Alma Mater Studiorum · Università di Bologna
Campus di Cesena

SCUOLA DI SCIENZE
Corso di Laurea in Ingegneria e Scienze informatiche

Named Data Networking for Computing in the Mobile Edge

Relazione finale in:

PROGRAMMAZIONE DI RETI

Relatore:
Prof. GIOVANNI PAU

Presentata da:
SIMONE PERACCINI

Seconda Sessione di Laurea
Anno Accademico: 2017-2018

PAROLE CHIAVE

Named Data Networking
Nomi
Interest
Data
ndnSIM
Edge computing

Ai miei genitori, Nadia e Giorgio,che mi hanno mostrato il valore del sacrificio e sono sempre stati pronti a sostenermi nelle difficoltà.

Indice

Introduzione

Internet nasce per applicazioni molto differenti da quelle attuali. Inizialmente, il suo scopo era quello di garantire comunicazioni stabili ed efficienti tra le sedi delle forze armate statunitensi oltre che tra le università e i centri di ricerca militare. In principio, sicuramente, nessuno si sarebbe mai immaginato l'odierna espansione di questa tecnologia, mentre oggi essa è uno strumento di cui praticamente nessuno può più fare a meno. Con gli anni, per permettere di superare i limiti progettuali e per renderlo un servizio fruibile a livello globale, gli esperti di reti sono stati costretti a trovare soluzioni innovative, che potessero integrarsi all'architettura di base. Un esempio è il Domain Name System(DNS) utilizzato per la risoluzione di nomi e che permette di associare a ciascun indirizzo un riferimento in formato testuale.

Oggigiorno l'Internet Protocol(IP), nonostante fosse stato progettato per conversazioni end-to-end, viene utilizzato ben oltre alle sue reali capacità. Proprio come fu per il sistema telefonico, internet risulta oggi un veicolo povero in confronto alla vasta gamma di contenuti che trasmette. Il motivo principale, sta proprio in un inadeguato approccio architetturale. Inoltre, le violazioni di sicurezza e gli attacchi informatici, attratti dal tremendo valore economico delle applicazioni, sono sempre più all'ordine del giorno. Nonostante i diversi tentativi di rendere sicuri i canali di comunicazione, la sicurezza delle reti rappresenta ancora oggi un tema molto delicato.

Nel 2010 il National Science Fondation(NSF) fonda quattro progetti intenti a trovare un paradigma che risolva i problemi della rete IP e che possa soddisfare le esigenze dell'odierna rete internet. L'idea è quella di creare una nuova concezione che porti alla conclusione del progetto Future Internet Architecture. Named Data Networking (NDN) è uno di questi, ed intende partire dal superamento dell'approccio Client-Server. Attualmente la rete richiede una continua creazione di contenuti e secondo i ricercatori sarà difficile continuare gestire un elevato numero di connessioni end-to-end che garantiscano sicurezza e allo stesso tempo buone prestazioni. Per questo NDN vuole distogliere l'attenzione dal "dove" trovare una determinata risorsa e concentrarsi più su "cosa" applicazioni ed utenti cercano. NDN vede come punto di partenza il superamento del concetto di indirizzo IP. In quest'ottica ,il progetto vuol far si che ogni risorsa di rete, dagli host ai dati, sia identificata da un nome. Gli host seguono innovative tecniche di forwarding e di caching, mentre la sicurezza viene garantita attraverso la firma dei dati.

Le caratteristiche di questo approccio, aprono scenari molto interessanti nel campo delle reti. Dato che ogni singolo device (dai più grandi super-computer ai più piccoli dispositivi wearable) ha la possibilità soddisfare un interesse, offre potenza di calcolo e può essere dotato di sensori, emerge la possibilità di dare vita a sistemi di calcolo parallelo, che svolgano task computazionalmente molto complessi in tempi notevolmente ridotti. Inoltre, questi aspetti si inseriscono perfettamente nello scenario del Fog ed Edge computing.

Proprio da questo scenario nasce il mio progetto di tesi: ideare un algoritmo che regoli la computazione cooperativa fra nodi appartenenti ad una rete NDN. Innanzitutto, si sono studiati i principi fondamentali del paradigma, impiegati poi nella realizzazione del sistema. Fra gli strumenti forniti da NDN si è preferito utilizzare ndnSIM, un simulatore di reti basato su NS-3. Grazie ad esso non si è incontrata alcuna problematica relativa all'hardware ed è stato possibile testare il sistema su un ampio numero di nodi.

Sommario

L'elaborato si sviluppa su quattro capitoli. Nel primo e nel secondo vengono trattati i concetti teorici che caratterizzano rispettivamente le architetture IP e NDN. Nel terzo capitolo vengono illustrati i principali strumenti forniti da NDN, concentrandosi su quelli impiegati nella realizzazione del sistema. Nel quarto vengono osservati i dettagli del protocollo mentre nel quinto si mostrano il design e i test.

- **Capitolo 1:** si osservano i concetti fondamentali dell'architettura e IP di cui si analizzano anche aspetti di particolare dettaglio. Si mostra il modello da cui parte il lavoro dei ricercatori del progetto Named Data Networking;
- **Capitolo 2:** si descrive dettagliatamente il paradigma architetturale Named Data Networking concentrandosi particolarmente su come vengano affrontate le funzionalità già fornite dal protocollo IP e su come ne vengano risolte determinate problematiche;
- **Capitolo 3:** si presentano le componenti software dell'architettura di NDN che vengono racchiuse nel Network Forwarding Deamon(NFD). Verranno inoltre illustrati gli strumenti forniti da ndnSIM e da ns-3, mostrando come essi integrino NFD.
- **Capitolo 4:** si mostrano i concetti di Fog e Edge computing spiegando quali aspetti del Cloud vadano a risolvere. Successivamente, vengono illustrati i dettagli del protocollo ideato, servendosi di diagrammi UML.
- **Capitolo 5:** si osservano le fasi di progettazione e sviluppo del prototipo installato sui nodi del simulatore. Dai grafici ricavati in fase di test, si conclude con alcune osservazioni riguardanti il comportamento del sistema.

Capitolo 1

Internet Protocol

1.1 IP e Trasmission Control Protocol

L'attuale suite di protocolli su cui è costruita la rete internet è TCP/IP che a sua volta si basa sul modello teorico ISO/OSI. L'immagine [1.1] mostra le loro strutture. Essa è definita da una pila protocollare a quattro livelli: Applicazione, Trasporto, Rete e Accesso alla rete. Il nome di quest'insieme di protocolli fu dato in funzione di Trasmission Controll Protocol (TCP) e di Internet Protocol (IP) che sono i due protocolli più importanti in esso definiti. Nelle prossime sezioni, andremmo ad osservare dettagliatamente il compito del protocollo IP che ricopre il ruolo di protocollo di rete.

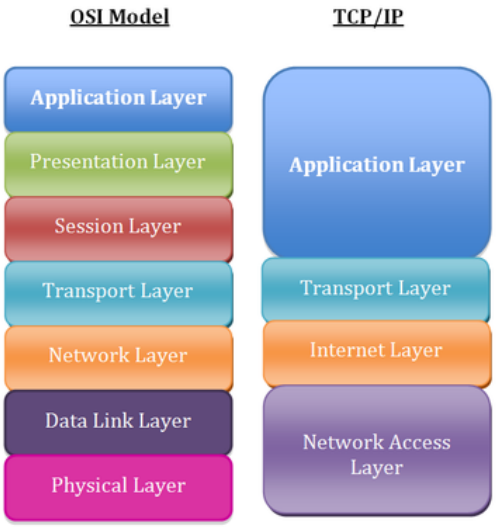

Figura 1.1: Stack protocollari del modello ISO/OSI e TCP/IP.

1.2 Funzionalità

Il protocollo IP ha il compito di definire in maniera formale quali modalità di interazione due sotto-reti debbano rispettare per far si che i dispositivi ad esse appartenenti possano comunicare fra loro. Esso è studiato per operare a commutazione di pacchetto in modalità connectionless. Le principali funzionalità, trattate dall'RFC 791 [7], sono le seguenti:

- **Indirizzamento:** definisce un modello di identificazione per distinguere i dispositivi connessi alla rete. Per fare ciò, IP definisce un indirizzo numerico chiamato indirizzo-IP. Esso viene utilizzato per l'indirizzamento fra sotto-reti differenti mentre a livello locale viene utilizzato l'indirizzamento fisico (MAC). A ciascun terminale, che intende connettersi alla rete Internet, viene assegnato un indirizzo-IP per mezzo di un protocollo di accesso (RARP, BOOTP o DHCP);

- **Instradamento:** è l'insieme di procedure che permette di individuare il percorso più adeguato alla interconnessione fra host appartenenti a due reti eterogenee. Sostanzialmente, consiste nel trovare il percorso che i pacchetti IP devono seguire per andare dalla rete sorgente fino alla rete di destinazione. Il percorso viene costruito step-by-step da ogni nodo verso cui è instradato il pacchetto fino al raggiungimento della destinazione.

Essendo IP un protocollo basato su pacchetti ne va a definire il Protocol Data Unit (PDU o IP header), il datagramma che viene incorporato al pacchetto ricevuto dal livello di trasporto. Come viene raffigurato nell'immagine [1.2] ogni livello della pila protocollare va ad incapsulare delle informazioni riguardanti il proprio strato; esse vengono interpretate dai dispositivi di rete che operano al rispettivo livello. Se necessario ,una volta entrati in rete, i pacchetti possono essere frammentati e riassemblati.

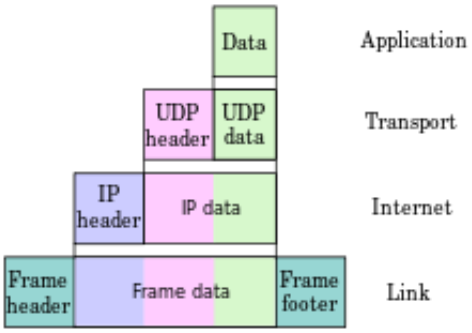

Figura 1.2: Incapsulazione dei dati nei livelli dello stack protocollare.

1.3 Indirizzamento

1.3.1 Indirizzi IP

Un indirizzo Ipv4 corrisponde ad una sequenza di 32 bit, che per semplicità vengono raggruppati in quattro gruppi da otto bit ciascuno, a ciascun gruppo corrisponde un numero decimale compreso tra 0 e 255. Convenzionalmente questi numeri, nella rappresentazione dotted-decimal, vengono separati da un punto. Ad esempio,l'indirizzo 172.16.254.1 corrisponde, in binario, a:

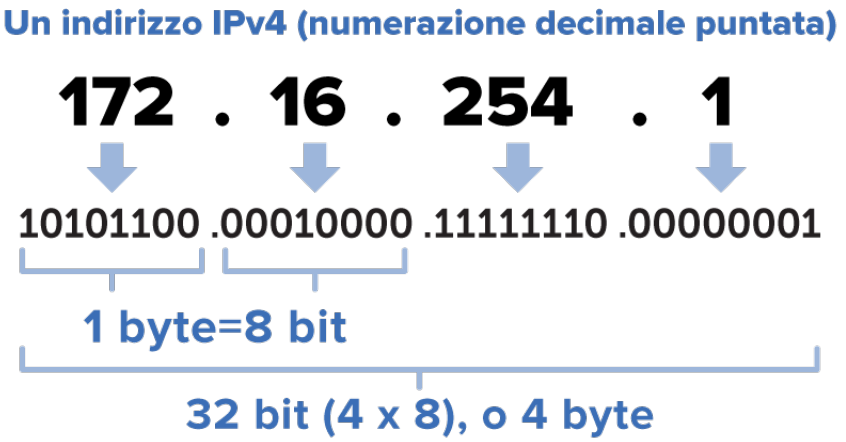

Figura 1.3: Indirizzo IP in formato decimale puntato.

Molto spesso l'indirizzo IP viene associato ad un host individuale, mentre è più corretto consideralo come identificatore di ciascun punto di interconnessione con la rete. Vengono detti multi-homed gli host che presentano pù interfacce di rete. Ogni indirizzo è logicamente suddiviso in due parti:

- **Network ID:** è il prefisso che identifica la rete a cui appartiene l'indirizzo, per questo gli host che appartenenti ad una medesima rete hanno il medesimo Network ID. Esso occupa la parte destra dell'indirizzo.;
- **Host ID:** identifica specifica interfaccia all'interno di una certa rete. Esso occupa la parte sinistra dell'indirizzo.

1.3.2 Netmask

La Netmask è una maschera di trentadue bit, in cui quelli appartenenti al Network ID hanno valore 1 e i restanti bdell'Host-ID hanno valore 0.

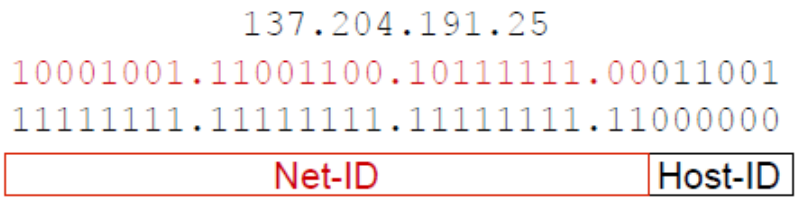

Figura 1.4: Divisione fra Network ID e Host ID.

Essa può essere rappresentata in tre modalità:

11111111.11111111.11111111.11000000 = 255.255.255.192 **Decimale**

11111111.11111111.11111111.11000000 = FF.FF.FF.C0 **Esadecimale**

11111111.11111111.11111111.11000000 = /26 **Abbreviata**

L'impiego di questa maschera verrà trattato nelle prossime sezioni in cui ci occuperemo dell'instradamento.

1.3.3 Indirizzi privati e riservati

Internet prevede che vi possano no essere reti pubbliche e reti private. Le private possiedono uno spazio di indirizzamento riservato e non sono raggiungibili direttamente dalla rete pubblica. Le reti private possono comunicare tra loro se almeno uno dei loro dispositivi fa parte anche della rete pubblica. Si elencano gli spazi di indirizzamento privato:

- da 10.0.0.0 a 10.255.255.255;
- da 172.16.0.0 a 172.31.255.255;
- da 192.168.0.0 a 192.168.255.255;

Il protocollo IP assegna ad alcuni indirizzi un ruolo specifico, che viene assunto in qualsiasi rete. Questi vengono chiamati indirizzi riservati:

0.0.0.0 è utilizzato per indicare l'host corrente;

Host-ID tutto a 0 è utilizzato per indicare la rete;

0.X.Y.Z è utilizzato per indicare un host della rete corrente senza specificarne il Net-ID.

255.255.255.255 rappresenta l'indirizzo di broadcast.

127.X.Y.Z è l'indirizzo di loopback utilizzato per redirigere i datagrammi agli strati superiori dell'host corrente.

1.3.4 Subnetting e Supernetting

I due metodi che permettono al protocollo IP di organizzare gerarchicamente indirizzi e reti sono subnetting e supernetting:

Il **Subnetting** è la procedura che permette di suddividere l'amministrazione di una rete in più "sub-network" logicamente separate. Essa prevede la suddivisione dell'Host ID in due parti : la prima identifica la sotto-rete mentre la seconda i suoi host. Questo metodo favorisce l'ottimizzazione dell'impiego degli indirizzi.

Il **Supernetting** è il procedura duale al subnetting e prevede il raggruppamento di più reti con indirizzi consecutivi in una sola entry. Questa è una tecnica usata principalmente nelle tabelle di routing, dove con una sola entry viene incorporato l'indirizzamento verso più reti.

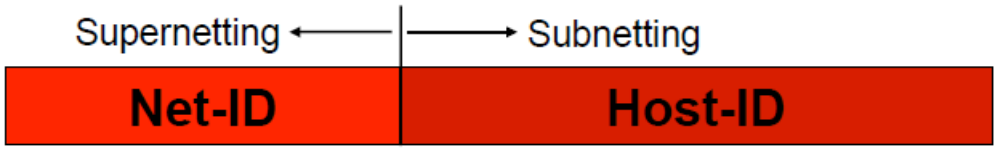

Figura 1.5: Raffigurazione del supernetting e del subnetting.

1.4 Datagramma IP

1.4.1 Formato del datagramma

L'header del pacchetto IPv4 racchiude tredici campi di cui uno opzionale chiamato Option. Tutti i campi vengono raccolti in notazione big-endian. L'immagine [1.6], tratta dal materiale didattico di programmazione di reti [2], ne mostra la struttura:

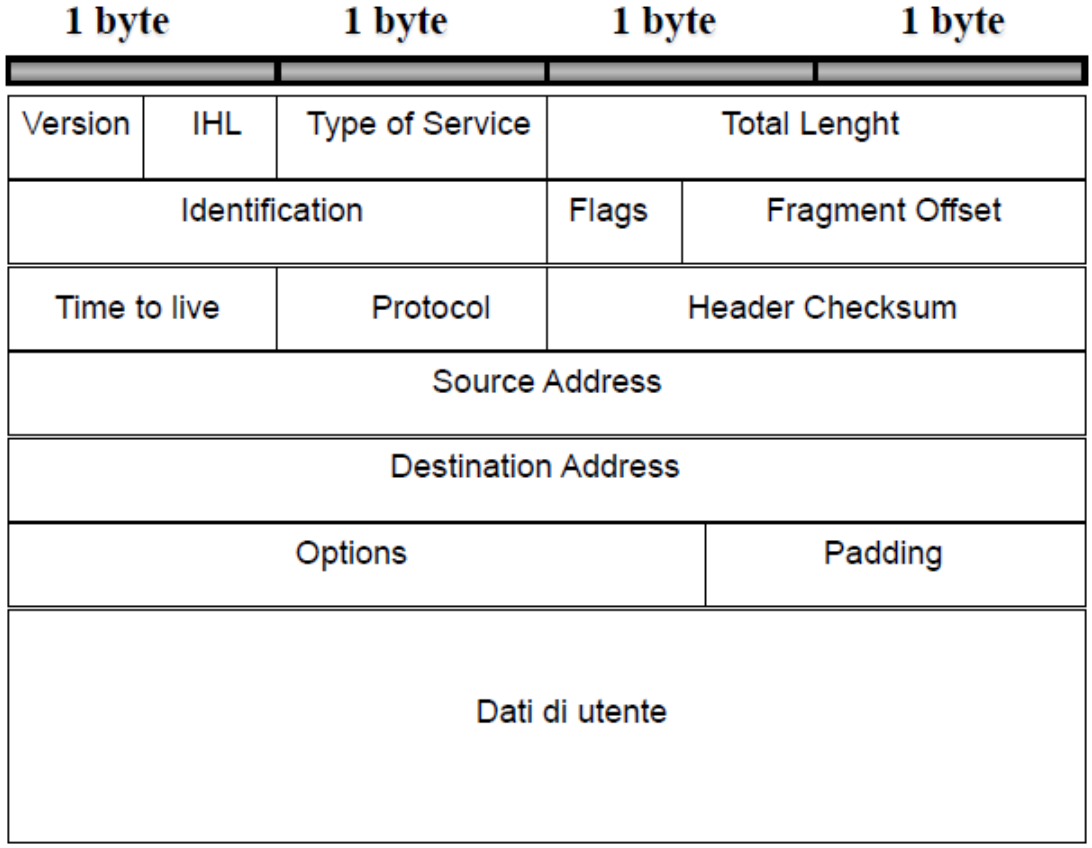

Figura 1.6: Struttura dell'header IP.

Analizziamo brevemente i campi:

- **Version:** indica il formato dell'intestazione attualmente è Ipv4;
- **IHL:** lunghezza dell'intestazione, espressa in parole di 32 bit. La lunghezza minima di questo valore è 5;
- **Type of service:** indica il tipo di servizio richiesto, usato per specificare la priorità;

- **Datagram length:** indica la lunghezza totale del datagramma espressa in bytes;
- **Identification:** valore intero che identifica univocamente il datagramma. Dato che i pacchetti ip possono essere frammentati esso indica anche a quale pacchetto corrisponde un determinato pacchetto;
- **Flag:** è una successione di tre bit, ciascuno con un significato specifico. Il bit 0 è sempre fisso a 0. Il bit 2 assume valore 0 per indicare che è consentita la frammentazione, 1 altrimenti. Il bit 3 è a 0 se il pacchetto è stato frammentato per l'ultima volta, 1 altrimenti.
- **Fragment offset:** nel caso il pacchetto venga frammentato questo campo ne indica la posizione in unità da 64 bit;
- **Time to live (TTL):**indica il numero massimo di nodi attraversabili. Il nodo sorgente attribuisce a TTL un valore superiore a 0 (solitamente si aggira da un minimo di 64 fino ad un massimo di 255) e ad ogni nodo che attraversa il datagramma esso viene posto a TTL = TTL – 1. Il primo nodo che vede TTL = 0 distrugge il datagramma;
- **Protocol:** indica a quale protocollo di livello superiore appartengono i dati del datagramma;
- **Header checksum:** è un controllo di errore della sola intestazione e viene ricalcolato da ogni nodo attraversato. Se esso non coincide con il valore che dovrebbe assumere il pacchetto viene scartato;
- **Source e Destination address:** indirizzo della sorgente e del destinatario;
- **Options:** contiene le opzioni relative al trasferimento del datagramma (registrazione del percorso, meccanismi di sicurezza). La sua lunghezza è variabile;
- **Padding:** sono bit privi di significato aggiunti per fare in modo che la lunghezza dell'intestazione sia multipla;

1.4.2 Frammentazione

Quando la dimensione massima concessa dall'interfaccia viene superata è necessatio attuare la frammentazione. La dimensiome massima è detta Maximum Trasmission Unit (MTU) e in rete internet equivale a 1500 byte. Generalmente qualunque nodo IP può frammentare un datagramma, ma generalmente nessun nodo intermedio riassembla, poiché questo compito è affidato al destinatario.

1.5 Instradamento

1.5.1 Tipologie di instradamento

L'instradamento nella rete IP può essere di due tipi:

- **Diretto;** si ha instradamento diretto quando i pacchetti attraversano una sola rete fisica, ovvero l'host sorgente e l'host destinatario fanno parte della stessa sotto-rete. Mostriamo un esempio figura [1.7].

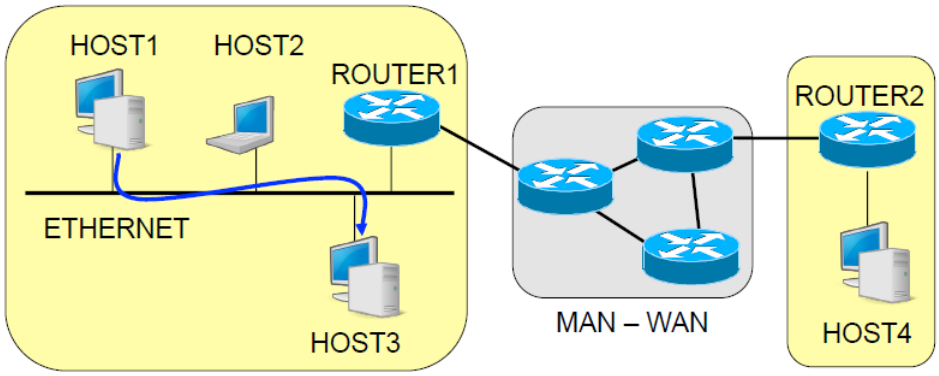

Figura 1.7: Esempio di instradamento diretto.

- **Indiretto**: si ha instradamento indiretto quando deve essere attraversata più di una rete fisica, ovvero l'host sorgente e l'host destinatario non fanno parte della stessa sottorete. Mostriamo un esempio figura [1.8].

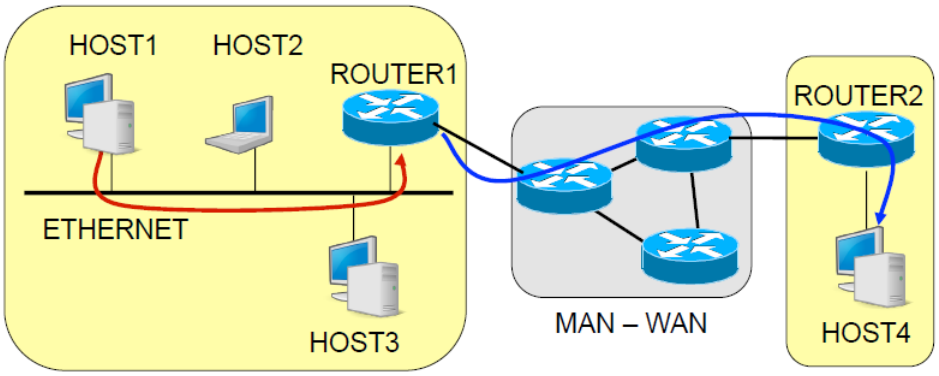

Figura 1.8: Esempio di instradamento indiretto.

1.5.2 Dispositivi di rete

Prima di osservare la tabella di routing è il meccanismo di Table LookUp è bene capire quali siano i ruoli del Router e del Gateway:

- **Router:** il router è un generico dispositivo di rete che ha il compito di instradare pacchetti.
- **Gateway:** un gateway è un ruoter a tutti gli effetti, ma introduce funzionalità aggiuntive. Nell'instradamento indiretto, esso ricopre un ruolo speciale, poiché funge da ponte tra reti eterogenee. Dunque, il suo scopo è quello di veicolare i pacchetti verso una rete differente.

Altri due concetti di cui chiariremo la differenza sono forwarding e routing:

- **Forwarding:** è il meccanismo che definisce verso quale interfaccia debbano essere instradati i pacchetti. Il suo esito è dato dalla lettura della tabella di instradamento(Table Lookup).
- **Routing:** il routing è una procedura che permette di costruire la Forwarding Information Base. Nel protocollo IP questo lavoro è assegnato a dei protocolli specifici(RIP, OSPF, ecc.)

1.5.3 Tabella di routing

Ogni volta che un router riceve pacchetti, deve capire verso quale interfaccia esso debba essere intradato. Se la sorgente non è direttamente connessa alla di destinazione, il dato viene inviato al gateway della rete di appartenenza, che stabilisce come il pacchetto debba essere indirizzato verso la destinazione. Perchiò ogni router deve tenere traccia delle possibili rotte d'instradamento, che vengono raccolte nella tabella di routing. Per ogni destinazione della routing table, vi sono le seguenti informazioni:

- **Destinazione(D):** indica l'indirizzo IP del nodo verso cui porta l'interfaccia;
- **Netmask(N):** indica la maschera della rete verso cui è indirizzato il route in riferimento;
- **Gateway(G):** indica l'indirizzo IP verso cui effettuare la consegna. Da questo valore è possibile capire se si opera per instradamento diretto o indiretto;
- **Interfaccia di rete (IF):** indica l'interdaccia fisica verso cui è indirizzato il route.
- **Metrica (M):** dato che per ogni destinazione possono esservi più route, questo campo serve a specificarne il "costo".

Nell'immagine [1.9] si mostra la tabella di routing Ipv4 vista da terminale.

```
IPv4 Route Table
===========================================================================
Active Routes:
Network Destination        Netmask          Gateway       Interface  Metric
          0.0.0.0          0.0.0.0         10.0.0.1        10.0.0.75     35
         10.0.0.0    255.255.255.0         On-link         10.0.0.75    291
        10.0.0.75  255.255.255.255         On-link         10.0.0.75    291
       10.0.0.255  255.255.255.255         On-link         10.0.0.75    291
        127.0.0.0        255.0.0.0         On-link         127.0.0.1    331
        127.0.0.1  255.255.255.255         On-link         127.0.0.1    331
  127.255.255.255  255.255.255.255         On-link         127.0.0.1    331
    192.168.56.0    255.255.255.0         On-link       192.168.56.1    281
    192.168.56.1  255.255.255.255         On-link       192.168.56.1    281
  192.168.56.255  255.255.255.255         On-link       192.168.56.1    281
        224.0.0.0        240.0.0.0         On-link         127.0.0.1    331
        224.0.0.0        240.0.0.0         On-link       192.168.56.1    281
        224.0.0.0        240.0.0.0         On-link         10.0.0.75    291
  255.255.255.255  255.255.255.255         On-link         127.0.0.1    331
```

Figura 1.9: Output tabella di routing Ipv4.

Nel mondo IP, le destinazioni sono tutte le reti IP esistenti nella topologia in esame. La topologia in figura [1.10] presenta presenti cinque reti IP, che corrispondono allo stesso numero di righe presenti nella routing table del nodo R1. Siccome il numero di righe dipende dal numero di destinazioni IP in tabella, esso è uguale in tutti i nodi della topologia. Ovviamente, essendo le destinazioni coincidenti con le reti IP presenti, tutti i router raccolgono la stessa lista di destinazioni raggiungibili, mentre cambiano i percorsi fatti per raggiungere quelle destinazioni.

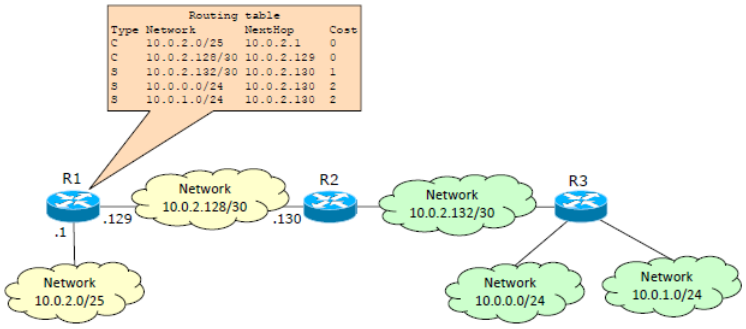

Figura 1.10: Esempio di topologia di rete con annessa tabella di routing.

1.5.4 Table Lookup

Il Table Lookup è la modalità secondo cui viene visitata la routing table nell'instradamento. Il meccanismo parte alla ricezione di ciascun datagramma ed è finalizzato a trovare il route adatto alla destinazione del pachetto. Esso comporta il confroto dell'indirizzo di destinazione con i campi di destinazione di ciascuna riga. Vediamo in seguito gli step della procedura:

1. **(IP datagramma AND netmask) = Result:** l'indirizzo di destinazione del datagramma è sottoposto all'AND bit a bit con la netmask della righa .

2. **(Result = destinazione) ?**: il risultato viene confrontato dola destinazione del route. Se combaciano il processo termina altrimenti si passa al route successivo.

	Destinazione	Netmask	Etc.
1	0.0.0.0	0.0.0.0	...
2	192.168.2.0	255.255.255.0	...
3	192.168.2.18	255.255.255.255	...

Datagramma con IP dest. = 192.168.2.18
Confronto prima con riga 3, poi con riga 2 e poi riga 1

```
192.168.002.018        bitwise AND
255.255.255.255
192.168.002.018 == 192.168.002.018
```

Figura 1.11: Esempio di lookup.

L'iterazione di questa procedura parte dalla riga che presenta la netmask con maggior bit a uno. Quando viene trovato il route idoneo si procede con l'instradamento verso la relativa interfaccia. Nel caso in cui non vi sia alcun route che permetta di raggiungere la destinazione viene generato un messaggio di errore. Quest'ultimo è solitamente indirizzato alla sorgente e viene trasmesso grazie al protocollo ICMP.

Capitolo 2

Named Data Networking

2.1 Nascita del progetto

Come accennato nelle sezioni precedenti, Named Data Networking(NDN) [10] [3] [7] fa parte dei cinque progetti fondati dal National Science Foundation negli Stati Uniti d'America per il Future Internet Architecture Program [1] [5]. In figura [2.1] se ne osservano i loghi. NDN pone le sue radici nel precedente progetto Content Centric Networking (CCN), presentato per la prima volta al pubblico nel 2006. L'obiettivo di NDN è quello di ideare un nuovo protocollo di rete, che parta dalle basi utili del protocollo IP ma allo stesso tempo ne vada a risolvere i limiti. Il progetto intende partire è il superamento del concetto di indirizzo IP, per lasciare il spazio all'assegnazione di nomi ai contenuti che vengono trasmessi in rete. Questa concezione stravolge completamente il paradigma di rete che passa dal "dove" trovare una risorsa al "cosa" cercare e sposta la sicurezza dei dati nei contenuti stessi. Nelle prossime sezioni si analizzano i concetti introdotti da NDN fino alla versione 0.6.1.

Figura 2.1: Loghi del progetto Named Data Networking.

2.2 Principi fondamentali

I ricercatori del progetto NDN hanno stilato una lista di linee guida utili alla progettazione dell'architettura. Se ne elencano i principi di particolare rilevanza:

- **Universality:** il protocollo deve essere comune a tutte le applicazioni e a tutte le tecnologie di rete. Esso si deve adattare a qualunque tipologia di rete a prescindere dalla tecnologia di base(Ethernet, Bluetooth, Wifi, ecc.);
- **Data Centricity and Data Immutability:** i dati trasmessi in rete devono essere univoci, nominati e immutabili. L'unico modo per accingere ad un dato è manifestare alla rete un'interesse;
- **Securing Data Directly:** la sicurezza deve essere intrinseca nei dati; in questo modo non vi è più la necessità di rendere sicuri i canali, ma viene lasciato ai consumatori il compito di verificarne la correttezza;
- **Hierarchical Naming:** i pacchetti devono contenere nomi gerarchici per abilitare il demultiplexing e fornire un contesto strutturato;
- **In Network Name Discovery:** gli interessi manifestati alla rete devono poter essere soddisfatti anche se non ne viene fornito il nome gerarchico completo. Se ne viene ricevuto un prefisso, i nodi devono essere in grado di ricostruire il nome completo in base alle convenzioni convenzioni con cui essi vengono assegnati;
- **Hop-by-Hop Flow Balance:** su ciascun collegamento, un pacchetto di interesse non deve riportare più di un pacchetto di dati. In questo modo si consente a ciascun nodo di controllare il carico sui relativi collegamenti;

2.3 Architettura a clessidra

L'architettura a clessidra [2.2] è ciò che rende il design Internet originale, elegante e potente. NDN vuole concentrarsi sul livello di rete che implementa le funzionalità necessarie all'interconnessione globale. La cosiddetta "vita sottile" è stata un fattore chiave per la crescita esplosiva di Internet, consentendo alle tecnologie di livello inferiore e superiore di innovare senza vincoli non necessari. Questo aspetto architetturale è proprio uno dei dettagli che NDN vuole mantenere e rafforzare. Il cambiamento concettuale è il seguente: mentre IP andava ad instradare pacchetti verso la destinazione NDN va a cercare i dati in base al loro nome.

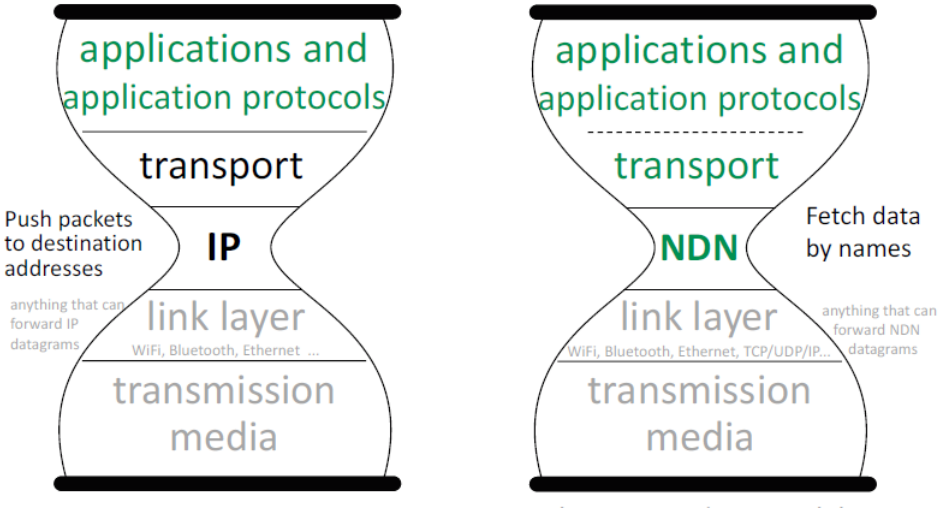

Figura 2.2: Cambiamento concettuale del modello a clessidra.

Un altro aspetto che NDN intende mantenere è la separazione fra routing e forwarding, dimostratasi necessaria per lo sviluppo di Internet. Questa divisione consente al piano di forwarding di operare mentre il sistema mentre il routing evolve nel tempo.

Uno degli aspetti che NDN intende stravolgere è la sicurezza, che nell'odierno piano architettuare appare come un ripensamento. Mentre negli attuali protocolli di sicurezza l'obiettivo è quello di rendere sicure le connessioni, NDN raggira il problema racchiudendo la sicurezza in una firma, che deve essere incapsulata in ogni pacchetto.

Mentre IP gode di un protocolli di trasporto che garantiscono l'equilibrio del traffico unicast, NDN intende realizzare il bilanciamento del flusso nella vita sottile. L'architettura dovrebbe facilitare la scelta dell'utente e la concorrenza, ove possibile. Sebbene non sia un fattore rilevante nel design originale di Internet, l'implementazione globale ci ha insegnato che "l'architettura non è neutrale". NDN fa uno sforzo consapevole per responsabilizzare gli utenti finali e consentire la concorrenza.

Nelle prossime sezioni sono illustrati i seguenti argomenti:

- Paccheti di Interesse e di Dati;
- Formato dei Nomi;
- Strategie riguardanti: instradamento, cachig, sicurezza e trasporto.

2.4 Pacchetti

Per abbandonare definitivamente l'identificazione per indirizzi ed abbracciare il paradigma a nomi NDN deve introdurre una nuova serie di strategie. Una di queste è la definizione dei pacchetti. Prima di introdurre il loro formato è bene chiarire l'idea che il progetto intende concretizzare. Per oltrepassare il modello a connessioni Client-Server, NDN mette al centro della comunicazione i il consumatore(destinatario dei dati). Contrariamente a quanto avviene nel protocollo IP, in cui ogni un dato viene richiesto ad una specifica sorgente, NDN prevede che il consumatore manifesti un interesse rivolto genericamente alla rete e che riguarda uno specifico pacchetto. Ogni router , seguendo determinate regole, inoltrerà il pacchetto verso una nuova interfaccia fino a quando l'interesse non raggiunge un nodo in grado di soddisfarlo (ovvero che possiede i dati richiesti) . A questo punto, il produttore risponde con un pacchetto contenente i dati, che ripercorrendo il percorso dell'interesse a ritroso viene consegnato a consumatore.

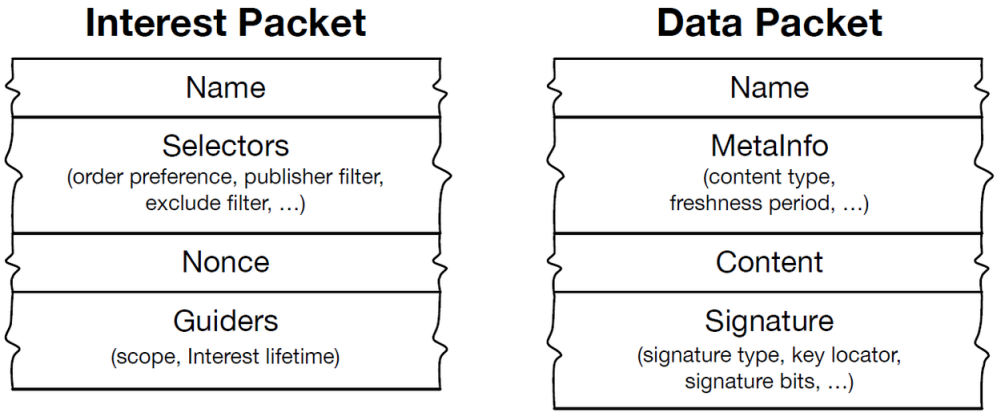

Figura 2.3: Interest Packet e Data Packet.

NDN si serve di due tipologie distinte di pacchetti : Interest Packet e Data Packet. Essi vengono mostrati nell'immagine [2.3] Prima di entrare nel dettaglio è necessario definire il formato con cui essi sono codificati ovvero il Type-Length-Value(TLV).

2.4.1 Formato TLV

Type-Length-Value è il formato con cui vengono definiti i pacchetti NDN. Esso segue una struttura gerarchica la cui radice è TVL0. Essa ne contiene il tipo(valore numerico) e a sua volta raccoglie altre strutture TLV. Le sub-TVL possono essere iterate fino a raggiungere il secondo livello di annidamento(generalmente si preferisce ridurre al minimo la nidificazione).

Questa rappresentazione permette di minimizzare la dimensione dei pacchetti ma allo stesso tempo garantisce flessibilità per le possibili estensioni del protocollo. Per questo motivo sia il campo Type (T) che il campo Length (L) assumono un formato ad ottetti di lunghezza variabile.

Il primo ottetto del numero riporta il numero effettivo o segnala se è presente una codifica a più ottetti, come definito di seguito:

- **primo ottetto $<$ 253:** il numero è codificato in quell'ottetto;
- **primo ottetto == 253:** il numero è codificato nei due ottetti seguenti, in ordine di byte netto;
- **primo ottetto == 254:** il numero è codificato nei quattro ottetti seguenti, in ordine di byte netto;
- **primo ottetto == 255:** il numero è codificato nei seguenti otto ottetti, in ordine di byte netto;

2.4.2 Interest Packet

L'Interest Packet è l'unico tipo di pacchetto che i nodi della rete NDN possono trasmettere in maniera attiva. Nel momento in cui un host mostra alla rete un interesse, chiunque sia in possesso di quel pacchetto è in grado di soddisfarlo rispondendo con un Data Packet. L'unico elemento necessariamente richiesto è il campo Name del pacchetto verso cui si mostra interesse, tutti gli altri campi sono opzionali. In seguito descriviamo i campi del pacchetto di interesse:

- **Name:** è l'unico campo obbligatorio degli Interest Packet. Se manca allora il pacchetto viene scartato dai nodi di rete;
- **CanBePrefix:** se presente indica che il campo Name può essere solo un prefisso;
- **MustBeFresh:** il campo che indica al Content Store se i dati sono ancora "freschi" o devono essere obbligatoriamente aggiornati;
- **ForwardingHint:** questo campo raccoglie una lista di deleghe in formato Link Object. Esse definiscono una serie di preferenze relative ai Content Store da cui viene soddisfatto l'interesse;
- **Nonce**: è un dato lungo quattro ottetti generato casualmente hop-by-hop. Serve alla rete per evitare il loop di Interest Packet;
- **InterestLifetime:** indica quanto tempo trascorre prima del timeout di interesse. Il valore è espresso in millisecondi ed è relativo all'istante in cui l'interesse è ricevuto.

- **HopLimit:** il campo è memorizzato in un byte ed indica quanti hop il pacchetto possa attraversare. Ad ogni hop il campo viene decrementato di uno (HopLimit = HopLimit - 1) e quando raggiunge valore zero i nodi che lo ricevono sono obbligati a scartarlo;
- **Parameters:** è un generico campo che parametrizza l'interesse;

2.4.3 Data Packet

Il Data Packet è il formato con cui vengono soddisfatti gli interessi; anch'esso presenta degli elementi facoltativi e degli elementi obbligatori. Assieme al nome in questo pacchetto si ha un altro dato obbligatorio: la Signature. Si tratta di un campo particolarmente importante dato che va a specificare l'origine del pacchetto. Visto che la sua codifica riguarda tutti gli altri campi la firma viene posta come ultimo parametro. Si illustrano i singoli campi:

- **Name:** campo obbligatorio che identifica il pacchetto;
- **MetaInfo:** metainfo raccoglie tre campi: **ContentType**, **FreshnessPeriod** e **FinalBlockId**. ContentType indica il tipo di contenuto; NDN fornisce una lista di tipi predefiniti. Il FreshnessPeriod specifica il periodo di "freschezza"; quando il periodo per cui il Content Store ha ospitato il pacchetto supera questo valore il dato deve essere aggiornato. Infine, il FinalBlockId raccoglie alcune informazioni aggiuntive indirizzate ai consumatori;
- **Content:** questo campo è facoltativo e può ospitare una qualsiasi sequenza di bit.
- **Signature:** la firma NDN viene definita come due blocchi TLV consecutivi: SignatureInfo e SignatureValue.
  - **SignatureInfo:** è incluso nel calcolo della firma e descrive come essa sia stata calcolata. Le informazioni da esso indicate sono l'algoritmo firma e i dettagli di certificazione come SignatureType e KeyLocator.

    Il SignatureType indica il tipo di firma utilizzata. Per ora NDN ne definisce queattro: DigestSha256, SignatureSha256WithRsa, SignatureSha256WithRsa e SignatureHmacWithSha256.

    Il KeyLocator specifica Nome del pacchetto contenente i dati di autenticazione della firma, che possono essere: chiave privata, certificato o KeyDigest. Nonostante esso sia definito come facoltativo deve sempre essere specificato qualora sia il metodo di cifratura a richiederlo. In caso contrario la cifratura non è efficace e il pacchetto viene riconosciuto come un dato non sicuro;
  - **SignatureValue**: racchiude il risultato del calcolo della firma;

2.5 Nomi

NDN prevede che i nomi siano strutturati gerarchicamente. Per esempio il nome di un video prodotto da "parc" può essere il seguente:

/parc/videos/WidgetA.mpg.

I nomi che vengono utilizzati per recuperare i dati a livello globale richiedono univocità mentre quelli destinati alla comunicazione locale possono essere basati sul contesto locale.

2.5.1 Scelta dei nomi

Per adottare un modello di comunicazione univoco, produttori e consumatori possono scegliere di creare delle convenzioni. Un esempio è dato dai nomi assegnati riferirsi a versioni e segmenti; il terzo segmento della prima versione del video potrebbe avere il seguente nome:

/parc/videos/WidgetA.mpg/1/3

Nonostante il vincolo di univocità, le applicazioni sono libere di scegliere lo schema di denominazione che più preferiscono. Ciò è possibile perchè che le convenzioni su nomi sono completamente "opache alla rete", ovvero i router non possono comprendere il significato dei nomi.

2.5.2 Recupero dati creati dinamicamente

Visto che l'instradamento degli interessi è basato sui nomi, NDN deve ideare una strategia che permetta di recuperare i dati generati dinamicamente. I consumatori devono essere in grado di identificarne il nome senza esserne a conoscenza. Per rendere tutto più semplice NDN permette di condividerli in base a nomi parziali. Si mostra un esempio:

Il consumatore richiede /parc/videos/WidgetA.mpg e recupera il pacchetto
/parc/videos/WidgetA.mpg/1/1

L'interesse "/parc/videos/WidgetA.mpg" nasconde il fatto che il dato richiesto sia suddiviso in più parti, permettendo al produttore di consegnare segmenti prodotti dinamicamente.

2.5.3 Formato del nome NDN

Anche i nomi adottano il formato TLV che viene iterato per due livelli: il primo indica il tipo di nome e la lunghezza mentre il secondo livello raccoglie i seguenti parametri:

- **GenericNameComponent**: rappresenta un nome generico per cui non è prevista alcun tipo di restrizione. Per la rappresentazione testuale, è spesso utilizzata la notazione URI. Essa non necessita di una parte iniziale che specifichi il protocollo(se specificato viene ignorato).I caratteri aggiuntivi utilizzano la codifica percentuale della sintassi URI;
- **ImplicitSha256DigestComponent**: è un componente digest implicito SHA-256 lungo 32 ottetti;
- **ParametersSha256DigestComponent**: è un componente digest implicito SHA-256 riguardante i parametri di interesse e lungo 32 ottetti;

Il digest ricopre un ruolo di particolare importanza poichè rende univoco ogni nome.

2.5.4 Funzionalità assodate ed aree di ricerca

La versione 0.6.1 di NDN implementa le seguenti funzionalità:

- Distribuzione dello stesso contenuto in momenti diversi;
- Richiesta concorrente di un pacchetto(multicast);
- Richieste di utenti da posizioni diverse;
- Adattamento all'intermittenza della connessione;

  Essendo un progetto molto recente, NDN è in costante evoluzione. Uno dei campi di ricerca più caldi riguarda le convenzioni dei nomi. I ricercatori stanno cercando di stilare serie di regole che guidino gli utenti a scegliere i nomi per i loro contenuti. Al momento queste linee guida vengono raccolte empiricamente dal test di applicazioni pilota.

2.6 Stateful Forwarding Plane

2.6.1 NDN Forwarding Deamon

NDN Forwarding Deamon definisce le strutture coinvolte nella gestione degli interessi all'interno di ciascun nodo della rete NDN. L'immagine [2.4] [7] descrive il modello su cui si basa il NFD:

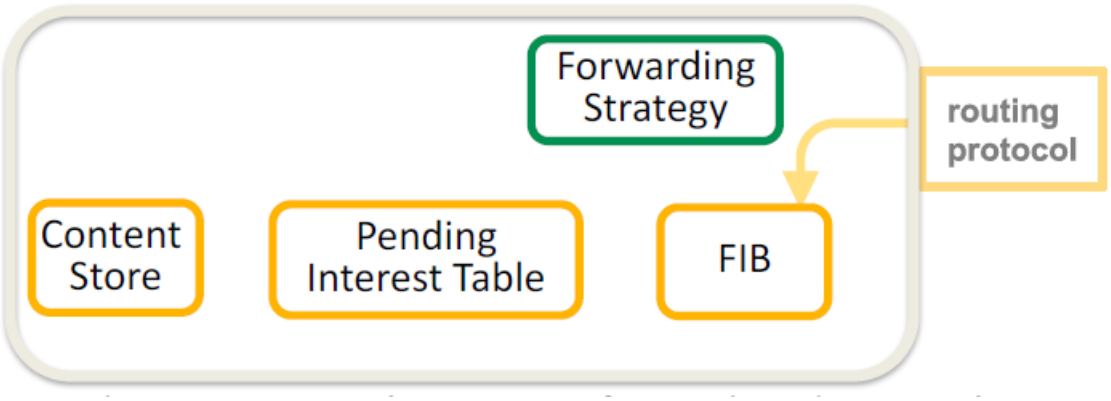

Figura 2.4: Modello concettuale di NDN Forwarding Deamon.

- **Content Store(CS):** contiene i nomi dei pacchetti raccolti dal nodo. E' la struttura dati che permette di realizzare il meccanismo di caching.
- **Pending Interest Table(PIT):** questa tabella registra e mantiene tutti gli interessi in sospeso. Essa raccoglie dati relativi al nome del pacchetto, all'interfaccia di ricezione e il timelife dell'interesse.
- **Forwarding Information Base(FIB):** rappresenta la tabella di forwarding che può essere popolata tramite due strategie differenti: autoapprendimento oppure protocolli routing. Essa associa a ciascuna interfaccia l'insieme dei nomi che può raggiungere.

2.6.2 Instradamento degli Interest Packet

La collaborazione delle strutture interne ad NFD da vita alla strategia di forwarding. L'elaborazione di un interessep parte dal Content Store, che per prima cosa controlla la presenza del pacchetto. In caso positivo si procede con la risposta, altrimenti l'interesse viene registrato nella PIT. Dopo la registrazione, si passa alla consultazione della FIB dove viene cercata una corrispondenza per instradare il pacchetto. Nel caso venga trovata si procede ad instradare l'interesse verso quell'interfaccia.

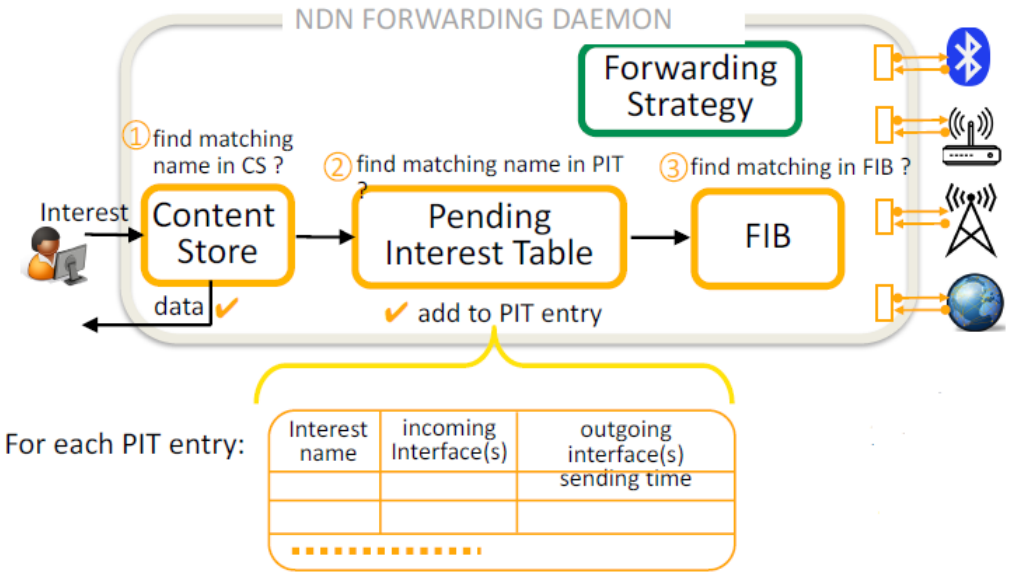

Figura 2.5: Passi seguiti dalla strategia di forwarding degli interessi.

Si analizzano gli step mostrati nell'immagine [2.5]:

1. Si controlla se il Content Store contiene il dato richiesto dall'interesse. In caso positivo invio il pacchetto contenente il dato verso l'interfaccia di provenienza, altrimenti si passa al secondo step;

2. Si controlla nella Pending Interest Table se sono presenti degli interessi per quel pacchetto. In caso negativo viene aggiunta alla tabella una riga contenente nome e interfaccia di provenienza, altrimenti si aggiunge alla riga esistente l'interfaccia. Si passa al terzo step;

3. Per ogni interfaccia della Forwarding Information Base, si controlla lo spazio di nomi che mi permette di raggiungere. L'interesse viene instradato verso le interfacce che permettono di raggiungere il pacchetto desiderato.

2.6.3 Instradamento dei Data Packet

NDN prevede che la modalità di instradamento dei Data Packet sia differente quella per gli interessi. Per prima cosa si controlla che la PIT registri interesse per quel nome, dopodichè il pacchetto viene instradato verso l'interfaccia da cui è provenuta la richiesta.

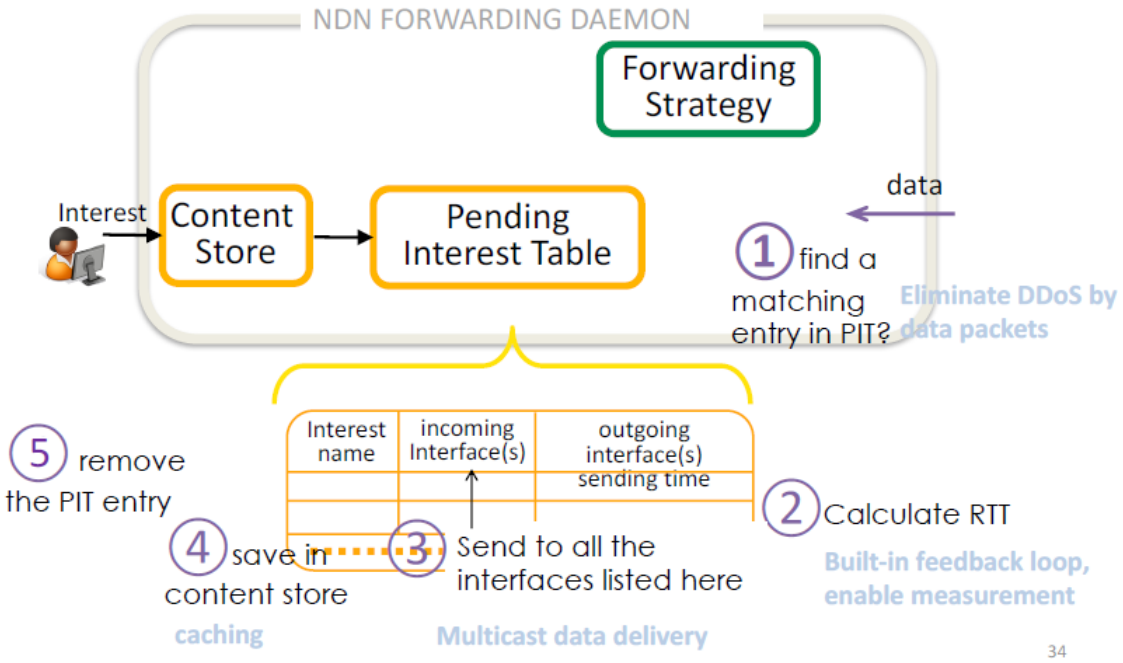

Figura 2.6: Passi dell'instradamento di un pacchetto di dati.

Si analizzano gli step mostrati nell'immagine [2.6]:

1. Controllo che il nome del pacchetto corrisponda ad una richiesta presente nella PIT. Nel caso sia presente si passa al secondo step, altrimenti esso viene immediatamente scartato; questo permette di evitare attacchi DDoS legati alla ricezione di Data Packet malevoli;

2. Si calcola il Round Trip Time(RTT), utilizzato per capire quale sia lo stato della rete;

3. Il Data Packet viene instradato verso l'interfaccia da cui proviene l'interesse;

4. Il dato viene memorizzato nel Content Store;

5. L'interesse viene rimosso dalla Pending Interest Table;

2.6.4 Vantaggi dell'approccio stateful

Uno dei principali cambiamenti introdotti da NDN è il passaggio ad un protocollo stateful. Mentre l'approccio stateless del protocollo IP, non richiede un'elevata occupazione di memoria e carichi computazionalmente eccessivi, NDN preferisce sacrificare in parte questi aspetti per dotare singoli nodi di maggiori funzionalità. Se ne osservano alcune:

- **Multicast**: nella PIT è possibile annotare tutte le interfacce da cui provengono interessi per lo stesso nome. Questo aspetto permette che il multicast venga implementato in maniera molto semplice.
- **Bilanciamento di flusso** : questa funzionalità può essere implementata monitorando le dimensioni della PIT. Supponiamo che la frequenza di interessi diretti verso una determinata interfaccia sia minore di quella dei pacchetti di dati ricevuti. Questo è un chiaro segno che la frequenza delle richieste è troppo elevata;
- **Difesa agli attacchi DDoS**: la dimensione della PIT e la velocità con cui essa aumenta possono indicare chiari sintomi legati a questi tipi di attacco.

Per evitare l'eccessivo accrescimento della PIT il demone NFD scarta gli interessi allo scadere del loro timelife. Da questi timeout vengono collezionati ulteriori dati grazie ai quali vengono apprese le prestazioni di rete verso le singole interfacce. Questa funzionalità permette di: rilevare perdite di pacchetti, valutare le migliori strategie di forwarding e rilevare guasti di rete. Se l'instradamento verso una specifica interfaccia porta ad un elevato numero di timeout si ha un chiaro segnale di malfunzionamento della rete.

2.7 Caching

NDN introduce il meccanismo di caching automatico. Memorizzando i dati ricevuti nel Content Store, i nodi di rete acquisiscono al capacità di soddisfare richieste future. Questo significa che un dato si può trovare in qualsiasi nodo della rete, ma non sappiamo da quale di questi arrivi la risposta. Mentre con il protocollo IP le richieste sono indirizzate a server specifici, con NDN gli interessi vengono mostrati genericamente alla rete, che capirà come soddisfarlo. Questo concetto viene raffigurato nell'immagine [2.7].

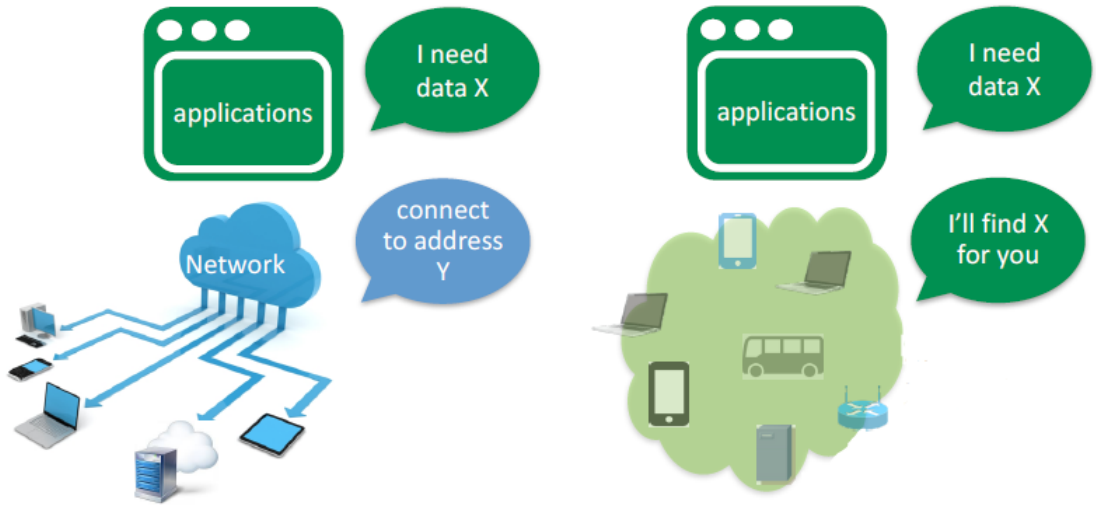

Figura 2.7: Passi della ricezione di un pacchetto di dati.

Il Content Store è paragonabile alla memoria buffer dei router, ma diversamente da essa ha la capacità di raccogliere dati. Questa strategia porta numerosi vantaggi alla prestazione di rete, sia per i dati statici che per quelli dinamici. Supponiamo che un pacchetto di dati venga perso all'ultimo hop. Se ciò avvenisse in una rete IP, il pacchetto dovrebbe ripercorrere tutti gli hop dalla sorgente fino alla destinazione, mentre in una rete NDN basterebbe ripresentare l'interesse alla rete ed il primo hop, avendo memorizzato il dato nel Content Store, sarebbe subito in grado di soddisfarlo. Uno dei principali problemi che presenta il caching è la privacy. Dal momento che si fornisce un dato alla rete non è possibile sapere quali Content Store lo memorizzino. Tuttavia NDN crede che questo problema sia trascurabile se si adottano le adeguate misure di sicurezza.

2.8 Sicurezza

Uno dei principali punti di forza di NDN è il modello di sicurezza Data Centric. Come mostrato in figura [2.8] esso prevede che la sicurezza delle informazioni sia racchiusa nei dati stessi. Sulla base di questo concetto, NDN crede che non sia più necessario rendere sicuri i canali di comunicazione, riducendo notevolmente i punti di vulnerabilità. I dati e i rispettivi nomi devono essere obbligatoriamente firmati e le applicazioni non possono contrastare questo meccanismo dato che tutti i dati privi di firma vengono scartati dalla rete.

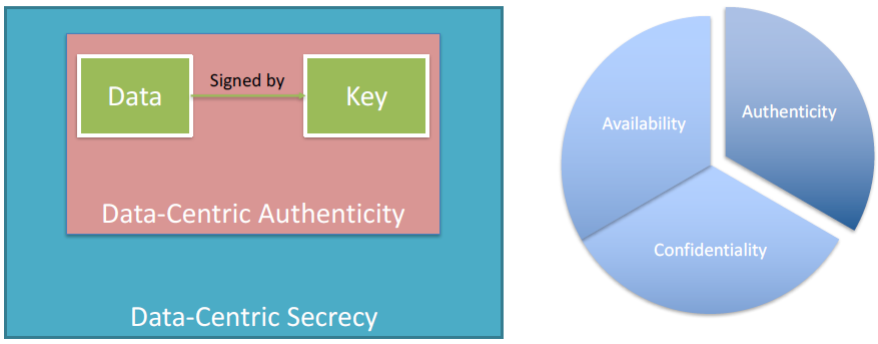

Figura 2.8: Modello Data-Centric Security.

La firma dei dati permette di autenticarne l'origine. Nel caso in cui questa non corrisponda a quella prevista basterà scartare il pacchetto. In questo semplice meccanismo viene racchiusa la relazione di trust fra produttore e consumatore, la quale viene completamente disaccoppiata da come(e da dove) essi vengano ottenuti.

Quindi, il modello di fiducia detto a "grana fine" consente ai consumatori di capire se un proprietario di chiave pubblica sia editore accettabile. Tuttavia, la sicurezza basata su questo tipo di chiavi si è sempre rivelata inefficiente e difficile da implementare, per questo essa richiede alcuni spunti di innovazione(ambito di ricerca). NDN prevede di utilizzare firme digitali efficienti ma da sole non sono bastano, sono necessari meccanismi flessibili per i rapporti di fiducia. Dato che le chiavi possono essere comunicate come dati la loro distribuzione è semplice e allo stesso tempo sicura. Infine visto che tutti i dati devono essere firmati NDN pone le basi anche per la sicurezza dei messaggi di routing.

2.9 Trasporto

L'architettura NDN decide di non adottare un livello di trasporto separato. Mentre nel protocollo IP esso è posto agli estremi del canale di comunicazione, NDN decide di integrarlo direttamente nel piano di inoltro. Multiplexing e il demultiplexing tra i processi applicativi viene effettuato direttamente utilizzando i nomi nel livello di rete(NDN) , mentre integrità e affidabilità sono affidate al livello applicativo. I traguardi raggiunti, permettono ad NDN di addattarsi egregiamente all'inaffidabilità della rete e di non avere particolari nell'integrazione di dispositivi mobili. L'inaffidabilità viene risolta in maniera molto semplice: gli interessi non soddisfatti entro un intervallo di tempo ragionevole vengono ritrasmessi alla rete. Questa strategia raggiunge risultati molto soddisfacenti senza che lo specifico compito venga assegnato ad un livello architetturale separato. Inoltre ogni nodo NDN effettua autonomamente il controllo di congestione e il controllo di flusso andando a monitorare le frequenze degli interessi e dei pacchetti di dati. Quando essi ricevono un numero eccessivo di Data Packet da una delle interfacce, reagiscono rallentando la frequenza degli interessi ad essa indirizzati. Si noti che questo approccio libera gli host terminali da queste responsabilità.

2.10 Tabella di confronto

Si è scelto di mostrare le principali differenze dei due protocolli con la seguente tabella.

NDN	IP
Pull: dati ricevuti solo a seguito di un interesse	Push: dati inviati attivamente
Identificazione dati	Identificazione host
Stateful	Stateless
Trasporto nel livello di rete	Livello di trasporto dedicato
Nessun riferimento a sorgente e destinatario nei pacchetti	I pacchetti indicano sorgente e destinatario
Caching	No caching
DNS non necessario	DNS necessario
Data Centric Security	Sicurezza del canale di comunicazione

Figura 2.9: Tabella di confronto dei protocolli IP e NDN.

Capitolo 3

Piattaforma NDN

3.1 Componenti software

Nonostante sia in piena fase ricerca accademica e molte delle sue componenti siano ancora molto sperimentali, la piattaforma NDN offre già una discreta base di applicativi, librerie, infrastrutture software e framework. Alcuni di essi sono mostrati nell'immagine [3.1].

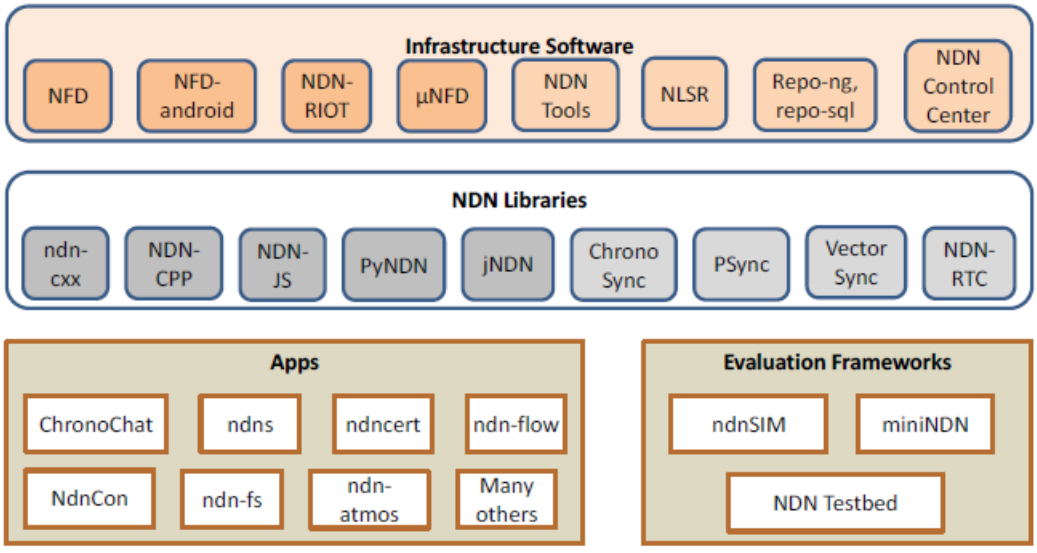

Figura 3.1: Strumenti software forniti dalla piattaforma NDN.

Queste componenti sono in continuo aggiornamento, non solo da parte dei ricercatori ma anche per mano della comunity di sviluppatori, a cui è data la possibilità di lavorare e contribuire al progetto.

3.2 Piattaforme supportate

Segue l'elenco delle piattaforme supportate:

- Sistemi desktop:
  - Ubuntu, OSX, FreeBSD, Fedora e altre distribuzioni Linux;
- Router domestici:
  - OPEN-WRT, DD-WRT;
- Dispositivi mobile:
  - Android e IOS(solo librerie);
- Internet of Things:
  - Arduino, ESP8266, RIOT-OS;
  - Raspberry Pi(disponibili i binari per NFD);
- Tecnologie Web:
  - Librerie Javascript come NDN-JS;
- Ambienti di virtualizzazione:
  - Vagrant;
  - Docker;

3.3 NDN Forwarding Deamon

NDN Forwarding Daemon(NFD) [9] [4] è un forwarder di rete che implementa ed evolve assieme al protocollo Named Data Networking. Con la distribuzione di guide per sviluppatori NDN permette ai membri della comunity di portare il proprio contributo nella realizzazione di algoritmi, librerie o applicativi. Proprio per questo in fase di progettazione si è prestata particolare attenzione ad aspetti come modularità ed estensibilità.

3.3.1 Moduli NFD

La funzione principale di NFD è quella di inoltrare Interest e Data Packet. Esso parte dall'astrazione del livello protocollare Data Link che viene concretizzato in Faces e mantiene tre strutture dati di base che si occupano dell'amministrazione dei pacchetti: Content Store(CS), Pending Interest Table(PIT) e Forwarding Information Base(FIB). Attraverso la collaborazione di queste componenti vengono supportate diverse strategie di inoltro. Nell'immagine [3.2] vengono mostrati i moduli e le loro relazioni:

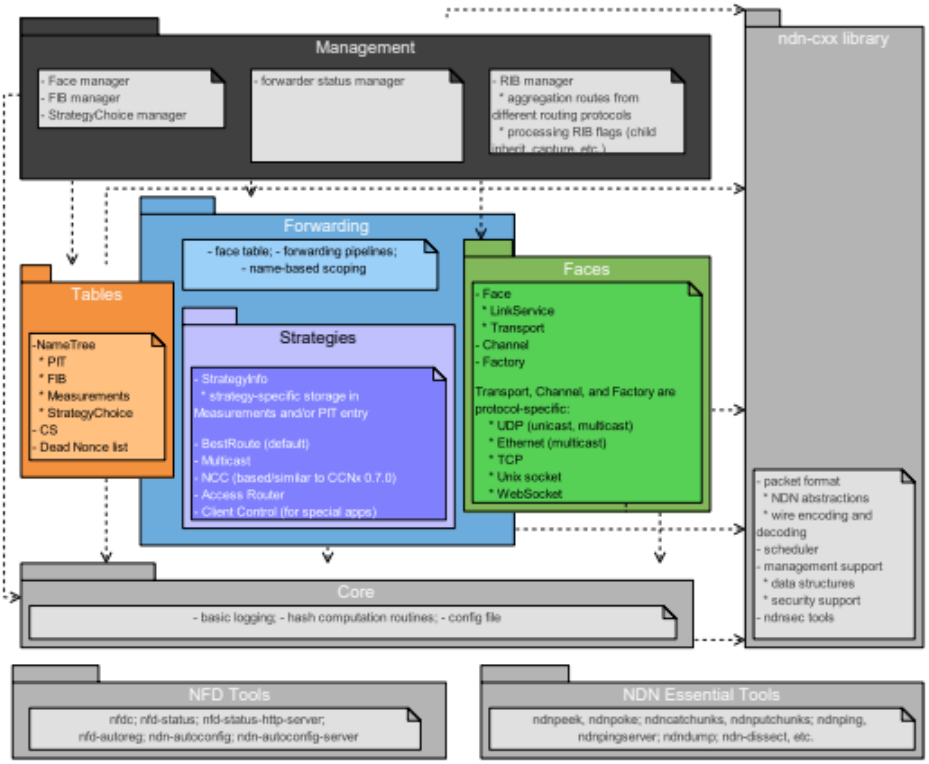

Figura 3.2: Strumenti software forniti dalla piattaforma NDN.

- **Librerie Core, Tools e ndn-cxx:** queste librerie raccolgono una serie di servizi condivisi, tra cui: file di configurazione, monitoraggio interfacce, semplici applicativi(ndn-ping) e molti altri moduli;
- **Faces:** implementa l'astrazione dell'interfaccia NDN e offre servizi di link e di trasporto.
- **Tables:** comprende le strutture dati utilizzare da NFD: Content Store, Pending Interest Table, Farwarding Information Base, StategyChoice e altre strutture sulle misure di prestazione;

- **Forwarding:** Definisce le funzionalità d'elaborazione dei pacchetti, che collaborano con Faces, Tables e Strategies. Strategies è un framework che permette di adottare diverse strategie di forwarding;
- **Management:** In questo package viene implementato l'NFD Management Protocol che consente alle applicazioni di interfacciarsi con l'NFD. Un semplice esempio riguarda lo scambio di interessi e dati tra applicazioni e demone;
- **RIB Management:** La Routing Information Base può essere aggiornata utilizzando protocolli di routing, registrando manualmente i prefissi applicativi oppure tramite configurazioni a riga di comando. Il compito principale di questo modulo è quello di generare una tabella di inoltro coerente che contenga le informazioni minime per la gestione dell'inoltro;

Il punto di approdo dei pacchetti è Faces; l'oggetto Face rappresenta una generalizzazione di interfaccia che può essere un'interfaccia fisica o un canale virtuale. Face è composta da un LinkService e da un Transport. Il LinkService fornisce servizi di alto livello, come frammentazione, assemblaggio e rilevamento degli errori, mentre il Transport si occupa della consegna dei pacchetti(Ethernet, UDP, TCP). Face legge flussi o datagrammi tramite le API di sistema operativo, e li consegna al livello di rete(NFD). A livello di rete ciascun pacchetto (Interesse, Dati o Nack) viene elaborato dopo essere stato inserito nelle pipeline, che definiscono una serie di passaggi da attuare su di essi. Fatto ciò viene attuato il piano di forwarding(illustrato nel capitolo precedente) che vede la collaborazione di Content Store, Pending Interest Table e Forwarding Information Base. Nel caso il pacchetto debba essere instradato verso un altro nodo verrà re-direzionato verso Face.

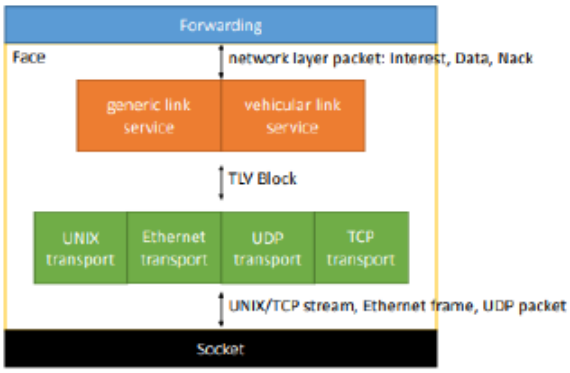

Figura 3.3: Schema operativo di Link Service e Transport all'interno di Face.

Dato che un'analisi dettagliata dell'intero NFD risulterebbe troppo lunga, si è scelto non toccare temi di maggiore dettaglio. Ciò non toglie che per avere una piena comprensione del forwarder sia necessario andare a toccare temi più specifici.

3.4 Piattaforme di sviluppo

Named Data Networking offre la possibilità di realizzare scenari di rete in contesti fisici oppure tramite un apposito simulatore. Due dei principali supporti offerti sono rispettivamente Testbed e nsnSIM.

TestBed: Testbed è una risorsa fisica condivisa, creata per supportare la ricerca. Nasce dalla collaborazione di circa trenta istituzioni tra centri di ricerca e università di tutto il mondo. Questa piattaforma offre svariati servizi tra cui router, servizi applicativi e tanti altri dispositivi. Lo scopo principale di Testbed è quello di permettere a qualsiasi ricercatore o semplice sviluppatore di testare il comportamento della rete NDN. Installando NFD su una propria macchina e seguendo le Policy di accesso definite da NDN è possibile inizare ad operare sulla piattaforma.

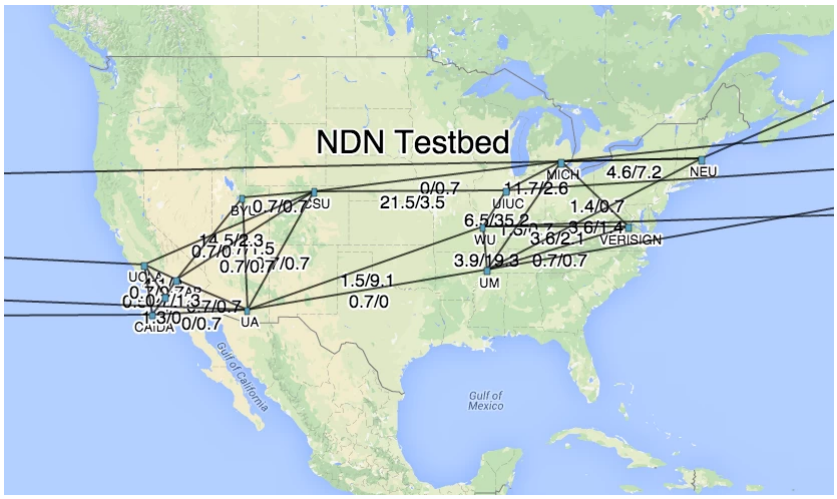

Figura 3.4: Porzione della piattaforma NDN Testbed situata in America.

ndnSIM: NdnSIM è un simulatore basato su ns-3, che permette di programmare scenari di rete ed eseguirli su un singolo calcolatore. Esso consente di definire il comportamento di ogni singolo nodo e di osservarne il comportamento tramite un Logger o un Visualizer. NdnSIM offre anche la possibilità di generare automaticamente grafici che mostrino nel dettaglio le statistiche di rete.

Visto che l'intero sistema è stato sviluppato su ndnSIM, nelle seguenti sezioni si osserveranno gli strumenti da esso offerti. Si parte da una breve introduzione del simulatore ns-3.

3.5 Ns-3

Ns-3 [3] [2] è un simulatore di rete ad eventi discreti, destinato principalmente alla ricerca e all'utilizzo educativo. Il progetto è impegnato nella costruzione di un solido nucleo di simulazione ben documentato, di facile debug e che soddisfa le esigenze dell'intero flusso di lavoro, dalla configurazione alla raccolta dati. Il design del simulatore è studiato per supportare simulazioni su reti IP ma una buona estensibilità lo rende in grado di lavorare anche con altri protocolli (ndnSIM ne è un esempio).Esso dispone di uno scheduler Realtime ingrado di gestire in maniera dinamica gli eventi istanziati dai nodi che compongono la topologia di rete.

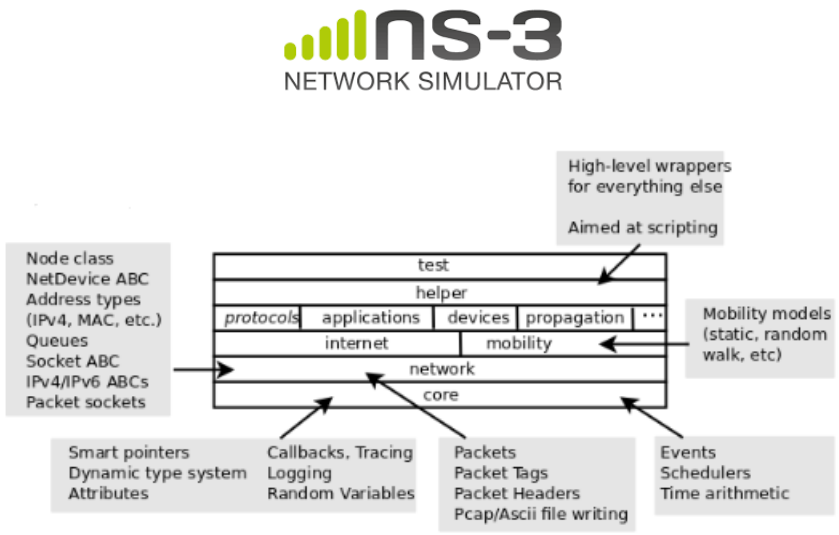

Figura 3.5: Logho e struttura del simulatore ns-3.

La struttura è simile a quella di una libreria che può essere collegata staticamente o dinamicamente a un programma principale C++ che definisce la topologia e lancia la simulazione. Tutte le API sono esportate anche in Python. Le fondamenta di ns-3 sono racchiuse nei moduli Core e Network dove vengono implementate genericamente le caratteristiche fondamentali di un simulatore di rete (non solo reti IP); Core descrive gli aspetti fondamentali di qualunque protocollo mentre Network definisce la struttura dei pacchetti. Proprio sopra questi due livelli viene integrato ndnSIM. Questa solida base permette ad ns-3 di andare a sviluppare nei livelli superiori aspetti di maggiore dettaglio come applicazioni, dispositivi di rete, modelli di movimento e modelli di propagazione. Infine, ns-3 sviluppa una serie di convenienti API wrapper chiamati Helper che facilitano notevolmente le chiamate a primitive di basso livello.

3.5.1 Panoramica

Ns-3 fornisce diversi strumenti che facilitano notevolmente lo sviluppo di simulazioni di rete e permettono di ottenere dei risultati notevolmente realistici. Se ne delineano alcuni fra più importanti:

- **Simulatore e Eventi:** il simulatore prevede l'esecuzione di eventi in momenti specificati dallo sviluppatore. All'inizio del programma si ha una coda di eventi ordinati cronologicamente; essi vengono eseguiti uno alla volta e al termine di ciascuna esecuzione si passa all'evento successivo. Una volta terminata la coda si conclude la simulazione. Per comprendere al meglio il funzionamento del simulatore è bene specificare il significato di "simulatore ad eventi discreti". Procediamo con un esempio: se la coda contiene due eventi, uno a 100 secondi e uno a 200, il simulatore esegue il primo per poi passare immediatamente al secondo, senza alcuna attesa temporale.

  La classe Simulator è il punto di accesso pubblico alla pianificazione di eventi, istanziabili grazie al metodo Simulator::Schedule(). Vediamo la pianificazione di un evento da eseguire fra 10 secondi:

```
97  void handler (int arg0, int arg1){
98    //handLer routine
99  }
100 //Event planning
101 EventId eventId = Simulator::Schedule(Seconds(10),&handler,this,arg0,arg1);
102 Simulator::Remove(eventId);
```

  L'esempio mostra secondo quale modalità sia possibile istanziare un evento. I parametri richiesti dal metodo sono: delay di esecuzione, il riferimento al metodo handler(), contesto di esecuzione e gli argomenti del metodo (cinque è il numero massimo consentito dal simulatore). Il metodo Schedule() restituisce un EventiID che identifica l'evento. Come mostrato nell'esempio, esso può essere utilizzato per rimuovere l'evento tramite il metodo Remove().

- **Numeri Random:** il Simulatore permette di schedulare eventi indicandone l'istante di esecuzione tramite oggetti della classe ns3::Time. Dato che il tempo viene trattato in maniera discreta è possibile incontrare casi di simultaneità che nell'ambito delle reti non hanno un comportamento definito. In modo da ovviare il più possibile a questo problema ns-3 fornisce un generatore di numeri pseudo-casuali, grazie ai quali è possibile istanziare eventi con cadenza randomica;

```
86  //Random variable between 0 and 100
87  Ptr<UniformRandomVariable> rand = CreateObject<UniformRandomVariable>();
88  double value = rand->GetValue(0,100);
```

- **Log:** la gestione dei log viene resa possibile dalla variabile globale NS_LOG. Questa strategia permette di monitorare e di eseguire il debug delle simulazioni per qualunque modulo in essa coinvolto. Dal punto di vista implementativo ns-3 fornisce una serie di macro che permettono di differenziare i log in base alla loro categoria (NS_LOG_INFO(), NS_LOG_ERROR() ecc..);

```
94  //Logging macro
95  NS_LOG_INFO("Log info");
96
```

- **Visualizer:** ns-3 offre anche esperienze di debug basate sul visualizer PyViz. Esso si può rivelare particolarmente utile nel caso in cui si voglia verificare il comportamento di modelli di mobilita oppure controllare che i pacchetti raggiungano la giusta destinazione. I nodi possono essere posizionati in spazi di due o tre dimensioni. Nell'immagine [3.6] se ne osserva l'interfaccia grafica.

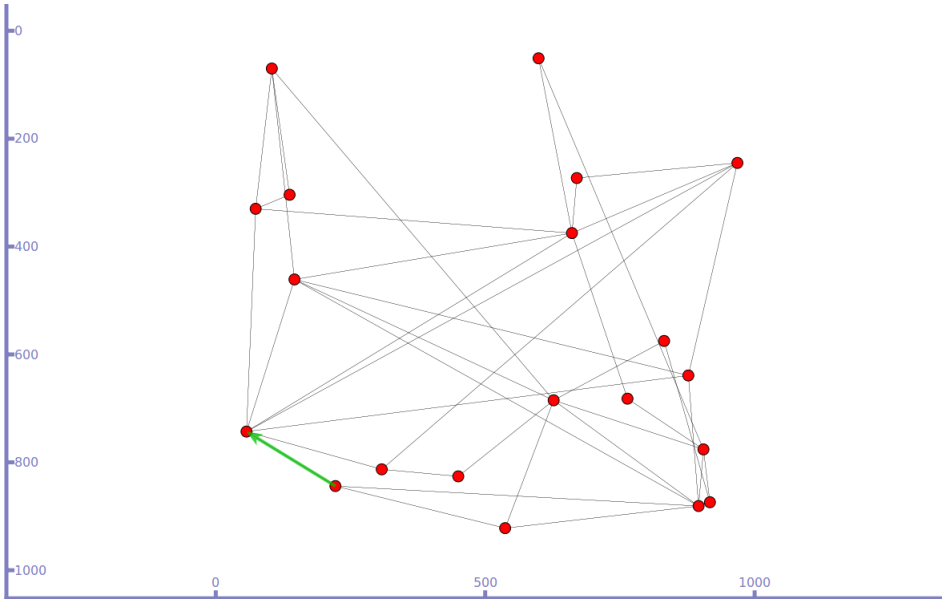

Figura 3.6: Interfaccia grafica del visualizer PyViz.

Per via dell'enorme numero di funzionalità fornite da ns-3 si è scelto di focalizzarsi principalmente sugli strumenti impiegati nell'elaborato. Questa sezione, inoltre, tralascia volutamente alcune componenti trattate nelle prossime sezioni.

3.6 ndnSIM

NdnSIM [6] [5] [6] è il simulatore di reti Named Data Networking(NDN) che attualmente raggiunge la versione 2.6. Nell'immagine [3.7] viene mostrato come esso si integri al simulatore ns-3:

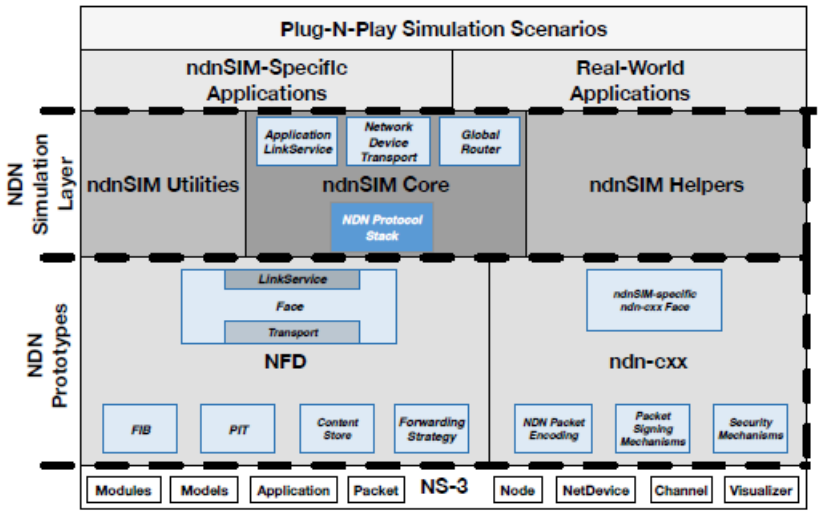

Figura 3.7: Integrazione di ndnSIM nel simulatore ns-3.

Alla base della struttura si trova ns-3 a cui si sovrappongono direttamente Network Forwarding Deamon e la libreria ndn-cxx (NDN C++ con eXperimental eXtensions). Visto che le simulazioni si appoggiano direttamente su NFD, per portare il codice sviluppato su una macchina fisica sono sufficienti alcune piccole modifiche. Immediatamente sopra troviamo, il livello di simulazione in cui ndnSIM integra Core, Utilies e Helpers. Il modulo Core, rappresenta il cuore di ndnSIM che racchiude il Protocol Stack NDN, l'Application LinkService, il Network Device Transport e il Global Router. Questa struttura fornisce gli elementi necessari allo sviluppo delle simulazioni.

3.6.1 Scenario applicativo

Per illustrare, in modo più chiaro possibile, come venga creato uno scenario si è scelto di mostrare alcuni esempi di codice, che fanno riferimento ad un'unica topologia di rete.

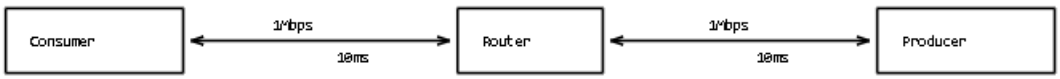

Figura 3.8: Esempio topologia scenario.

La topologia mostrata in figura [3.8] riguarda una semplice connessione point-to-point fra un Consumer e un Producer che sono collegati ad un router centrale. Un Consumer è un nodo che manifesta interessi alla rete, mentre il Producer è un nodo che prende l'incarico di soddisfarlo. In seguito si osservano le sezioni di codice più importanti:

- **Prestazioni di rete:** per rendere la simulazione simile ad una situazione reale, viene predisposta la tecnologia di rete, andandone a specificare le prestazioni(frequenza, delay e numero massimo di pacchetti).

```
4  // Setting default parameters for PointToPoint links and channels
5  Config::SetDefault("ns3::PointToPointNetDevice::DataRate", StringValue("1Mbps"));
6  Config::SetDefault("ns3::PointToPointChannel::Delay", StringValue("10ms"));
7  Config::SetDefault("ns3::QueueBase::MaxPackets", UintegerValue(10));
```

- **Nodi e topologia di rete**: il simulatore prevede che tutti i nodi siano contenuti in un oggetto della classe NodeContainer, la quale permette di definirne il numero grazie al metodo Create(). Impiegando i nodi istanziati è possibile definire la struttura della topologia di rete, che nell'immagine [3.8] è una semplice p2p. E' bene precisare che le funzionalità offerte da ns-3 permettono di creare topologie di rete molto più articolate rispetto a quella presa in esame.

```
18 // Creating nodes
19 NodeContainer nodes;
20 nodes.Create(3);
21
22 // Connecting nodes using two links
23 PointToPointHelper p2p;
24 p2p.Install(nodes.Get(0), nodes.Get(1));
25 p2p.Install(nodes.Get(1), nodes.Get(2));
```

- **Installazione NDN stack**: per permettere ai nodi di operare secondo il protocollo NDN è necessario installare su ciascuno di essi il relativo stack di rete; ciò è reso possibile grazie alla classe ndn::StackHelper.

```
27  // Install NDN stack on all nodes
28  ndn::StackHelper ndnHelper;
29  ndnHelper.SetDefaultRoutes(true);
30  ndnHelper.InstallAll();
```

- **Forwarding Strategy**: La classe ndn::StrategyChoiceHelper permette di scegliere la strategia di forwarding più consona al sistema che si intende realizzare. Il simulatore offre solo quattro strategie predefinite, ma fornisce anche gli strumenti che ne semplifichino lo sviluppo personale. Le strategie predefinite sono:
  - **best-route:** inoltra i pacchetti verso il nodo successivo con costo minore;
  - **ncc:** propone una reinterpretazione della strategia NCCx 0.7.2;
  - **multicast:** inoltra i pacchetti verso tutti gli upstream corrispondenti alla FIB fornita;
  - **client-control:** la strategia permette al Consumer di scegliere l'interfaccia d'uscita per ciascun interesse;

  Nell'esempio seguente viene scelta la strategia multicast.

```
32  // Choosing forwarding strategy
33  ndn::StrategyChoiceHelper::InstallAll("/prefix", "/localhost/nfd/strategy/multicast");
```

- **Installazione applicazioni**: la classe ndn::AppHelper permette di installare applicazioni differenti sui singoli nodi. Nello scenario proposto vengono installate le applicazioni ns3::ndn::ConsumerCbr e ns3::ndn::Producer; la prima mostra interessi alla rete ad una frequenza fissa mentre la seconda si occupa semplicemente di soddisfarli. Le seguenti porzioni di codice mostrano l'installazione degli applicativi;

```
35  // Installing applications
36  // Consumer
37  ndn::AppHelper consumerHelper("ns3::ndn::ConsumerCbr");
38  // Consumer will request /prefix/0, /prefix/1, ...
39  consumerHelper.SetPrefix("/prefix");
40  consumerHelper.SetAttribute("Frequency", StringValue("10")); // 10 interests a second
41  consumerHelper.Install(nodes.Get(0));                        // first node
```

```
// Producer
ndn::AppHelper producerHelper("ns3::ndn::Producer");
// Producer will reply to all requests starting with /prefix
producerHelper.SetPrefix("/prefix");
producerHelper.SetAttribute("PayloadSize", StringValue("1024"));
producerHelper.Install(nodes.Get(2)); // Last node
```

- **Istruzioni finali**: Una volta effettuate tutte le configurazioni è possibile definire il periodo di simulazione; a seguito di questa operazione vengono richiamati i metodi Simulator::Run() e Simulator::Destroy() che avviano e rimuovono la simulazione;

```
Simulator::Stop(Seconds(20.0));

Simulator::Run();
Simulator::Destroy();
```

In seguito vengono mostrati gli strumenti forniti da ndnSIM per il calcolo della FIB e la gestione del CS:

- **Calcolo FIB**: Nelle topologie di rete più articolate è necessario attuare la strategia di routing, che permette la costruzione della Forwarding Information Base. Ciò è reso possibile grazie all'Helper della classe GlobalRoutingHelper, che dopo essere stata installata sui nodi della rete permette di costruire la FIB tramite il metodo CalculateRoutes();

```
// Installing global routing interface on all nodes
ndn::GlobalRoutingHelper ndnGlobalRoutingHelper;
ndnGlobalRoutingHelper.InstallAll();

//Network configuration....

// Add /prefix origins to ndn::GlobalRouter
ndnGlobalRoutingHelper.AddOrigins(prefix, producer);
// Calculate and install FIBs
ndn::GlobalRoutingHelper::CalculateRoutes();
```

- **Poliche di caching:** Quando il Content Store raggiunge una dimensione eccessiva, i nodi di rete sono costretti a sostituire dei dati. In questo contesto si delineano Policy che adottano strategie differenti; quelle offerte da ndnSIM sono le seguenti:
  - **LRU(default):** sostituisce il dato utilizzato meno recentemente;
  - **FIFO:** sostituisce il dato memorizzato in cache da più tempo;
  - **LFU:** sostituisce il dato utilizzato meno frequentemente;

- **Random:** sostituisce il dato randomicamente;
- **Nocache:** disattiva completamente il caching;

La configurazione è permessa dai metodi SetCsSize() e SetPolicy() della classe ndn::StackHelper;

```
// Install NDN stack on all nodes
ndn::StackHelper ndnHelper;
ndnHelper.setCsSize(100);
ndnHelper.setPolicy("nfd::cs::lru");
ndnHelper.InstallAll();
```

3.6.2 Applicazioni personalizzate

Per gli sviluppatori che intendono realizzare un'applicazione personalizzata, ndnSIM offre la classe ndn::App. Tramite l'estensione di questa classe, che definisce un comportamento di base, è possibile specificare la reazione ai principali eventi di rete, ovvero ricezione dati(OnData) e ricezione interessi(OnInterest).

```
#include "ns3/ndnSIM/apps/ndn-app.hpp"

namespace ns3 {

class CustomApp : public ndn::App {
public:
  // register NS-3 type "CustomApp"
  static TypeId GetTypeId();

  // (overridden from ndn::App) Processing upon start of the application
  virtual void StartApplication();

  // (overridden from ndn::App) Processing when application is stopped
  virtual void StopApplication();

  // (overridden from ndn::App) Callback that will be called when Interest arrives
  virtual void OnInterest(std::shared_ptr<const ndn::Interest> interest);

  // (overridden from ndn::App) Callback that will be called when Data arrives
  virtual void OnData(std::shared_ptr<const ndn::Data> contentObject);

private:
  void SendInterest();
};

}
```

Figura 3.9: Intefaccia CustomApp che estende ndn::App.

Capitolo 4

NDN-enabled edge computing protocol

4.1 Introduzione

Con il progresso dell'Internet of Things(IoT) le unità di calcolo vengono sempre più integrate a qualunque tipo di oggetto. Questa tendenza, sta portando alla proliferazione di dispositivi sempre più specializzati, ai quali si da la possibilità di cooperare scambiando e condividendo dati. La progettazione dei sistemi embedded chiede di destare particolare attenzione a fattori come peso e dimensioni, che influiscono inevitabilmente sulla capacità di calcolo e sulla memoria; non a caso questi dispositivi hanno capacità solitamente limitate. L'intento del protocollo è quello di ampliare la collaborazione di questi sistemi, permettendo loro lo scambio reciproco di task. Anche se questa funzione non migliora le capacità dei singoli dispositivi, permette di accrescere notevolmente la complessità dei compiti assegnati all'insieme. Questo approccio si rivela particolarmente utile anche nel caso in cui un calcolatore debba allocare più task di quelli concessi dalle proprie capacità computazionali; quando esse sono sature il calcolatore potrà comunicare con gli altri dispositivi in rete per trovane uno in grado di soddisfare la propria richiesta.

Il primo obiettivo che si prefigge questa tesi è quello di ideare un protocollo che permetta ai nodi di una rete NDN di cooperare scambiandosi l'esecuzione di task. Esso impiega un algoritmo già in parte sviluppato dal professor. Giovanni Pau nell'ambito del progetto V-NDN [4]. Il secondo step è quello di impiegare il protocollo ideato nello sviluppo di un applicativo che può essere installato sui nodi del simulatore ndnSIM. Infine si effettuano alcun test che aiutino a comprendere il comportamento del sistema permettano la stesura di alcuni grafici.

4.2 Evoluzione del modello Cloud

L'idea che sta alla base del protocollo inserisce perfettamente nel contesto del Fog [1] e dell'Edge[8] computing che rappresentano l'evoluzione del modello Cloud. Per capire meglio quali siano le differenze fra questi approcci se ne osserveremo gli aspetti fondamentali. L'attuale architettura Cloud viene ideata raccogliere molteplici servizi di rete. Questi ultimi possono riguardare diverse tipologie come Software as a Service(SaaS), Data as a Service(DaaS) e Hardware as a Service(HaaS). Un esempio concreto è rappresentato dal servizio iCloud di Apple, che permette agli utenti di sincronizzare i loro dati su diversi dispositivi grazie ad uno storage di servizi centralizzato. Il Cloud computing offre numerosi vantaggi ,sia agli utenti che alle imprese, tuttavia presenta ancora diversi limiti. Esso non si rivela efficace nelle applicazioni sensibili alla latenza, che richiedono nodi nelle vicinanze in grado di soddisfare i loro requisiti di ritardo. Oggi un numero notevole di implementazioni su Internet, in particolare nell'ambito Internet of Things, richiede supporti che vanno in netto contrasto con questo modello. Tra i più importanti vi sono: la latenza, la mobilità, la geo-distribuzione e la geo-localizzazione. Per questo motivo è sorta l'esigenza di trovare una strategia che fosse in grado di affrontare questi problemi. Il Fog computing sembra avere tutte le carte in regola per affrontare questo tema. Esso è un livello architetturale che si interpone fra Cloud e Internet delle cose.

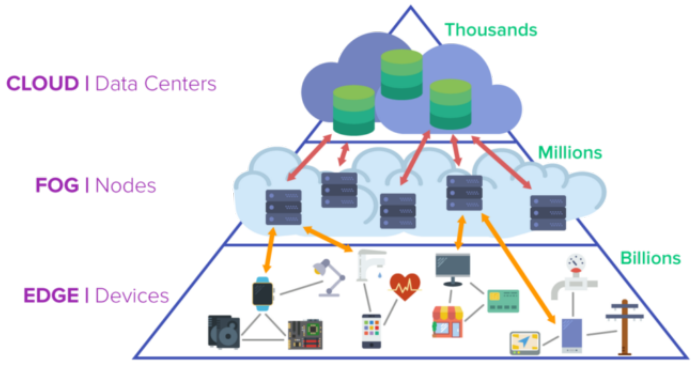

Figura 4.1: Architetture Cloud, Fog e Edge computing.

Tuttavia è bene sottolineare gli aspetti che lo differenziano da una semplice estensione del Cloud. Il primo fra questi è sicuramente la più ampia distribuzione geografica, che oltre a fornire una minore latenza facilita il supporto per la mobilità. La vicinanza dei nodi della "nebbia", risulta un grosso vantaggio la dove le applicazioni debbano distinguere l'identità di un dispositivo dalla sua posizione geografica. Infine, un altro aspetto che facilita il movimento è la prevalenza di comunicazioni wireless, le quali permettono terminali di comunicare tra loro ma allo stesso tempo di trasmettere informazioni ai nodi del Fog. Infine, è dai dispositivi dell'Internet delle cose che nasce il concetto di Edge

computing. Questo paradigma coinvolge tutte le computazioni svolte ai margini della rete tra cui si può evidenziare il calcolo cooperativo. Concettualmente l'Edge computing va ad abbracciare tutti i task bassa latenza, aspetto sarebbe difficile curare con il solo Cloud.Il protocollo realizzato si colloca proprio nel campo dell'Edge computing.

4.3 Analisi

4.3.1 Requisiti del protocollo

In funzione di quanto esposto nella sezione precedente si è deciso che le comunicazioni dovessero avvenire tramite wifi. Inoltre, per semplificare la trasmissione si è scelto che tutti i dispositivi siano in ascolto su unico canale condiviso. Lo scenario da cui parte la realizzazione dell'elaborato riguarda una rete di dispositivi sui quali è installato lo stack NDN. Le unità di calcolo possono essere dotati di sensori e svolgono diversi task in autonomia. Un dato su cui si concentra l'attenzione è l'occupazione percentuale della CPU. Supponiamo che, come in figura [4.2] uno dei nodi appartenenti alla rete raggiunga il 100% di CPU ma abbia la necessità di svolgere un altro task che ne richiede il 10%; l'intento è quello di creare un protocollo di comunicazione che permetta a qualunque nodo incontri lo stesso problema di affidare lo svolgimento del task ad un dispositivo nelle vicinanze. Chiamo Customer i nodi che presentano richiesta alla rete ed Helper il nodi che si propongono per l'esecuzione dell'incarico.

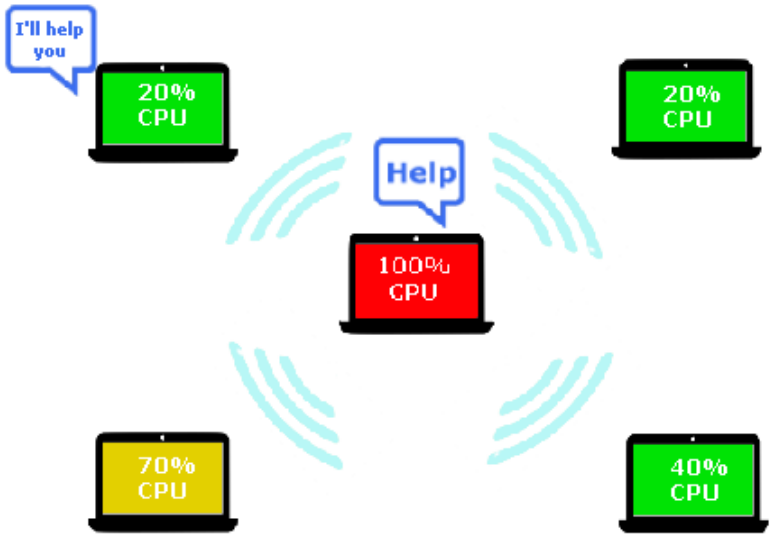

Figura 4.2: Rappresentazione richiesta di aiuto.

Si stila la lista di linee guida da rispettare nella realizzazione del protocollo:

- **Restrizione dei ruoli:** i nodi appartenenti alla rete possono presentare e soddisfare richieste senza alcun tipo di limitazione. L'unico vincolo è associato ai Customer, i quali avendo presentato una richiesta di aiuto alla rete non possono assumere il ruolo di Helper.
- **Molteplicità delle richieste:** i Customer possono gestire più richieste contemporaneamente;
- **Molteplicità degli incarichi:** gli Helper possono prendere in carico più task contemporaneamente;
- **Criterio di adesione:** i nodi possono proporsi per la risoluzione del task solo se riescono a soddisfare l'impegno percentuale richiesto. Supponiamo che un nodo con CPU al 90% riceva una richiesta del 20%. In queso caso, la richiesta supera le reali capacità del nodo che non potrà accettare l'incarico.
- **Bilanciamento del carico:** il protocollo seguito dai nodi della rete deve permettere il bilanciamento dei carichi di lavoro. In altre parole esso deve fare in modo che gli incarichi vengano assegnati ai nodi col minor carico computazionale.
- **Prestazioni:** la fase d'accordo Customer ed Helper deve essere effettuata nel minor tempo possibile e l'Helper dove eseguire il task entro un periodo prestabilito.
- **Perdita di pacchetti:** appoggiandosi su un canale di comunicazione inaffidabile, il protocollo deve essere in grado di adattarsi all'eventuale perdita di pacchetti.
- **Univocità dell'incarico:** i Customer possono scegliere uno solo dei nodi appartenenti alla rete.
- **Baso traffico di rete:** il protocollo dovrà cercare di limitare il più possibile il traffico di rete.

Si precisa che queste linee guida offrono un prospetto di una versione stabile e completa del protocollo. Nonostante gran parte delle funzionalità siano state implementate, diversi aspetti sono ancora da migliorare e perfezionare.

4.4 Design del protocollo

4.4.1 Algoritmo di elezione

L'illustrazione dell'algoritmo parte dallo scenario in figura [4.3] , in cui sono raffigurati i calcolatori 'A','B','C' e 'D'. Ciascuno di essi sta gestendo un proprio calcolo computazionale e il nodo 'D' ha raggiunto il 100% della CPU. Esso deve comunicare con gli altri dispositivi, per capire quale di essi stia gestendo il minor carico computazionale. E' possibile comprendere in maniera molto semplice che l'esecuzione del task venga assegnata al nodo 'A', tuttavia 'D' non possiede informazioni sugli altri nodi della rete e per poter identificare un possibile helper deve necessariamente interrogarli.

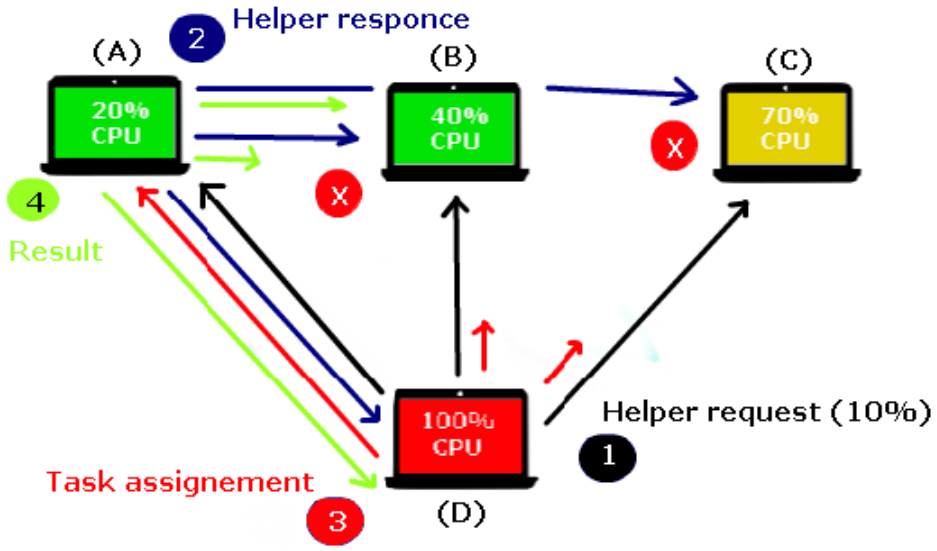

Figura 4.3: Scenario di partenza dell'algoritmo.

L'algoritmo si compone di 4 step principali:

1. **Richiesta di aiuto:** i nodi della rete che necessitano che qualcun altro esegua un task per loro, mandano in broadcast una richiesta di aiuto, che viene ricevuta da tutti i nodi nelle vicinanze;

2. **Risposta con forwarding timeout:** i nodi che possono soddisfare la richiesta si "candidano" come possibili Helper. La risposta non è però immediata, ma avviene allo scadere del Forwarding Timeout(FT). Proprio in questo meccanismo di attesa risiede l'aspetto fondamentale dell'algoritmo. Il periodo ad esso associato, è inversamente proporzionale alla CPU libera del calcolatore, così che i nodi con minor

carico computazionale riescano a rispondere più velocemente rispetto a quelli con carico maggiore. In figura [4.3] abbiamo che FreeCPU(A)=80, FreeCPU(B)=60 e FreeCPU(C)=30. Supponendo che il ritardo massimo sia di 1s, si hanno i seguenti valori di FT:

- FT(A) = 1s/80 = 0,0125s;
- FT(B) = 1s/60 = 0,0166s;
- FT(C) = 1s/30 = 0,0333s;

Seguendo questa strategia il nodo 'A' risponde alla richiesta per primo. Il messaggio di risposta ha una duplice funzione, poichè va a bloccare i nodi che si trovano ancora in attesa del timeout. Questa strategia permette di alleggerire notevolmente il traffico di rete dato che evita le risposte dei nodi 'B' e 'C';

3. **Assegnazione del task:** il Customer non deve fare altro che assegnare il compito al primo nodo da cui riceve risposta. Per permettere l'accordo via broadcast e utilizzando il protocollo NDN verranno aggiunti dei metadati ai pacchetti scambiati. Il ruolo di queste informazioni verranno presentati nelle prossime sezioni.

4. **Esecuzione del task e consegna risultato**: una volta ricevuto l'incarico, l'Helper procede con l'esecuzione del task alla fine del quale risponderà con il risultato.

Questi step, rappresentano i passi fondamentali dell'algoritmo, impiegati nella realizzazione del protocollo . Essi non implicano l'utilizzo di un protocollo di rete specifico, anzi, potrebbero essere utilizzati anche con protocolli diversi da NDN.

4.4.2 Modello Interesse-Dato

Lavorando con il protocollo NDN è stato necessario adottare il modello Interesse-Dato che implica un approccio alla comunicazione e allo scambio di dati completamente diverso. Mentre il protocollo IP permette operare in modalità PUSH, con NDN è possibile lavorare solo con approccio PULL. Dall'analisi del contesto sono emerse due possibili strategie:

- **Modello di candidatura:** questa strategia prevede che il Customer lavori inviando unicamente interessi. Il Customer avvia il protocollo chiedendo ai nodi della rete un loro "pacchetto di candidatura" e riceve risposta dai nodi che possono prendere in carico il lavoro. Una volta individuato l'Helper, il Customer procede inviando l'interesse per il risultato, a cui il nodo incaricato risponde dopo lo svolgimento del lavoro;

- **Modello con certificato di incarico:** questa strategia prevede che il primo interesse inviato dal Customer riguardi direttamente il risultato. I nodi che intendono candidarsi non rispondono con un pacchetto dati, ma "rimbalzano" un interesse che ha come oggetto il "certificato di assunzione". Una volta individuato l'Helper, il Customer rispondere con il pacchetto che ne certifica l'assunzione. Infine, dopo aver eseguito il task, l'Helper consegna il pacchetto dati contenente il risultato;

La scelta tra queste due strategie è ricaduta sul modello a "certificato di incarico". Si precisa che non vi è una motivazione particolare, semplicemente essa è sembrata una soluzione più elegante ed in grado di incapsulare la richiesta in maniera solida.

4.4.3 Caso Base

In questa sezione si analizzano nel dettaglio interessi e dati scambiati per il raggiungimento dell'accordo. Per ciascun pacchetto si illustra la struttura del nome, andandone a spiegare il significato dei singoli parametri.

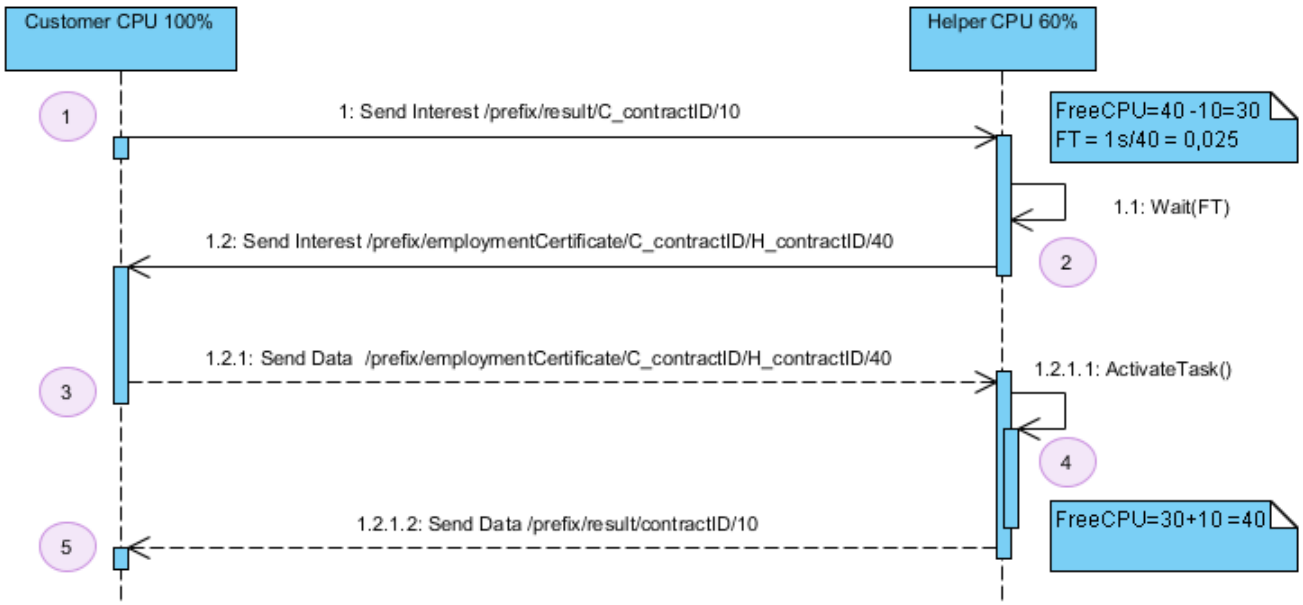

Figura 4.4: Diagramma di sequenza del caso base senza perdita di pacchetti.

In figura [4.4] sono mostrati i pacchetti scambiati da un Customer e dall'Helper a cui viene affidata l'esecuzione del task. Per difficoltà di raffigurazione non vengono mostrati altri nodi che avrebbero potuto soddisfare la richiesta.

1. **Result Interest:** '/prefix/result/C_contractID/10' è un esempio di nome che può essere assunto dall'interesse che avvia la procedura di accordo. Esso presenta diversi campi suddivisi dal carattere '/'. Si mostra ciascuno di essi nel dettaglio:

   - **Prefisso:** il campo "prefix" indica un prefisso generico. Nell'esempio esso non assume nessuna funzionalità;
   - **Result:** il campo "result" indica l'oggetto dell'interesse. Esso chiede in maniera esplicita il risultato del task che può essere eseguito dall'Helper solo dopo aver raggiunto l'accordo e aver concretamente eseguito il task;
   - **Customer Contract ID:** al riscontro di una necessità il nodo crea un Customer Contract, identificato dal " C_contractID". Il campo composto da due parti: una che identifica il nodo e una che identifica il contratto. La duplice informazione permette ai nodi di rete di prendere in considerazione solamente i pacchetti broadcast che li riguardano. Il concetto di Customer Contract viene ampliato nel prossimo capitolo;
   - **Carico della Richiesta:** il valore "10" fornisce un'informazione quantitativa sul carico della richiesta. Esso è stato assegnato trascurando volutamente il fatto che in uno scenario reale il carico computazionale dipenda dall'hardware della macchina;

   Il Customer, una volta inviato l'interesse, imposta un timeout al cui scadere viene inviato nuovamente lo stesso interesse. Questo periodo viene definito in maniera tale da dare il tempo all'Helper di presentare la propria candidatura.

2. **Certificate Interest**: i nodi che ricevono l'interesse controllano se possono soddisfare il carico da esso indicato. In caso positivo, riservano il carico di CPU richiesto per poi procedere alla candidatura. La risposta avviene a seguito di un Forwarding Timeout, il cui periodo viene calcolato sulla base della CPU libera all'istante di ricezione della richiesta. '/prefix/employmentCertificate/C_contractID/H_contract/40/timestar è il nome dell'interesse di risposta, di cui andiamo ad analizzare i campi:

   - **Prefisso:** il campo "prefix" indica un prefisso generico. Come per l'interesse riguardante il risultato non assume alcuna funzionalità;
   - **Emplyement Certificate:** il campo "employementCertificate" indica che l'oggetto dell'interesse è un certificato di incarico. Se il nodo riceve risposta a questo interesse sa che il compito è stato assegnato a lui;
   - **Customer Contract ID:** il campo "C_contractID" viene aggiunto perchè l'interesse possa essere interpretato unicamente al relativo Customer. Il valore equivale a quello che il campo assume nell'interesse '/prefix/result';

- **Helper Contract ID:** come il Customer, anche l'Helper istanzia il proprio Helper Contract che viene identificato dal campo "H_contractID".Il campo composto da due parti: una che identifica il nodo e una che identifica il contratto di candidatura. Questo identificativo ha funzione analoga a quella assunta da "C_contractID" e permette all'Helper di capire quale pacchetto contente un certificato sia indirizzato unicamente a lui. Il concetto di Helper Contract viene ampliato nel prossimo capitolo;
- **CPU libera:** il campo indica la CPU libera dell'Helper. Dato che il Customer riceve quest'interesse dal nodo con il carico minore esso può rappresentare un feedback sullo stato della rete. Questo valore è stato introdotto per eventuali sviluppi futuri ma non è stato impiegato in alcun modo nell'applicativo;

Una volta inviato l'interesse, l'Helper imposta un timeout al cui scadere l'Helper Contract viene annullato. Naturalmente questo periodo è definito in maniera tale da dare il tempo al Customer di rispondere con il pacchetto dati. Allo scadere del timeout viene rilasciata la CPU riservata per il contratto.

3. **Employment Certificate Data**: alla ricezione dell'interesse per il certificato di incarico, il Customer risponde con il pacchetto dati corrispondente. Negli sviluppi futuri del protocollo è previsto che il contenuto possa trasportare i parametri di esecuzione del task. Questo pacchetto sancisce l'assegnazione del compito. Dato che da questo momento il Customer si mette in attesa del risultato viene impostato un timer il cui periodo è superiore al tempo di esecuzione del task. Se allo scadere di questo timer non si è ancora ricevuto il risultato allora viene riavviata da capo la procedura di accordo. E' bene specificare che in questa fase potrebbe essere adottato un meccanismo che permetta di capire se l'esecuzione del task in ritardo, tuttavia si è preferito lasciare questo aspetto ad eventuali sviluppi futuri;

4. **Result Data**: Alla ricezione del certificato di incarico l'Helper procede con l'esecuzione del task; una volta completato esso risponde con il pacchetto dati contenente il risultato. Infine, viene rilasciata la CPU riservata per il contratto;

5. **Ricezione risultato**: alla ricezione del risultato la richiesta è considerata definitivamente conclusa.

4.4.4 Adattamento alla perdita di pacchetti

Si osservano le principali strategie adottate dal sistema per adattarsi alla perdita di pacchetti:

Ritrasmissione interesse risultato: il customer da cui nasce la richiesta, dopo avere trasmesso l'interesse '/prefix/result...' imposta un timer che definisce il periodo in cui esso si aspetta risposta da parte di un Helper. Se, al termine di questo periodo, nessun Helper ha risposto allora il Customer ritrasmette l'interesse. Il diagramma nell'immagine [4.5] mostra come avvenga questa ritrasmissione.

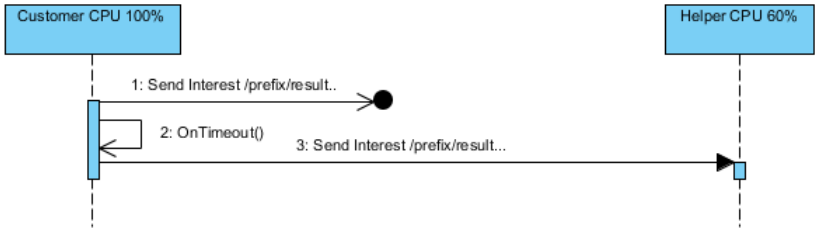

Figura 4.5: Esempio di perdita dell'interesse per il risultato.

Lo stesso meccanismo permette al sistema di adattarsi anche alla perdita di interessi per il certificato di assunzione. L'immagine [4.6] ne mostra l'esempio. Si è scelto di non ritrasmettere l'interesse per il certificato perchè avrebbe introdotto troppo traffico di rete. Se si adottasse questa soluzione si otterrebbe una ritrasmissione ogni qual volta un nodo non sia scelto per l'esecuzione di un task.

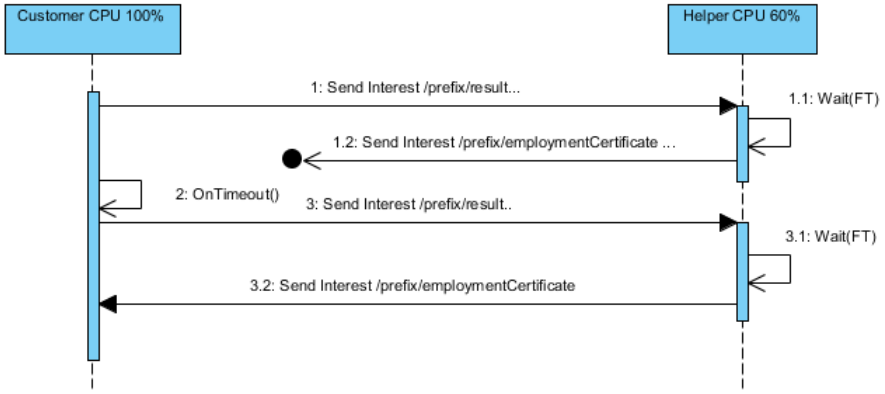

Figura 4.6: Esempio di perdita dell'interesse per il certificato di assunzione.

La gestione della perdida del certificato di incarico o del risultato, è una delle parti che ha destato maggiori difficoltà. L'applicativo realizzato per il contesto della tesi non offre ancora una buona gestione di queste perdite, tuttavia si sono delineate le basi concettuali che ne potrebbero rappresentare una soluzione.

Three Way Handshake: Per risolvere il problema ci si è ispirati al modello Three Way Handshake utilizzato dal protocollo TCP per instaurare il canale di comunicazione. Nel nostro caso esso viene impiegato per accertarsi che i pacchetti di dati scambiati siano stati correttamente consegnati al destinatario. Alla ricezione dei pacchetti dato (riguardanti certificato o risultato) i destinitari del pacchetto risponderanno con un interesse di acknowledgment(ack), che serve a notificarne la corretta ricezione.

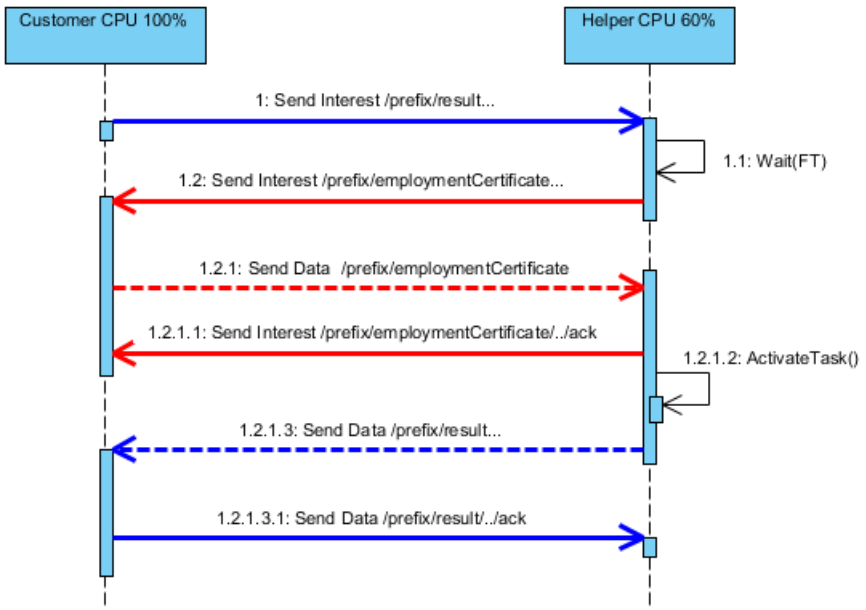

Figura 4.7: L'immagine mostra i due Three Way Handshake(TWH) impiegati dal protocollo. Il TWH relativo al risultato in colore blu, mentre quello per il certificato di incarico in rosso.

Nel caso in cui l'ack non venga ricevuto vi sono due possibilità: o è stato perso il pacchetto dati oppure è andato smarrito l'ack stesso. A prescindere dal pacchetto che è stato perso, se l'ack tarda troppo ad essere ricevuto, si procedere con la ritrasmissione del pacchetto dati. La procedura viene ripetuta fino a quando il mittente non ottiene la conferma dell'avvenuta ricezione. Nel diagramma [4.8] come il protocollo si adatta alla perdita del pacchetto dati e dell'ack.

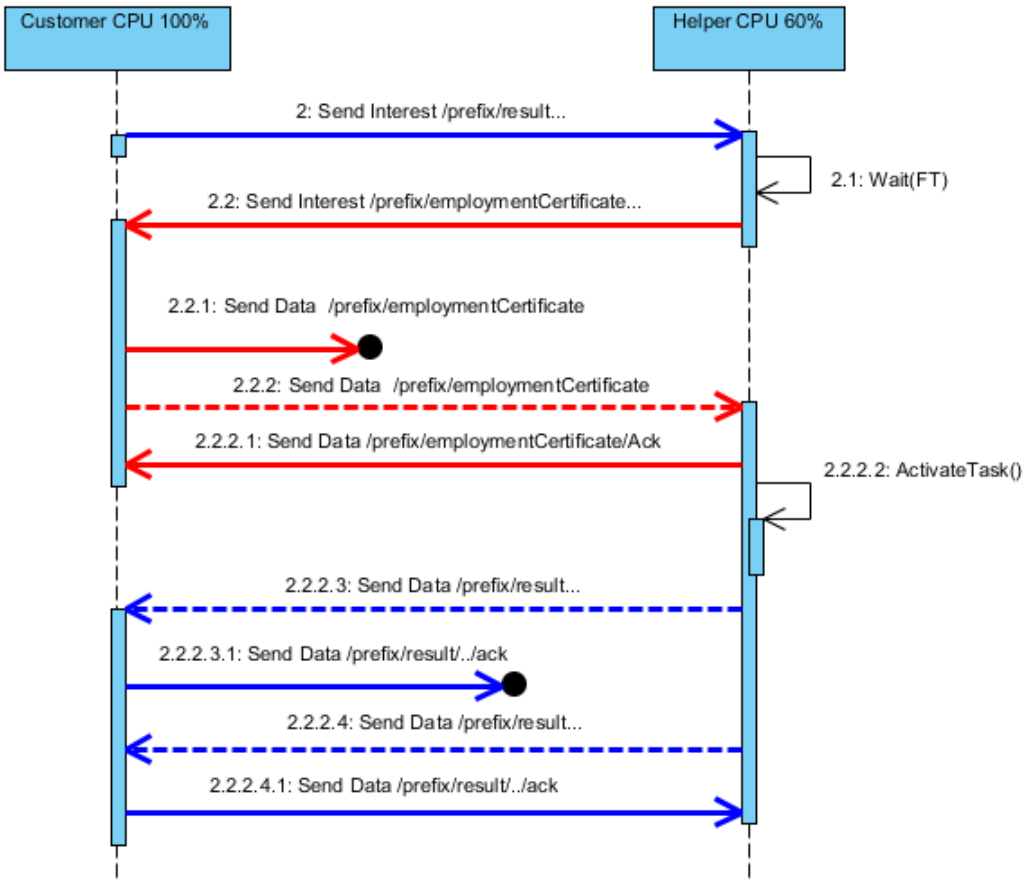

Figura 4.8: Adattamento del Tree Way Handshake alla perdita del pacchetto dati e dell'ack.

4.4.5 Timelife degli interessi

Il protocollo vede il coinvolgimento di due tipi di interessi: uno per il risultato e uno per il certificato di incarico. Come previsto da NDN entrambi devono avere un proprio timelife che nel nostro caso deve essere assegnato ad hoc. Per l'interesse riguardante il certificato di incarico non si osservano particolari restrizioni, il suo timelife deve essere equivalere al tempo di risposta del Customer. E' necessario destare attenzione all'interesse riguardante il risultato, dato che il calcolo del suo timelife deve includere il periodo di accordo(assegnazione certificato) ma anche il periodo di esecuzione del task. Nel caso in cui al timelife venga assegnato un periodo e la scadenza avverrà prima della consegna del risultato, il quale non potrà più essere riconsegnato correttamente.

4.4.6 Intercettazione interessi certificato

Come in qualunque comunicazione radio, il wifi prevede che i messaggi scambiati sullo stesso canale vengano ricevuti da tutti i dispositivi nelle vicinanze. Nella fase di candidatura per esecuzione del task i nodi che rispondono più velocemente "bloccano" la candidatura di quelli ancora in attesa del forwarding timeout.

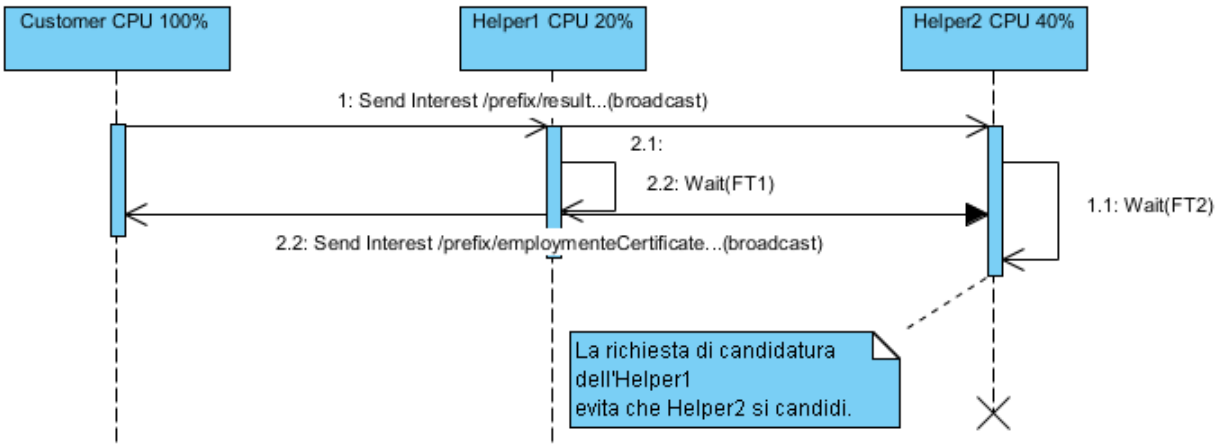

Figura 4.9: Adattamento del Tree Way Handshake alla perdita del pacchetto dati e dell'ack.

Nell'immagine [4.9] si osservano un Customer e due Helper. Gli Helper hanno rispettivamente impieghi del CUP del 20% e del 40%. Come definito dal protocollo il periodo di attesa(FT1) dell'Helper1 è minore dell'attesa(FT2) dall'Helper2; conseguentemente la candidatura dell'Helper1 viene ricevuta dall'Helper2 prima egli possa mostrare la propria. Dal messaggio esso comprende che qualcuno lo anticipato e rinuncia alla candidatura.

Capitolo 5

Sviluppo del protocollo su ndnSIM

5.1 Analisi dei requisiti

Si vuole realizzare un software che permetta a calcolatori dotati di un'interfaccia wifi di cooperare secondo le regole stabilite dal protocollo ideato. Esso può essere testato esclusivamente sui nodi del simulatore ndnSIM, che permette di valutarne il comportamento in diversi scenari che posso coinvolgere un numero di nodi differente. Nella versione raggiunta, l'applicativo offre la possibilità configurare i nodi di rete, definendone il carico computazionale tramite un semplice valore numerico; inoltre, può essere definito se essi debbano assumere ruolo di Customer o di Helper. Dato che l'interesse ricade principalmente sul corretto funzionamento del protocollo si è preferito simulare l'esecuzione del task definendo un periodo di occupazione della CPU. Il sistema predispone una struttura adeguata alla gestione di più accordi sia per i Customer che per gli Helper, anche se questa funzionalità è stata testata esclusivamente sugli Helper. Andiamo a osservare i risultati raggiunti per le singole linee guida del protocollo:

- **Restrizione dei ruoli:** il sistema è stato testato in modo che questa funzionalità fosse definita a priori;
- **Molteplicità delle richieste:** non testato;
- **Molteplicità degli incarichi:** gli Helper riescono a soddisfare un molteplice numero di richieste in maniera soddisfacente;
- **Criterio di adesione:** l'adesione alle richieste avviene esclusivamente se gli Helper possiedono le adeguate risorse computazionali;
- **Bilanciamento del carico:** il corretto bilanciamento del carico è stato osservato principalmente nelle simulazioni che coinvolgono pochi nodi. Con l'aumentare dei nodi e del traffico di rete il comportamento del sistema non aderisce perfettamente al comportamento sperato.

- **Prestazioni:** il sistema offre solitamente buoni tempi di risposta, eccezion fatta per i casi in cui si ha una perdita di pacchetti.

- **Perdita di pacchetti:** come specificato nel capitolo precedente, il sistema non offre ancora una buona gestione delle perdite ma si predispone per una semplice introduzione della funzionalità.

- **Univocità dell'incarico:** non si osservano casi in cui il task viene affidato a due Helper differenti.

- **Basso traffico di rete:** questo aspetto non è stato curato particolarmente. L'unica strategia attuata riguarda l'intercettazione di interessi per certificato di incarico.

Per via dei numerosi aspetti che devono essere ancora essere trattati adeguatamente, esso non può essere considerato un'applicativo completo ma ne rappresenta una buona base.

5.2 Design

5.2.1 Modello concettuale

L'applicativo realizzato ha come fulcro l'entità Worker a cui viene assegnata l'amministrazione dei task. Il Worker installato sui singoli nodi può comportarsi da Customer o da Helper in base alla configurazione ad esso assegnata. Tramite interessi broadcast, i Customer mostrano delle "opportunità di lavoro", che saranno assegnate agli Helper che presenteranno per primi la loro candidatura. Essi non fanno altro che instaurare degli accordi di prestazione che prevedono regole e scadenze sia per i Customer che per gli Helper. Questi concetti, finiscono in maniera molto minimale i moduli che verranno illustrati nelle prossime sezioni:

- **Worker:** la classe Worker definisce il comportamento dei nodi alla ricezione di ogni tipologia di pacchetto. Essa si occupa anche della gestione dei contratti.

- **Contract:** definisce i dettagli contrattuali comuni alle due tipologie di contratto:

  - **CustomerContract:** definisce regole e scadenze seguite dal Customer;
  - **HelperContract:** definisce regole e scadenze seguite dall'Helper;

  I Worker gestiscono il progresso dei contratti nei quattro stati che essi possono assumere: contrattazione, lavoro, conclusione e fallimento.

5.2.2 Worker

La classe ns3::ndn::Worker è il cuore dell'applicativo. Questo componente ha l'incarico prendere decisioni alla ricezione dei diversi messaggi definiti dal protocollo. Essa estende la classe ns3::ndn::AbstractWorker che a sua volta estende la classe ns3::ndn::App. Quest'ultima raccoglie le funzionalità base per gli applicativi che si appoggiano sullo stack protocollare NDN. Il diagramma delle classi [4.9] mostra la gerarchia delle classi presentate.

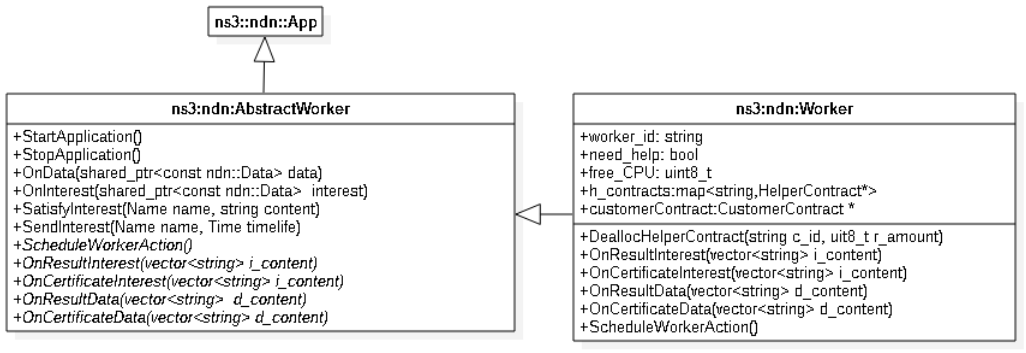

Figura 5.1: Diagrammi delle classi relativo a ns3::ndn::App, ns3::ndn::AbstractWorker e ns3::ndn::Worker.

Si osservano le classi nel dettaglio:

- **ns3::ndn::AbstractWorker:** questa classe astratta implementa le funzionalità di base offerte dai nodi della rete, che sono la capacità di trasmettere e ricevere pacchetti appartenenti alle classi ndn::Interest e ndn::Data. Utilizzando pattern template method, il comportamento della classe viene caratterizzato a seconda del pacchetto ricevuto. Si illustrano brevemente i metodi che implementano queste funzionalità:
  - **OnInterest:** il metodo viene eseguito ad ogni interesse ricevuto. Esso va ad analizzare il nome dell'interesse e ne estrapola i parametri. A seconda del nome queste informazioni sono passate ai metodi virtuali *OnResultInterest()* e *OnCertificateInterest()* i cui comportamenti vengon definiti nella classe ns3::ndn::Worker;
  - **OnData:** il metodo va ad analizzare il nome del dato e ne estrapola i parametri. A seconda del nome queste informazioni vengono fornite ai metodi virtuali *OnResultData()* e *OnCertificateData()* i cui comportamenti sono definiti nella classe ndn::ns3::Worker;

- **SatisfyInterest:** il metodo ha il compito di comporre e spedire un pacchetto dati. Dopo dopo la configurazione, in cui vengono indicati nome, freshness period e firma, il metodo utilizza le API fornite da NDN per instradare il pacchetto.
- **SendInterest:**il metodo ha il compito di comporre e spedire un interesse. Dopo la configurazione in cui vengono indicati nome, timelife e nonce il metodo utilizza le API NDN per instradare il pacchetto;

Un'altro compito di particolare importanza è la configurazione della FIB che grazie alla classe ndn::FibHelper avviene in maniera molto semplice e intuitiva.

- **ns3::ndn::Worker:** questa classe è il cuore del'applicativo installato sui nodi del simulatore. Ciascun Worker possiede un identificatore alfanumerico e gode di una percentuale di CPU libera rappresentata da un valore intero che può andare da 0 a 100. Un campo booleano, definito al momento dell'inizializzazione, stabilisce se nel corso della simulazione il nodo debba comportarsi da Helper o da Customer. In base al ruolo assunto, ciascun nodo amministra dei CustomerContract o degli HelperContract. I metodi *OnResultInterest()*, *OnCertificateInterest()*, *OnCertificateData()* e *OnResultData()* definiscono le regole secondo cui vengono istanziati e aggiornati i contratti di prestazione; inoltre, dato che ciascun nodo intercetta tutti i pacchetti trasmessi in rete, essi filtrano esclusivamente quelli che li riguardano.
  - **OnResultInterest:** il metodo viene chiamato alla ricezione ogni result interest. I nodi che hanno la possibilità di candidarsi istanziano un HelperContract;
  - **OnCertificateInterest:** il metodo viene chiamato alla ricezione di un interesse per il certificato di incarico. Se esso è il primo interesse di candidatura ricevuto, allora il Customer procederà alla consegna del certificato, annotando l'avvenuto nel proprio CustomerContract;
  - **OnCertificateData:** il metodo viene chiamato alla ricezione di un certificato di incarico. Il nodo da cui proviene la candidatura potrà quindi procedere all'esecuzione del task, che nel nostro caso consiste in una semplice attesa;
  - **OnResulData:** il metodo viene chiamato alla ricezione di un pacchetto contenente un risultato. Il Customer da cui proviene la richiesta ne registra il contenuto concludendo definitivamente il CustomerContract;

5.2.3 Contract

Per garantire una solida coordinazione fra Customer ed Helper si è scelto di incapsulare i dettagli d'incarico in dei contratti. La classe ns3::ndn::Contract definisce una generica struttura di contratto che viene specializzata nelle classi ns3::ndn::CustomerContract e ns3::ndn::HelperContract. Il ciclo di vita di un contratto prevede quattro possibili stati definiti nell'enumerazione State:

- **Bargaining:** il contratto si trova in questo stato dall'istante della sua creazione fino all'inizio dell'esecuzione del task.
- **Working:** il contratto si trova in questo stato dall'inizio dell'esecuzione del task fino alla sua conclusione.
- **Concluded:** il contratto è considerato concluso se il risultato è stato consegnato correttamente.
- **Failed:** il contratto fallisce se non vie portato a compimento. Gli HelperContract che non ricevono certificato di incarico vengono considerati falliti.

Ciascun contratto, inoltre, prevede che i periodi di contrattazione(Bargaining) e di lavoro(Working) abbiano una durata massima definita nell'istante de loro inizio. Infine, ogni contratto ha un riferimento al Worker che lo ha istanziato in modo da poterlo notificare in maniera attiva. Per esempio l'HelperContract, una volta eseguito il task, chiede subito al Worker di deallocarlo. Nell'immagine 5.2 osserviamo il diagramma delle classi dei contratti sviluppati.

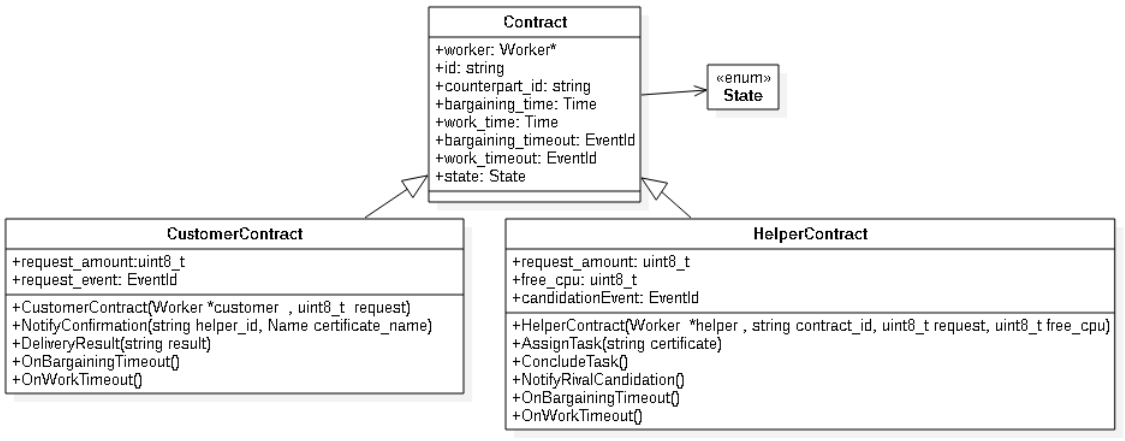

Figura 5.2: Diagramma delle classi relatico ai contratti implementati.

Ciascun accordo di lavoro, viene racchiuso in una coppia di contratti, uno per il Customer e uno per l'Helper. Essi vengono amministrati grazie all'assegnazione di due identificatori che verranno scambiati durante la procedura d'accordo. Mostriamo il dettaglio dei contratti implementati:

- **ns3::ndn::CustomerContract:** questo contratto racchiude le procedure rispettate dal Customer al fine della corretta esecuzione del task. Il costruttore, dopo aver registrato i dettagli contrattuali(ammontare della richiesta e id), istanzia subito un evento che invii l'interesse di nome "/prefix/result...". L'evoluzione del contratto è affidata al Worker, che grazie ai metodi *NotifyConfirmation()* e *DeliveryResult()* può indicare l'helper incaricato ed infine annotare la corretta ricezione del risultato. Un altro compito di particolare importanza riguarda la gestione degli eventi di timeout, che possono essere aggiunti o rimossi dallo scheduler in base all'evoluzione del contratto.

- **ns3::ndn::HelperContract:** questo contratto racchiude la procedura rispettata dall'Helper per presentare la propria candidatura. Nel caso esso venga scelto per l'esecuzione del task allora il contratto si evolve passando alla fase di lavoro. Il costruttore dopo aver registrato i dettagli contattuali (id,ammontare richiesta e CPU libera) istanzia l'interesse per il certificato di incarico; come definito dall'algoritmo l'evento di tramissione viene fissato a seguito di un periodo d'attesa calcolato in base al carico computazionale. L'evoluzione del contratto è affidata al Worker, che grazie ai metodi *AssignTask()*, *NotifyRivalCandidation()* può annotare l'assegnamento del task oppure annullare il contratto se un altro helper lo ha preceduto nella candidatura. Come il CustomerContract questo contratto racchiude la gestione dei timeout.

5.3 Scenario applicativo

Per comprendere al meglio secondo quali modalità venga impostata una simulazione, si è scelto di mostrarne un semplice esempio. Nello specifico andremo installare l'applicativo Worker su cinque nodi, due dei quali opereranno da Customer mentre i restanti tre opereranno da Helper.

```
#include "ns3/core-module.h"
#include "ns3/network-module.h"
#include "ns3/applications-module.h"
#include "ns3/wifi-module.h"
#include "ns3/mobility-module.h"
#include "ns3/internet-module.h"
#include "ns3/ndnSIM-module.h"
using namespace std;
namespace ns3 {
NS_LOG_COMPONENT_DEFINE("ndn.WifiExample");
int main(int argc, char* argv[]) {
  /*Configurazione rete*/
  Config::SetDefault("ns3::WifiRemoteStationManager::FragmentationThreshold", StringValue("2200"));
  Config::SetDefault("ns3::WifiRemoteStationManager::RtsCtsThreshold", StringValue("2200"));
  Config::SetDefault("ns3::WifiRemoteStationManager::NonUnicastMode", StringValue("OfdmRate24Mbps"));
  ////////////////////
  WifiHelper wifi = WifiHelper::Default(); //Configurazione comunicazione Wifi
  wifi.SetStandard(WIFI_PHY_STANDARD_80211a); //Configurazione standard 80211a
  wifi.SetRemoteStationManager("ns3::ConstantRateWifiManager", "DataMode",StringValue("OfdmRate24Mbps"));

  YansWifiChannelHelper wifiChannel; // Helper per configurazione canale comunicazione
  wifiChannel.SetPropagationDelay("ns3::ConstantSpeedPropagationDelayModel"); // Velocità di propagazione costrante
  wifiChannel.AddPropagationLoss("ns3::ThreeLogDistancePropagationLossModel"); //Modello di perdita dei pacchetti .
  wifiChannel.AddPropagationLoss("ns3::NakagamiPropagationLossModel");

  YansWifiPhyHelper wifiPhyHelper = YansWifiPhyHelper::Default();
  wifiPhyHelper.SetChannel(wifiChannel.Create());
  wifiPhyHelper.Set("TxPowerStart", DoubleValue(5));
  wifiPhyHelper.Set("TxPowerEnd", DoubleValue(5));

  NqosWifiMacHelper wifiMacHelper = NqosWifiMacHelper::Default();
  wifiMacHelper.SetType("ns3::AdhocWifiMac");

  Ptr<UniformRandomVariable> randomizer = CreateObject<UniformRandomVariable>(); Ge
  randomizer->SetAttribute("Min", DoubleValue(0));
  randomizer->SetAttribute("Max", DoubleValue(10));

  MobilityHelper mobility;
  mobility.SetPositionAllocator("ns3::RandomBoxPositionAllocator", "X", PointerValue(randomizer),
                                "Y", PointerValue(randomizer), "Z", PointerValue(randomizer));

  mobility.SetMobilityModel("ns3::ConstantPositionMobilityModel"); // Mobilità a posizione fissa
  NodeContainer nodes;
  nodes.Create(5);
  // 1. Installazione wifi sui nodi
  NetDeviceContainer wifiNetDevices = wifi.Install(wifiPhyHelper, wifiMacHelper, nodes);
```

Figura 5.3: Esempio di simulazione pt.1.

```
  // 2. Installazione modello di mobilità
  mobility.Install(nodes);
  // 3. Installazione dello stack NDN
  NS_LOG_INFO("Installing NDN stack");
  ndn::StackHelper ndnHelper;
  ndnHelper.SetOldContentStore("ns3::ndn::cs::Lru", "MaxSize", "1000");
  ndnHelper.SetDefaultRoutes(true);
  ndnHelper.Install(nodes);
  // Best routing come strategia di forwarding
  ndn::StrategyChoiceHelper::Install(nodes, "/", "/localhost/nfd/strategy/best-route");
  // 4. Installazione delle applicazioni
  ndn::AppHelper w_0("ns3::ndn::Worker");
   w_0.SetAttribute("WorkerID", StringValue("W00"));
   w_0.SetAttribute("Free_CPU", IntegerValue(0));
   w_0.SetAttribute("Overrun", BooleanValue(true));
   w_0.Install(nodes.Get(0));
   ndn::AppHelper w_1("ns3::ndn::Worker");
   w_1.SetAttribute("WorkerID", StringValue("W01"));
   w_1.SetAttribute("Free_CPU", IntegerValue(0));
   w_1.SetAttribute("Overrun", BooleanValue(true));
   w_1.Install(nodes.Get(1));
   ndn::AppHelper w_2("ns3::ndn::Worker");
   w_2.SetAttribute("WorkerID", StringValue("W02"));
   w_2.SetAttribute("Free_CPU", IntegerValue(90));
   w_2.SetAttribute("Overrun", BooleanValue(false));
   w_2.Install(nodes.Get(2));
   ndn::AppHelper w_3("ns3::ndn::Worker");
   w_3.SetAttribute("WorkerID", StringValue("W03"));
   w_3.SetAttribute("Free_CPU", IntegerValue(80));
   w_3.SetAttribute("Overrun", BooleanValue(false));
   w_3.Install(nodes.Get(3));
   ndn::AppHelper w_4("ns3::ndn::Worker");
   w_4.SetAttribute("WorkerID", StringValue("W04"));
   w_4.SetAttribute("Free_CPU", IntegerValue(40));
   w_4.SetAttribute("Overrun", BooleanValue(false));
   w_4.Install(nodes.Get(4));
  Simulator::Stop(Seconds(50.0)); // Simulato timer
  Simulator::Run();
  Simulator::Destroy();
  return 0;
}
}

int main(int argc, char* argv[]){
  return ns3::main(argc, argv);
}
```

Figura 5.4: Esempio di simulazione pt.2.

L'esempio proposto nelle immagini [5.3] e [5.4], mostra gli strumenti forniti da ns-3 per la connessione wifi. Come è possibile comprendere dal codice, essi offrono la possibilità configurare una vasta gamma di parametri. Alcuni esempi sono lo standard di trasmissione(nel nostro caso IEEE 802.11a), la velocità di propagazione e il criterio di perdita dei pacchetti. Inoltre, la classe MobilityHelper permette di definire lo spazio in cui i nodi sono racchiusi e secondo quale criterio essi debbano essere posizionati. Nell'esempio i nodi vengono posizionati in modo random all'interno di un cubo e mantengono la stessa posizione per l'intera simulazione.

Una volta avviata la simulazione, i Customer mostrano la loro richiesta alla rete nell'arco di un secondo. Ciascuna richiesta, per azione del protocollo, viene assegnata ad un Helper il quale, dopo l'esecuzione del task, consegna il risultato. Nell'immagine 5.5 osserviamo l'output della simulazione.

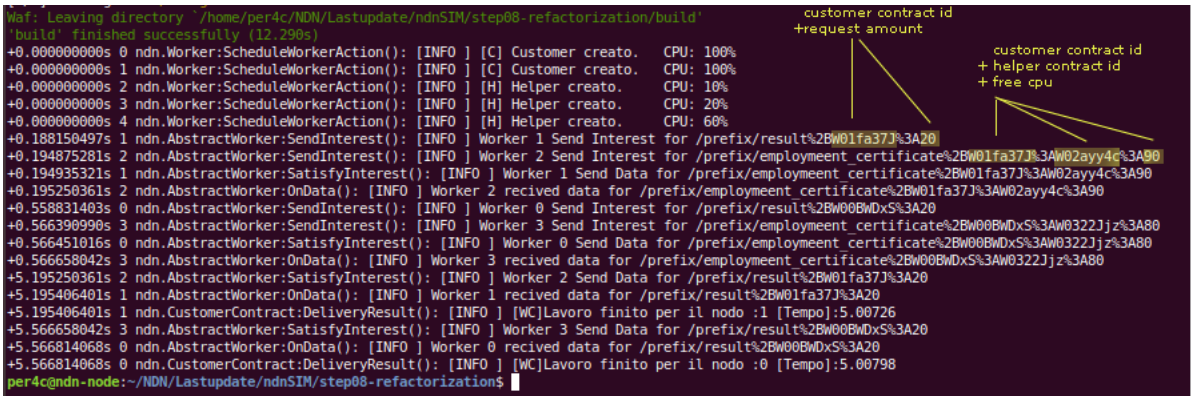

Figura 5.5: Output della simulazione d'esempio.

Le macro offerte dal simulatore permettono allo sviluppatore di personalizzare l'output, mostrando solo gli eventi a cui è interessato. Le righe della simulazione sono ordinate cronologicamente e seguono uno schema predefinito: istante di esecuzione, nodo sorgente e contenuto del log. Dall'output si osserva che il nodo numero 1 è il primo ad effettuare una richiesta del 20% di CPU. Come previsto da dal protocollo essa viene accolta dal nodo numero 2 che con il 10% di CPU è il nodo con il minor carico computazionale. Di conseguenza la CPU del nodo 2 passa al 30%. Mentre viene eseguito il task commissionato dal nodo 1, anche il nodo 0 espone una richiesta alla rete. Quest'ultima viene assegnata al nodo 3 perchè in questo istante, con il 20% di CPU occupata, è quello con minor carico di lavoro. Dopo un'attesa di circa cinque secondi, che corrisponde al periodo dei task, gli Helper procedono alla consegna del risultato che viene ricevuto correttamente ai Customer. Si precisiamo che i caratteri utilizzati per la suddivisione dei metadati (%3A e %2B) non hanno alcuna rilevanza, essi sono stati adottati per suddividere in maniera differente i dati aggiuntivi.

L'esempio in figura è uno degli scenari più semplici che sia stato testato, tuttavia incrementando il numero di nodi e configurando le richieste in maniera differente (in termini di periodo di esecuzione e di carico computazionale) è possibile impostarne di molto più complessi e articolati.

5.4 Ambiente di lavoro

Il sistema è stato sviluppato e testato installando ndnSIM su una macchina virtuale basata su Ubuntu 18.04. Tra gli aspetti positivi forniti dal simulatore è possibile individuare la scalabilità e la portabilità degli scenari sviluppati. Una volta installato il simulatore, per testate il proprio scenario è stato sufficiente affiancarvelo seguendo le best-practies /Scite indicate dal team di sviluppo di ndnSIM.

5.5 Test e sviluppo

Per via della natura altamente sperimentale del progetto si è preferito adottare un modello si sviluppo a cascata, che ha permesso di evolvere il sistema sulla base dei progressi ottenuti step-by-step. A seguito della fase di sviluppo e del perfezionamento il software ha subito assunto il comportamento desiderato. Inoltre non si segnala alcuna problematica specifica legata al protocollo NDN. Nella maggioranza delle situazioni il sistema risponde in tempi molto buoni mentre è visibilmente minore il numero delle circostanze in cui questi non combaciano con quelli sperati. Nello specifico questi casi riguardano perdite di pacchetti, aspetto consapevolmente non curato durante la fase di sviluppo. Testando diversi scenari, si è potuto osservare quali fossero i parametri che più incidono sulle prestazioni e a cui sarebbe necessario destare attenzione nel caso si volesse impiegare il sistema.

Lo scenario ideale su cui testare il protocollo, prevede che i nodi coinvolti nella simulazione gestiscano i propri task in maniera dinamica. Questa impostazione permetterebbe di apprezzare a pieno il comportamento del sistema e di coglierne maggiori aspetti da perfezionare. Purtroppo ricreare questo tipo di scenario sul simulatore è apparso particolarmente complesso perciò si è preferito testare il comportamento assegnando staticamente un carico computazionale.

Per mettere alla prova il sistema si è scelto osservarne il comportamento di fronte alle situazioni, ovvero la gestione di più richieste di lavoro contemporanee. Nello specifico andiamo ad osservarne il comportamento in due casi:

- **Caso peggiore:** si osserva il comportamento del sistema nel caso in cui il carico di lavoro richiesto dai customers superi le risorse degli helper.
- **Richieste a frequenza costante:** si osserva il comportamento del sistema chiedendo a degli helper di soddisfare molteplici richieste generare a frequenza costante.

Nel il primo dei due test si è scelto di coinvolgere 20 helpers configurati allo 0% di CPU e che possono prendere in consegna l'esecuzione di task fino all'80%. Contemporaneamente vengono istanziati 380 customers che nell'arco di un millisecondo dall'avvio effettuano le loro richieste. In questo caso i task commissionati prevedono un carico computazionale del 20% per un periodo di 5 secondi. Nel grafico mostrato dall'immagine [5.6] ogni punto di colore blu identifica una richiesta. L'asse verticale indica il tempo che intercorre tra l'istante in cui nasce la richiesta e l'effettiva consegna del risultato. L'asse orizzontale indica l'istante in cui l'interesse è stato creato.

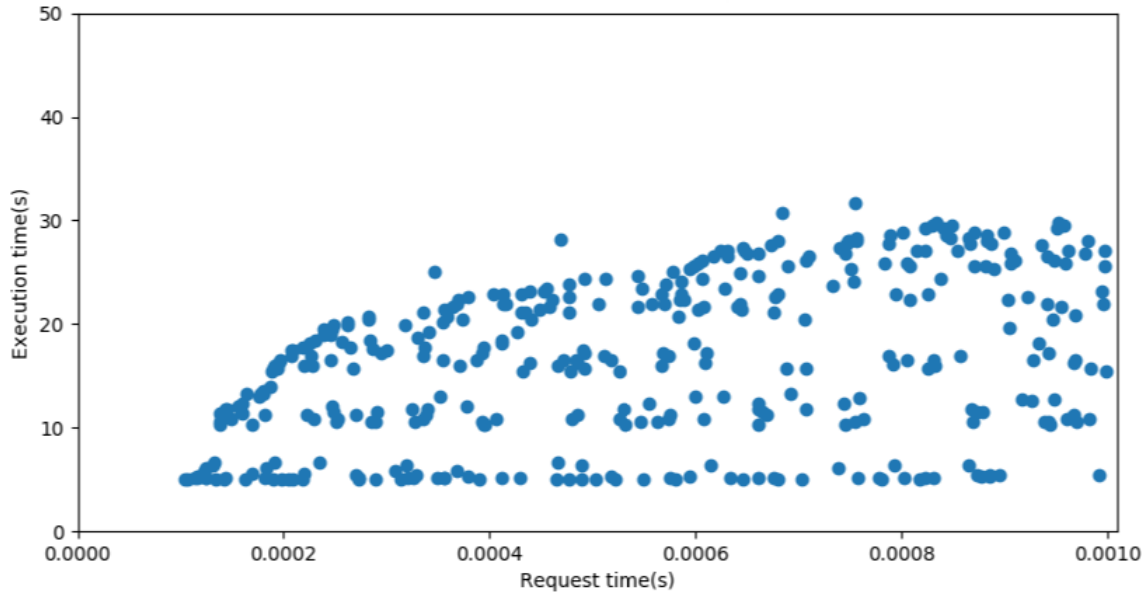

Figura 5.6: Simulazione in cui 20 helpers soddisfano 380 richieste(periodo(T)= 5s, richiesta(R)= 20%).

Valutando le risorse offerte dagli helper capiamo che le richieste non possono essere soddisfatte tutte nello stesso momento (per la precisione esse vengono colmate da 80 richieste). Questo è il motivo per cui dal grafico si possono distinguere delle bande orizzontali in corrispondenza di 5, 10 ,15 ,20 e 25 secondi. Dopo 5 secondi le risorse dei primi task presi in carico iniziano ad essere liberate lasciando che le richieste in attesa possano essere soddisfatte. Tuttavia, anche dopo questi 5 secondi vi sano ancora task costretti ad attendere. Questo meccanismo viene iterato fino a quando tutte le richieste non vengono accontentate. L'esempio conferma a grandi linee il comportamento atteso ma data la disomogeneità delle bande orizzontali si denota che l'algoritmo necessita di alcune ottimizzazioni.

Nel secondo test sono stati impiegati sempre 20 helper a cui è stata assegnata la stessa configurazione(CPU=0% maxCPU=80%). Ad essi vengono commissionate 50 richieste al secondo per i primi 10 secondi di simulazione. Anche in questo caso i task richiedono un carico computazionale del 20% per una durata di 5 secondi.

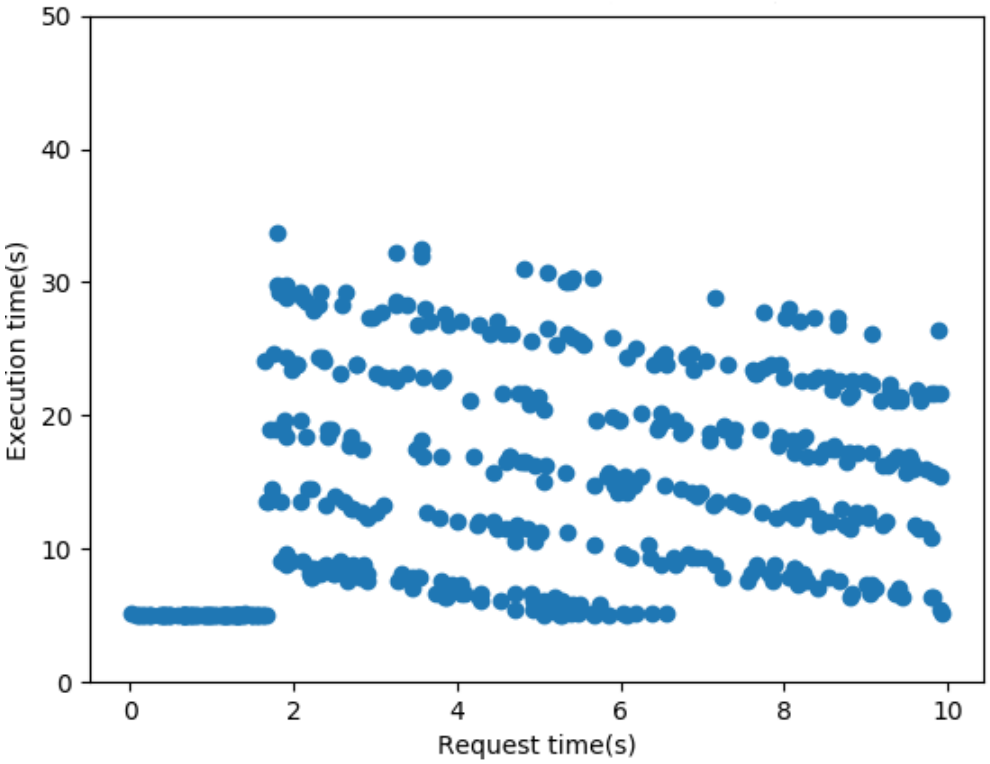

Figura 5.7: 20 helpers soddisfano 50 richieste al secondo(periodo(T)= 5s, richiesta(R)= 20%).

Le informazioni rappresentate nel grafico [5.7] equivalgono a quelle mostrate nell'immagine [5.6]. Il tempo con cui vengono risolti i task rimane costante a 5 secondi fino circa 2s mentre da lì in avanti si osserva un andamento a bande inclinate. Ragionando sul grafico si è capito che i ritardi vengono propagati secondo questa modalità perchè le risorse dei 20 helper vengono colmate prima che esse possano essere liberate. Se consideriamo che il totale delle risorse equivale a 1600(80% per ciascuno dei 20 helper) e che la richiesta al secondo è del 1000 (20% per ciascuno dei 50 customer), si comprende che esse dovrebbero essere colmate dopo circa 1,6s(dato confermato dal grafico). Infine, notiamo che alcune delle richieste nate attorno ai 5s vengano risolte in tempi molto vicini al periodo di esecuzione del task. Questo comportamento è giustificato dal fatto che il protocollo non adotta alcun tipo strategia di prenotazione ma prevedere che i task

in attesa continuino a mostrare richieste. Al contrario, una strategia di prenotazione garantirebbe che le richieste vengano soddisfatte secondo il loro ordine cronologico.

Grazie a questo grafico si è capito che come si potesse regolare il comportamento del sistema in base alla frequenza delle richieste, il loro periodo, il loro ammontare e ovviamente le risorse disponibili. Per questo motivo si mostrano altre simulazioni che aiutano a comprendere il comportamento del sistema al variare di questi parametri.

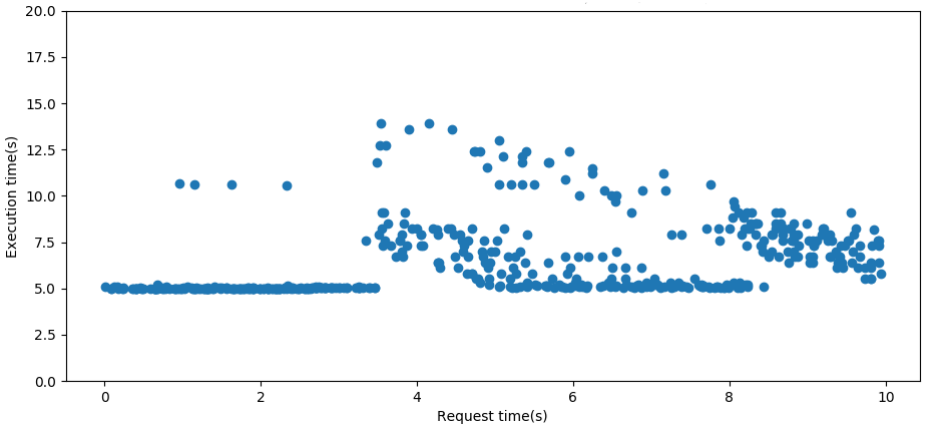

Figura 5.8: 20 helpers soddisfano 50 richieste al secondo(periodo(T)= 5s, richiesta(R)= 10%).

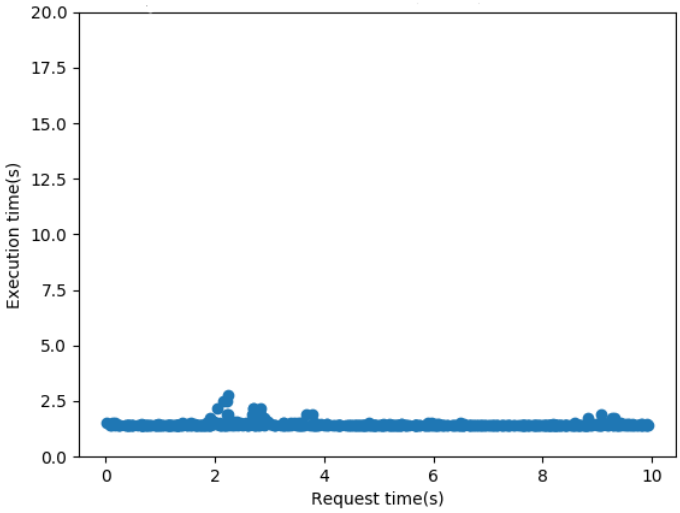

Figura 5.9: Venti helpers soddisfano 50 richieste al secondo(periodo(T)= 1.4s, richiesta(R)= 20%).

Nel grafico [5.8] l'ammontare delle richeste viene ridotto al 10% ma dato che le risorse vengono colmate prima dei 5 secondi allora permangono task costretti ad un'attesa. In quello successivo [5.9], invece,viene ridotto il periodo del task facendo in modo che le risorse vengano liberate prima che possano raggiungere la saturazione; in questo modo esse vengono utilizzate a pieno regime.

Conclusione e sviluppi futuri

Il progretto di tesi intrapreso si è rivelato notevolmente stimolante e gratificante. Toccare con mano i particolari di un progetto altamente sperimentale come Named Data Networking, parte del progetto statunitense Future Internet Architecture , mi ha fatto sentire per la prima volta coinvolto, in maniera operativa, in un programma di ricerca. Oltre all'apprendimento dei concetti che caratterizzano il paradigma NDN, questa esperienza mi ha permesso di lavorare con nuovi strumenti come i simulatori di reti ndnSIM e ns-3. Essi hanno richiesto l'approfondimento del linguaggio C++ e la programmazione ad eventi, argomenti trattati ma mai approfonditi durante il percorso di studi. Per la realizzazione dei grafici, si è impiegato per la prima volta il linguaggio di programmazione python. Dal punto di vista teorico sono stati compresi i concetti di Fog e Edge computing, evoluzioni del paradigma Cloud che si propongono come uno degli aspetti chiave nel progresso di Internet of Things.

Nella prima fase del lavoro si è svolto un sostanzioso studio dell'architettura NDN che, nell'ottica della realizzazione di un protocollo, ha richiesto di andare a osservare anche aspetti di particolare dettaglio. Una volta chiarito il panorama, si è scelto sviluppare il prototipo sul simulatore ndnSIM. NDN permette di installare il demone NFD anche su macchine fisiche, ma questa strategia non avrebbe permesso di testare il sistema come con il simulatore. Nonostante il sistema non appaia troppo articolato, la parte di sviluppo è stata sicuramente quella più lunga e complessa. Lo studio e la comprensione di nuovi strumenti sono i motivi principali dell'ampia durata di questa fase. Dati i lunghi tempi di esecuzione delle simulazioni, anche i test hanno portato via molto tempo. Questo aspetto, unito alla possibilità di lavorare solo su una macchina virtuale, ha portato alla scelta di mostrare i comportamenti più interessanti del sistema tramite degli snapshot di breve durata. Il lavoro svolto ha portato al raggiungimento degli obiettivi preposti. Il risultato conferma che il paradigma NDN permetta di realizzare protocolli di computazione cooperativa. Il prototipo non rappresenta ancora una versione completa ed efficente ma ne pone sicuramente una buona base concettuale.

L'esito del progetto offre una vasta gamma di sviluppi futuri, dei quali se ne elencano alcuni fra i più rilevanti:

- **Ampliamento:** ampliare il sistema sviluppando e testando la perdita di pacchetti;
- **Configurazione parametri:** la configurazione del forwarding timeout si è rilevata un'aspetto molto rilevante termini di prestazioni. Questo tema è stato trattato in maniera empirica ma sarebbe interessante studiare nel dettaglio quale sia la regolazione ottimale di questo parametro;
- **Test dinamici:** una volta migliorate le prestazioni, sarebbe interessante effettuare una serie di test specifici che permettano di comprendere il comportamento del sistema in scenari più dinamici. Un inizio potrebbe essere quello di rendere variabili i carichi computazionali per poi passare all'introduzione di task concreti.
- **Apprendimento automatico:** le comunicazioni broadcast permettono ai nodi di ascoltare tutte le conversazioni, reperendo così informazioni sugli altri dispositivi. Questo aspetto apre la possibilà di analizzare questi dati, applicando metodi di machine learning che possono essere utili all'ottimizzazione delle prestazioni.
- **Nuove interpretazioni:** la soluzione proposta rappresenta solo una delle possibili interpretazioni del problema e probabilmente ne esistono di migliori. Potrebbe essere interessante e avvincente cercare di scoprire nuove strategie che permettano di raggiungere lo stesso risultato in maniera ancora più semplice.

A livello personale mi ritengo molto soddisfatto del lavoro svolto, in quanto sono riuscito a confrontarmi discretamente con problemi di ricerca attuali, apportando così il mio piccolo contributo al progetto Named Data Networking. Grazie questa esperienza di lavoro autonomo, le cui linea guida sono state fornite dal Professor Giovanni Pau, ho potuto esprimere a pieno le mie capacità e allo stesso tempo affrontare i miei limiti. In futuro, mi farebbe molto piacere portare avanti gli aspetti non trattati nel percorso di tesi, in modo da poter raggiungere una versione completa del protocollo.

Rigraziamenti

Le persone a cui voglio di indirizzare il ringraziamento più grande sono i miei genitori, Giorgio e Nadia. Il loro affetto e i loro saggi consigli mi hanno accompagnato in tutto il percorso di vita e si sono rivelati lo strumento più utile per raggiungere questo risultato.

In secondo luogo vorrei ringraziare i miei fratelli, Mirco e Matteo, che mi hanno sempre sostenuto e mi sono sempre stati vicini a partire dai piccoli problemi della vita quotidiana.

Un ringraziamento speciale va ai miei nonni, Giulio, Elisa e Rosina che sono sempre stati pronti a coccolarmi e a regalarmi un sorriso. Grazie alle loro storie ho capito quale sia il valore degli sforzi e dei sacrifici.

Grazie a tutti gli amici con cui ho potuto ridere e scherzare ma che allo stesso tempo hanno avuto la forza di condividere anche i momenti più difficili. Un pensiero particolare a quegli amici che, per lontananza o impegni, ricevono meno attenzioni ma non per questo sono meno importanti.

Infine vorrei rivolgere un sentito ringraziamento al professor Giovanni Pau, che ha sempre saputo aiutarmi con preziosi consigli ed è riuscito dedicarmi parte del suo tempo nonostante i numerosi impegni. Oltre che per i suoi insegnamenti, vorrei ringraziarlo per il piacevole rapporto professore-studente che egli è riuscito a instaurare, aspetto e che mi ha fornito quella motivazione in più per dare sempre il meglio di me stesso.

Bibliografia

[1] Flavio Bonomi, Rodolfo Milito, Jiang Zhu, and Sateesh Addepalli. Fog computing and its role in the internet of things. In *Proceedings of the First Edition of the MCC Workshop on Mobile Cloud Computing*, MCC '12, pages 13–16, New York, NY, USA, 2012. ACM.

[2] Franco Callegati. Il protocollo di internet. teaching slides.

[3] Gustavo Carneiro. Ns-3: Network simulator 3. In *UTM Lab Meeting April*, volume 20, 2010.

[4] Giulio Grassi, Davide Pesavento, Giovanni Pau, Rama Vuyyuru, Ryuji Wakikawa, and Lixia Zhang. Vanet via named data networking. In *Computer Communications Workshops (INFOCOM WKSHPS), 2014 IEEE Conference on*, pages 410–415. IEEE, 2014.

[5] Spyridon Mastorakis, Alexander Afanasyev, Ilya Moiseenko, and Lixia Zhang. ndnsim 2: An updated ndn simulator for ns-3. *Dept. Comput. Sci., Univ. California, Los Angeles, Los Angeles, CA, USA, Tech. Rep. NDN-0028*, 2016.

[6] Spyridon Mastorakis, Alexander Afanasyev, and Lixia Zhang. On the evolution of ndnSIM: an open-source simulator for NDN experimentation. *ACM Computer Communication Review*, July 2017.

[7] Jon Postel et al. Rfc 791: Internet protocol, 1981.

[8] Weisong Shi, Jie Cao, Quan Zhang, Youhuizi Li, and Lanyu Xu. Edge computing: Vision and challenges. *IEEE Internet of Things Journal*, 3(5):637–646, 2016.

[9] NFD Team. Nfd developer's guide.

[10] Lixia Zhang, Deborah Estrin, Jeffrey Burke, Van Jacobson, James D Thornton, Diana K Smetters, Beichuan Zhang, Gene Tsudik, Dan Massey, Christos Papadopoulos, et al. Named data networking (ndn) project. *Relatório Técnico NDN-0001, Xerox Palo Alto Research Center-PARC*, 157:158, 2010.

Sitografia

[1] Future internet architecture project. http://www.nets-fia.net/.

[2] Manual ns-3. https://www.nsnam.org/docs/manual/html/index.html.

[3] Named data networking. https://named-data.net/.

[4] Nfd wiki. https://redmine.named-data.net/projects/nfd/wiki.

[5] Rfc fiap. https://tools.ietf.org/html/rfc1287.

[6] Website ndnsim. http://ndnsim.net/current/.

[7] Lixia Zhang. Ndn overview. https://named-data.net/tutorials/milcom2017/milcom2017-ndn-2-overview.pdf.