alla componente SBI, come per i dispositivi reali, in modo tale che potessero comunicare con il controller. Successivamente per dimostrare la politica basata sul servizio e il ricalcolo delle configurazioni, abbiamo aggiunto artificialmente delle condizioni di ritardo e packet loss all'interno di uno switch presente nel percorso. Ad esempio, abbiamo utilizzato il comando *switch2 tc qdisc add dev switch2-eth2 root netem delay 200ms* per aggiungere un ritardo di 200ms oppure il comando *switch2 tc qdisc add dev switch2-eth2 root netem loss 10%* per simulare una perdita di pacchetti del 10%. Inoltre, abbiamo disattivato un link tra due switch con il comando *link switch6 switch7 down*.

Questi cambiamenti sono stati monitorati grazie a un probe in Python, una funzionalità che rileva lo stato di integrità delle istanze dell'applicazione. Inizialmente, si avvia l'agent che, ascoltando gli eventi della componente di Context, crea tre Kpi Id differenti per loss, latenza e capacità ogni volta che viene creato un servizio 4.5. Successivamente si attiva un secondo script che riesce a monitorare le metriche di loss e latenza attraverso il comando ping del terminale e la capacità tramite iperf per poi inviarle attraverso una socket all'agent. Prima di avviare lo script è necessario attivare il server iperf sul dispositivo finale del servizio.

L'agent, ottenuti i relativi valori, li inoltra alla componente di monitoring tramite una KPI

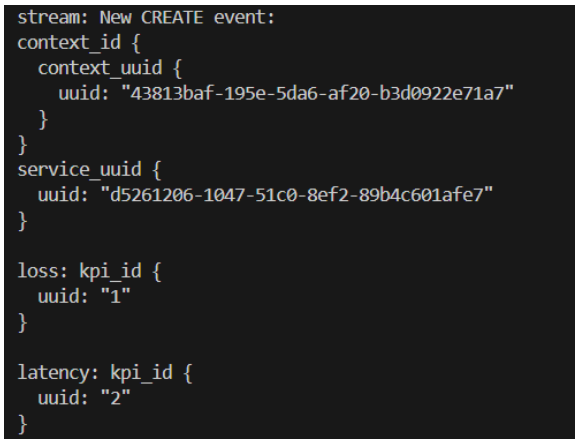

Figura 4.5: Creazione delle due KPI id

composta dal valore, il kpi id corretto e un timestamp.

La componente, ricevute le metriche, riconosce che è avvenuto un cambiamento nella rete e solleva un allarme per una potenziale violazione della politica che sarà poi validata dalla componente di Policy. Questo evento causa l'esecuzione dell'azione predefinita nella politica, ovvero l'aggiornamento del percorso. Quando il nuovo tragittto è calcolato dalla Path Comp, la componente di Service compila una lista di configurazioni di dispositivi per validare gli aggiornamenti seguita da un'altra lista per la cancellazione delle vecchie configurazioni. Queste istruzioni sono tradotte in regole P4 dalla componente SBI prima di essere imposte al piano dati tramite il P4Runtime.

In tutte e tre le tipologie di alterazione della rete, si è riscontrato un cambiamento del percorso volto a rispettare le condizioni desiderate.

4.3 Esperimento 2

8 sw, latenza pkt loss cambio tolopogia, aggiunta link, esteso probe, aggiunta porte per mininet

4.4 Esperimento 3

cambio topologia, nuovo script mininet (preso da internet con cambio bw), aggiunte porte, aggiunto iperf, aggiunta capacità

5. Conclusioni

All'interno di questo elaborato sono state inizialmente descritte le motivazioni che stanno spingendo la ricerca di nuove tecnologie per superare le limitazioni dell'infrastruttura di rete tradizionale. Abbiamo presentato il paradigma SDN come uno dei più promettenti per conseguire tale obiettivo e sono state discusse le implementazioni più popolari per i controller SDN. Ci siamo concentrati sull'architettura e sulle funzionalità di TeraFlow, un controller SDN emergente progettato per le reti B5G, e abbiamo analizzato i suoi meccanismi di creazione e gestione dei servizi di rete. La sperimentazione si è focalizzata sulla creazione di servizi di connettività tra due end points, l'introduzione di politiche di rete specifiche, e la verifica della risposta del sistema a condizioni introdotte artificialmente. I test sono stati resi possibili dall'emulatore di rete Mininet. La valutazione è stata fatta su prove di raggiungibilità e di performance che, pur essendo abbastanza semplici, rappresentano un punto di partenza per analisi più complesse ed esegubili su reti più ampie.
In una rete reale, tuttavia, andrebbero svolti ulteriori accertamenti per garantire le stesse performance.
Durante la sperimentazione si sono riscontrate delle problematiche dovute alla mancanza di implementazione di alcune funzioni che hanno limitato lo sviluppo completo delle funzionalità desiderate.
Nonostante queste limitazioni, l'approccio sperimentale adottato ha permesso di ottenere risultati significativi, evidenziando la capacità del sistema di rispondere a condizioni di rete variabili e di applicare politiche di gestione del traffico in modo efficace.
Per lavori futuri si potrebbe pensare di implementare il servizio tramite interfaccia Web. Attualmente la creazione dei servizi tramite WebUI non supporta il tipo L2MV utilizzato in questa sperimetazione.
Si potrebbe inoltre sperimentare con topologie più grandi e politiche più complesse per approfondire ulteriormente le capacità e i limiti del sistema. Infine, si potrebbero verificare le stesse prove con altri protocolli e tipi di switch differenti da P4, ampliando così le possibilità di applicazione e la robustezza delle soluzioni proposte.

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 [77]. 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.

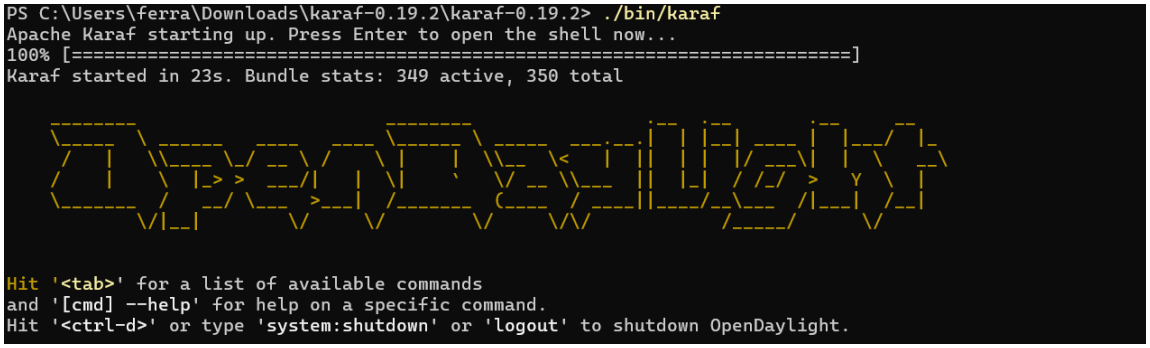

Figura 6.1: Installazione ODL

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

```
feature:list
```

Per installarle invece

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

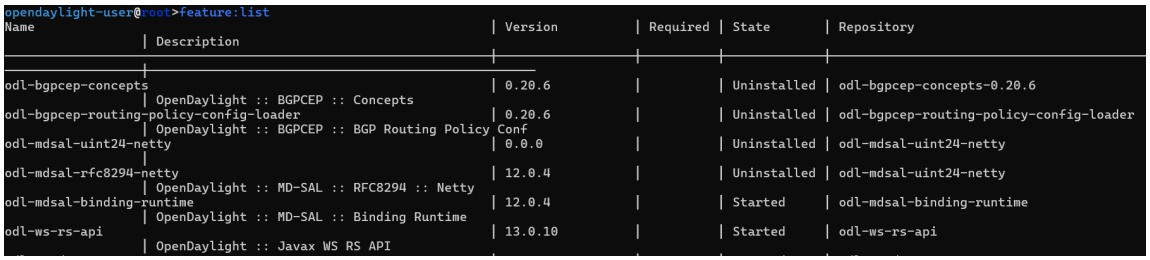

Figura 6.2: Alcune features disponibili

Bibliografia

[1] B. Martini, M. Gharbaoui, and P. Castoldi. Intent-based zero-touch service chaining layer for software-defined edge cloud networks. *Computer Networks*, 212:109034, 2022.

[2] Andrea Biancini, Mauro Campanella, Luca Prete, and Fabio Farina. Software defined networking esperienze openflow e l'interesse per cloud. 04 2013.

[3] Maede Zolanvari. Sdn for 5g. 2015.

[4] Le prospettive del 5g. *aeit*, 104(11/12):39–41, 2018. `https://www.aeit.it/aeit/edicola/aeit/aeit2018/aeit2018_06_cisa/aeit2018_06_riv.pdf`.

[5] Laura Zanotti. Intent-based networking (ibn): significato e vantaggi del networking basato sugli intenti. *ZeroUno*, 2019.

[6] Yiming Wei, Mugen Peng, and Yaqiong Liu. Intent-based networks for 6g: Insights and challenges. *Digital Communications and Networks*, 6(3):270–280, 2020.

[7] Etsi. `https://www.etsi.org/`.

[8] Onos. `https://opennetworking.org/onos/`.

[9] Opendaylight. `https://www.opendaylight.org/`.

[10] Daniel Barattini. Supporto a micro-servizi per controller ad alta scalabilità e affidabilità. Master's thesis, Alma mater studiorum - universita' di Bologna, 2020.

[11] Routing information protocol. *IBM*, 2023. `https://www.ibm.com/docs/en/i/7.3?topic=routing-information-protocol`.

[12] Muhammad Fauzan Rafi Sidiq Widjonarto. Application of dijkstra algorithm on ospf routing protocol and its effect in modern networks, 2018. 13518147.

[13] William Stallings. *Foundations of Modern Networking: SDN, NFV, QoE, IoT, and Cloud*. Pearson Education, 2015. Accessed: 6 September 2024.

[14] Diego Kreutz, Fernando Ramos, Paulo Veríssimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *ArXiv e-prints*, 103, 06 2014.

[15] Open networking foundation. `https://opennetworking.org/`.

[16] Yustus Eko Oktian, SangGon Lee, HoonJae Lee, and JunHuy Lam. Distributed sdn controller system: A survey on design choice. *Computer Networks*, 121:100–111, 2017.

[17] Nicholas Brasini. *Analisi e sviluppo di un'interfaccia web per gestire controller SDN*. PhD thesis, Alma mater studiorum - universita' di Bologna, 2017.

[18] Babak Darabinejad. An introduction to software-defined networking. *International Journal of Intelligent Information Systems*, 3:71, 11 2014.

[19] Diego Kreutz, Fernando M. V. Ramos, Paulo Esteves Veríssimo, Christian Esteve Rothenberg, Siamak Azodolmolky, and Steve Uhlig. Software-defined networking: A comprehensive survey. *Proceedings of the IEEE*, 103(1):14–76, 2015.

[20] B. Martini, M. Gharbaoui, and P. Castoldi. Intent-based zero-touch service chaining layer for software-defined edge cloud networks. *Computer Networks*, 212:109034, 2022.

[21] *MEC*. url: `https://www.etsi.org/technologies/multi-access-edge-computing`.

[22] Nfv. `https://www.etsi.org/technologies/nfv`.

[23] Ietf. `https://www.ietf.org/`.

[24] Hao Yu, Hamid Rahimi, Carl Janz, et al. Building a comprehensive intent-based networking framework: A practical approach from design concepts to implementation. *Journal of Network and Systems Management*, 32:47, 2024.

[25] Nathan Sousa, Nazrul Islam, Danny Perez, and Christian Esteve Rothenberg. Policy-driven network traffic rerouting through intent-based control loops. 07 2019.

[26] B. Martini, M. Gharbaoui, and P. Castoldi. Intent-based network slicing for sdn vertical services with assurance: Context, design and preliminary experiments. *Future Generation Computer Systems*, 142:101–116, 2023.

[27] Bhargavi Goswami. Experimenting with onos scalability on software defined network. *Journal of Advanced Research in Dynamical and Control Systems*, 10:1820–1830, 01 2019.

[28] Osgi. `https://www.osgi.org/`.

[29] Apache karaf. `https://kafka.apache.org/`.

[30] Ashwin Rajaratnam, Ruturaj Kadikar, Shanthi Prince, and M. Valarmathi. Software defined networks: Comparative analysis of topologies with onos. In *2017 International Conference on Wireless Communications, Signal Processing and Networking (WiSPNET)*, pages 1377–1381, 2017.

[31] Putri Monika, Ridha Negara, and Danu Sanjoyo. Performance analysis of software defined network using intent monitor and reroute method on onos controller. *Bulletin of Electrical Engineering and Informatics*, 9(5):2065–2073, 2020.

[32] Davide Sanvito, Daniele Moro, Mattia Gullì, Ilario Filippini, Antonio Capone, and Andrea Campanella. Onos intent monitor and reroute service: enabling plug&play routing logic. *2018 4th IEEE Conference on Network Softwarization and Workshops (NetSoft)*, pages 272–276, 2018.

[33] Lfnetworking. `https://lfnetworking.org/`.

[34] Gonzalez Carlos Javier. *Management of a heterogeneous distributed architecture with the SDN*. PhD thesis, 12 2017.

[35] Giacomo Ondesca. *Monitoraggio di rete con tecnologie SDN (Software-Defined Networking)*. PhD thesis, Politecnico di Torino, 2021.

[36] Benedetta Contigiani. *Improving network management with Software-Defined Networking*. PhD thesis, Università degli studi di Camerino, 2016.

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

[38] Confluence. Network intent composition proposal, 2021.

[39] Kubernetes. hhttps://kubernetes.io/.

[40] *CNCF*. url: https://www.cncf.io/.

[41] T Kubernetes. Kubernetes. *Kubernetes. Retrieved May*, 24:2019, 2019.

[42] Brendan Burns, Joe Beda, Kelsey Hightower, and Lachlan Evenson. *Kubernetes: up and running*. " O'Reilly Media, Inc.", 2022.

[43] *Indagine statistica sull'uso dei microservizi*. url: https://www.statista.com/statistics/1233937/microservices-adoption-level-organization/.

[44] Xabier Larrucea, Izaskun Santamaria, Ricardo Colomo-Palacios, and Christof Ebert. Microservices. *IEEE Software*, 35(3):96–100, 2018.

[45] Zhijun Ding, Song Wang, and Changjun Jiang. Kubernetes-oriented microservice placement with dynamic resource allocation. *IEEE Transactions on Cloud Computing*, 11(2):1777–1793, 2023.

[46] Alberto Mozo, Amit Karamchandani, Luis Cal, Sandra Gómez-Canaval, Antonio Pastor, and Lluis Gifre. A machine-learning-based cyberattack detector for a cloud-based sdn controller. *Applied Sciences*, 13:4914, 04 2023.

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

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

[49] TeraFlow. *5GPPP*, 2021. url: https://5g-ppp.eu/teraflow/.

[50] Daniel Adanza, Lluis Gifre, Pol Alemany, Juan-Pedro Fernández-Palacios, Oscar González de Dios, Raul Muñoz, and Ricard Vilalta. Enabling traffic forecasting with cloud-native sdn controller in transport networks. *Computer Networks*, 250:110565, 2024.

[51] D5.3 final demonstrators and evaluation report. *teraflow-h2020.eu*, 2023.

[52] D1.4: Final project periodic report. *teraflow-h2020.eu*, 2023.

[53] D3.1: Preliminary evaluation of life-cycle automation and high performance sdn components. *teraflow-h2020.eu*, 2022.

[54] D2.2 - preliminary requirements, architecture design, techno-economic studies and data models. *teraflow-h2020.eu*, 2023.

[55] Teraflowsdn core components. https://www.teraflow-h2020.eu/blog/teraflowsdn-core-components, 2021.

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

[57] D3.2: Final evaluation of life-cycle automation and high performance sdn components. *teraflow-h2020.eu*, 2023.

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

[59] K. Indrasiri and D. Kuruppu. *gRPC: Up and Running: Building Cloud Native Applications with Go and Java for Docker and Kubernetes*. O'Reilly Media, 2020.

[60] Jong-Kun Lee. A group management system analysis of grpc protocol for distributed network management systems. In *SMC'98 Conference Proceedings. 1998 IEEE International Conference on Systems, Man, and Cybernetics (Cat. No.98CH36218)*, volume 3, pages 2507–2512 vol.3, 1998.

[61] grpc. https://grpc.io/.

[62] F. Paolucci, A. Sgambelluri, M. Dallaglio, F. Cugini, and P. Castoldi. Demonstration of grpc telemetry for soft failure detection in elastic optical networks. In *2017 European Conference on Optical Communication (ECOC)*, pages 1–3, 2017.

[63] Ricard Vilalta, Noboru Yoshikane, Ramon Casellas, Ricardo Martínez, Shohei Beppu, Daiki Soma, Seiya Sumita, Takehiro Tsuritani, Itsuro Morita, and Raul Muñoz. Grpc-based sdn control and telemetry for soft-failure detection of spectral/spacial superchannels. In *45th European Conference on Optical Communication (ECOC 2019)*, pages 1–4, 2019.

[64] Hari Krishna and Rinki Sharma. Comparative study of orchestration using grpc api and rest api in server creation time: An openstack case. *International Journal of Computer Networks & Communications (IJCNC)*, 16(1), jan 2024. Available at SSRN: `https://ssrn.com/abstract=4710302`.

[65] Hackfest 3. `https://labs.etsi.org/rep/groups/tfs/-/wikis/TFS-HACKFEST-3`.

[66] Link alla macchina virtuale per l'hackfest 3. `https://drive.google.com/file/d/1OaukXmAC1uaeIAChkEpvBB9mSkUHimsR/view`.

[67] Mininet. `https://mininet.org/`.

[68] Francesco Cristiano. *Emulazione distribuita di reti di telecomunicazioni su piattaforma Mininte*. PhD thesis, Alma Mater Studiorum- Università di Bologna, 2013.

[69] Mininet overview. `https://mininet.org/overview/`.

[70] Nikhil Handigol Bob Lantz and Vimal Jeyakumar Brandon Heller. *Introduction to Mininet*. url: `https://github.com/mininet/mininet/wiki/Introduction-to-Mininet`.

[71] Mininet.org. *Mininet Walkthrough*. url: `https://mininet.org/walkthrough/`.

[72] Ya Gao and Zhenling Wang. A review of p4 programmable data planes for network security. *Mobile Information Systems*, 2021(1):1257046, 2021.

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

[74] Panagiotis Famelis, Georgios P. Katsikas, Vasilios Katopodis, Carlos Natalino, Lluis Gifre Renom, Ricardo Martinez, Ricard Vilalta, Dimitrios Klonidis, Paolo Monti, Daniel King, and Adrian Farrel. P5: Event-driven policy framework for p4-based traffic engineering. In *2023 IEEE 24th International Conference on High Performance Switching and Routing (HPSR)*, pages 1–3, 2023.

[75] P4 Language. *Behavioural model (bmv2) reference P4 software switch*, March 2023. Available: https://github.com/p4lang/behavioral-model.

[76] Wikipedia contributors. Abilene network — Wikipedia, the free encyclopedia. https://en.wikipedia.org/w/index.php?title=Abilene_Network&oldid=1220796057, 2024. [Online; accessed 17-August-2024].

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