POLITECNICO DI TORINO

Corso di Laurea in Ingegneria Informatica (Computer Engineering)

Tesi di Laurea Magistrale

Sviluppo di un'interfaccia orchestratore per eNodeB OpenAirInterface5G (OAI)

Relatore
prof. Claudio Ettore Casetti

Candidato
Corrado Monaco

Tutor aziendale Telecom Italia
Ing. Gian Michele Dell'Aera

Anno accademico 2018-2019

Alla mia famiglia.

Sommario

Gli ultimi anni hanno visto una rapida evoluzione delle reti mobili e la rete di accesso sta via via diventando più complessa e articolata. Con l'avvento del 4G, si è cercato di dare una prima risposta alla suddetta complessità, introducendo soluzioni in grado di agevolare i più comuni processi di configurazione e ottimizzazione. Alcuni esempi di tali processi sono la riduzione dei problemi dovuti alla mobilità degli utenti, il bilanciamento del traffico ed il miglioramento della copertura del segnale radio.

I paradigmi di *softwarization* e *virtualization*, che hanno visto le prime applicazioni nell'accesso radio 4G, troveranno piena realizzazione nei sistemi mobili di quinta generazione (5G), abilitando maggiore automazione, flessibilità ed agilità nei processi di realizzazione ed ottimizzazione. Il sistema 5G aumenterà sensibilmente le prestazioni necessarie per i nuovi servizi: bit-rate fino a alcuni Gbit/s per una sempre migliore fruizione di contenuti multimediali, latenze end-to-end di pochi millisecondi per servizi ed alta affidabilità per applicazioni *mission-critical* e servizi di *safety*.

Oggigiorno la grande eterogeneità di tecnologie, dovuta al dispiegamento di più gamme di frequenza per le varie tecnologie radio ed alla frammentazione del parco terminali in rete, per prestazioni e funzionalità, porta ad una nuova complessità che richiede dunque nuove soluzioni per poter essere gestita al meglio.

Il concetto di *Self Organizing Networks* (SON) è una prima risposta all'esigenza degli operatori di far fronte a questo incremento di complessità nella gestione dei

processi di configurazione, ottimizzazione e assurance delle reti. L'obiettivo primario è quello di attuare una gestione per mezzo della raccolta di indicatori di qualità e, sulla base di questi, individuare in "real time" le azioni in grado di migliorare le prestazioni, risolvere problemi locali e garantire QoS.

Obiettivo di questa tesi è stato pertanto quello di implementare un *RAN Intelligent Controller* (RIC) in grado di gestire molteplici nodi di accesso radio e di istanziare delle sessioni di misurazioni (*Minimization of Drive Tests*). Nel dettaglio, si vuole usare il terminale dell'utente per collezionare informazioni pubbliche circa i devices con lo scopo di ridurre gli sforzi operativi, migliorare la *performance* e la qualità della rete e, allo stesso tempo, ridurre i costi di manutenzione.

Il lavoro di tesi è stato strutturato nel seguente modo.

Parte I: parte informativa che fornisce le nozioni principali per comprendere propriamente il lavoro in esame.

Il **Capitolo 1** introduce il concetto di Open RAN, fornendo una visione generale della sua architettura, le caratteristiche salienti di essa ed i principali motivi che spingono operatori come Telecom Italia ad esserne interessati.

Nel **Capitolo 2** viene illustrato il concetto di *RAN Intelligent Controller* (RIC), descrivendone i principi architetturali, ed il concetto di *Self Organizing Network* (SON) su cui si basa l'interazione orchestratore-controllore.

Parte II: parte sperimentale che espone la soluzione proposta da Telecom Italia nell'implementazione del RIC.

Indice

Parte I

Sezione informativa

Capitolo 1

Open RAN

1.1 Alleanza Open RAN

L'Open RAN (O-RAN) Alliance, fondata nel febbraio 2018 e gestita da un comitato operativo composto da 15 operatori e da un comitato tecnico (TSC), è un'alleanza di livello mondiale che ha come obiettivo primario proprio quello di rivoluzionare la *Radio Access Network* (RAN) indirizzando nuovi livelli di "*openness*" rispetto alla generazione precedente. Lo scopo è definire un'architettura di RAN basata su white-box hardware ed interfacce standardizzate in modo da separare l'hardware dal software e rispecchieranno dunque i principi fondamentali di "*intelligence*" e "*openness*" di Open RAN.

1.2 5G, Open RAN e la svolta tecnologica richiesta

1.2.1 Verso il 5G

Il traffico dati mobile ha visto un aumento drammatico negli ultimi tempi a causa della rapida adozione degli smartphone e del loro supporto per i servizi che necessitano di dati come lo streaming video. Si prevede che questa tendenza continui a crescere nei prossimi anni, creando una serie di problematiche per le reti mobili del futuro. Inoltre, la necessità di far fronte a nuovi paradigmi di comunicazione come

il *Machine-to-Machine* (M2M) per realizzare i nuovi servizi dell'*industry 4.0*, rende lo sviluppo di una rete sempre più performante un processo necessario.

Le suddette osservazioni delineano dunque come il fattore principale per determinare l'avvento del 5G sia l'avanzamento tecnologico nella rete mobile: reti e componenti più flessibili, intelligenti ed efficienti dal punto di vista energetico. I requisiti chiave della nuova rete 5G includeranno infatti: un aumento della velocità dei dati fino a 1000 volte superiore, la possibilità di supportare un numero più alto di dispositivi (100x) e latenza di comunicazione ridotta ad 1 millisecondo [1].

In termini più tecnici, la rete 5G verterà su tre pilastri fondamentali:

- Alte prestazioni: *enhanced Mobile Broadband* (**eMBB**)
- Maggiore scalabilità: *massive Machine Type Communications* (**mMTC**)
- Bassa latenza: *Ultra-Reliable and Low-Latency Communications* (**URLLC**)

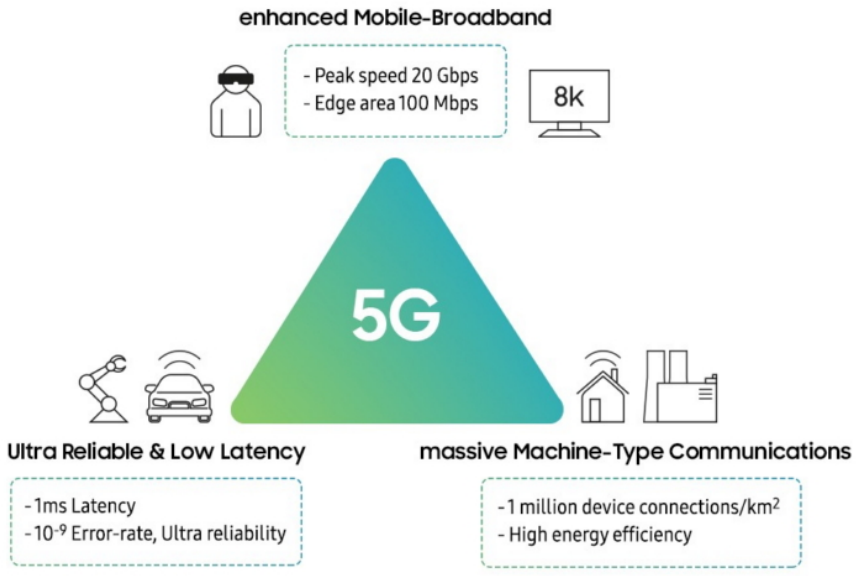

Figura 1.1. *Casi d'uso 5G*

Nell'architettura RAN tradizionale (4G) una determinata area è divisa logicamente in celle ed ognuna di essa è coperta da una stazione radio base (eNodeB), tipicamente posta ad una certa altezza rispetto all'area di copertura. Questa distribuzione capillare delle stazioni base sul territorio permette ai dispositivi mobili, all'interno di una cella, di sfruttare la stazione base più vicina per comunicazioni di tipo voce e/o dati. Il segnale viene quindi prima trasmesso alla stazione base via radio e poi alla rete centrale via cavi o collegamenti radio ad alte frequenze.

L'architettura di una stazione radio base può essere diviso in due blocchi fondamentali:

- **RRH** (*remote radio head*) è l'antenna da cui viene trasmesso il segnale radio, viene posta in una posizione elevata rispetto al terreno ed ha il compito di convertire il segnale digitale in RF (Radio Frequenza) e vice versa

- **BBU** (*baseband unit*) è collegata al RRH tramite fibra ottica dedicata ed è posta ai piedi della stazione radio base. É responsabile della trasformazione dei dati ricevuti attraverso una rete IP a pacchetto in un segnale digitale adatto ad essere trasmesso in aria (l'interfaccia fisica sfruttata dalla RAN)

In breve si potrebbe dire che la BBU possiede la *digital function*, mentre RRH quella analogica[2].

La presenza nell'architettura RAN di un unico grande blocco funzionale come la BBU ha portato negli anni alla diffusione di soluzioni mono-fornitori chiuse. Questa tipologia di mercato non è adatta ai requisiti chiave della rete 5G che richiede un incremento della densità di nodi al fine di garantire le prestazioni attese. Un'architettura RAN tradizionale rende difficoltosa sia in termini di spesa operativa (*OPEX*) che di spesa capitale (*CAPEX*) la diffusione capillare di una rete 5G.

1.2.2 O-RAN

Per raggiungere delle prestazioni maggiori (servizi *eMBB*), il 5G sfrutterà uno spettro più ampio ad alte frequenze. La forte attenuazione di un segnale ad alte frequenze è una delle maggiori cause dell'aumento del numero di stazioni base.
Questo processo che porta ad una maggiore densità delle reti di quinta generazione e di norma indicato con il termine di *densification*, imporrà significativi cambiamenti soprattutto per le reti di trasporto e di backhaul.

La visione di Open RAN è proprio quella di permettere l'evoluzione dell'architettura RAN tradizionale delineando tre aree di interesse:

- Disaccoppiare il *control plane*[1] dal *data plane*[2]
- Creare uno stack software modulare della base station che opera su un *common-off-the-shelf* (COTS) hardware
- Esporre interfacce *north/south-bound* aperte per la gestione e il coordinamento delle funzioni radio

La rapida e massiccia installazione delle *high frequency base stations*, faciliterebbe gli operatori nel soddisfare le aspettative della nuova rete 5G.
La tecnologica più significativa quando si definiscono architetture di rete ultra-dense è un'architettura radio di tipo centralizzata (***Centralized-RAN***) che vede come obiettivo principale quello di "raggruppare" le BBU, in modo da ottimizzare i costi di gestione migliorare le prestazioni e la scalabilità sfruttando la concentrazione della potenza computazionale in un'unica sede. La base della architettura C-RAN vede la separazione delle RRHs dalle BBUs, con la conseguente aggregazione di queste ultime in un *BBU Pool*.

[1] *determina il percorso che il traffico percorrerà attraverso la rete*
[2] *la parte della rete che effettivamente trasporta il traffico*

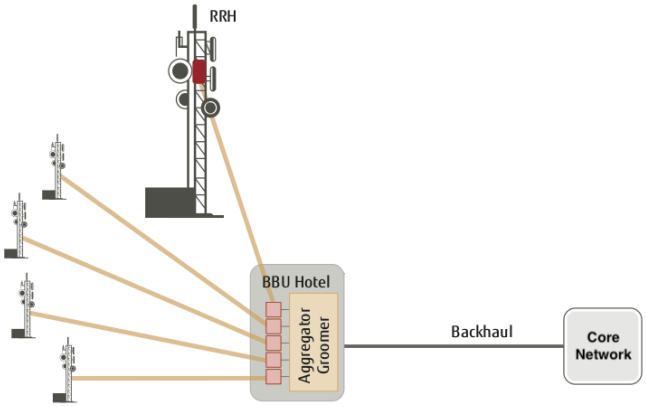

Figura 1.2. Rete di accesso radio centralizzata (*C-RAN*)

[3]

Nasce dunque la necessità di una rete di trasporto estremamente performante per consegnare i segnali campionati dalle antenne (RRH) alla *centralized BBU Pool*.

Antenna Configuration	CPRI Communication Rate for LTE Bandwidth			
	5 MHz	**10 MHz**	**20 MHz**	**20 MHzx2**
2x2 MIMO	614.4 Mbit/s (IP Rate 37.5 Mbit/s)	1.2288 Gbit/s (IP Rate 75 Mbit/s)	2.4576 Gbit/s (IP Rate 150 Mbit/s)	4.9152 Gbit/s (IP Rate 300 Mbit/s)
4x2 (4x4) MIMO	1.2288 Gbit/s (IP Rate 75 Mbit/s)	2.4576 Gbit/s (IP Rate 150 Mbit/s)	4.9152 Gbit/s (IP Rate 300 Mbit/s)	4.9152 Gbit/s (IP Rate 300 Mbit/s)

Figura 1.3. La velocità di comunicazione del CPRI varia in base alla configurazione dell'antenna ed alla larghezza di banda LTE[4]

Nell'architettura C-RAN il termine "***fronthaul***" descrive proprio questa rete di trasporto. BBUs e RRHs potrebbero trovarsi a diversi chilometri di distanza e la

loro comunicazione è basata sul protocollo semi-proprietario CPRI. La realizzazione della *BBU Pool* ha come grosso vantaggio quello di permettere a tutte le BBUs di condividere la stessa infrastruttura fisica, lo stesso sistema di alimentazione e lo stesso sistema di raffreddamento.

Alla luce di quanto detto, un aspetto molto importante in Open RAN è quello di porre enfasi sulla sostenibilità energetica di un sistema centralizzato. È molto più semplice infatti adempire ai concetti di energia rinnovabile su una *BBU Pool* piuttosto di gestire innumerevoli siti dove vengono istallate poche stazioni base. Nonostante i requisiti sopramenzionati richiedono innovazione in varie dimensioni che vanno dalla scalabilità ad architetture di rete mobile più nuove e flessibili, non è detto che il fronthaul e l'architettura C-RAN saranno obbligatori nella realizzazione della rete 5G, ma ad oggi sono sicuramente una soluzione valida per il lancio di quest'ultima.

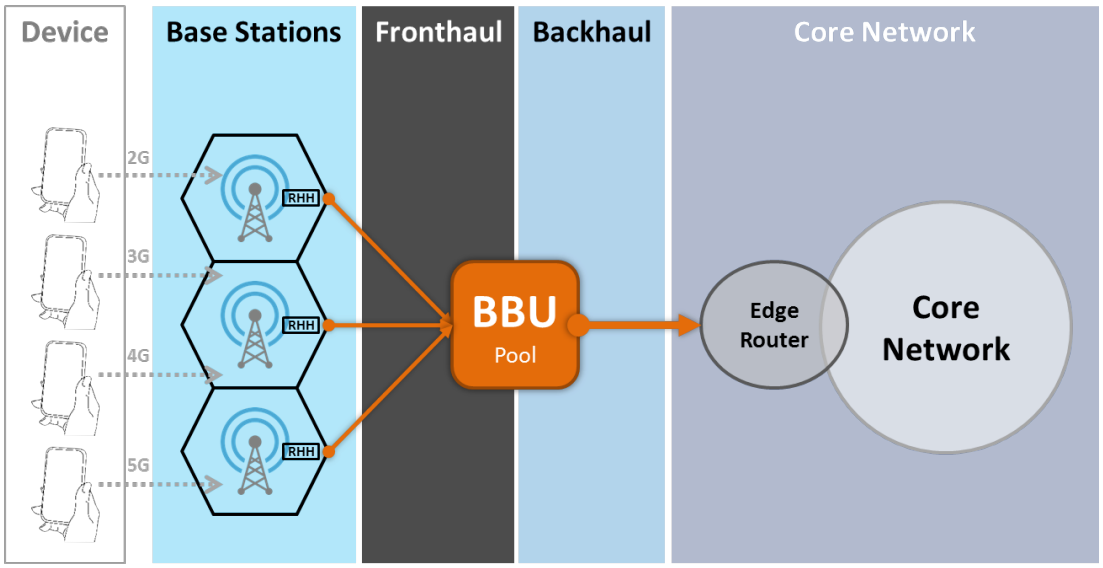

Figura 1.4. Trasformazione dalla rete mobile tradizionale in C-RAN. La BBU viene spostata dalla stazione base ad una *BBU Pool* dotata di connessione "Fronthaul"[5]

1.2.3 SDN

Delineata dagli studi del gruppo C-RAN, la visione di O-RAN vede come uno dei punti focali la separazione del *data plane* dal *control plane*, rimuovendo quest'ultimo dall'hardware di rete ed implementandolo in software. Ciò consente un accesso programmatico e, di conseguenza, rende l'amministrazione della rete più flessibile. La *Software Defined Network* (SDN) è infatti considerata tra le tecnologie chiave per l'evoluzione della rete mobile[1]. In particolare, le idee che stanno alla base di SDN vedono il disaccoppiamento del *data plane* dal *control plane* mediante API ben definite e l'introduzione di maggiore flessibilità nella rete attraverso la programmabilità. La RAN è la parte della rete mobile che senza dubbio può beneficiare dalle idee di SDN.
Una delle ragioni è che tutte le tecnologie usate per migliorare l'efficienza spettrale e la scalabilità richiedono un alto livello di coordinazione tra le stazioni base, cosa che SDN può naturalmente abilitare (si veda la densificazione delle celle e l'uso di diverse tecnologie di accesso radio).

1.2.4 Perché O-RAN

O-RAN Alliance è stata creata per accelerare la realizzazione di prodotti (nodi radio) in grado di supportare un'architettura aperta e comune, questi aspetti sono fondamentali per l'infrastruttura wireless di prossima generazione. Con il concetto di aperta si intende un'architettura le cui componenti sono descritte tramite interfacce aperte e note a tutti, quando si parla di comune si intende fare in modo che vengano adottate dalla maggior parte dei venditori di infrastrutture radio. L'*O-RAN Alliance* è costituita al contempo da operatori di rete, università e un'ampia comunità di fornitori hardware/software al fine di garantire innovazione e concorrenza aperta.

Il settore delle telecomunicazioni sta vedendo la virtualizzazione dell'infrastruttura, in continua crescita, combinarsi con l'intelligenza integrata per offrire servizi sempre più agili e funzionalità avanzate ai suoi clienti. In tal senso l'*O-RAN Alliance* è in prima linea nella definizione dei requisiti per questa trasformazione. Un'infrastruttura RAN che supporta le specifiche *O-RAN Alliance*, combinata con l'aumento della virtualizzazione RAN e l'intelligence basata sui dati, dovrebbe consentire una riduzione significativa dei costi operativi e di implementazione permettendo una più rapida diffusione della tecnologia.

Seguire le specifiche O-RAN, permetterà agli operatori di godere della flessibilità *multi-vendor* all'interno dell'architettura RAN in modo da ridurre la dipendenza da un singolo fornitore. Questo approccio devolverà le reti al di fuori della dipendenza da "*secret sauce*" specifica del fornitore (si veda l'implementazione di un protocollo CPRI specifico o di architetture RAN differenti tra i fornitori).

I vantaggi generali di un'architettura basata su un controllore centralizzato in grado di gestire e riconfigurare dinamicamente la modalità di trasmissione radio sono molteplici:

- Riduzione dei costi operativi per la *cross-compatibility* (ovvero per gestire la compatibilità tra due fornitori diversi)
- Nessuna necessità da parte dell'operatore di dover soddisfare requisiti sistemistici specifici per ogni fornitore
- La standardizzazione di interfacce comuni facilita il supporto di piattaforme software per *mobile edge computing* (MEC)
- Semplifica la gestione della Rete di Accesso 5G

Per raggiungere questi obiettivi, il lavoro di O-RAN Alliance incarnerà due principi fondamentali:

- ***Openness***: ovvero la definizione di interfacce aperte tra i diversi moduli dell'architettura radio
- ***Intelligence***: ovvero l'introduzione di moduli di controllo (come il *Radio Intelligence Controller*) capaci di riconfigurare in modo autonomo la rete seguendo le policy impostate dell'operatore

Capitolo 2

Controllore intelligente per la RAN (RIC)

2.1 Principi architetturali: interfaccia non-Real Time/near-Real Time

Come anticipato nel precedente capitolo, la RAN sta subendo una migrazione graduale da una architettura di tipo distribuita ad una di tipo centralizzata, chiamata anche *Cloud-RAN*. Ciò determinerà la sostituzione delle apparecchiature tradizionali con prodotti *general purpose*, interessando sia la core *network* che la *radio access network*.

Uno *split* flessibile tra i diversi elementi di elaborazione, nell'architettura *C-RAN*, consentirà di trovare un ottimo *trade-off* tra l'elaborazione centralizzata (CU/*Radio Cloud Center*) e quella all'edge (DU/*Remote Radio Unit*). Ogni *software function* presenta diversi requisiti hardware, perciò solo designando un'opportuna divisione della rete, gli operatori potranno ottimizzare i costi relativi ad esso, la semplicità operazionale ed il collegamento di trasporto *fronthaul*.

In modo da realizzare la suddetta divisione funzionale, è utile ragionare in termini di requisiti *Real Time* (RT) e *non-Real Time* (non-RT). In altre parole, i moduli funzionali che non necessitano di bassa latenza (nell'ordine dei secondi o più) saranno dislocati centralmente con lo scopo di coordinare l'area di copertura, mentre

quelli con alti requisiti real-time (nell'ordine dei millisecondi) saranno ancora locati nelle stazioni base.

Nella architettura appena descritta *Central Unit* (CU) e *Distributed Unit* (DU) possono essere definite come segue:

- **CU** è una importante parte dell'architettura che contiene una collezione di funzioni facilmente virtualizzabili in quanto non richiedono un hardware ad alte prestazioni. Le principali funzionalità supportate dalla central unit sono: il layer PDCP (*Packet Data Convergence Protocol*) e il *Radio Resource Control* (RRC). Tutte queste funzionalità, come già detto, non necessitano di bassissima latenza e sono dunque ideali per essere eseguite centralmente su hardware comune (*orchestration layer*). L'ente di standardizzazione che si occupa del 5G(il 3GPP) ha inoltre previsto la separazione del *control plane* (CP) dallo *user plane* (UP) come una caratteristica di rete obbligatoria. Il principale vantaggio proveniente dalla separazione delle funzionalità di *Control Plane* e *User Plane* deriva dal poter allocare risorse computazionali all' *User Plane* solo quando necessario (in presenza di utenti nella rete). Questo offre una maggiore efficienza ed una migliore gestione delle risorse radio.

- **DU** richiede alte prestazioni sia in termini di latenze che di capacità, coinvolge le funzionalità del livello più basso del nodo radio che in 5G si chiama *gNodeB* (RLC/MAC/PHY). Dunque decisioni quali *radio network scheduling*, *retrasmission* e *beams reconfiguration* sono prese all'edge della rete. Ad oggi queste funzioni richiedono un grosso carico computazionale e di conseguenza l'efficienza energetica è fondamentale. Proprio per questo è molto più efficienti eseguirle su hardware specifico piuttosto che su hardware *general purpose*.

Le suddette funzionalità di controllo di un nodo radio devono interfacciarsi on i due elementi prima citati e sono dislocate nel *RAN Intelligent Controller* (RIC).

Nel dettaglio i due blocchi in cui può essere diviso:

- **RAN Intelligent Controller (RIC) non-Real Time (non-RT) layer** è lo strato che si occuperà principalmente della *policy management* e quindi di "istruire" il RIC near-RT sul *layer* sottostante. Il modello verrà dunque distribuito per provvedere al RIC near-RT la capacità di modificare i comportamenti della RAN in base agli obiettivi della *policy*. L'interfaccia rilevante in questo scenario, designata dall'architettura O-RAN, è la *A1 interface* che permette la comunicazione tra il RIC non-RT e l'Orchestratore RAN
- **RAN Intelligent Controller (RIC) near-Real Time (near-RT) layer** si occupa fondamentalmente della comunicazione con la rete sottostante (il gNodeB) basandosi sulle informazioni ricevute dal RIC non-RT, per mezzo dell'interfaccia *A1*. Caratteristica del RIC near-RT è proprio quella di riconfigurare la rete usando le interfacce in real-time (*E2 interface*). La suddetta interazione tra il RIC non-RT ed il RIC near-RT, permette la realizzazione di una *Self Organized Network* (SON).

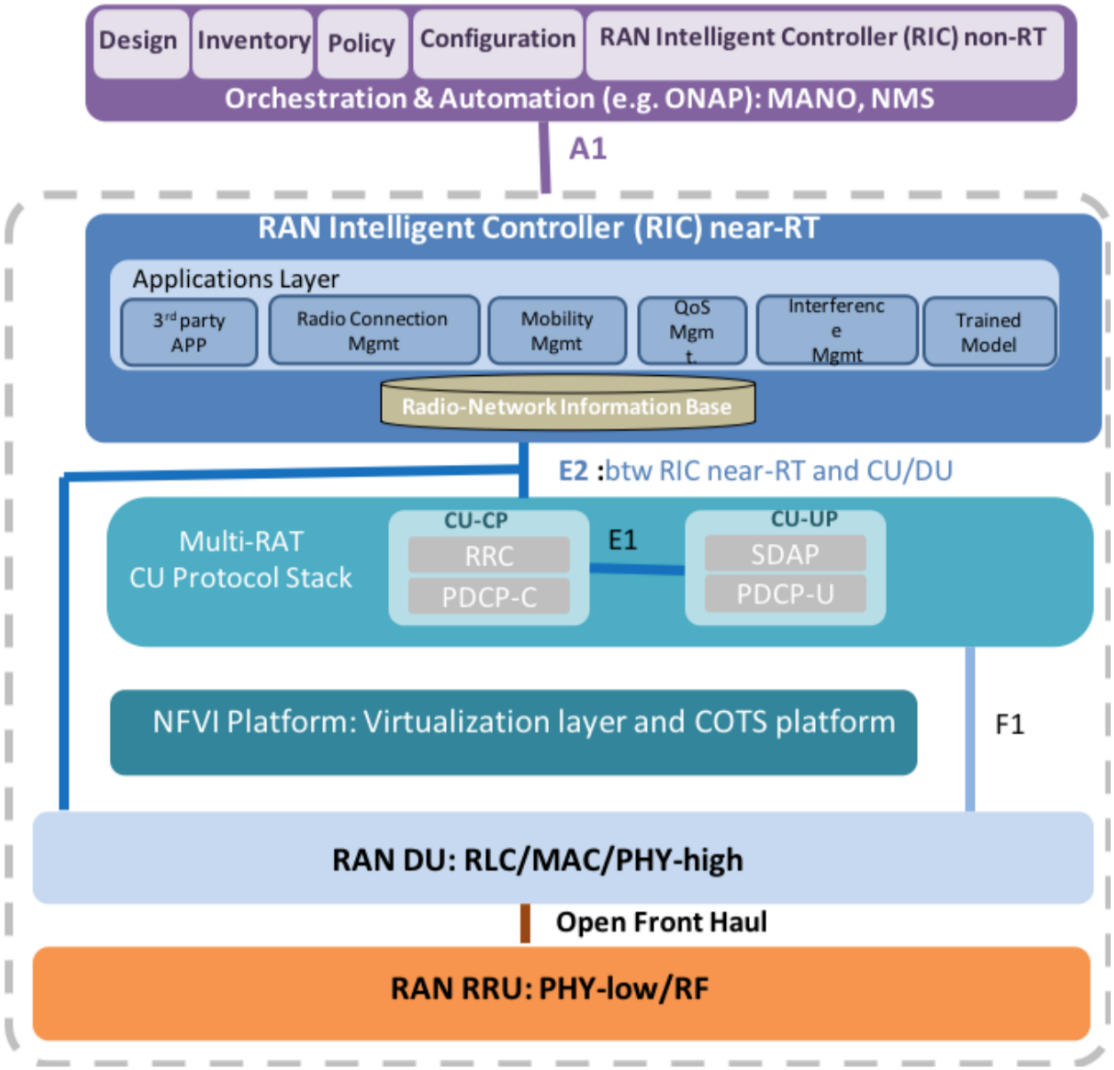

Figura 2.1. Alleanza *Open RAN*: architettura di riferimento

2.2 Funzionalità di *Self-Organized Network* abilitate dal RIC

Il concetto di *Self-Organized Network*, riferito alle reti cellulari, è un fattore chiave per il miglioramento delle attività di *Operations*, *Administration* e *Maintenance* (OAM).

In altre parole lo scopo principale di SON è quello di semplificare la gestione e massimizzare l'automazione delle reti *wireless*.

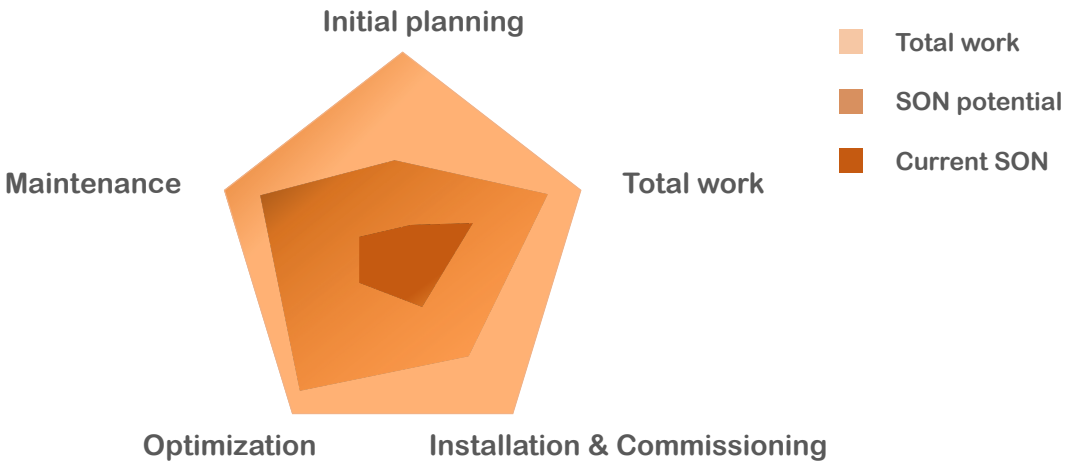

Figura 2.2. Panoramica dello stato di automazione corrente per le reti *wireless*

"The Next Generation SON should have the capability of full awareness of current status and the ongoing changes, the ability to do necessary analysis to determine optimal network parameter values, and the ability to implement the network adjustment, and thus minimize the human intervention as much as possible, and also to provide network maintenance in an optimal and timely fashion[6]"

SON dovrebbe essere in grado di identificare automaticamente i diversi tipi di scenari, provvedendo differenti strategie di configurazione in accordo alla *policy* distribuita dal *RAN Intelligent Controller* non-RT.
Scopo del RIC è infatti quello di sfruttare le funzionalità di SON per gestire la crescente complessità delle reti, sempre in continua evoluzione.

Sono stati definiti degli *use cases* significativi, che possono essere classificati in base alle fasi del ciclo di vita di un sistema cellulare (self-configuration, self-healing and self-optimization):

- *Slicing*
- *Coverage Extension*
- *Mobility Management*
- *Minimization Drive Tests*
- *Load balancing*

Nel dettaglio il lavoro di tesi si è concentrato su una implementazione *custom* della funzionalità di *Minimization Drive Tests* (MDT).

2.2.1 Minimization Drive Tests

MDT permettono agli operatori di collezionare misure sulla qualità del segnale ricevuto degli *User Equipment* (UEs) insieme alle informazioni relative all'interferenza presente, se disponibili. Lo scopo di queste misure è un continuo monitoraggio della qualità della rete riducendo comunque i costi operazionali.
Tra gli obiettivi di questa funzionalità ci sono la standardizzazione delle soluzioni per l'ottimizzazione della copertura, mobilità, capacità e QoS.

2.3 Divisione funzionale in OpenAirInterface (OAI)

Nata nel 2014, la "OpenAirInterface Software Alliance" (OSA) è una organizzazione francese no-profit il cui obiettivo è quello di provvedere software e strumenti

per supportare la ricerca delle reti di quinta generazione. OpenAirInterface (OAI) è una iniziativa open source che provvede l'implementazione software di una *base station* LTE (OAI eNodeB), un *user equipment* (OAI UE) ed una *core network* (OAI EPC1) eseguibile su sistemi di elaborazione *general purpose*. Il software OAI EPC è chiamato *openairCN* mentre il software di accesso rete va sotto il nome di *openair5G*.

Lo stack protocollare del eNodeB è suddiviso in 3 parti: la RRU, ossia una evoluzione della classica RRH, la *Radio Aggregation Unit* (RAU), il quale controlla diverse RRUs ed infine il *Radio Control Center* (RCC) che controlla a sua volta diverse RAUs. Nello specifiche 3GPP del 5G il RCC non è altro che la CU, mentre il RAU è la DU, implementate entrambe da OAI. In conclusione il software del sistema LTE copre l'intero protocol stack dello standard 3GPP, sia in *E-UTRAN* che in EPC.

2.3.1 USRP X-series/B-Series

OAI è progettato per essere un software "agnostico" alle piattaforme RF hardware. Può essere infatti interfacciato con apparecchiature di terze parti senza particolare sforzo. Supporta i prodotti della famiglia Ettus *USRP B-series/X-series* (tramite l'*Ettus USRP Hardware Driver software*) largamente usati negli ambienti di ricerca. Un esempio di RRU è la scheda USRP B210 usata proprio in *Telecom Italia Lab*.

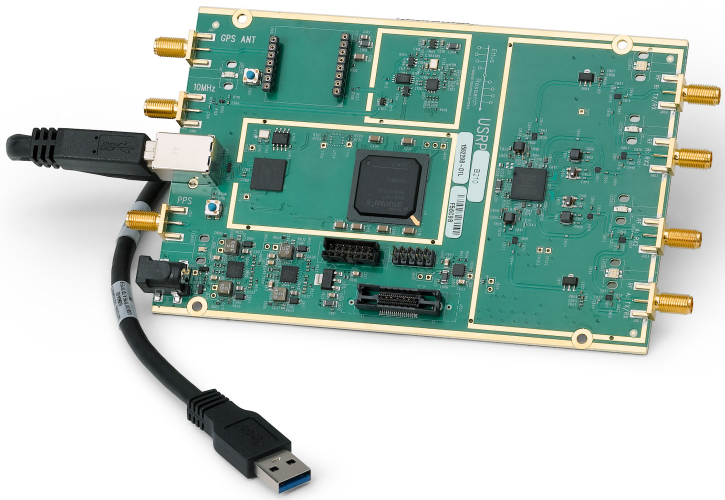

Figura 2.3. *Ettus USRP B210*

Parte II

Sezione sperimentale

Capitolo 3

Architettura

3.1 Introduzione

Questo capitolo tratta il primo approccio avuto con l'ambiente di lavoro e tutta la componentistica. L'intero lavoro di tesi, dalla fase di sviluppo alla fase di *debugging/testing* del software, è stato svolto all'interno del laboratorio di *Telecom Italia* (Tlab).
Nel dettaglio l'obiettivo di questa prima fase è stato quello di prendere dimestichezza con tutta la rete di laboratorio ed in particolare comprendere il funzionamento della *Core Network* (EPC) e degli *evolved Node Base* (eNB), in questo caso implementati dalla piattaforma OAI (*openair interface 5G*).
Successivamente si è passati all'analisi dei requisiti circa l'implementazione del *controller layer*. L'architettura dell'applicazione, la scelta del linguaggio di programmazione, le modalità di lancio e la valutazione del *database* sono stati infatti alcuni dei punti focali su cui si è concentrata questa prima fase di analisi.

3.2 Rete di laboratorio e soluzioni esistenti

In modo da semplificare le operazioni con la rete di test, EPC ed eNB sono state dislocate in differenti macchine fisiche, dati i diversi requisiti HW e SW.
Il software eNB, come già detto è stato implementato attraverso OAI e richiede PC

basati su architettura *Intel*. Per le funzionalità *Radio* di queste ultime sono state usate le schede *Ettus USRP B210*[7].

La rete di laboratorio usata, ai fini degli obiettivi di tesi, presenta la seguente configurazione:

- 4 OAI eNodeB
  - 4 PC Dell
  - USRP B210
- 2 EPC usabili con OAI
  - *openairCN*
  - ***Amarisoft***

In accordo a quanto detto precedentemente, il software OAI *eNodeB* è stato installato su quattro differenti PC ed ognuno configurato con una USRP B210 in modo da provvedere le funzionalità radio. La EPC usata è invece quella *Amarisoft*, preferita per maggiore stabilità a quella implementata da OAI.

All'inizio del lavoro di tesi la configurazione di laboratorio presentava già la definizione dell'interfaccia E2, uno dei componenti principali dell'architettura O-RAN (1).

Fondamentale per la comunicazione tra il *controller* ed i nodi sottostanti, quest'ultima è stata realizzata da un precedente lavoro di sviluppo in TIM[8]. Lo sviluppo dell'interfaccia ha interessato la modifica del codice sorgente di OAI e la scrittura di un OAI agent per permetterne la codifica/decodifica di informazioni, da/verso le opportune strutture dati del software OAI[8].

Il codice OAI *agent*, scritto in Python, è composto essenzialmente da due *threads* paralleli che, per sincronicità, sono stati implementati anche nella piattaforma OAI, modificando il codice sorgente C di quest'ultima. Il lavoro coordinato di questi due

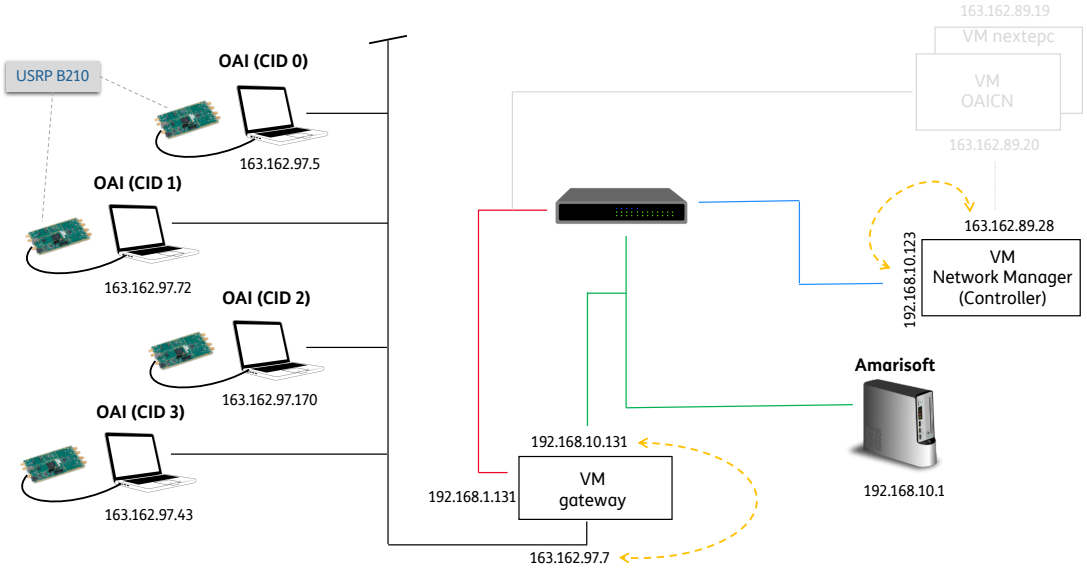

Figura 3.1. *Rete di test (TIM - LAB B0011)*

software, eseguiti simultaneamente su ognuno dei quattro *splinter*[1] , permette da un lato l'estrazione dei parametri di interesse da OAI propagandoli ai livelli soprastanti, dall'altro la riconfigurazione della comunicazione tra UE e rete.

Più in dettaglio, come si evince dalla Figura 3.2, le informazioni raccolte dal *reading thread* vengono trasmesse all'agente OAI, mentre le informazioni fornite dal RAN *controller* all'agente OAI vengono trasmesse al *writing thread*.
Ogni nodo OAI "pubblica" il proprio status periodicamente (attualmente ogni 5 secondi, ma configurabile) in modo da annunciarlo al *controller* mentre l'informazione trasmessa dal RAN *controller* al nodo OAI è inviata solo quando quest'ultimo necessita di una riconfigurazione.

[1]PCs basati su architettura Intel adibiti alla funzionalità di *eNodeB*

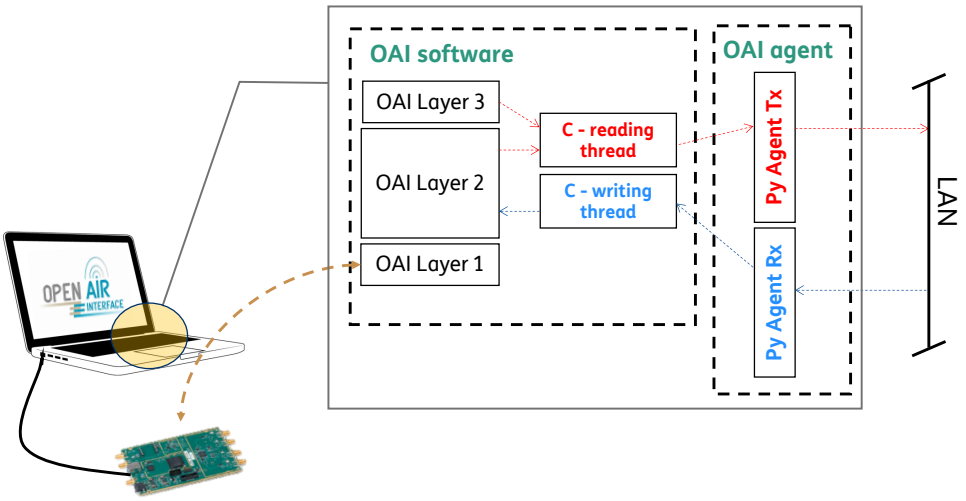

Figura 3.2. Agente OAI e modifiche apportate al software *OpenAirInterface* per *eNodeB*

Nel seguito si analizzano i principali parametri di interesse che definiscono l'interfaccia E2:

eNodeB OAI $\Longrightarrow$ RAN controller

- QCI
- CQI
- RSRP/RSRQ
- TMSI/RNTI

eNodeB OAI $\Longleftarrow$ RAN controller

- *Scheduler RBG (Resource Block Group) assigment for a specific UE*
- *RRC Connection Re-Configuration Modification*

Prima di descrivere in dettaglio l'implementazione del *controller* (Sezione 3.3), si propone una panoramica dell'architettura generale (fisica-logica) ed un possibile scenario di interesse che illustra il funzionamento della E2 *interface* (Figura 3.3). Sono raffigurati i componenti essenziali del *controller layer* ed un possibile *use case* di interazione con il nodo OAI. La serializzazione delle informazioni è stata basata sulla notazione JSON, ideale anche per l'interscambio di dati su internet[2].

[2]mantiene la compatibilità per l'interscambio di dati sull'interfaccia A1 (I)

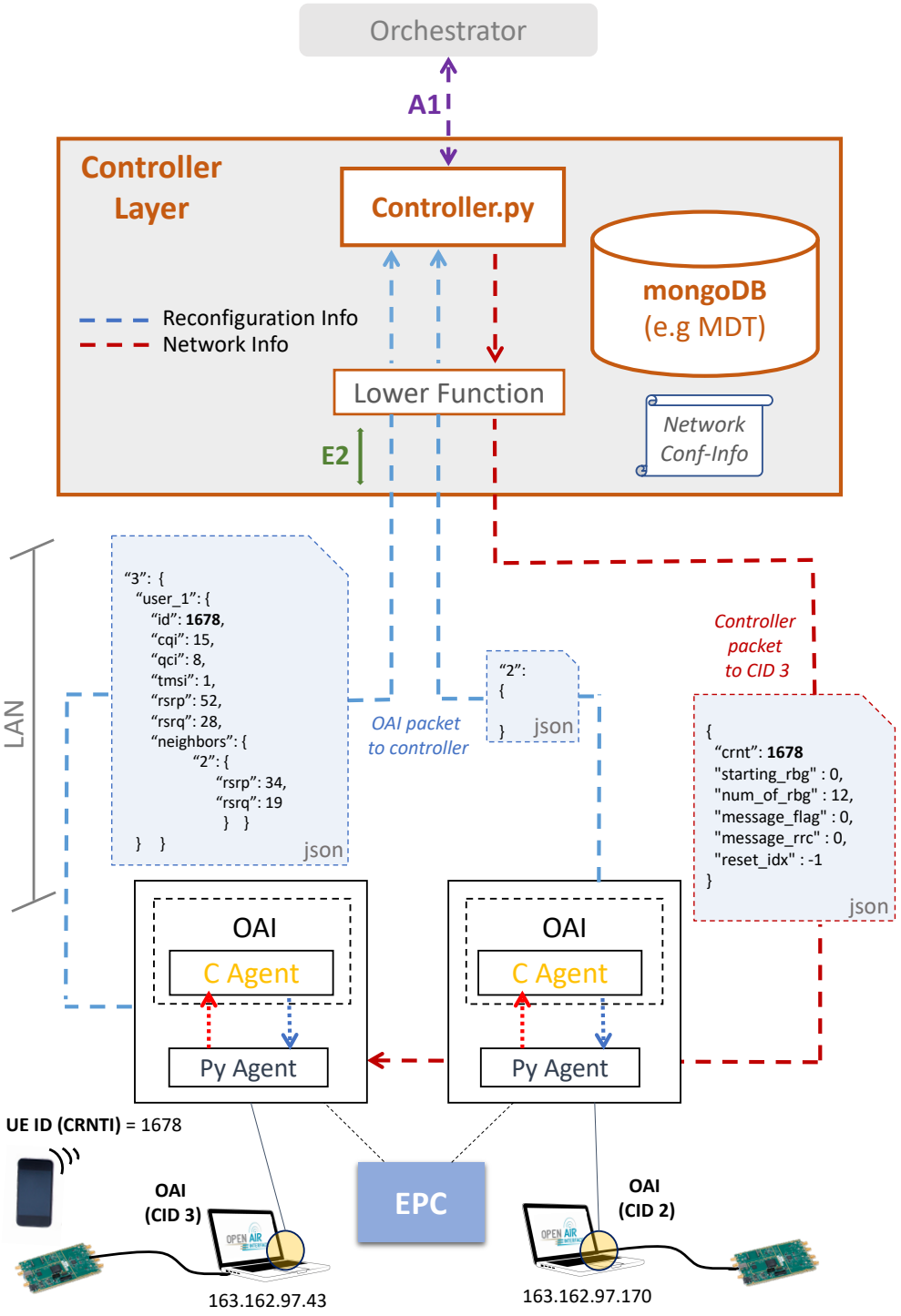

Figura 3.3. Architettura dello strato *controller*: caso di interazione con i nodi di rete

3.3 Strato controllore

3.3.1 Analisi dei requisiti

Uno dei requisiti principali nella realizzazione del *controller layer* (blocco Controller.py nella Figura 3.3) è stato quello di lanciare lo script fornendo diverse opzioni in input. Questo al fine di ottenere una gestione più flessibile del *controller layer*, dando all'utente la possibilità di scegliere la configurazione iniziale dell'applicazione. Le opzioni si focalizzano principalmente sulla possibilità di scegliere la *core network* su cui i nodi sono agganciati e la possibilità di selezionare la modalità di lancio del nodo.

La necessità di poter scegliere la *core network* è stata dettata dal fatto che, come visto nel precedente paragrafo, il laboratorio ne presenta più di una. Per il momento questa opzione è stata settata di *default* sul valore "AMARISOFT", in quanto più stabile più stabile rispetto all'altra *core network* a disposizione. Tuttavia in futuro potrebbero essere a disposizione altre *core network*. Il *tag* dell'opzione è "*-c, –core*" (*core network's name*) di tipo *required*.

Per quanto riguarda la modalità di lancio, si è optato per l'implementazione di due modalità, *normal mode* e *debug mode* utilizzata quest'ultima in fase di test e durante lo sviluppo di nuove funzionalità del *controller layer*.

3.3.2 Scelte implementative

Quando lo script è lanciato in *debug mode* una *common-line interface* (CLI) permette all'amministratore di rete di sfruttare le funzionalità del *controller layer* in loco, bypassando l'orchestratore.
In *normal mode*, si attiva invece la comunicazione con l'orchestratore tramite un servizio web basato sul protocollo HTTP (I).

Nel seguito un esempio d'uso dello script.

Unix terminal

```
MBP-di-Corrado: Corrado$ python3 run.py -h
usage: run.py [-h] [-m M] -c C [-n NODES [NODES ...]]
optional arguments:
-h, --help            show this help message
                      and exit
-m M, --mode M        normal or debug mode (d or n)
-c C, --core C        core network's name
-n NODES [NODES ...]  eNodeB list to monitor
```

In entrambe le modalità, il controllore comunica con gli eNodeB per mezzo dell'interfaccia E2, precedentemente discussa.

3.3.3 Modalità debug

Realizzata per mezzo di una interfaccia da linea di comando (CLI), questa modalità espone un elenco con il quale l'utente è in grado di visualizzare e/o manipolare lo stato della rete. Se e solo se durante il lancio dello script è verificata la presenza dell'opzione *-m* con valore *d*, viene visualizzato l'elenco delle funzionalità offerte dal controllore.
Nel dettaglio, la gestione dell'interfaccia e l'esecuzione dei relativi comandi scelti dall'utente sono gestite da un opportuno *thread* che viene lanciato in fase di inizializzazione dell'istanza *controller*. Per una questione di pulizia del codice anche le operazioni relative all'elenco sono state organizzate all'interno di una classe, seguendo dunque i concetti della *object oriented programming* (OOP).

L'interfaccia appena discussa visualizza le seguenti opzioni:

- TURN ON/OFF NODE

  In tale videata vengono richiesti l' identificativo/i dell'eNodeB da attivare o disattivare. Il programma instaurerà dunque una comunicazione con lo *splinter*, conoscendone il corrispondente indirizzo IP grazie al file di configurazione locato nel *controller layer*.

  Al fine di realizzare la comunicazione è stato scelto il protocollo *Secure Shell v2* (SSH2) in quanto l'applicativo necessita di stabilire una connessione sicura con i terminali remoti.

  Per tale fine è stata usata la libreria *paramiko 2.4.x* di *python*, in quanto ampiamente documentata e considerata una delle migliori soluzioni per instaurare connessioni sicure basate sul protocollo SSH2.

  Per mezzo di questa libreria è possibile eseguire l'opportuna applicazione per avviare l'*eNodeB*, direttamente sullo *splinter* desiderato.

  Una volta sicuri che entrambi, *eNodeB* ed EPC, sono stati correttamente configurati ed avviati, è possibile connettere un UE al nodo.

- CHECK RUNNING NODES

  Attraverso questa funzionalità l'utente riesce ad avere una visione migliore della condizione della rete. In particolare si possono specificare uno o più identificativi di nodo per avere informazioni circa lo stato di questi ultimi e quindi capire quali *eNodeB* siano attivi e quali invece no. Se nessun input è inserito, la funzione verificherà lo stato di tutti i nodi sotto la supervisione del *controller*.

  Più in dettaglio il *main thread* lanciato dalla classe *controller* cattura i messaggi da parte degli *eNodeB* ed aggiorna in tempo reale un'apposita struttura dati, utilizzata proprio per tenere traccia dei nodi attivi.

  Prima di descrivere in dettaglio l'algoritmo (3.3.5), vediamo quali sono i dati a disposizione e come l'applicativo possa riceverli/fornirli ed elaborarli.

  Per il trasporto dei messaggi si è preferito l'utilizzo del protocollo UDP in

quanto più rapido ed efficiente, inoltre essendo l'informazione trasmessa ad intervalli regolari non è necessaria nessuna conferma circa l'avvenuta ricezione del messaggio.

In questo caso il programma *controller* apre un *datagram socket* in ascolto (*listening*) ed è quindi il *server* UDP, mentre l'agente OAI è il programma *client* UDP in quanto apre il *datagram socket* dedicato all'invio dei dati. Quindi con lo scopo di contattare il *server* periodicamente. Questa attività permette quindi di monitorare costantemente la rete, andando a verificare che un nodo sia attivo e/o che la comunicazione sia stata correttamente instaurata.

Nel seguito un *eNodeB sample message*:

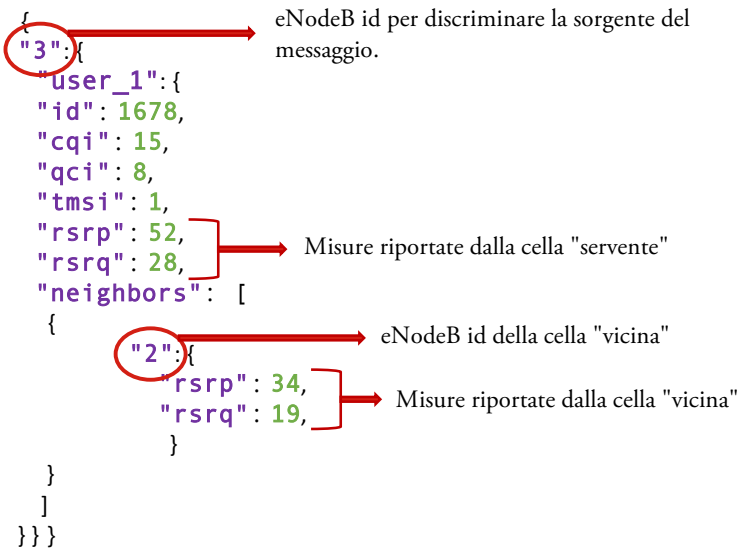

Figura 3.4. Messaggio inviato da un *eNodeB OAI* verso il *RAN controller*

- SEND RRC MEASUREMENT

  L'informazione inviata dal *controller* verso gli eNodeB OAI è invece trasmessa solo quando un nodo necessita di una riconfigurazione. In questo caso il messaggio è relativo ad uno specifico UE, identificato dal campo *crnti*, e contiene un campo usato per trasmettere un *radio resource control* (RRC) *message*.

Esaminando il caso d'uso MDT, questa funzionalità è stata sfruttata per configurare un *measurement report* su un determinato UE, in modo da permettere a quest'ultimo di comunicare alla rete le informazioni richieste. In genere, si tratta di misurazioni delle celle circostanti riguardo la potenza e la qualità del segnale ricevuto (e.g. RSRP, RSRQ). Nel seguito un RAN *controller sample message*:

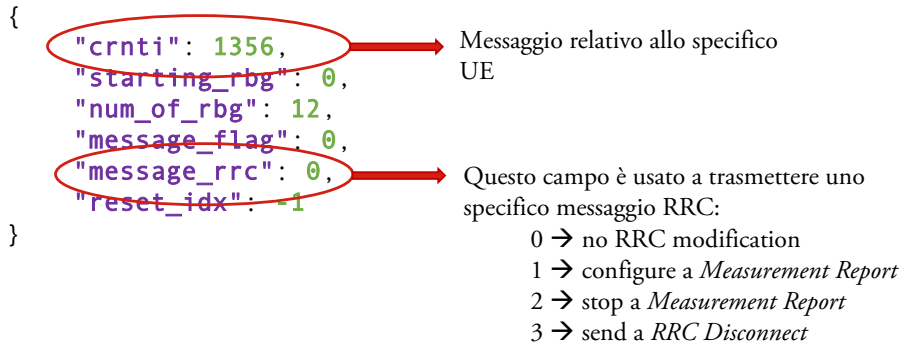

Figura 3.5. Messaggio inviato dal *RAN controller* verso un *eNodeB OAI*

3.3.4 Modalità normal

E' importante notare come tutte le funzionalità, offerte nella modalità debug appena descritta, possono essere sfruttate anche da remoto grazie all'interfaccia A1. Nella modalità normal, l'orchestratore può dunque comunicare con il controller, istruendolo in modo da incrementare l'automazione della rete di laboratorio.

Si è scelto di implementare la comunicazione tra orchestratore e controllore tramite l'esposizione di servizi REST, che risulta lo che risulta lo standard attualmente in uso insieme al SOAP per la comunicazione di dati o la fruizione di servizi tramite protocollo HTTP/HTTPS. La scelta del protocollo REST contrariamente al protocollo SOAP è stata fatta considerando la snellezza e la pulizia del codice. L'architettura generale dell'interfaccia A1 espone una URI per ognuna delle tre resource:

eNodeBs

espone un unico metodo GET con il quale si possono recuperare le informazioni circa i nodi ubicati all'interno di una certa area geografica. L'orchestratore fa una richiesta al servizio web, arricchita da una lista di coordinate e da una lista di tecnologie che descrivono rispettivamente l'area di interesse e le caratteristiche del nodo. Alla URI base viene infatti aggiunta una parte di *query-string* in cui si specificano le coordinate di un'area geografica.
Una volta ricevute le coordinate, il server definirà per mezzo di un'opportuna libreria un oggetto *GeoJSON* contenente una geometria di tipo *Polygon*. Usato per interrogare e filtrare opportunamente il database circa i nodi situati all'interno di una certa area.
Nella risposta verrà inoltre aggiunta l'informazione circa lo stato del nodo, sfruttando la funzionalità precedentemente discussa. Il server aggiornerà l'oggetto *Response* in base allo stato (*real-time*) del nodo specifico, ritornandolo al chiamante dopo averne fatto il *marshaling* della classe equivalente.

eutran

utilizzato dall'orchestratore per avere informazioni dettagliate riguardo la modalità di accesso della rete (interfaccia radio). Anche questa risorsa dispone di un solo *endpoint*, in particolare un metodo GET che produce un oggetto *EUTRAN generic cell*. Il *marshaling* del metodo fornirà anche in questo caso un JSON. La cella, con tutte le necessarie informazioni, è identificata dall'orchestratore per mezzo dell'id ottenuto dalla precedente interazione con la risorsa *eNodeBs*. In particolare per indirizzare la risorsa in questione è stato usato il metodo *Path Parameter*.
Dopo questa seconda interazione con il server, l'orchestratore avrà tutte le informazioni necessarie per chiedere al controller di istanziare un determinato MDT monitorando le EUTRAN cell desiderate (previa selezione da parte dell'utente).

mdt

a differenza dei due *endpoint* appena descritti, il metodo esposto da questa risorsa è una POST ed ha lo scopo di creare una istanza (documento) di MDT nell'opportuna collezione del database. Il *Content-Type* del *payload* dovrà essere necessariamente JSON, contenente l'oggetto MDT. Una volta ricevuta la richiesta il metodo farà prima di tutto l'*unmarshaling* da JSON all'oggetto specificato dallo *Schema*, successivamente la validazione per mezzo di un *jsonschema* definito ed infine si andrà ad inserire l'MDT nel database. Se i processi di *unmarshaling* e *validation* vanno a buon fine, il server restituirà una *Response* all'orchestratore contenente il riferimento del documento creato (ObjectId) con status code "*201 - CREATED*". Altrimenti lo status code sarà "*400 - BAD REQUEST*". E' importante notare che, da scelta implementativa, non sono stati permessi inserimenti di MDT duplicati. In tal caso il server restituirà un "*409 - CONFLICT*" come status code, interrompendo l'operazione.

3.3.5 Algoritmo

In modo da implementare gli algoritmi principali del controller, l'applicazione è stata basata sul paradigma multithread e, come già detto, realizzata mediante il linguaggio Python.
Nel dettaglio le operazioni svolte dal programma per realizzare l'algoritmo di *monitoring* della rete (come mostrato in Figura 3.6):

1. il processo padre creerà un nuovo thread, incaricato di gestire i messaggi UDP in arrivo dalla rete

2. il *thread* apre un *datagram socket* in ascolto sull'indirizzo IP e sulla porta specificati nel file di inizializzazione "*config.ini*"

3. a questo punto il *thread* sfrutterà il *socket* e rimarrà in ascolto per eventuali messaggi

4. fase di *learning*: ad ogni pacchetto ricevuto si aggiorna un'apposita struttura dati che memorizzerà i nodi dei messaggi più recenti. Per non saturare la struttura dati si è optato di "rinfrescarla" dopo un certo numero di occorrenze (=3)

5. rinfrescare la struttura dati permetterà all'algoritmo di rilevare automaticamente eventuali nodi spenti/malfunzionanti andando quindi ad aggiornare un opportuno file di *log*

L'algoritmo appena descritto viene eseguito costantemente ed in parallelo al normale funzionamento del programma *controller*. In questo modo l'applicativo è in grado di monitorare la condizione *real-time* della rete, andando semplicemente a leggere la suddetta struttura dati.

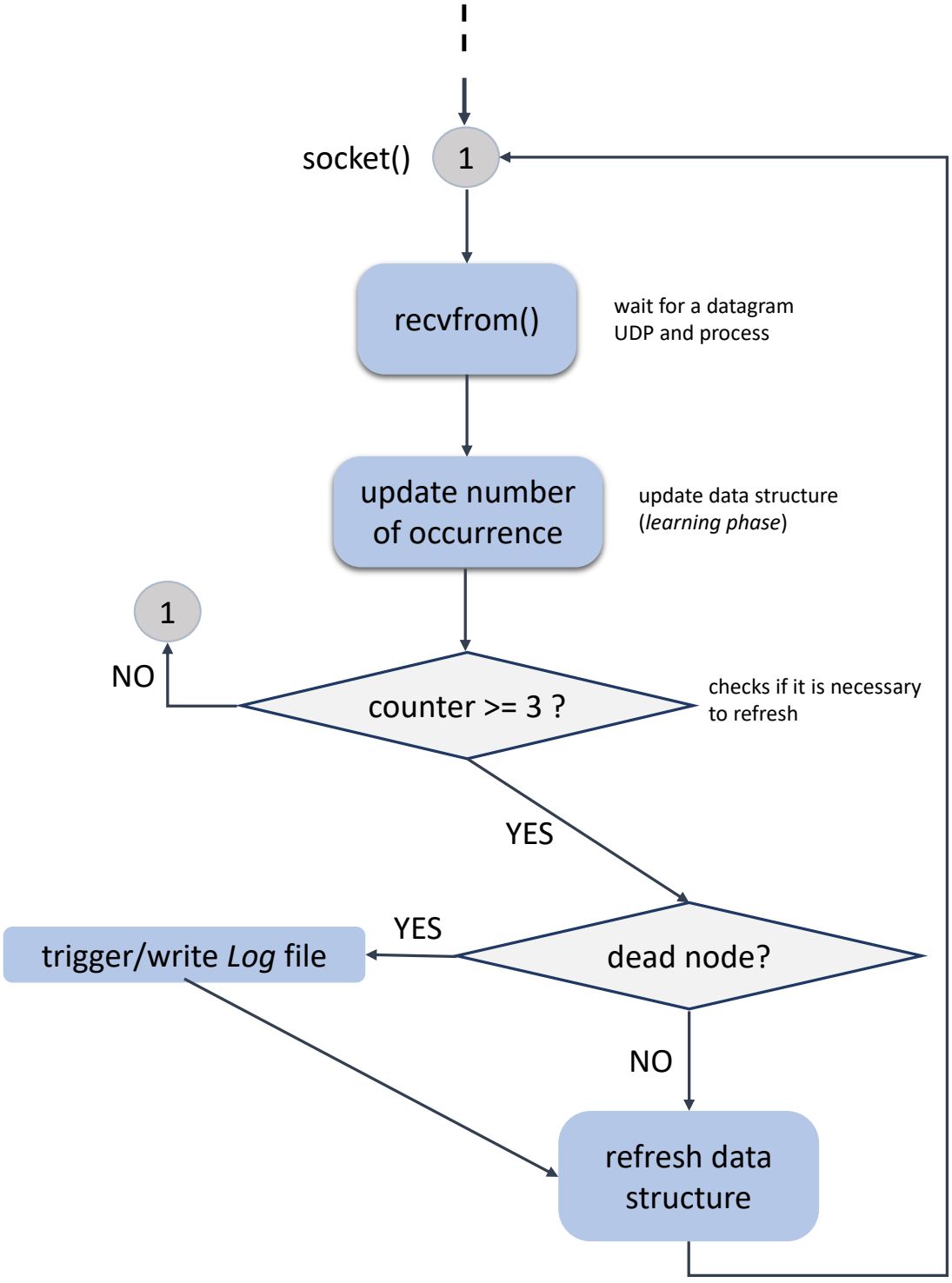

Figura 3.6. Flusso di esecuzione dell'algoritmo gestito dal *thread* di servizio della classe *Controller*

Capitolo 4

Minimization of Drive Tests

4.1 Introduzione

Le reti di comunicazione mobile devono essere monitorate ed ottimizzate in modo da provvedere buona copertura e qualità del servizio. Si pensi per esempio al problema dei "buchi di copertura" dovuti alle nuove costruzioni che isolano alcune aree. Per rilevare e migliorare tali problemi, le misurazioni radio sono necessarie. Queste misure possono essere effettuate con una attrezzatura specifica direttamente alla stazione base oppure da *Drive Tests* (DTs) per coprire l'intera area. In questo caso le misurazione sono collezionate per mezzo di automobili equipaggiate con un'attrezzatura specifica. Generare ed analizzare questi dati provoca però una grossa spesa operativa (OPEX) e mostra lo stato della rete solo ad un definito tempo e ad una definita posizione[12].

In modo da ridurre gli sforzi operazionali per i DTs, sono state studiate soluzioni specifiche che vanno sotto il nome di "*Minimization Drive Tests*" (MDTs). L'idea principale è quella di usare ogni dispositivo loggato in rete per collezionare dati di misurazioni, portando dunque una automazione e configurazione di queste ultime. Con MDT, gli operatori di rete mobile riescono a raccogliere da remoto le misurazioni che indicano la qualità di servizio della rete,sperimentata dagli stessi utenti. Ciò si traduce in una più ampia applicazione di casi d'uso che consente il monitoraggio e l'ottimizzazione della rete, senza la necessità di test di guida convenzionali.

4.2 Casi d'uso

Uno degli obiettivi principali per gli operatori telefonici è quello di massimizzare la copertura di rete e minimizzare l'uso dell'hardware. Ed ecco quindi come i MDT svolgono un ruolo basilare in questo contesto.

Alcuni dei principali casi d'uso per i MDTs sono: *coverage optimization*, *mobility optimization*, *capacity optimization*, *parametrization for common channels* e *Quality of Service* (QoS) *verification*.
La *coverage optimization* per esempio, è un aspetto molto importante per gli operatori, in quanto può facilmente influenzare la *user-experience* dell'utente. Si possono distinguere i seguenti sotto casi:

- *coverage hole detection*: aree in cui si verificano interruzioni di chiamate
- *identification of weak coverage*: aree in cui il livello del segnale è inferiore al livello pianificato
- *detection of excessive interference*: a causa della eccessiva sovrapposizione delle aree di copertura delle celle, possono verificarsi interferenze che degraderanno inevitabilmente la capacità della rete
- *overshoot coverage detection*: può accadere che una cella servente rilevi un forte segnale da parte di un'altra cella, nonostante questa sia molto lontana (*overshoot*)

L'attività della tesi è stata incentrata sulla possibilità di attivare sessioni MDTs basate sull'area di copertura. L'area su cui effettuare una sessione MDT viene selezionata dallo strato orchestratore ed inviata al *controller* mediante apposite APIs. Quest'area è definita come una lista di celle ed opzionalmente arricchita da una lista di UEs che devono effettuare le misure di qualità del segnale ricevuto.

Lo strato orchestrazione segnala il *reporting* delle misure al *RAN Intelligent Controller* che, a sua volta, indirizza l'eNodeB/UE per mezzo di connessioni RRC. Le misurazioni dovrebbero dunque permettere al *controller layer* di collezionare i dati opportunamente ed infine riportarli all'orchestratore.
I parametri collezionati, riguardanti indicatori di qualità e di potenza del segnale radio, vengono salvati in un apposito database e successivamente analizzati.

4.3 Implementazione

La raccolta delle misure da parte del *controller*, è stata implementata mediante una funzionalità che analizza e filtra tutto il traffico di rete. Come per la funzionalità di monitoraggio della rete (3.3.5) anche questa è basata sul paradigma *multithreading*. Per una migliore organizzazione del codice si è scelto di delegare tutte le funzionalità di gestione degli MDTs ad una classe specifica, "*MDTManager*".
Inizializzata dalla classe "*Controller*", quest'ultima lancia un nuovo *thread* che si occuperà dell'analisi dei singoli pacchetti e di gestire l'interazione con il database. La classe, in particolare il *thread* di servizio, gestirà quindi un *datagram socket*, configurato anche in questo caso grazie ai parametri di rete definiti nel file "*config.ini*" (Figura 4.1).
L'indirizzo IP sarà ovviamente lo stesso di quello usato dal thread gestito dalla classe Controller, mentre il numero porta diverso, così da discriminare le due funzionalità.

```
[MDT]
ip_address: 163.162.89.28
port: 6222
mdt_collection_header: mdt_collection_header
mdt_collection_data: mdt_collection_data
```

Figura 4.1. Sezione MDT del file di configurazione "config.ini"

4.3.1 Algoritmo di raccolta

Nel seguito sono descritti i passi eseguiti dall'algoritmo di raccolta dati[1] (come mostrato in Figura 4.2):

1. il processo padre creerà un nuovo thread, incaricato di gestire i messaggi UDP in arrivo dalla rete

2. il *thread* apre un *datagram socket* in ascolto sull'indirizzo IP e sulla porta specificati nel file di inizializzazione "*config.ini*"

3. a questo punto il *thread* sfrutterà il *socket* e rimarrà in ascolto per eventuali messaggi

4. quando un pacchetto è ricevuto si verifica la presenza di almeno un UE *attached* al nodo; altrimenti il flusso di esecuzione ritorna al punto 3

5. il database viene interrogato per constatare la presenza di un MDT che soddisfi i parametri del pacchetto ricevuto e che la sequenza temporale di quest'ultimo comprenda il *system time*; altrimenti il flusso di esecuzione ritorna al punto 3

6. l'ultimo controllo serve per verificare se il *controller* abbia già inviato all'UE lo specifico messaggio RRC; in caso negativo si abilitano le misure sull'UE selezionato e si ritorna al punto 3

7. se invece il messaggio di *Measurement Report* è stato precedentemente inviato, allora i dati sono validi per essere collezionati nell'opportuno(i) MDT

E' importante notare come, parallelamente all'algoritmo appena descritto, venga eseguito un altro controllo con il fine di monitorare eventuali MDT "*expired*". Questo controllo si basa ancora sul *system time* e, nel caso in cui quest'ultimo sia

[1] si noti come, data la stessa struttura dell'algoritmo di *monitoring*, i passi 1,2 e 3 sono i medesimi del precedente

maggiore del valore "*end*", il valore del campo "*status*" dell'MDT verrà aggiornato da "*pending*" a "*completed*".

4.4 Database

4.4.1 MongoDB

MongoDB è uno dei più noti database non relazionali (o NoSQL). Si tratta di una soluzione orientato ai documenti, che sfrutta il formato JSON per la memorizzazione e la rappresentazione dei dati.Esso memorizza i documenti in JSON, offrendo agli utenti la facilità d'uso e la flessibilità dei documenti JSON insieme alla velocità e alla ricchezza di un formato binario leggero (BSON).

Usato nell'implementazione del *database*, mongoDB ha permesso di sfruttare il traffico della rete di laboratorio per la memorizzazione dei documenti. Il formato *JSON like* di quest'ultimi ha reso più flessibile lo storage dei pacchetti attribuiti ai vari MDT, permettendo di concentrarsi solo sull'algoritmo di memorizzazione che sulla gestione dei dati.

4.4.2 Struttura

A differenza dei database relazionali, basati sulle tabelle, quelli non relazionali (NoSQL) sono organizzati in *collection*. Una collezione è un insieme di documenti. Ogni documento è identificato da un *Object_Id* univoco all'interno della collezione.

La suddetta struttura di MongoDB ha portato alla realizzazione di 2 collezioni per la gestione degli MDT. Si è optato infatti per la suddivisione della collezione principale in due sotto collezioni, anche per una questione prestazionale. Le due collezioni che gestiscono l'intestazione ed il contenuto dei dati MDT sono state chiamate rispettivamente *"MDT_collection_header"* e *MDT_collection_data*.

- MDT_collection_header

  il formato del documento enfatizza solo le informazioni specifiche della sessione

MDT e non il contenuto dei dati (misure raccolte). Ogni sessione MDT, non appena creata, è indirizzata da un'opportuno documento in questa collezione, contenente le informazioni necessarie circa l'istante di inizio e fine della raccolta dati, una la lista di celle da monitorare, una lista di UE (vuota se non specificata nel *body* della post), una *username*, uno *status* ed infine un campo *Object_Id* generato automaticamente da mongoDB.
La scelta di avere un numero limitato di informazioni, per i dati di intestazione, porta ad un miglioramento prestazionale nell'interrogare il *database*. Inoltre la quadrupla tempo di inizio/fine, lista di celle e lista di UEs permette di discriminare efficacemente le varie campagne MDT.

- MDT_collection_data
  un documento con lo stesso *Object_Id* del documento di intestazione nella collezione *"MDT_collection_header"*. Oltre all'Id ed alla lista di celle (per comodità duplicata), quest'ultima collezione contiene un campo *collected_data*. Struttura dati contenente appunto tutte le misure dei terminali raccolte per quella campagna MDT. Essendo quest'ultima (generalmente) di dimensione non trascurabile, è usata solo per altri inserimenti nel *database*, evitando quindi di recuperare tutta la struttura dati ogni volta che si vuole interrogare il DB per verificare l'esistenza di uno specifico MDT.

4.4.3 Abilitazione misure

E' possibile che ci siano più sessioni MDT aventi un *overlapping* delle celle. La stessa cella può dunque essere monitorata da più utenti/sessioni, che per motivi diversi hanno bisogno di collezionare le misure gestite da quella cella. La potenziale condivisione di celle da parte delle diverse sessioni MDT conduce ad una maggiore complessità nella gestione delle misure dei terminali mobile. Questa problematica è stata risolta sfruttando una collezione apposita nel *database*, la *measures_collection*. In particolare è stato studiato un algoritmo che permettesse di tenere traccia di quei

terminali su cui sono state attivate precedentemente le misure.
Alla creazione di una nuova sessione MDT, prima di inviare un messaggio di riconfigurazione RRC, l'algoritmo verifica che il terminale in questione non siano già stato riconfigurato,andando ad interrogare la suddetta collezione. Nella fase di completamento di una sessione MDT, la collezione è altrettanto gestita, eliminando gli opportuni *TMSI* da quest'ultima.

Grazie all'algoritmo appena descritto, che sfrutta il *database*, si è potuta gestire in modo corretto la abilitazione/disabilitazione del *reporting* delle misure sugli UE idonei.

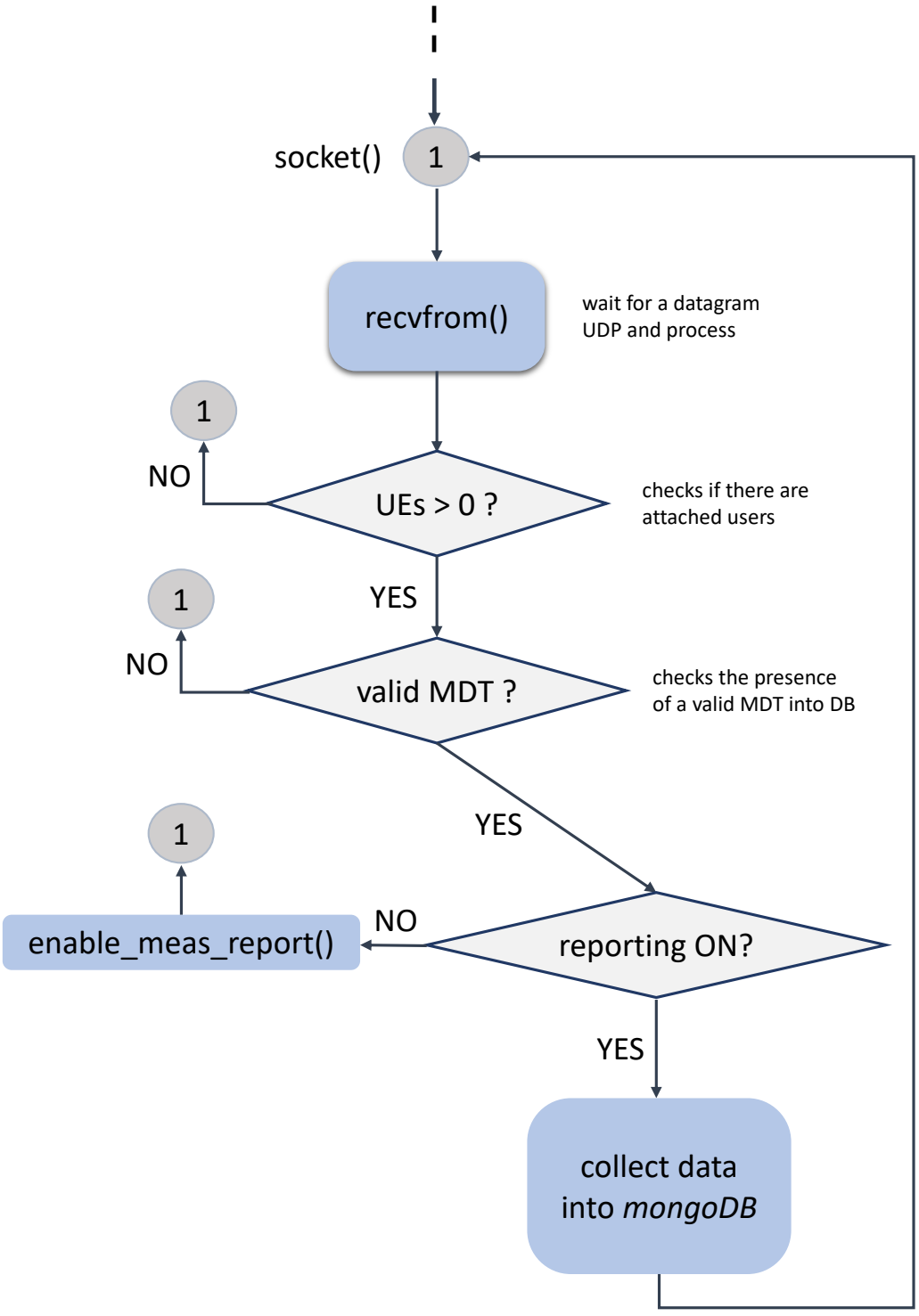

Figura 4.2. Flusso di esecuzione dell'algoritmo gestito dal *thread* di servizio della classe *MDTManager*

Capitolo 5

Conclusioni

Il lavoro di tesi descritto ha visto il raggiungimento dei seguenti risultati:

- realizzazione di un strato *controller* che permetta la gestione della rete di accesso radio
- implementazione di due modalità di lancio:
  - modalità *debug*: permette la gestione a basso livello degli eNodeB della rete di accesso
  - modalità *normal*: impostazione preliminare di un'interfaccia di comunicazione con l'orchestratore, basata sul protocollo HTTP
- capacità dello strato controllore di abilitare il *reporting* di misure, circa la qualità del segnale, da parte dei terminali *mobile*; nel dettaglio si è riusciti a gestire in modo autonomo le diverse sessioni MDT.

Le principali criticità che si sono individuate nel corso della ricerca hanno riguardato principalmente la gestione, temporale e capillare, delle varie sessioni MDT. E' stato infatti necessario mantenere aggiornata un'apposita struttura dati che identificasse i terminali aventi già il *reporting* delle misure abilitato.
Altra problematica è stata riscontrata nella individuazione delle sessioni temporali degli MDT. Identificando quindi l'istante esatto in cui attivare e disattivare il *reporting* delle misure su uno specifico terminale (UE).

Tuttavia, ai fini di un utilizzo reale e un test sul campo, sarebbe interessante sostituire la piattaforma software OAI, utilizzata in questo lavoro di tesi, con componenti commerciali.

In conclusione il lavoro effettuato ha portato comunque dei risultati interessanti, che costituiscono un nuovo passo avanti verso il concetto di *Self-Organized Network* per le reti *wireless*, semplificandone dunque la gestione e massimizzandone l'automazione.
Il SON di prossima generazione dovrebbe avere infatti la capacità di eseguire l'analisi necessaria per determinare i valori dei parametri di rete migliori e di provvederne una manutenzione in modo ottimale e tempestivo, riducendo quindi al minimo l'interazione umana.

Appendice I

Interfaccia A1

I Introduzione

REpresentational State Transfer (REST)

REpresentational State Transfer (REST) è uno stile di architetturale software[9] che definisce una serie di vincoli da utilizzare per la creazione di servizi Web. I servizi Web conformi allo stile architetturale REST, definiti servizi Web RESTful, garantiscono l'interoperabilità tra i sistemi informatici su Internet. I servizi web RESTful consentono ai sistemi richiedenti di accedere e manipolare rappresentazioni testuali delle risorse Web utilizzando un insieme di operazioni stateless uniformi e predefinite. In un servizio web RESTful, le richieste fatte all'URI di una risorsa susciteranno una risposta con un payload formattato in HTML, XML, JSON o in qualche altro formato. La risposta può confermare che alcune modifiche sono state apportate alla risorsa memorizzata e la risposta può fornire collegamenti ipertestuali ad altre risorse correlate o raccolte di risorse. Quando viene utilizzato HTTP, essendo il più comune, le operazioni disponibili sono GET, POST, PUT, DELETE e altri metodi CRUD HTTP predefiniti.

JavaScript Object Notation (JSON)

JavaScript Object Notation (JSON) è un formato leggero di scambio di dati[10]. È infatti facile per gli umani leggere e scrivere, e per le macchine analizzare e generare. Si basa su un sottoinsieme del linguaggio di programmazione JavaScript ed è completamente indipendente dalla lingua, ma utilizza convenzioni familiari ai programmatori della famiglia di linguaggi C, inclusi C, C ++, C#, Java, JavaScript, Perl, Python e molti altri. Queste proprietà rendono JSON un linguaggio di interscambio dati ideale.

GeoJSON

GeoJSON[11] è un formato aperto utilizzato per archiviare una collezione di geometrie spaziali i cui attributi sono descritti attraverso *JavaScript Object Notation*. Le geometrie possibili sono punti, linee spezzate, poligoni, e collezioni multiple di queste tipologie.
Le geometrie GeoJSON non devono però necessariamente rappresentare entità geografiche: ad esempio, i software di navigazione assistita possono usarlo per descrivere l'area di copertura del servizio

II Documentazione del servizio web

/eNodeBs

GET /

Descrizione:

Recupera le informazioni riguardanti la stazione base (eNodeB) che specifica i seguenti requisiti: 'technologies', 'coordinates' (cioè ubicata all'interno del poligono disegnato dall'utente)

Parametri richiesti:

Nome	Tipo parametro	Richiesto	Descrizione
coordinates	Query String	NO	Lista di coordinate (Polygon)
technologies	Query String	NO	Lista di tecnologie

Esempio richiesta:

163,162,89,28:5055/**eNodeBs***?coordinates*=40.078811,-76.730422,41.078811,-74.730422,
40.078811,-74.730422,39.961879,-76.730422,39.961879,-76.730422,40.078811,-76.730422

Risposte:

200 OK


```
[
  {
    "id": "TO01XX2_2",
    "eNBID": 10101,
    "f_type": "OAI_eNodeB_Function",
    "status": "ON",
    "cells": [
        2,
        3
    ]
  }
]
```


404 NOT FOUND


```
{
    "error": "eNodeB Function non trovata!"
}
```


/eutran

GET /{id}

Descrizione:

Ritorna al client i dettagli circa la EUTRAN Generic cell specificata dall'***id***.

Parametri richiesti:

Nome	Tipo parametro	Richiesto	Descrizione
id	Path Parameter	YES	Identificativo cella E-UTRAN

Risposte:

200 OK


```
[
  {
    "id": "TO01XX2",
    "localCellId": 2,
    "pci": 2,
    "PLMN": 10101,
    "eu_type": "FDD",
    "earfcnDl": 3400,
    "earfcnUl": 21400,
    "BandWidth": 20,
    "ueList": [
        1678
    ]
  }
]
```


404 NOT FOUND


```
{
    "error": "EUTRAN Generic cell non trovata!"
}
```


/mdt

POST /

Descrizione:

Endpoint usato dal client (orchestratore) per istanziare una nuova sessione MDT (Minimitazion Drive Tests).

Parametri richiesti:

Nome	Tipo parametro	Richiesto	Descrizione
start	RequestBody	YES	Tempo di inizio misurazioni
end	RequestBody	YES	Tempo di fine misurazioni
cells_id	RequestBody	YES	Lista di celle da monitorare
ue_list	RequestBody	NO	Lista di UEs da monitorare

Risposte:

201 CREATED

```
{
    "start": 1550416283,
    "end": 9999999999,
    "cells_id": [
        1,
        3
    ],
    "ue_list": [
        1678
    ]
}
```

409 CONFLICT

```
{
"error": "MDT gia' esistente: duplicato non permesso"
}
```

415 UNSUPPORTED MEDIA TYPE

```
{
"error": "L'entita' della richiesta e' in un
          formato non supportato"
}
```

422 UNPROCESSABLE ENTITY

```
{
"error": "La validazione non ha avuto successo!"
}
```

Bibliografia

[1] *"FlexRAN: A Flexible and Programmable Platform for Software-Defined Radio Access Networks"* (Xenofon Foukas, Navid Nikaein, Mohamed M. Kassem, Mahesh K. Marina e Kimon Kontovasilis)

[2] *"5G, C-RAN, and the Required Technology Breakthrough"* https://medium.com/@miccowang/5g-c-ran-and-the-required-technology-breakthrough-a1b2babf774

[3] *"Centralized Radio Access Network (C-RAN) Transport"* https://www.fujitsu.com/us/Images/Centralized-Radio-Access-Network-C-RAN-Transport-Application-Note.pdf

[4] *"CPRI and OBSAI"* https://www.anritsu.com/en-US/test-measurement/technologies/cpri-and-obsai

[5] *"Scenario eCPRI fronthaul in 5G networks"* https://cdn-images-1.medium.com/max/1540/1*cn-zu3IrkAMPrw-iWM7Nig.png

[6] *"White Paper by TELUS and Huawei"* https://www-file.huawei.com/-/media/corporate/pdf/mbb/next-gene-ation-son-for-5g.pdf

[7] *"USRP B210 datashit"* https://www.ettus.com/all-products/ub210-kit/

[8] *"M. Gucciardo, I. Tinnirello, G.Dell'Aera, M. Caretti, "A Flexible 4G/5G Control Platform for fingerprint-based indoor localization", accepted at SMILING, INFOCOM workshops, 29 April 2019"*

[9] *"Representational state transfer"*
https://en.wikipedia.org/wiki/Representational_state_transfer

[10] *"JavaScript Object Notation"*
https://www.json.org/

[11] *"The GeoJSON Format"*
https://tools.ietf.org/html/rfc7946#page-9

[12] *"Minimization of Drive Tests (MDT) in Mobile Communication Networks"* (*Daniel Baumann*, Department of Computer Science, Technische Universitat Munchen)

Elenco delle figure