UNIVERSITÀ DI PISA

DIPARTIMENTO DI INGEGNERIA DELL'INFORMAZIONE

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

PROGETTAZIONE ED IMPLEMENTAZIONE DI UN SERVIZIO DISTRIBUITO DI LOOK-UP DELLE RISORSE PER L'INTERNET DELLE COSE

MARCO LUPI

RELATORI

PROF. ENZO MINGOZZI

PROF. GIOVANNI STEA

DOT. CARLO VALLATI

DOT. GIACOMO TANGANELLI

ANNO ACCADEMICO 2013/2014

Ai miei Genitori e
a mia sorella Ilaria

INDICE

INDICE DELLE FIGURE

INDICE DELLE TABELLE

INTRODUZIONE

L'internet delle cose [IoT[1]] [23] è un nuovo concetto che sta crescendo in questi anni riferito all'estensione di internet al mondo degli oggetti. L'idea è quella di rendere gli oggetti intelligenti, permettendogli di comunicare dati su se stessi o informazioni raccolte ed accedere informazioni pubblicate da altri in modo da far acquisire agli oggetti stessi un ruolo attivo grazie al collegamento alla rete. Grazie a questo l'internet del futuro prevede la concezione di nuovi servizi, sensibili al contesto, che avranno la capacità di migliorare la qualità della vita degli utenti.

Per realizzare tale visione è di primaria importanza lo sviluppo di meccanismi efficaci, per scoprire le risorse e le informazioni rese disponibili dagli oggetti. Questi meccanismi sono necessari per trovare le informazioni, anche se l'esatta posizione e la forma d'immagazzinamento di queste ultime sono inizialmente sconosciute al richiedente. I Servizi di Discovery (DS) sono finalizzati a colmare questa lacuna. Riguardo alla progettazione dei DS ci sono vari approcci da poter seguire che portano a proposte distinte di architetture da utilizzare, una delle quali è l'architettura P2P-DHT che verrà analizzata in questo lavoro.

Il contributo di questo lavoro consiste nella progettazione e implementazione di un servizio distribuito di look-up delle risorse per l'internet delle cose. La soluzione proposta adotta un approccio peer-to-peer, grazie all'uso di tabelle hash distribuite (DHT[2]), per garantire scalabilità, affidabilità, robustezza e facilità di manutenzione del sistema complessivo. A differenza della maggior parte dei servizi peer-to-peer presenti in letteratura per il supporto dell'IoT, che implementano solo la ricerca su un singolo attributo (ad esempio l'identificatore dell'oggetto), la soluzione proposta è in grado di gestire query multi-attributo e range query. Questo comporta che non è necessario conoscere il valore esatto dell'identificatore dell'oggetto che si desidera raggiungere, come invece avverrebbe nelle normali DHT, permettendo alle applicazioni client di scoprire le informazioni e le capacità funzionali di oggetti che non sono stati ancora individuati e non sono necessariamente in

[1] IoT = Internet of Things
[2] DHT = Distributed Hash Table

prossimità fisica. Un altro vantaggio è che nella maggior parte dei casi una risorsa può rendere disponibili dati su attributi differenti e quindi non possiamo utilizzare una sola chiave per inserirla all'interno della DHT, dovremmo invece rendere disponibile una chiave per ogni attributo che la risorsa possiede in modo da poterla identificare sotto tutti i suoi aspetti.

L'approccio proposto permetterà, quindi, di poter ricercare una risorsa sulla base delle proprietà dell'oggetto che la fornisce senza conoscere a priori l'identificatore dello stesso, come invece avviene nelle normali DHT.

Questo lavoro di tesi si è principalmente diviso in due parti principali: un primo lavoro di ricerca e analisi delle implementazioni di DHT multi-attributo già presenti in letteratura anche utilizzate in campi differenti dall'IoT e successivamente la progettazione e l'implementazione del sistema seguita dalla valutazione delle performance.

La tesi sarà organizzata in questo modo: nel Capitolo 2 inquadriamo i requisiti funzionali per i Discovery Services, ed eseguiamo una panoramica e un'analisi di cinque approcci consolidati che sono presi nella letteratura e nella pratica industriale. Il Capitolo 3 riguarda la descrizione dei vantaggi dell'architettura scelta a confronto con le altre scartate. Il Capitolo 4 descrive il lavoro di implementazione svolto e le scelte imlementative. Il Capitolo 5 presenta la descrizione della valutazione delle prestazioni. Il Capitolo 6 tratterà le conclusioni e i possibili lavori futuri.

INTERNET DEGLI OGGETTI

L'Internet degli Oggetti (IoT) è un nuovo concetto che sta rapidamente guadagnando terreno nello scenario delle moderne telecomunicazioni wireless. L'idea di base di questo concetto è la presenza pervasiva di cose o oggetti che ci circondano (ad esempio: identificatori a radiofrequenza(RFID), sensori, attuatori, telefoni cellulari, ecc) che sono in grado di interagire tra loro e cooperare con i loro vicini per raggiungere obiettivi comuni.

Indubbiamente, il principale punto di forza l'idea degli oggetti sarà l'elevato impatto che avrà su vari aspetti della vita quotidiana e sul comportamento dei potenziali utenti. Dal punto di vista di un utente privato, gli effetti più evidenti dell'introduzione di questa tecnologia saranno visibili sia in ambiti domestici sia di lavoro. In questo contesto, la domotica, l'e-healt, sono solo alcuni esempi di possibili scenari applicativi in cui questa nuova tecnologia svolgerà un ruolo di primo piano nel prossimo futuro. Allo stesso modo, dal punto di vista dell'ambiente lavorativo, le conseguenze più evidenti saranno ugualmente visibili in settori quali l'automazione e la produzione industriale, la logistica, l'attività e i processi di gestione e il trasporto intelligente di persone e merci.

Partendo dalle considerazioni appena fatte, non dovrebbe sorprendere che l'IoT è stato incluso dal US National Intelligence Council nella lista delle sei "Rivoluzioni civili tecnologiche" con potenziale impatto sulla potenza economica degli Stati Uniti. Il NIC prevede che nel 2025 Internet risiederà nelle cose di tutti i giorni: confezioni alimentari, mobili, documenti cartacei, e molte altre cose. Sono evidenziate anche le possibili minacce derivanti da una diffusa adozione di tale tecnologia. Infatti, si evidenzia che gli oggetti di uso quotidiano potrebbero diventare rischi per la sicurezza dell'informazione, l'internet degli oggetti potrebbe incrementare tali rischi in maniera molto più ampia rispetto ad oggi.

In realtà, molte questioni complesse sono ancora da affrontare in più devono essere superati pregiudizi sociali e tecnologici prima che l'idea degli oggetti sia ampiamente accettata. Enti importanti stanno utilizzando dispositivi collegati tra loro, fornendogli un grado sempre più elevato di intelligenza, consentendo loro adattamento all'ambiente e comportamento autonomo, garantendo nel contempo la fiducia, la privacy e la sicurezza. Inoltre, l'idea degli oggetti pone diversi nuovi problemi riguardanti gli aspetti di networking. In realtà, gli oggetti che compongono la IoT saranno caratterizzati da poche risorse sia in

termini di calcolo e capacità di energia. Di conseguenza, le soluzioni proposte devono prestare particolare attenzione all'efficienza delle risorse oltre che ad evidenti problemi di scalabilità.

Diversi reparti di standardizzazione e di ricerca sono attualmente coinvolti in attività di sviluppo di soluzioni per soddisfare i requisiti tecnologici evidenziati. Questo capitolo fornisce un quadro dello stato attuale della ricerca sull'IoT :

- Fornisce ai lettori una descrizione delle diverse visioni dell'Internet of Things, provenienti da diverse comunità scientifiche;
- Recensisce le tecnologie esistenti e illustra quali sono i principali vantaggi della diffusione nella vita quotidiana;
- Offre un'analisi delle principali problematiche che la comunità scientifica deve ancora affrontare.

2.1. Idee alla Base dell'IoT

Consultando la letteratura, un lettore interessato potrebbe avere una reale difficoltà a capire cosa significa veramente l'IoT, che idee fondamentali stanno alla base di questo concetto e quali implicazioni sociali, economiche e tecniche si avranno con il pieno utilizzo degli oggetti.

Iniziamo con il nome "Internet of Things" che sintatticamente si compone di due termini, il primo spinge verso una visione di rete orientata all'internet degli oggetti, mentre il secondo sposta il focus sugli "oggetti" in generale e la loro integrazione in un quadro comune.

Le differenze, a volte sostanziali, nelle visioni dell'internet degli oggetti sottolineano il fatto che le parti interessate, alleanze commerciali, ricerca e organismi di standardizzazione si avvicinano al problema o da una prospettiva "orientata all'Internet", o "orientata agli Oggetti", a seconda dei loro specifici interessi e finalità.

Non si deve dimenticare, comunque, che le parole "Internet" e "Things", quando sono messe insieme, assumono un significato che introduce un livello di innovazione dirompente nel mondo dell'ICT[1] di oggi. Infatti, "Internet delle cose" significa semanticamente *"una rete mondiale di oggetti interconnessi e indirizzabili in modo univoco, sulla base di protocolli di*

[1] Information and Communication Technology

comunicazione standard". Ciò implica un numero enorme di oggetti (eterogenei) coinvolti nel processo.

L'identificazione univoca di un oggetto e la rappresentazione e memorizzazione delle informazioni scambiate sono due delle questioni più difficili, queste portando direttamente ad una terza prospettiva di avvicinamento al problema, "orientata alla Semantica".

L'Internet degli oggetti sarà il risultato della convergenza delle tre visioni principali affrontate in precedenza.

Il nome "Internet degli Oggetti" è stato attribuito agli Auto-ID Labs, una rete mondiale di laboratori di ricerca accademica nel campo della tecnologia RFID. Questa istituzione, insieme ad EPCglobal, fin dalla loro nascita, hanno avuto come obiettivo l'architettura e la definizione di precisi standard per l'Internet degli oggetti. Tali standard sono finalizzati principalmente a migliorare la visibilità dell'oggetto (cioè la tracciabilità di un oggetto e la consapevolezza del suo status, la posizione corrente, ecc.).

Secondo gli autori di [24], la tecnologia RFID si trova ancora in prima linea a guidare la visione sull'IoT. Questo è una conseguenza della maturità di RFID, basso costo, e forte sostegno da parte della comunità. Tuttavia, essi affermano che un ampio portafoglio di dispositivi, reti e servizi di tecnologie finirà per costituire L'Internet degli oggetti. Near Field Communications (NFC) e Wireless Sensor Networks e Actuator Networks (WSAN) insieme RFID sono riconosciuti come i "componenti che collegheranno il mondo reale con il mondo digitale".

Come detto prima, vale la pena notare che la visione "orientata alla semantica" dell'Internet degli oggetti è disponibile in letteratura [25-28]. L'idea alla base è che il numero di elementi coinvolti nell'Internet del Futuro è destinato a diventare estremamente elevato. Pertanto, le questioni relative al modo di rappresentare, conservare, interconnettere, ricercare e organizzare le informazioni generate dagli oggetti diventerà molto impegnativo. In questo contesto, le tecnologie semantiche potrebbero svolgere un ruolo chiave. In realtà, si potrebbero sfruttare adeguate soluzioni di modellazione per la descrizione degli oggetti, ambienti di esecuzione semantica e architetture che soddisfino le esigenze dell'Internet degli oggetti, che salvino i dati in modo scalabile conservazione e che permettano la comunicazione tra le varie infrastrutture.

2.2. Problematiche da Affrontare

In questa sezione, saranno mostrati i temi di ricerca più importanti che devono essere sviluppati per soddisfare i requisiti che caratterizzano lo scenario dell'internet degli oggetti. Più in particolare, nella sezione 2.3.1 ci concentreremo su come affrontare i problemi e di networking, mentre nella sezione 2.3.2 descriveremo i problemi legati alla sicurezza e privacy.

Nella Tabella 2 sono riassunti i temi di ricerca aperti e le cause per le quali sono cruciali per l'internet degli oggetti.

Tabella 1. Problematiche da Affrontare per l'IoT

Problemi Aperti	Descrizione della causa
Standard	Ci sono diverse proposte di standardizzazione ma non sono integrate in un quadro globale
Supporto alla mobilità	Ci sono diverse proposte per i singoli oggetti, ma nessuna per il supporto alla mobilità nello scenario IoT, dove la scalabilità e l'adattabilità delle tecnologie eterogenee rappresentano un problema cruciale
Naming	Object Name Server (ONS) sono necessari per mappare un riferimento a una descrizione di un oggetto specifico e al relativo identificatore, e viceversa
Protocolli di Trasporto	I protocolli di trasporto esistenti falliscono negli scenari dell'internet degli oggetti, a partire dalla configurazione della connessione fino ai meccanismi di controllo della congestione
Caratterizzazione del traffico e supporto QoS	L'IoT genererà traffico dati con modelli che si prevede saranno significativamente diversi da quelli osservati nell'Internet corrente. Di conseguenza, sarà anche necessario definire nuovi requisiti di QoS e schemi di supporto
Autenticazione	L'autenticazione è difficile nell'IoT in quanto richiede infrastrutture di autenticazione appropriate che non saranno disponibili in scenari dell'internet degli oggetti. Inoltre, gli oggetti hanno scarse risorse rispetto dispositivi informatici correnti. Anche l'attacco man-in-the-middle è un problema serio.

Integrità dei dati	È solitamente assicurata proteggendo i dati con password. Tuttavia, la lunghezza delle password supportate da tecnologie dell'IoT, è in molti casi troppo breve perché fornisca alti livelli di protezione.
Privacy	Molte informazioni private su una persona possono essere raccolte senza che questa ne sia a conoscenza. Con le attuali tecniche il controllo sulla diffusione di tali informazioni è impossibile.
Digital Forgetting	Tutte le informazioni raccolte su una persona da parte degli oggetti possono essere conservate a tempo indeterminato. Anche tecniche di data mining possono essere utilizzate per recuperare facilmente ogni informazione anche dopo diversi anni.

2.2.1. Problemi di Networking

L'IoT includerà un numero incredibilmente elevato di nodi, ognuno dei quali produrrà contenuti che devono essere ricavabili da qualsiasi utente autorizzato indipendentemente dalla sua posizione. Attualmente, il protocollo IPv4 identifica ogni nodo attraverso un indirizzo di 4 byte. È ben noto che il numero di indirizzi IPv4 disponibili sta diminuendo rapidamente e presto raggiungerà lo zero. Pertanto, è chiaro che devono essere utilizzate altre politiche di indirizzamento diverse da quelle utilizzate da IPv4.

Per questo motivo, è stato proposto IPv6 per nodi wireless a bassa potenza. Gli indirizzi IPv6 vengono espressi mediante 128 bit e pertanto, è possibile definire 10^{38} indirizzi, che dovrebbero essere sufficienti ad identificare qualsiasi oggetto. Di conseguenza, possiamo pensare di assegnare un indirizzo IPv6 a tutte le cose che fanno parte della rete. Tuttavia, dato che le etichette RFID utilizzano identificatori 64-96 bit, come standardizzato da EPCglobal, sono necessarie soluzioni per consentire l'indirizzamento di tag RFID nelle reti IPv6. Recentemente, è stata studiata[29] l'integrazione dei tag RFID nelle reti IPv6 e sono state proposte metodologie per integrare gli identificatori RFID e gli indirizzi IPv6.

È importante sottolineare che in nessuna delle metodologie proposte viene affrontato il problema della mobilità dell'RFID. Infatti, l'assunzione comune è che ogni RFID può essere raggiunto attraverso un dato gateway tra la rete e il sistema RFID. Ne consegue che, sono necessari meccanismi adeguati per sostenere la mobilità negli scenari dell'internet degli

oggetti. In questo modo, il sistema complessivo sarà costituito da un gran numero di sottosistemi con caratteristiche estremamente diverse. In passato, sono state proposte diverse soluzioni per la gestione della mobilità [31]; tuttavia queste soluzioni, dovrebbero dimostrare la loro validità negli scenari dell'internet degli oggetti, cioè si dovrebbe indagare su quali problemi ci potrebbero essere in termini di scalabilità e adattabilità andandole ad utilizzare in ambienti così eterogenei.

Un altro problema riguarda il modo in cui si ottengono gli indirizzi. Nell'Internet tradizionale qualsiasi indirizzo host è identificato da dei query server chiamati Domain Name Servers (DNS). Obiettivo dei DNS è quello di ricavare l'indirizzo IP di un host da un certo nome in ingresso. Nell'IoT, le comunicazioni sono solite verificarsi tra (o con) gli oggetti, pertanto, deve essere introdotto il concetto di Object Name Service (ONS), che associa un riferimento a una descrizione dell'oggetto specifico e al relativo tag RFID [32]. L'identificatore tag viene mappato in un Internet Uniform Reference Locator (URL), che punta a informazioni rilevanti dell'oggetto. Nell'IoT, gli ONS dovrebbero operare in entrambe le direzioni, cioè, dovrebbe essere in grado di associare la descrizione dell'oggetto specificato a un determinato tag di identificazione RFID, e viceversa. Invertire questa funzione non è facile e richiede un servizio adeguato, che si chiama Object Code Mapping Service (OCMS). Le caratteristiche desiderate per OCMS sono riportate in [33], dove viene suggerito un approccio P2P per migliorare la scalabilità. Tuttavia, si noti che il design e la valutazione delle OCMS in complessi ambienti operativi, come l'IoT, sono ancora questioni aperte.

Inoltre è necessaria una nuova concezione del livello di trasporto per l'IoT. I principali obiettivi del livello di trasporto sono: garantire l'affidabilità end-to-end e eseguire il controllo di congestione end-to-end. Nell'Internet tradizionale, il protocollo utilizzato a livello di trasporto per comunicazioni affidabili è il Transmission Control Protocol (TCP). TCP è insufficiente per l'IoT, a causa delle motivazioni spiegate in [23]. Pertanto, TCP non può essere efficacemente utilizzato per il controllo della trasmissione end-to-end in oggetti. Fino ad oggi, non sono state proposte soluzioni per l'Internet degli oggetti.

Un altro tema di ricerca importante per quanto riguarda gli aspetti di networking è legato alla caratterizzazione del traffico. È noto che le caratteristiche del traffico in reti di sensori dipendono fortemente dallo scenario applicativo. Questo non è un problema se ci si focalizza sul flusso di traffico all'interno della rete wireless del sensore stesso. Le complicazioni sorgono quando, secondo il paradigma degli oggetti, i nodi sensori diventano

parte di Internet. Infatti, in questo scenario, Internet sarà attraversato da una grande quantità di dati generati da reti di sensori che hanno scopi differenti e quindi, caratteristiche del traffico estremamente diverse. Inoltre, poiché la distribuzione su larga scala e i sistemi RFID distribuiti sono ancora all'inizio, le caratteristiche dei relativi flussi di traffico non sono state studiate finora, e pertanto il traffico che attraverserà l'IoT è completamente sconosciuto.

Infine, è necessaria la caratterizzazione e modellazione del traffico unitamente ad un elenco di requisiti di traffico per elaborare soluzioni adeguate per sostenere la qualità del servizio (QoS). Questo problema è ancora del tutto inesplorato in sistemi RFID.

2.2.2. Problemi di Sicurezza e Privacy

Ci sarà resistenza all'introduzione dell'IoT finché non ci sarà la fiducia del pubblico sul fatto che l'utilizzo di queste tecnologie non causerà gravi minacce alla privacy. Tutto il parlare e il lamentarsi a seguito dell'annuncio da parte del rivenditore italiano Benetton sul piano di contrassegnare una linea completa di abbigliamento (circa 15 milioni di dispositivi RFID) è stata la prima, chiara conferma di questa sfiducia nei confronti dell'uso che verrà fatto dei dati raccolti dalle tecnologie dell'internet degli oggetti.

Le preoccupazioni del pubblico sono, infatti, propense a concentrarsi su un certo numero di questioni di sicurezza e privacy.

Sicurezza

L'IoT è estremamente vulnerabile agli attacchi per diversi motivi. Primo, spesso i suoi componenti trascorrono la maggior parte del tempo incustoditi; e quindi, è facile attaccarli fisicamente. In secondo luogo, la maggior parte delle comunicazioni sono senza fili, il che rende estremamente semplice le intercettazioni. Infine, la maggior parte dei componenti dell'internet degli oggetti sono caratterizzati da basse capacità sia in termini di risorse energetiche che informatiche e, quindi, non possono attuare programmi complessi a sostegno della sicurezza.

In particolare, i principali problemi legati alle preoccupazioni sulla sicurezza sono l'autenticazione e l'integrità dei dati. L'autenticazione è difficile poiché richiede di solito infrastrutture di autenticazione appropriate e server che raggiungano il loro obiettivo attraverso lo scambio di messaggi appropriati con altri nodi. Nelle IoT tali approcci non sono fattibili, dato che i tag RFID passivi non possono scambiarsi messaggi con i server di

autenticazione. Lo stesso ragionamento si applica (in un modo meno restrittivo) anche per i nodi sensore.

In questo contesto, sono state proposte diverse soluzioni per le reti di sensori [34]. Tuttavia, le soluzioni esistenti possono essere applicate solo quando i nodi sensore sono considerati come parte di una rete di sensori collegata al resto di Internet tramite alcuni nodi che giocano il ruolo di gateway. Negli scenari dell'Internet degli oggetti, invece, i nodi sensore devono essere visti come nodi di Internet, in modo che sia possibile autenticarsi anche da nodi non appartenenti alla stessa rete di sensori.

Negli ultimi anni, sono state proposte alcune soluzioni per sistemi RFID, però, tutte hanno gravi problemi. Ad esempio nessuna delle soluzioni esistenti può evitare l'attacco man-in-the-middle. Si consideri il caso in cui un nodo viene utilizzato per identificare qualcosa o qualcuno e, di conseguenza, consente di accedere ad un determinato servizio o una certa area (si consideri un passaporto elettronico, o alcune chiavi basate su RFID). L'attacco raffigurato in Figura 1 potrebbe essere eseguito con successo.

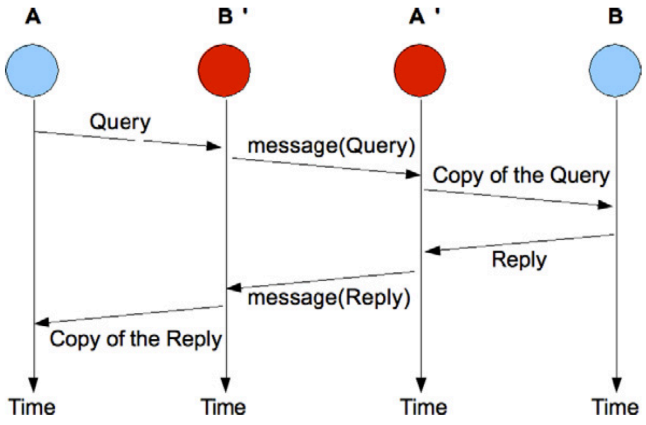

Figura 1. Attacco man-in-the-middle

Si consideri il caso in cui A è il nodo che vuole autenticare altri elementi del sistema attraverso un meccanismo RF e che un attaccante vuole rubare l'identità dell'elemento B (si noti che questo B può essere qualsiasi oggetto dell'IoT). L'attaccante disporrà due "ricetrasmettitori". Il primo vicino ad A, che noi chiameremo B' e il secondo vicino a B, che noi chiameremo A'. L'idea di base è quella di fare credere che B' è B, e far a B che A' è A. A questo scopo, il nodo B' trasmetterà la richiesta di autenticazione ricevuta dal nodo A ad A'. A' trasmetterà tale segnale in modo che B possono riceverlo. Osservare che il segnale trasmesso da A' è una replica esatta del segnale trasmesso da A. Di conseguenza, è impossibile per il

nodo B comprendere che il segnale non è stato trasmesso da A e quindi, risponderà con la sua identificazione. Il nodo A' riceverà la risposta e la trasmette al nodo B', che la ritrasmette al nodo A. Il nodo A non può distinguere che tale risposta non sia stata inviata del nodo B, e quindi, identificherà il B' come B e di conseguenza gli fornirà l'accesso. Questo metodo può essere eseguito indipendentemente dal fatto che il segnale sia criptato o meno.

Soluzioni che assicurano l'integrità dei dati dovrebbero garantire che un avversario non possa modificare i dati della transazione senza che il sistema rilevi il cambiamento. Il problema dell'integrità dei dati è stato ampiamente studiato in tutti i sistemi informatici e di comunicazione tradizionali ed esistono alcuni risultati preliminari per le reti di sensori, ad esempio, [35]. Tuttavia, sorgono nuovi problemi quando si parla di sistemi RFID integrati in Internet e di come i dati spendano la maggior parte del tempo incustoditi. I dati possono essere modificati da avversari mentre sono immagazzinati nel nodo o quando attraversano la rete. Per proteggere i dati contro il primo tipo di attacco, possiamo proteggere la memoria con delle tecnologie di tag, questo tipo di soluzioni sono state proposte anche per le reti di sensori wireless [36]. Si osservi, però che queste soluzioni proposte per sostenere l'integrità dei dati quando si parla di sistemi RFID hanno problemi seri. La lunghezza della password supportato dalla maggior parte delle tecnologie di tag è troppo breve per fornire elevati livelli di protezione. Inoltre, anche se sono supportate password più lunghe, la loro gestione resta ancora un compito impegnativo, soprattutto quando sono coinvolti enti appartenenti a organizzazioni diverse, come nel caso dell'internet degli oggetti.

Infine, si ricorda che tutte le soluzioni proposte per supportare la sicurezza, fanno uso di metodologie di crittografia. Gli algoritmi crittografici più utilizzati spendono grandi quantità di risorse in termini di energia e di larghezza di banda sia alla sorgente sia alla destinazione. Tali soluzioni non possono essere adottate dall'IoT, dato che includerà componenti (come i tag RFID e nodi sensori) che sono fortemente limitati in termini di energia, capacità di comunicazione e di calcolo.

Privacy

Il concetto di privacy è profondamente radicato nella nostra società, è riconosciuto in tutte le legislazioni dei paesi civili e, come abbiamo già detto, le preoccupazioni circa la sua protezione si sono dimostrate essere un ostacolo significativo contro la diffusione delle tecnologie coinvolte nell'IoT. Le preoccupazioni sulla privacy sono infatti ben giustificate. In

realtà, i modi in cui la raccolta dei dati, data mining, e il provisioning sarà realizzato nell'IoT sono completamente diversi da quelli che noi oggi conosciamo e ci sarà un numero incredibile di dati personali da raccogliere. Pertanto, per le persone, sarà impossibile controllare la divulgazione dei propri dati personali.

L'IoT rappresenta davvero un ambiente in cui la privacy delle persone è seriamente minacciata in diversi modi. Inoltre, mentre nell'Internet tradizionale i problemi della privacy sorgono per lo più per gli utenti (individui che giocano un ruolo attivo), negli scenari IoT problemi privacy nascono anche per persone che non utilizzano alcun servizio IoT ma che ne sono coinvolte.

Di conseguenza, la privacy dovrebbe essere garantita del fatto che gli individui possano controllare quali dei loro dati personali vengono raccolti, chi sta raccogliendo tali dati, e quando questo sta accadendo. Inoltre, i dati personali raccolti devono essere utilizzati solo con lo scopo di supportare servizi autorizzati; e, infine, i dati di cui sopra dovrebbero essere memorizzati solo fin quando è strettamente necessario.

Al fine di garantire che i dati personali raccolti siano utilizzati solo per supportare servizi autorizzati, sono state proposte soluzioni che di solito si basano su un sistema chiamato *privacy broker* [37]. Il proxy interagisce con l'utente da un lato e con i servizi dall'altro. Di conseguenza, si garantisce che il provider ottenga solo le informazioni relative all'utente che sono strettamente necessarie. L'utente può impostare le preferenze del proxy. Quando nelle reti di sensori sono inclusi sistemi RFID, allora il proxy opera tra loro ed i servizi. Si noti tuttavia che in questo caso l'individuo non può impostare e controllare le politiche utilizzate dai *privacy broker*. Inoltre, si osserva che tali soluzioni basate su proxy soffrono di problemi di scalabilità.

Infine, sono ancora agli inizi gli studi per quanto riguarda il *digital forgetting* in quanto questo è stato riconosciuto come un problema importante solo recentemente. Infatti, poiché il costo della memoria è in discesa, la quantità di dati memorizzabili aumenterà drammaticamente. Di conseguenza, vi è la necessità di creare soluzioni che eliminano periodicamente informazioni che non sono utili all'obiettivo che deve essere raggiunto.

2.3. Considerazioni sull'Internet degli Oggetti

L'internet degli oggetti è in rapida crescita e sviluppo, ma per riuscire ad imporsi nella vita di tutti i giorni deve risolvere ancora moltissimi problemi, primo fra tutti, la fiducia degli utenti ma, questa, solo il tempo e i continui sviluppi della ricerca potranno riuscire a conquistarla. Grazie alla ricerca si sono fatti passi da gigante in questo nell'IoT ma molte cose sono ancora da migliorare, da sviluppare o meglio ancora da adattare.

Per raggiungere la visione che molti ricercatori condividono, un primo problema da risolvere nello sviluppo dell'IoT è di sicuro quello dell'indirizzamento e della ricerca dei dati attraverso un sistema scalabile in grado di supportare un altissimo numero di partecipanti. Questo proprio perché la visione che si ha dell'internet degli oggetti è un sistema su larghissima scala che permetta ad oggetti sempre più intelligenti di poter comunicare tra loro. Il problema dell'indirizzamento e del reperimento dei dati è un problema ancora aperto all'interno dell'IoT, ci sono comunque alcune proposte ma nessuna di queste è ancora stata realizzata in modo concreto.

Andando a guardare in letteratura possiamo trovare una lista di requisiti funzionali che un servizio di ricerca pensato specificatamente per l'IoT dovrebbe supportare. I requisiti funzionali descrivono le funzionalità e i servizi che un sistema dovrebbe supportare. I requisiti non funzionali invece rappresentano le qualità o i vincoli del sistema, ad esempio le performance, l'installabilità, ecc.

Requisiti funzionali importanti per un servizio di ricerca o Discovery Service (DS) per l'Internet degli oggetti, possono essere raccolti dai documenti di EPCglobal, ad esempio, la specifica ONS [3]. I requisiti raccolti fanno attenzione agli aspetti funzionali, alle prestazioni, alla disponibilità e all'integrità, così come alla riservatezza dei dati del fornitore, ma trascurano la prospettiva del cliente.

Per formulare i requisiti generali in modo oggettivo, useremo il termine OID (Object Identifier), poiché un DS non dovrebbe essere limitato a servire solo codici EPC (Electronic Product Code), ma anche altri sistemi di numerazione, ad esempio basati sul valore letto da uno specifico sensore. Sulla base della letteratura citata, elenchiamo la seguente serie di requisiti funzionali di alto livello per un DS:

- *Supporto Flessibile per OID*: il DS deve essere flessibile per supportare vari schemi per gli OID.

- *Possibilità di pubblicare:* deve essere concessa la possibilità di pubblicare dati aggiuntivi oltre l'OID ai nodi che ne hanno l'autorizzazione, ad esempio l'indirizzo del server che fornisce informazioni aggiuntive per quell'OID.
- *Pubblicatori Multipli*: devono esserci pubblicatori multipli, indipendenti e autorizzati.
- *Ricerca di un OID*: il DS deve rendere disponibile una lista di server che forniscono dati legati a quel preciso OID.
- *Ricerca su un attributo (opzionale)*: il DS deve rendere disponibile una lista di server che forniscono informazioni legate a quel valore di quel preciso attributo.
- *Aggiornamento.*
- *Cancellazione*.

Questa lista di requisiti funzionali è costruita sulla base delle necessità che si hanno negli scenari attuali. In questa tesi andremo ad analizzare le varie proposte di Discovery Service per l'IoT e andremo a vedere quali e quanti requisiti funzionali andranno a soddisfare.

STATO DELL'ARTE

Le attuali tendenze verso l'internet del futuro prevedono la concezione di nuovi servizi sensibili al contesto per migliorare la qualità della vita degli utenti finali. L'Internet delle cose si prevede di contribuire a questa visione ambiziosa proponendo modelli e meccanismi che consentano la creazione di reti di "cose intelligenti" su larga scala. Sono necessari quindi dei meccanismi efficaci per scoprire le risorse e le capacità disponibili per realizzare tale visione. Elencheremo in questo capitolo le architetture esistenti per l'implementazione del Servizio di Discovery elencandone punti deboli e di forza [1].

3.1. Approcci Esistenti

In questa sezione descriveremo le architetture esistenti per l'implementazione di un Servizio di Discovery.

3.1.1. EPCglobal

Il concetto di DS è stato inizialmente formulato da EPCglobal in [4] ma lo standard[1] che dovrebbe definire l'architettura e le interfacce dei DS non è stato ancora pubblicato. La letteratura [5], comunque, ci fornisce una descrizione ad alto livello dell'architettura EPCglobal mostrata in Figura 1.

L'EPCglobal DS è organizzato come un servizio di ricerca che memorizza i riferimenti (URI), legati agli EPC, su un EPCIS[2] corrispondente. Tale servizio consentirà ad un cliente che è alla ricerca di dati relativi ad un EPC specifico di identificare l'EPCIS in grado di fornire questi dati.

I seguenti passaggi descrivono il processo di scambio di dati tra il cliente, il DS, e il relativo EPCIS: Nella fase (0a-b), le aziende notificano al DS gli eventi che sono stati recentemente aggiunti ai loro EPCIS. (1)Il client vuole ottenere i dati su tutti gli eventi che

[1] http://www.epcglobalinc.org/standards/
[2] EPC Information Services

sono legati a un determinato numero EPC e invia una query corrispondente al DS, ricevendo, in risposta, i riferimenti a EPCIS che contengono tutti gli eventi legati al numero EPC interrogato. (2)Il client invia questi riferimenti al Domain Name Systems (DNS), che risolve i riferimenti in indirizzi IP. Utilizzando questi indirizzi IP, (3a-b), il client interroga l'EPCIS in grado di fornire dati relativi al corrispondente EPC.

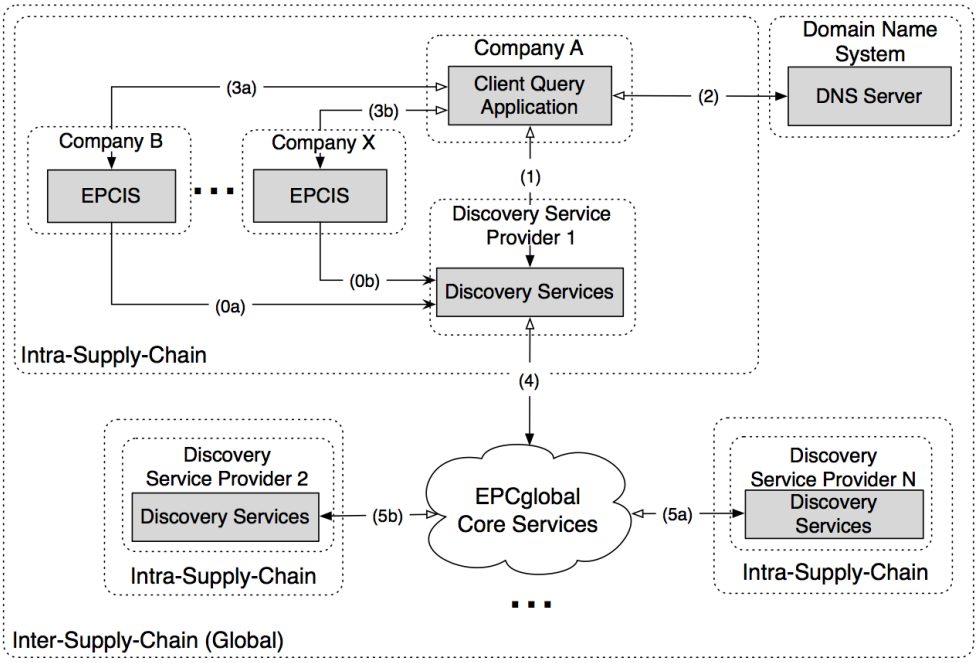

Figura 2. EPCglobal Discovery Service

Nell'architettura descritta, inizialmente, i partecipanti al servizio possono non essere consapevoli dell'EPCIS degli altri, ma le interfacce dei DS (ad esempio, il suo URL) devono essere noti in anticipo. In più è possibile che quando alcuni dati lasciano un determinato DS, le loro informazioni siano ancora presenti in altre istanze di DS.

Secondo EPCglobal, il DS costituirà uno degli EPCglobal Core Services [5] - Servizi che saranno gestiti da EPCglobal o suoi delegati. Il ruolo effettivo di EPCglobal non è ancora stato definito. Una delle possibilità è che EPCglobal fornirà servizi che consentano l'interoperabilità dei DS indipendenti (punti 4, 5a-b) ma ancora i documenti disponibili non rivelano alcun dettaglio.

3.1.2. Progetto BRIDGE

Il progetto BRIDGE, sostenuto dall'UE, si rivolge a un ampio spettro di problemi legati all'implementazione della tecnologia RFID in Europa. Secondo la visione BRIDGE, i DS possono essere implementati come servizi di directory, e distribuiti sia come un singolo server o come una rete di server che forniscono servizi di look-up per EPICIS contenenti dati sugli oggetti identificati da codici EPC.

Quattro differenti architetture vengono considerate in BRIDGE:

- *Directory of Resources:* In questo approccio, le risorse (EPCIS) registrano informazioni sulla disponibilità dei dati corrispondenti ai numeri EPC nel DS. I client interrogano il DS per ottenere i riferimenti agli EPCIS che contengono i dati sui numeri EPC di loro interesse. In seguito essi contattano gli EPCIS per i dati dettagliati.
- *Notification of Resources*: In questo approccio, i client inizialmente si devono iscrivere al DS per ricevere una notifica in merito a determinati numeri EPC. Un EPCIS trasmette al DS informazioni sul set di numeri EPC di cui contiene i dati relativi. Il DS poi invierà le notifiche sulla disponibilità di dati, per i numeri EPC di loro interesse, ai client. Per ottenere dati dettagliati, i client interrogano l'EPCIS corrispondente.
- *Notification of Clients*: EPCIS pubblicano informazioni sulla disponibilità dei dati per determinati numeri EPC alle DS. I clients, invece, usano il DS per notificargli le risorse di loro interesse, in particolare gli comunicano i numeri EPC che vogliono seguire. Se le informazioni su questi numeri EPC sono aggiornate, il DS informa il client, e lui stesso si occupa di contattare gli EPCIS associati per informazioni dettagliate.
- *Query Propagation:* Gli EPCIS pubblicano nel DS informazioni sulla disponibilità dei dati corrispondenti ai numeri EPC. I client utilizzano il DS per l'inoltro di domande direttamente agli EPCIS contenenti informazioni sui numeri EPC di loro interesse. Le risorse rispondono con informazioni dettagliate.

Nell'articolo a cui facciamo riferimento[7], è stato implementato un prototipo di DS, questa implementazione è stata utilizzata come architetture di riferimento per il progetto BRIDGE. Il prototipo è basato sul modello *Directory of Resources* con la componente di

archiviazione implementata come LDAP[1]. L'architettura di alto livello del prototipo è identica alla parte interna dell'architettura EPCglobal DS (Figura 1).

3.1.3 Afilias

Afilias Discovery Service è stato sviluppato da Affilias Inc. e mira a risolvere 5 questioni principali dell'IoT:

a. Identificazione univoca di oggetti, anche se identificati da diverse Autorità per l'identificazione.
b. Retro-compatibilità con sistemi di identificazione già esistenti.
c. Preoccupazione riguardante una singola istituzione di controllo che è esterna a chi pubblica informazioni nel sistema.
d. Praticità, scalabilità e apertura alla concorrenza.
e. Fiducia e sicurezza del sistema.

Per supportare catene di fornitura multiple e indipendenti Afilias DS utilizza un protocollo Web Services chiamato Extensible Supply-chain Discovery Service (ESDS) [8]. ESDS fornisce la descrizione dei concetti, dello schema e dei comandi per l'implementazione degli ESDS Client Query Applications. Alcuni progetti pilota sono stati avviati con aziende partner, ad esempio, nel settore del trasporto aereo. Per i partecipanti al programma pilota ancora in esecuzione, Afilias fornisce l'infrastruttura DS, supporto tecnico e un toolkit per lo sviluppo di un'interfaccia client.

La Figura 2 mostra l'architettura e la procedura di discovery di ESDS. Ogni catena di fornitura partecipante ha a un'istanza di un server ESDS che gestisce la pubblicazione di eventi (0a-b) e le richieste di individuazione del servizio (1) all'interno della supply chain. Questi sono solitamente seguiti da una risoluzione DNS (2) e dall'accesso EPCIS (3a-b). Se un client richiede dati esterni, il server ESDS invia una ricerca globale (4) ad Afilias DS. Questa ricerca viene indirizzato ad altri server ESDS (5). Qualsiasi server ESDS locale risponderà alle ricerche globali in entrata (6a-b), se possiede i dati richiesti.

[1] LDAP (Lightweight Directory Access Protocol) è un protocollo standard per l'interrogazione e la modifica dei servizi di directory come un elenco aziendale di email o una rubrica telefonica o più in generale qualsiasi raggruppamento di informazioni che può essere espresso come record di dati ed organizzato in modo gerarchico.

Per ora, il sistema dipende registri ospitati localmente. Inoltre, ESDS specifica i meccanismi di sicurezza per l'accesso autenticato ai dati locali. Ogni azienda partecipante può definire i diritti di accesso per i propri dati.

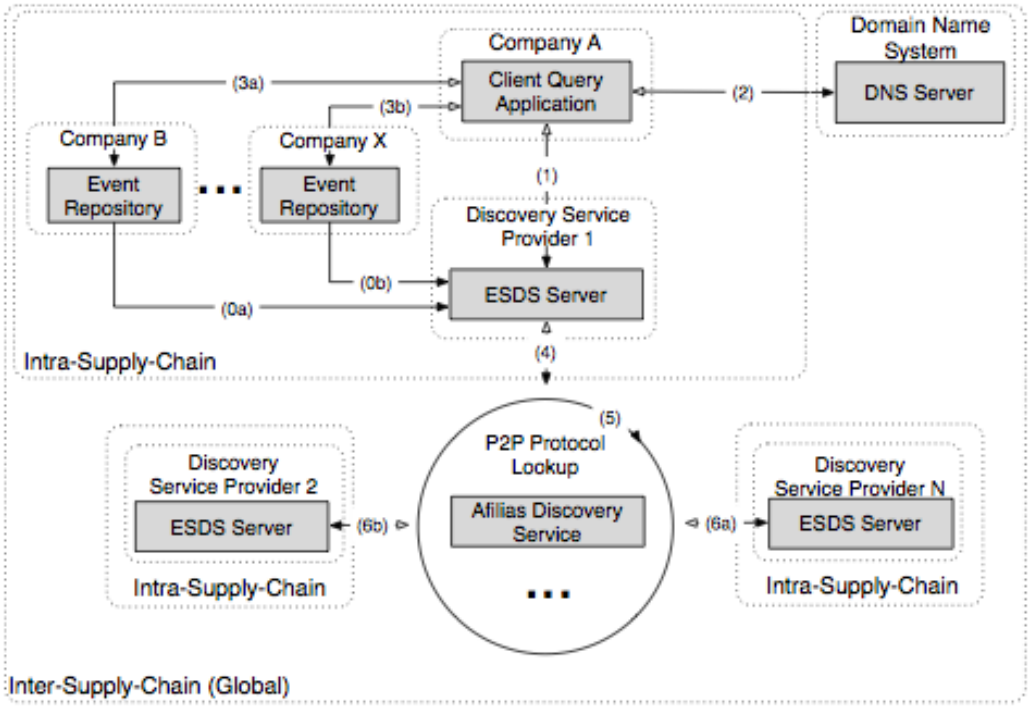

Figura 3. Afilias Discovery Services

3.1.4 ID@URI

Il DS ID@URI è stato sviluppato nel contesto del Dialog project2, iniziato nel 2003 come un progetto open source con l'obiettivo di sviluppare un sistema di tracciamento a livello mondiale[9]. L'ID@URI è stato progettato per utilizzare gli standard di denominazione esistenti che consentono l'interoperabilità e l'integrazione con i sistemi informativi noti. Ha due componenti principali: Client e Object Agent. Il client è utilizzato per la lettura degli identificatori del prodotto e si collega all'Object Agent identificato dall'identificatore dell'oggetto ID@URI. Mentre la parte URI si riferisce al dominio dell'Object Agent appartenente alla società che ha prodotto l'oggetto, l'ID è univoco ed è assegnato a livello locale presso l'azienda produttrice quando viene creato l'oggetto.

Dato che tutti i dati relativi ad un prodotto vengono gestiti da un solo Object Agent e sono memorizzati nel suo dominio, deve essere scoperta una sola fonte di dati. Le funzioni del

processo di scoperta sono le seguenti: Nella fase (0), un'applicazione client che vuole pubblicare i dati su un evento-ID legato all'oggetto, prima si risolve l'URI dell'oggetto all'indirizzo IP del corrispondente Object Agent; poi, (1) l'Object Agent è contattato dal client e (2) il dato evento viene memorizzato nella corrispondente Event Repository locale. Quando un client vuole accedere agli eventi che fanno riferimento a un oggetto identificato da un ID, (3) una Query Application risolve l'URI in un indirizzo IP e (4) si collega all'Object Agent, che (5) utilizza l'ID dell'oggetto per il recupero dei dati dal Repository per poi inviarli indietro al client.

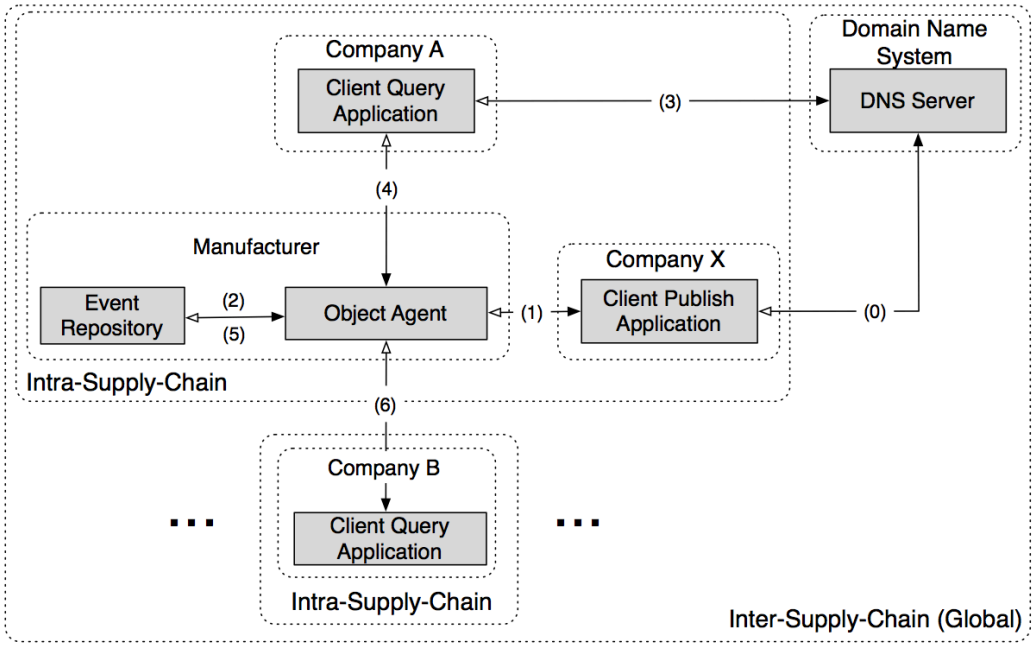

Figura 4. ID@URI Discovery Service

3.1.5 Approccio Peer-2-Peer (DHT-P2P)

I sistemi Peer-to-Peer Systems (P2P) sono alternative, altamente distribuite, alle architetture di rete di servizi classici e possono essere considerate come un cambiamento di paradigma dalla classica architettura Client-Server a un nuovo paradigma con una più o meno equa distribuzione delle responsabilità e del carico tra i partecipanti. Sistemi P2P soprattutto strutturati con tabelle hash distribuite (DHT), offrono un'elevata robustezza ai guasti (ad esempio, non hanno nodi principali speciali come nei DNS), e distribuiscono responsabilità e carico tra i partecipanti in modo sistematico per mezzo di una struttura topologica a strati[10].

Dato che le DHT offrono funzionalità di ricerca di identificatori (ad esempio, un EPC), possono costituire una base altamente scalabile e robusta per DS globali. In [11] è presentata un'implementazione di un DS DHT-based chiamato OIDA, basato su Bambù DHT, testato su circa 350 nodi distribuiti globalmente della piattaforma PlanetLab. In [12], i risultati della simulazione (utilizzando Pastry DHT) hanno mostrato la fattibilità di reti P2P costituite anche da 20.000 nodi.

La Figura 4 mostra come i DS basati su DHT possono essere integrati in un'applicazione EPCglobal. Una volta che un'applicazione EPC presso una società partner ha rilevato un evento che coinvolge un EPC, i dati corrispondenti possono essere pubblicati in un Event Repository (0a-c). Per consentire ad altri partner di scoprire la EPCIS corrispondente, l'Event Repository utilizza un client peer locale per inserire una coppia (chiave = h(EPC), valore = documento EPCIS) nel sistema DHT, dove h è creato con una funzione di hash, ad esempio SHA-1. Un'applicazione client, che vuole scoprire i dati su un determinato EPC, può utilizzare la DHT Lookup Interface tramite il proprio Peer Client Locale per ottenere gli indirizzi IP dei Repositories dell'evento corrispondente (1) e recuperare tutti i dati memorizzati per un EPC (2).

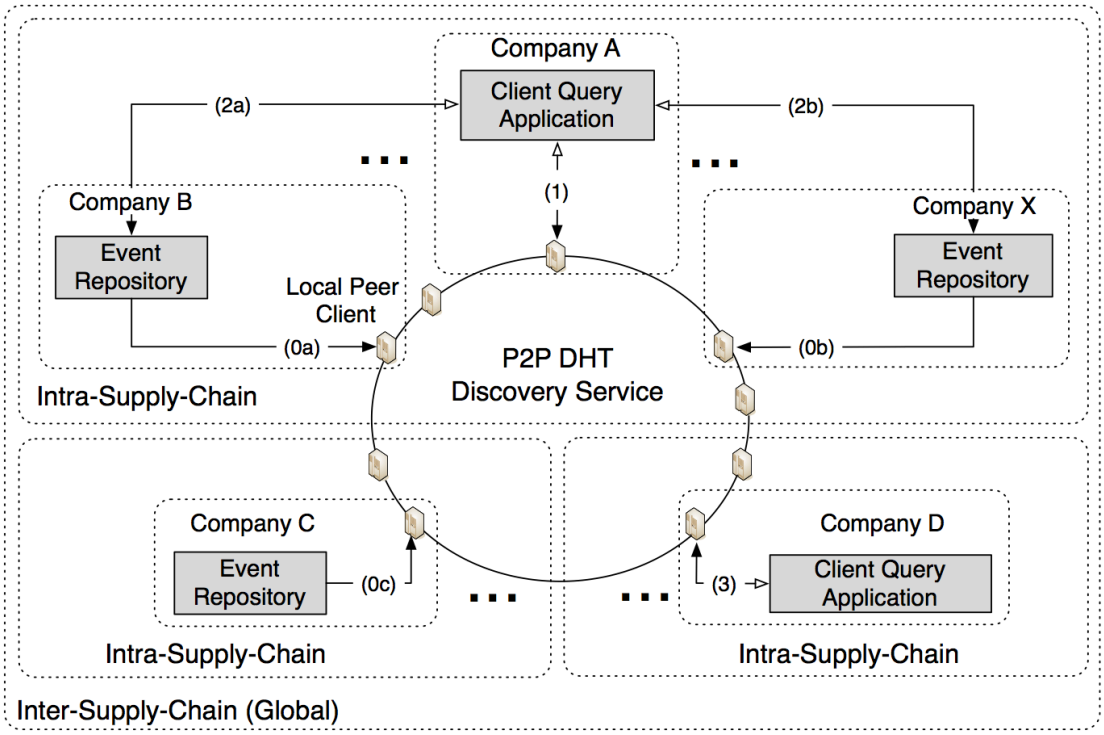

Figura 5. P2P-DHT Discover Service

3.2. Comparazione delle architetture

In questa sezione confronteremo i diversi approcci presentati in precedenza in base alla valutazione della qualità.

La valutazione della qualità si concentra sugli aspetti tecnici degli approcci e si basa sulla norma ISO/IEC 9126 [13] per la valutazione della qualità del software e sulla ricerca e revisione della letteratura rilevante.

3.2.1. Valutazione della qualità

Per valutare la qualità dei diversi approcci DS, abbiamo condotto una ricesca in letteratura [1] per definire le categorie di qualità importanti. Abbiamo identificato tre principali fonti per la definizione della qualità dei DS. In primo luogo, lo standard ISO / IEC 9126 [13], questo definisce un modello di qualità applicabile a qualsiasi tipo di software. Si compone di sei principali categorie di qualità con relative sotto-categorie:

- Funzionalità (idoneità, accuratezza, interoperabilità, conformità, sicurezza),
- Affidabilità (maturità, tolleranza ai guasti, e recuperabilità),
- Usabilità (comprensibilità, apprendibilità e operatività),
- Efficienza (comportamento nel tempo, comportamento delle risorse),
- Manutenibilità (analizzabilità, mutevolezza, stabilità, testabilità),
- Portabilità (adattabilità, installabilità, conformità, e sostituibilità).

In secondo luogo, sulla base di ricerche e revisioni della letteratura abbiamo individuato le seguenti sei categorie di requisiti per DS:

- La proprietà dei dati,
- La sicurezza,
- Il rapporto di affari di design indipendente,
- La crescita organica,
- Scalabilità e qualità del servizio.

In terzo luogo, le seguenti sottocategorie sono state specificate nel progetto BRIDGE:

- La scalabilità orizzontale,
- Collo di bottiglia,
- L'aggiornamento,

- La ricerca dei dati,
- L'organizzazione dei dati,
- Record con vari campi,
- La garanzia e correttezza del risultato.

Dalla letteratura abbiamo trovato una tabella che racchiude queste tre fonti principali, eliminando i requisiti che possono non essere applicati alle architetture DS e aggiungendone alcuni che invece dovrebbero (privacy, controllo degli accessi e tracciabilità)

Tabella 2. Comparazione della Qualità dei diversi Discovery Services

Categorie	Descrizione	EPC-global	Afilias	ID@URI	DHT-P2P
Funzionalità					
Stabilità	Il DS soddisfa i requisiti funzionali	+	+	+	+
Accuratezza	I risultati delle query sono completi e corretti	o	o	o	o
Interoperabilità	L'architettura DS incoraggia la partecipazione, fornendo interfacce e specifiche.	+	+	+	+
Sicurezza	Il DS protegge la riservatezza, l'integrità e la disponibilità dei dati pubblicati e interrogati.	o	o	o	+
Privacy	Il DS assicura e protegge la privacy del cliente.	-	-	-	+
Controllo degli accessi	Chi pubblica informazioni è in grado di definire e controllare i diritti di accesso.	o	o	-	O
Tracciabilità	Chi pubblica è in grado di tenere traccia delle richieste sui suoi dati e sul loro utilizzo.	o	o	-	-
Affidabilità					
Tolleranza ai guasti, Recuperabilità	Anche se parti del sistema complessivo non sono funzionanti, il DS può assicurare una certa	o	o	o	+

	qualità di servizio.				
Efficienza					
Scalabilità Comunicazioni	Il DS non ha colli di bottiglia nello svolgimento dell'aggiornamento di dati multipli e operazioni di ricerca.	-	-	-	+
Scalabilità Risorse	Nuovi partecipanti possono facilmente unirsi al DS. Non vi è alcun limite al numero di partecipanti.	+	+	+	+
Manutenibilità					
Analizzabilità	Fallimenti nel DS possono essere rilevati e riparati, la configurazione può essere aggiustata.	+	+	+	-
Mutevolezza	Cambiamenti nei rapporti commerciali non devono influenzare il modo in cui una società interagisce con il DS.	o	o	o	+
Portabilità					
Installabilità	I DS possono essere facilmente integrati in sistemi informativi esistenti.	+	+	+	+
Conformità	Il DS è conforme con lo standard di denominazione.	+	+	+	+
Espandibilità	È possibile sviluppare estensioni per il DS	+	+	+	+

Valutazione di qualità (+ = buono, o = preoccupante, - = mancante)

Riferendoci ai *requisiti funzionali* visti in precedenza, nessuno dei DS analizzati sembra offrire funzionalità di "Ricerca su un attributo", anche se questa funzionalità nel futuro IOT potrebbe diventare molto importante.

Nessuno degli approcci considerati è in grado di garantire *accuratezza* dei risultati delle query, poiché i dati sono forniti da una terza parte che può omettere o creare alcuni valori, o semplicemente negare l'accesso se le informazioni richieste sono troppo sensibili. Una caratteristica naturale per i DS, e quindi svolta da tutti gli approcci, è il loro obiettivo di

fornire interoperabilità, vale a dire, tutte le descrizioni, le interfacce e le specifiche sono a disposizione del pubblico.

In termini di *sicurezza*, ad eccezione di DHT-P2P, per tutti gli approcci si stanno discutendo dettagliati provvedimenti volti a tutelare la riservatezza e l'integrità dei dati memorizzati e trasmessi per garantire la disponibilità dei relativi servizi. Lo stesso vale per la *privacy* di un client, anche in questo caso si stanno discutendo approcci che possano impedire al fornitore del DS di identificare il cliente (ad esempio, l'indirizzo IP). In contrasto a ciò, l'architettura P2P-DHT nasconde naturalmente il client, poiché una query raggiunge la destinazione in più di un hop all'interno della rete P2P. Tutti e quattro gli approcci assumono che i partecipanti siano in grado di definire i diritti di *controllo degli accessi* per i propri dati. Dal momento che i DS non sono (fisicamente) ospitati dalle parti interessate, il *loro controllo sulle voci pubblicate* nel DS è limitato. Tuttavia, in EPCglobal, Afilias, e soprattutto anche per approcci DHT-P2P si presume che gli eventi effettivi siano memorizzati in repository controllati dalle parti interessate. Solo nell'approccio ID@URI, i dati dell'evento devono essere trasferiti ad un terzo, il costruttore.

Per quanto riguarda la *tracciabilità*, approcci di EPCglobal e Afilias presuppongono che i soggetti interessati siano in grado di monitorare l'accesso ai loro dati, compreso l'utilizzo. Nel caso P2P-DHT la tracciabilità diventa un problema molto grande, dato che i DS sono rappresentati da migliaia di peers molti dei quali non sono a conoscenza dell'informazione. Per quanto riguarda *affidabilità* ed *efficienza*, gli approcci che assumono che le istanze del DS vengano distribuite centralmente, inevitabilmente inducono un collo di bottiglia. Misure come *replica* e il *bilanciamento del carico* possono migliorare la *tolleranza ai guasti*, il *recupero* e la *scalabilità*. Tuttavia, è solo l'approccio basato su DHT-P2P che elimina completamente il collo di bottiglia e introduce meccanismi che garantiscono al DS di rimanere in funzione, anche se (importanti) parti dell'infrastruttura di base non sono più disponibili. Nessuno degli approcci impone restrizioni esplicite riguardo il numero massimo di partecipanti, ciò potrebbe minare la *scalabilità*.

Le categorie *analizzabilità* e *mutevolezza* descrivono quanto è semplice mantenere il sistema. La natura centralizzata di EPC-global, di Afilias e di ID@URI, in linea di principio, permette di localizzare e identificare gli errori e i fallimenti. D'altra parte, l'architettura di DHT-P2P, con la sua struttura molto più flessibile e distribuita, può rendere molto difficile la localizzazione dei guasti. Allo stesso tempo, tale flessibilità conferisce a questo modello un

vantaggio quando le relazioni commerciali delle parti interessate cambiano e questi cambiamenti vanno ad influenzare i flussi di dati tra di loro. Con P2P, le modifiche sono naturalmente propagate attraverso l'intero overlay. Gli altri approcci possono richiedere aggiornamenti di parecchi DS o, in certi casi, anche una distribuzione di nuove istanze di DS. Per quanto riguarda l'*installabilità*, la facilità d'implementazione è uno dei presupposti di tutti gli approcci. Lo stesso si può dire di *conformità* ed *espandibilità*.

3.2.2. Vantaggi e Svantaggi

Riassumendo le caratteristiche viste finora dei vari approcci di DS, si può affermare che l'approccio EPCglobal è ancora in via di sviluppo. Componenti del metodo relativamente maturo Afilias saranno probabilmente integrati nello standard EPCglobal, mentre il prototipo del progetto BRIDGE è molto simile all'approccio EPCglobal. Questi tre condividono gli stessi vantaggi e svantaggi. Due approcci molto contrastanti sono ID@URI e DHT-P2P. Entrambi hanno diversi vantaggi e svantaggi: ID@URI è facile da implementare, ma dipende dal produttore che, da solo, ha la responsabilità di fornire le informazioni sull'oggetto nella catena di fornitura. DHT-P2P offre invece una soluzione molto scalabile e flessibile, offre sicurezza sia nella trasmissione dei dati sia nel sistema stesso in caso di nodi malfunzionanti infine offre privacy al client. Uno svantaggio di questa soluzione è il dover migliorare il supporto alla tracciabilità dei dati.

La soluzione DHT-P2P si è comunque rilevata vincente rispetto alle altre analizzate ed è proprio da questa constatazione iniziale che parte il nostro lavoro. L'idea è di creare una rete DHT-P2P che implementa la possibilità di effettuare query multi-attributo e range query, dato che nessuno degli approcci proposti in questo capitolo lo supporta.

3.3. Architetture DHT-P2P Esistenti

In questo capitolo presenteremo alcune tecnologie DHT-P2P esistenti in letteratura che supportano richieste multi-attributo e le range query, analizzandone vantaggi e svantaggi. Ne sceglieremo poi una e la andremo a progettare ed implementare per l'utilizzo nell'internet degli oggetti.

3.3.1. MAAN

MAAN (Multi-Attribute Addressable Network)[14] è una rete P2P utilizzata nei Grid Information Services[1]. MAAN implementa le range query mappando i valori degli attributi in una rete Chord[15] attraverso una locality preserving hash function uniformemente distribuita, cioè una funzione hash che mantiene la "posizione" del dato. Per comprendere meglio una locality preserving hash function possiamo far riferimento alla seguente definizione:

Una funzione H è una locality preserving hash function se soddisfa la seguente proprietà: $H(v_i) < H(v_j)$ *se e solo se* $v_i < v_j$*, e se un intervallo* $[v_i, v_j]$ *è suddiviso in* $[v_i, v_k]$ *e* $[v_k, v_j]$*, l'intervallo* $[H(v_i), H(v_j)]$ *corrispondente deve essere diviso in* $[H(v_i), H(v_k)]$ $[H(v_k), H(v_j)]$.

In Chord sia ai nodi sia ai dati viene assegnato un identificatore di m-bit. Un dato che ha chiave *k* viene assegnato al nodo che ha ID maggiore e più vicino a *k*. Chord necessita che ogni nodo si crei una propria "finger table", all'interno di questa tabella ci saranno gli indirizzi di alcuni nodi della DHT. Ad esempio, un nodo *n* avrà all'interno della sua "finger table" in posizione *i-esima* l'indirizzo del nodo che conterrà il dato per la chiave $\left((n + 2^{i-1}) mod\ 2^m\right)$.

[1] I sistemi Grid sono un'infrastruttura di calcolo distribuito, utilizzati per l'elaborazione di grandi quantità di dati, mediante l'uso di una vasta quantità di risorse. In particolare, tali sistemi permettono la condivisione coordinata di risorse all'interno di un'organizzazione virtuale.

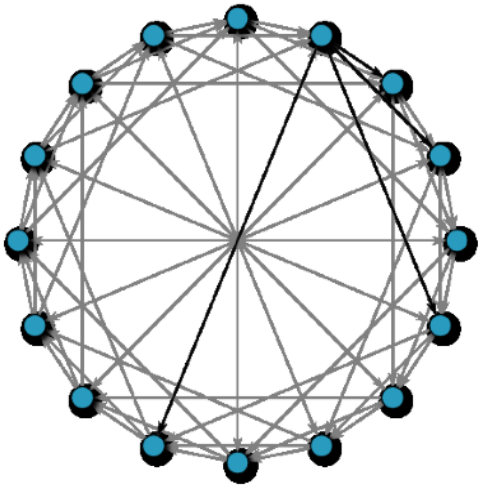

Figura 6. Rete Chord. La "finger table" per uno dei nodi è evidenziata.

In MAAN, le risorse sono registrate come una serie di coppie attributo-valore e possono essere effettuate range query multi-attributo. MAAN utilizza la funzione hash SHA1 per assegnare un identificatore di m-bit per ogni nodo e per assegnare un valore ad un attributo di tipo stringa. Tuttavia, per gli attributi con valori numerici MAAN utilizza una funzione hash che preserva la località per assegnare ad ogni valore di un attributo un identificatore nello spazio m-bit.

Supponiamo di avere un attributo con valori numerici nell'intervallo $[v_{min}, v_{max}]$. Possiamo utilizzare una semplice funzione hash che preserva la località, nell'articolo è riportata la seguente $H(v) = (v - v_{min}) \times \frac{(2^m - 1)}{(v_{max} - v_{min})}$, dove $v \in [v_{min}, v_{max}]$ ma qualsiasi funzione hash locality preserving potrebbe andare bene. Quindi, ogni valore di attributo v, ha il corrispondente identificatore $H(v) \in [0, 2^m - 1]$ nello spazio degli identificatori. MAAN utilizza poi lo stesso metodo di Chord ed assegna valore dell'attributo v al nodo successore di $H(v)$.

Supponiamo che il nodo *n* vuole ricercare valori *v* compresi tra *l* e *u* (*l* < *v* < *u)* dove *l* e *u* sono, rispettivamente, il limite inferiore e limite superiore. Il nodo *n* compone una richiesta di ricerca e utilizza l'algoritmo di routing di Chord per indirizzare la richiesta al nodo n_l, il successore di $H(l)$. La richiesta di ricerca sarà composta come segue *RICERCA(k,R,X)* dove *k* è la chiave utilizzata per l'instradamento in Chord, R è il range desiderato [*l,u*] e X è il contenitore dei risultati, quindi inizialmente vuoto. Quando n_l riceve la richiesta di ricerca,

aggiunge a X i dati che si trovano all'interno del range, controlla poi se lui è stesso è anche il successore di *u,* in tal caso ritorna i dati ad *n,* altrimenti invia la richiesta al nodo successivo.

Invece di supportare la ricerca di un solo attributo, MAAN estende l'algoritmo di routing, presentato sopra, supportando le ricerche multi-attributo. In questa configurazione multi-attributo, si assume che ogni risorsa abbia M attributi a_1, a_2,. . . , a_m e ogni attributo abbia i suoi valori, avremo quindi delle coppie $\langle a_i, v_i \rangle$, dove $1 \leq i \leq M$. Per ogni attributo a_i i valori dei suoi attributi v_i saranno nell'intervallo $[v_{i\ min}, v_{i\ max}]$ e saranno conformi ad una distribuzione rappresentata da una funzione di distribuzione $D_i(v)$. Viene generata quindi la seguente funzione hash locality preserving $H_i(v) = D_i(v) \times (2^m - 1)$ per ogni attributo a_i. Con queste funzioni hash possiamo mappare tutti i valori degli attributi nello stesso spazio Chord di m-bit. Ogni risorsa registrerà le informazioni , coppie ⟨attributo, valore⟩, al nodo n_i successore di $H(v_i)$ per ogni valore di attributo v_i, dove $1 \leq i \leq M$. La richiesta di registrazione per il valore dell'attributo v_i viene instradata al suo nodo successore utilizzando l'algoritmo di instradamento di Chord. Quando un nodo riceve una richiesta di registrazione di una risorsa x con il valore di attributo $a_i = v_{i\,x}$ e informazioni sulla risorsa r_x, aggiunge la coppia $\langle v_{i\,x}, r_x \rangle$ nella lista corrispondente all'attributo a_i.

Quando un nodo esegue una ricerca multi-attributo prepara le sotto-query singolarmente e alla fine interseca i risultati. Per una query multi-attributo su M attributi MAAN impiega $O(\sum_{i=1}^{M}(\log N + K_i))$ salti per risolvere la query dove K_i è il numero di nodi che partecipano alla risoluzione.

3.3.2. Mercury

Mercury[16] supporta query multi-attributo e range query isolando i nodi del sistema in gruppi chiamati "attribute hubs". Questa partizione è di tipo logico, vale a dire che ogni nodi fisico può far parte di più nodi logici. In più Mercury esegue un esplicito bilanciamento del carico muovendo i nodi e cambiando le loro responsabilità secondo i carichi. Questo consente la combinazione di un buon bilanciamento del carico con il supporto alle range query. Tuttavia, un effetto collaterale è che non è più garantita l'uniformità della distribuzione del carico. Un problema riconosciuto di Mercury è la bassa scalabilità in caso di molti attributi, questo è dovuto dalla distribuzione non uniforme dei dati, dato che non viene usata una

funzione Hash, devono essere quindi implementati meccanismi alternativi per mantenere basso il numero di hop.

Gli hub possono essere pensati come una dimensione ortogonale su uno spazio multidimensionale rappresentante tutti gli attributi. Il salto sul primo nodo determinerà quale dimensione attraverserà il percorso. Il resto del routing è unidimensionale e si basa sui valori del singolo attributo del dato.

Ogni nodo in Mercury deve costruire e mantenere la seguente serie di collegamenti:

- Nodo successore e predecessore all'interno dell'attribute hub.
- k collegamenti a lunga distanza per un efficiente instradamento intra-hub
- Vari collegamenti ad altri hub. Il collegamento tra hub implica che ogni nodo conosce almeno un rappresentante per ogni hub del sistema

I collegamenti a lunga distanza vengono creati utilizzando un preciso metodo, un nodo *n* che gestisce un range di valori [l,r) costruisce i suoi collegamenti a lunga distanza in questo modo: calcola da un intervallo unitario I = [0,1] un valore $x \in$ I utilizzando la funzione armonica di distribuzione di probabilità: $p_n(x) = \frac{1}{(n \log x)}$ $se\ x \in \left[\frac{1}{n}, 1\right]$. Il nodo contatterà poi un nodo *n'* (utilizzando il protocollo di routing) che gestisce il valore $r + (M_a - m_a)x$ nel suo stesso hub, una volta trovato lo inserirà tra i suoi collegamenti a lunga distanza. Ogni nodo avrà una routing table di dimensione k + 2 (il nodo successivo e il precedente) dove k può essere diverso per ogni nodo.

L'algoritmo di routing è semplice: prendiamo un nodo n_i incaricato del range $[l_i, r_i)$ e indichiamo con *d* la distanza in ordine orario tra due nodi. Quando un nodo deve ricercare un valore v, questo sceglie il vicino n_i che minimizza la distanza $d(l_i, v)$. Prendiamo m_a e M_a rispettivamente valore minimo e massimo per l'attributo a.

$$d(a,b) = \begin{cases} b - a & if\ a \leq b \\ (M_a - m_a) + (b - a) & if\ a > b \end{cases}$$

Partendo dal presupposto che gli intervalli per nodo sono uniformi, si dimostra che il numero di salti del routing per ogni valore all'interno di un hub sarà $O\left(\frac{1}{k} log^2\ n\right)$. Questa garanzia si basa sulle analisi di Kleinberg sulle small-world networks[17].

3.3.3. LORM

LORM-DHT (Low-Overhead Range-query Multi-attribute)[19] a differenza di altri approcci basati su DHT, si basa su una rete DHT a grafo composto in grado di distribuire informazioni sulle risorse tra i nodi sfruttando la struttura del grafo composto. Inoltre, è in grado di gestire un alto numero di risorse e sistemi con caratteristiche dinamiche (alto numero di ingressi o cancellazione di nodi). I risultati sperimentali dimostrano l'efficienza di Lorm in confronto con altri approcci. Lorm riduce drasticamente la manutenzione e l'overhead sulla scoperta delle risorse.

LORM è costruito basandosi su una DHT chiamata Cycloid[18]. In Lorm ogni nodo è responsabile per le informazioni sulle risorse di un attributo specifico all'interno di un intervallo di valori, approfittando della struttura grafico composto di Cycloid che collega i cluster da un ciclo.

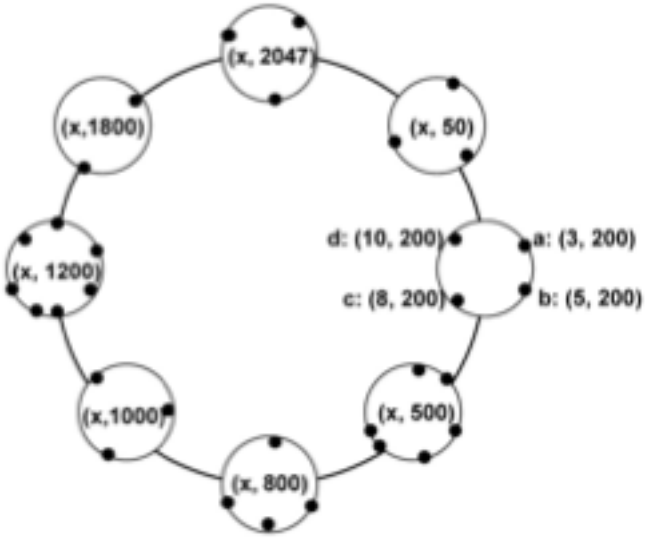

Figura 7. Struttura di Cycloid

In Cycloid ogni oggetto o nodo ha un ID che è l'hash del nome dell'oggetto o dell'indirizzo IP del nodo. La rete fornisce due operazioni principali: inserimento (chiave, oggetto) e ricerca (chiave) che, rispettivamente, memorizzano un oggetto in un nodo o recuperano l'oggetto. Il messaggio di operazione è trasmesso da nodo a nodo in base all'algoritmo di instradamento, fino a raggiungere il nodo che possiede l'oggetto. Ogni nodo mantiene una tabella di routing che registra i suoi vicini all'interno del cluster. L'ID di ogni nodo o oggetto in Cycloid è rappresentato da due indici (k, a_{d-1}, a_{d-2}… a_0), dove *k* è chiamato indice ciclico ed *a* indice cubico. L'indice cubico è un intero compreso tra [0, d-1] mentre

l'indice cubico è un numero binario tra [0, 2^d-1] che sta ad indicare il cluster di appartenenza, dove d è la dimensione della DHT. Come mostrato nella Figura 6 i nodi con lo stesso indice cubico sono ordinati secondo k nel loro cluster. Il nodo con il k più alto è detto nodo primario del cluster. Tutti i cluster sono connessi secondo il loro indice cubico. Ogni nodo mantiene una tabella di routing in cui sono memorizzati gli indirizzi dei nodi vicini: il nodo successore, il predecessore, i nodi primari del cluster predecessore e del successore.

L'idea di base di LORM è di rendere ogni cluster responsabile per le informazioni di un solo attributo e distribuire i dati tra i nodi basandosi sul valore o sulla sua descrizione. LORM assegna ad ogni risorsa una Cycloid ID (k, a_{d-1}, a_{d-2}… a_0). L'indice cubico a_{d-1}, a_{d-2}… a_0 rappresenta l'attributo *a* e l'indice ciclico rappresenta il valore dell'attributo. Quindi l'indice cubico servirà a differenziare i cluster mentre l'indice ciclico stabilirà la posizione del nodo all'interno del cluster. Le informazioni verranno distribuite in un rete Cycloid come da Figura 7.

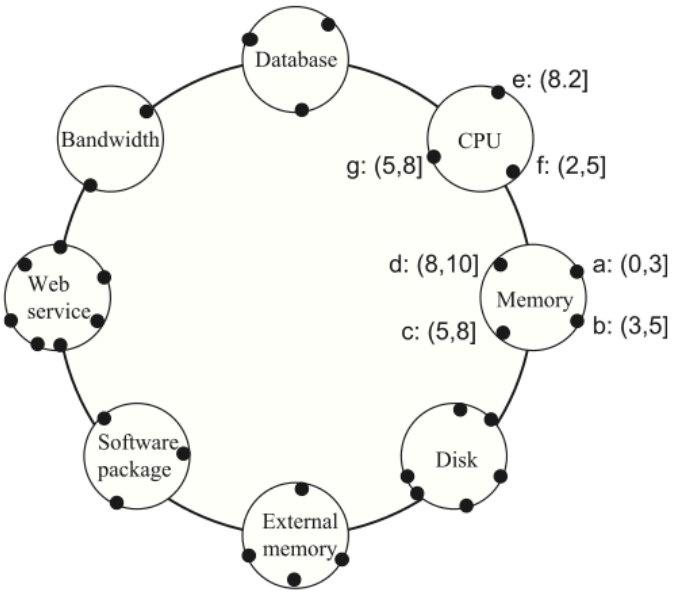

Figura 8. Struttra di LORM

Per gli indici cubici viene utilizzata una funziona hash tipo SHA1 per garantire uniformità della rete ed evitare le collisioni, è utilizzata anche per indici ciclici che si basano su stringhe. In caso di indici ciclici numerici si utilizza un funzione hash di tipo locality preserving questo per rendere possibile operazioni di range query. Si assume che ogni risorsa abbia un valore massimo π_{max} ed uno minimo π_{min}. La funzione hash locality preserving sarà:

$$H = (\pi - \pi_{min}) \times \frac{(d-1)}{(\pi_{max} - \pi_{min})}$$

dove π è il valore per l'attributo *a* all'interno del range[π_{min}, π_{max}] e d è la dimensione massima dell'ID, ma, come per MAAN, si può scegliere una qualsiasi funzione hash che preservi la località.

Come in tutte le DHT, le due operazioni più importanti sono l'inserimento e la ricerca. Per quanto riguarda la prima una risorsa può inserire un dato attraverso l'interfaccia *Insert(rescID, rescInfo)* dove *rescInfo = < a, dπ_a, ip_addr(i) >* racchiude tutte le informazioni sulla risorsa (a = attributo, *dπ_a* = valore inserito, *ip_addr(i)* = indirizzo IP di chi fornisce il dato), mentre il *rescID* è praticamente il Cycloid ID. LORM per inserire il dato quindi cercherà prima di raggiungere un nodo del cluster che gestisce l'attributo del dato inserito, poi inserirà l'informazione nella posizione corretta data dall'hash function locality preserving. Per l'operazione di ricerca, il nodo richiedente invia *Lookup(rescID, rescInfo),* dove il campo ip_addr di rescInfo rimane vuoto. Questo verrà riempito con operazioni di join sui nodi che possiedono i dati di interesse. Per le range query, la richiesta verrà inviata da nodo a nodo fino a quando non cade sull'ultimo nodo che gestisce l'estremo superiore del range.

3.3.4. Comparazione tra MAAN, Mercury e LORM

In questo capitolo andremo a comparare le diverse architetture DHT presentate in precedenza grazie all'ausilio della letteratura[19]. Analizzeremo le loro performance in termini di costi di mantenimento della struttura del sistema, del mantenimento delle informazioni sulle risorse ed efficienza nella ricerca delle informazioni. La letteratura che ci ha permesso di valutare le differenze tra i vari sistemi prende in esame esperimenti con 2048 nodi, 200 attributi e 500 valori per ogni attributo.

3.3.4.1. Costi di mantenimento del sistema

Nelle DHT ogni nodo deve mantenere un numero di vicini all'interno della sua routing table, pertanto, il mantenimento delle routing table costituisce una buona parte dei costi di mantenimento.

L'articolo [19] mostra il numero di collegamenti mantenuti nelle routing table di LORM e Mercury, possiamo vedere come il numero di link mantenuti da Mercury è drasticamente superiore a quelli di LORM.

Per quanto riguarda il mantenimento delle risorse, sarebbe bene distribuire le informazioni tra i nodi, il più uniformemente possibile. Nell'articolo[19] si vede bene che LORM riesce a ridurre la dimensione media delle directory che mantengono i dati rispetto a

MAAN, quest'ultimo duplica le informazioni da mantenere ed ha quindi bisogno del doppio delle operazioni di mantenimento. Si ha quindi che sia LORM che Mercury hanno un migliore bilanciamento della distribuzione delle informazioni rispetto a MAAN.

3.3.4.2. Efficienza nella ricerca delle informazioni

Definiamo *nodi contattati* tutti i nodi che sono coinvolti in un'operazione di ricerca delle risorse.

Dall'articolo [19], preso come riferimento, possiamo osservare che per risolvere una query su un qualsiasi attributo *m* in una rete di *n* nodi, LORM e Mercury riescono a ridurre il numero totale di *nodi contattati* rispetto a MAAN di, rispettivamente, un fattore $\frac{\log(n)}{d}$ dove *d* è il numero di cluster di LORM e un fattore due. Mentre, per risolvere una range query LORM, in media, può ridurre il numero di nodi visitati a $\frac{m(n-d)}{4}$ rispetto a MAAN e Mercury.

Per quanto riguarda le range query multi-attributo LORM riesce a ridurre i nodi contattati a $m \times n$ (caso peggiore) dove m è il numero degli attributi e n è il numero dei nodi all'interno del cluster, nettamente meglio di Mercury e MAAN, rispettivamente $m(\log n + n)$ e $m(2 \log n + n)$.

3.3.5. Sommario

In sintesi LORM ha prestazioni superiori per quanto riguarda i metodi di ricerca multi-attributo analizzati:

- Genera più bassi costi di manutenzione del sistema.
- Produce una migliore distribuzione bilanciata del carico che favorisce il mantenimento delle informazioni e la ricerca dei dati.
- È più efficiente per quanto riguarda la ricerca dei dati in termini di salti tra i vari nodi.
- È più efficiente nella velocità di risposta.
- È più elastico ai cambiamenti.

PROGETTAZIONE E IMPLEMENTAZIONE

Come visto nel capitolo precedente, LORM risulta il vincitore per quanto riguarda bilanciamento delle informazioni e costi di manutenzione del sistema. Abbiamo quindi deciso di rifarci alla sua struttura per implementare la nostra DHT-P2P orientata all'internet delle cose. Questo capitolo tratterà le scelte di progettazione per il nostro sistema e le scelte implementative fatte prima e durante la scrittura del codice.

4.1. Progettazione del Sistema

4.1.1. Caratterizzazione del sistema

Basandoci su LORM e quindi avendo uno scheletro della DHT derivante da Cycloid[18] siamo andati a studiare più a fondo il meccanismo utilizzato da Cycloid per la gestione dei singoli cluster. Cycloid utilizza una qualsiasi DHT mono-attributo per la gestione dei cluster. Dopo un confronto tra due delle più utilizzate DHT mono-attributo, Pastry[20] e Chord[21], abbiamo deciso di utilizzare la prima, poiché ha la possibilità di scorrere i nodi sia in senso orario sia in senso antiorario e ha all'interno della routing table una lista, il leaf set, in cui sono presenti gli indirizzi dei nodi numericamente più vicini, al contrario di Chord che può girare solo in senso orario e che non possiede una lista di nodi vicini. Questa particolarità ci sarà utile nel diminuire il tempo di ricerca dei dati per quanto riguarda l'esecuzione di una range query.

Come linguaggio di sviluppo abbiamo scelto di utilizzare Java per la sua alta portabilità su diverse piattaforme, la sua semplicità e la grande disponibilità di librerie, anche perché data la sua grande diffusione ci aspettavamo di riuscire a trovare più facilmente un'implementazione Java di Pastry su cui basarci per andare a sviluppare il nostro sistema. La scelta è ricaduta su FreePastry[1], sviluppata dalla "Rice University di Houston, USA" e dal "Max Plank Institute for Software Systems di Saarbrücken, Germania". Questa si è rivelata un'ottima scelta dato che il sito presenta degli ottimi tutorial per la creazione e l'utilizzo della DHT e un'ottima documentazione che spiega nel dettaglio il comportamento delle singole classi, cosa

[1] http://www.freepastry.org

che ci è stata molto utile quando siamo andati ad implementare lo scambio di messaggi tra i vari nodi.

Un'altra cosa da scegliere è stata la funzione hash locality preserving da utilizzare all'interno dei cluster per mantenere la località dei dati e rendere possibili le range query. Per questa scelta ci siamo attenuti all'articolo di LORM[19] che riporta la seguente funzione:

$$H = (\pi - \pi_{min}) \times \frac{(d-1)}{(\pi_{max} - \pi_{min})}$$

dove π è il valore per l'attributo all'interno del range$[\pi_{min}, \pi_{max}]$ e d è il valore massimo dell'ID.

Per il livello superiore, cioè per il livello in cui sono presenti i master degli attributi, che chiameremo **DHT Globale** e sarà gestita anch'essa con Pastry, abbiamo scelto di cifrare le stringhe degli attributi, ad esempio "Temperature" con la funzione hash SHA1. Questa verrà utilizzata anche per valori di attributi che non necessitano di preservare la località.

Abbiamo deciso che ogni partecipante alla DHT può ospitare più nodi e quindi partecipare a più cluster, quindi dobbiamo necessariamente differenziare il nodo fisico da quello virtuale.

- **Nodo Fisico**: è il partecipante alla DHT, ogni nodo fisico può racchiudere in se più nodi virtuali, specificati al momento dell'ingresso nella DHT e dipendenti dalle sue caratteristiche computazionali.
- **Nodo Virtuale**: è il nodo che partecipa ad un solo cluster, ogni nodo virtuale ha una sua routing table, quindi un suo preciso indirizzo, e memorizza dati differenti dagli altri.

Per fare un esempio, un nodo fisico, al momento di accedere alla DHT, tramite un nodo che già ne fa parte, specifica che gestirà *temperatura* e *umidità*, saranno quindi creati due nodi virtuali, (in realtà tre ma vedremo successivamente perché), che si posizioneranno all'interno dei rispettivi cluster, in caso questi esistano già.

Per facilitare le operazioni di un nodo fisico si è scelto che ognuno di essi avrà un nodo virtuale connesso alla DHT Globale da cui partiranno non solo le richieste di inserimento e di ricerca di un dato ma anche le richieste di connessione ad uno specifico cluster in fase di creazione dei nodi virtuali. Questo è appunto il terzo nodo virtuale di cui si è parlato sopra. Questo nodo è detto **Nodo Gestore** ed è il primo ad essere creato perché appunto deve gestire tutte le varie richieste di creazione dei nodi virtuali nei cluster.

Ma come facciamo a verificare l'esistenza di un cluster? Semplicemente ogni cluster/attributo ha un suo **Nodo Master** che è attaccato alla DHT Globale e che sarà il referente per quell'attributo. Il nodo master di un attributo è creato nel momento in cui un nodo che decide di entrare nella DHT dichiara di voler gestire un attributo non ancora presente in essa, in questo caso il nodo gestore creerà un altro nodo virtuale che verrà inserito nella DHT Globale in posizione H("attributo") dove H è la funzione hash SHA1.

Vediamo in Figura 8 i diversi tipi di **Nodo Fisico** sia in presenza di Nodi Master che senza. Come possiamo vedere i nodi sono racchiusi nel **Manager dei Nodi**, questo manager si occupa, oltre che della creazione di tutti i nodi, di effettuare le richieste di lookup e di inserimento inviandole ai nodi che sono in grado di eseguirle, come vedremo nel capitolo successivo.

Le richieste, lookup e inserimento dei dati, vengono effettuate tramite un client che si connette al nodo ed effettua le operazioni. Il nodo fisico per permettere ciò implementa un server socket sulla porta 9000 che gestisce connessioni multiple e quindi più richieste in parallelo.

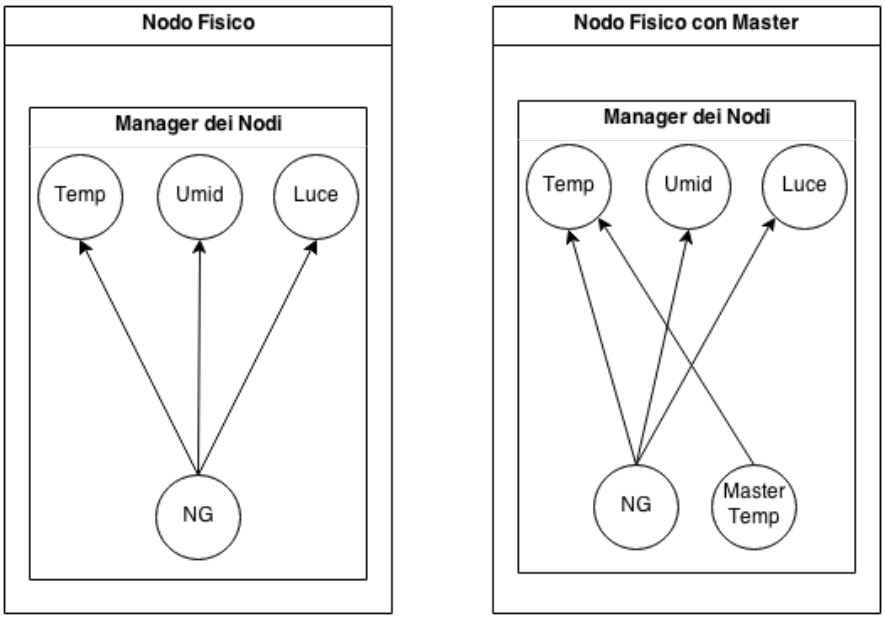

Figura 9. Esempi di Nodo Fisico

La Figura 9 mostra una possibile configurazione di una rete con tre attributi e tre nodi. Possiamo osservare che ogni Nodo Fisico ha un Nodo Gestore (**G**) connesso alla DHT Globale, come introdotto prima, da questo partiranno sia le richieste di entrata nel cluster dei nodi virtuali sia le operazioni di inserimento o lookup dei dati. Vediamo che i Nodi Master (**M**) sono stati disegnati doppi, questo perché sono realmente due nodi virtuali differenti, uno connesso

alla DHT Globale e l'altro connesso al cluster, grazie a questo una richiesta di lookup proveniente dalla DHT Globale passerà, prima per il nodo connesso ad essa, che si occuperà di far effettuare la richiesta al nodo connesso al cluster.

Per alleggerire il carico del Master, non tutte le richieste passeranno da lui, se il nodo fisico può svolgerle da solo non ha senso appesantire ulteriormente il nodo master. Questo accade quando il nodo fisico che esegue, ad esempio, una lookup, ha già un nodo virtuale all'interno del cluster coinvolto nella richiesta, non ha senso quindi dover coinvolgere il master dell'attributo dato che, il nodo all'interno del cluster può rispondere lui stesso alla query diminuendo di molto i tempi di risposta.

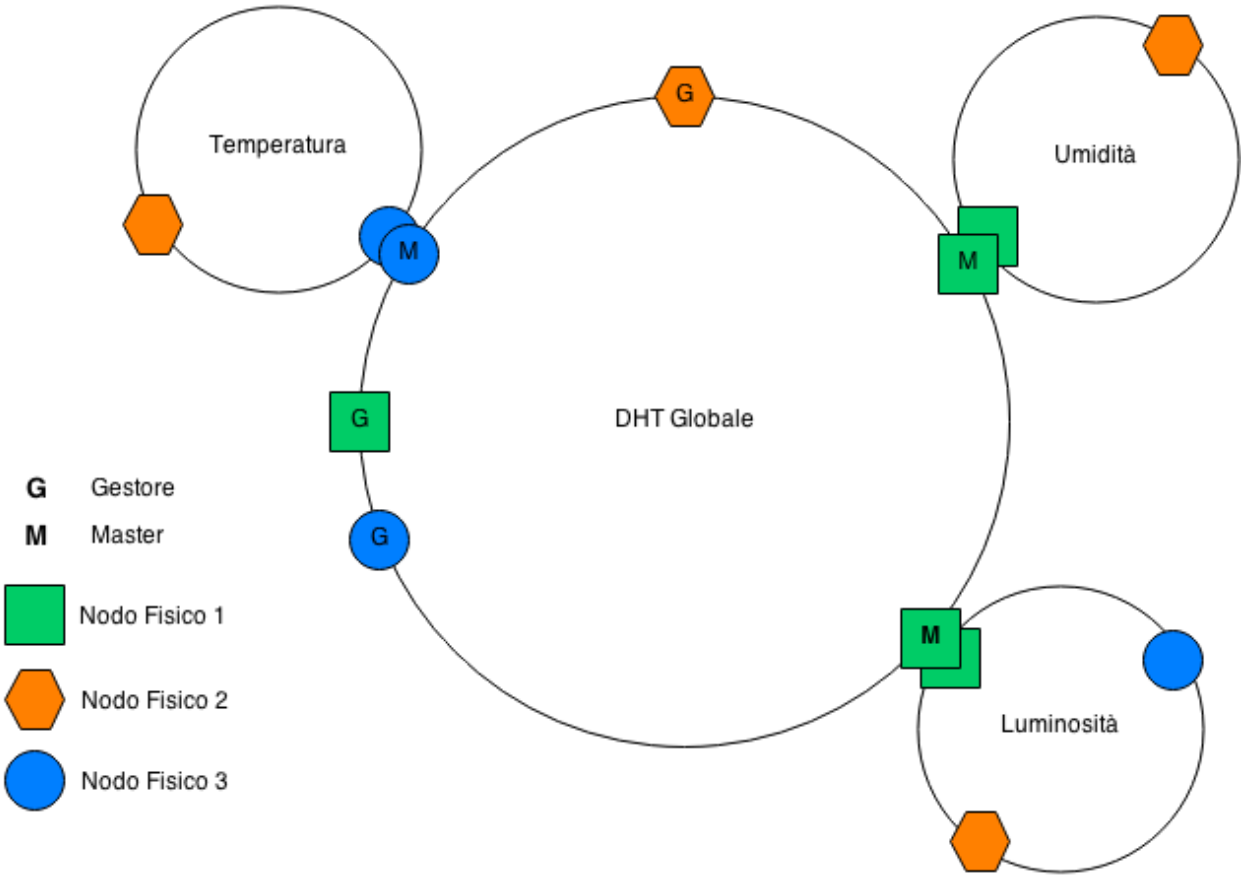

Figura 10. Esempio di DHT con 3 Attributi e 3 Nodi Fisici

Infine per facilitare la richiesta delle operazioni e quindi facilitare anche le operazioni di valutazione delle performance abbiamo deciso di implementare un socket server che gestisce le connessioni di più client contemporaneamente grazie al multi threading ed esegue le operazioni da loro richieste. Il thread aperto dalla connessione con il client richiede le operazioni direttamente al Manager dei Nodi che una volta ricevuti i risultati li passa al thread stesso che si occupa di ritrasferirli al client che ha effettuato la richiesta.

4.1.2. Comportamento della DHT e Diagrammi di Sequenza

In questo capitolo descriveremo il comportamento della DHT per quanto riguarda le operazioni principali: ingresso di un nodo nella rete, inserimento di un dato, lookup e range lookup di un dato.

4.1.2.1. Ingresso di un nuovo nodo nella rete

In primis andiamo ad analizzare l'ingresso di un nuovo nodo all'interno della DHT. Per accedere alla DHT il nodo deve conoscere l'indirizzo (IP e porta) di un nodo già connesso alla rete, eccetto che non sia lui il primo e quindi il nodo da cui la rete avrà origine. L'indirizzo non deve però essere quello di un Nodo Virtuale qualsiasi presente all'interno del Nodo Fisico, utilizzato come nodo di boot, ma deve essere necessariamente quello di un Nodo Virtuale collegato alla DHT Globale (quindi Nodo Gestore o qualsiasi Nodo Master).

La Figura 10 presenta un diagramma di sequenza su come avviene effettivamente il primo passaggio dell'ingresso di un nuovo nodo. Come possiamo vedere il primo passo che viene effettuato è la creazione del Nodo Gestore, senza il quale non si potrebbero effettuare tutte le operazioni seguenti.

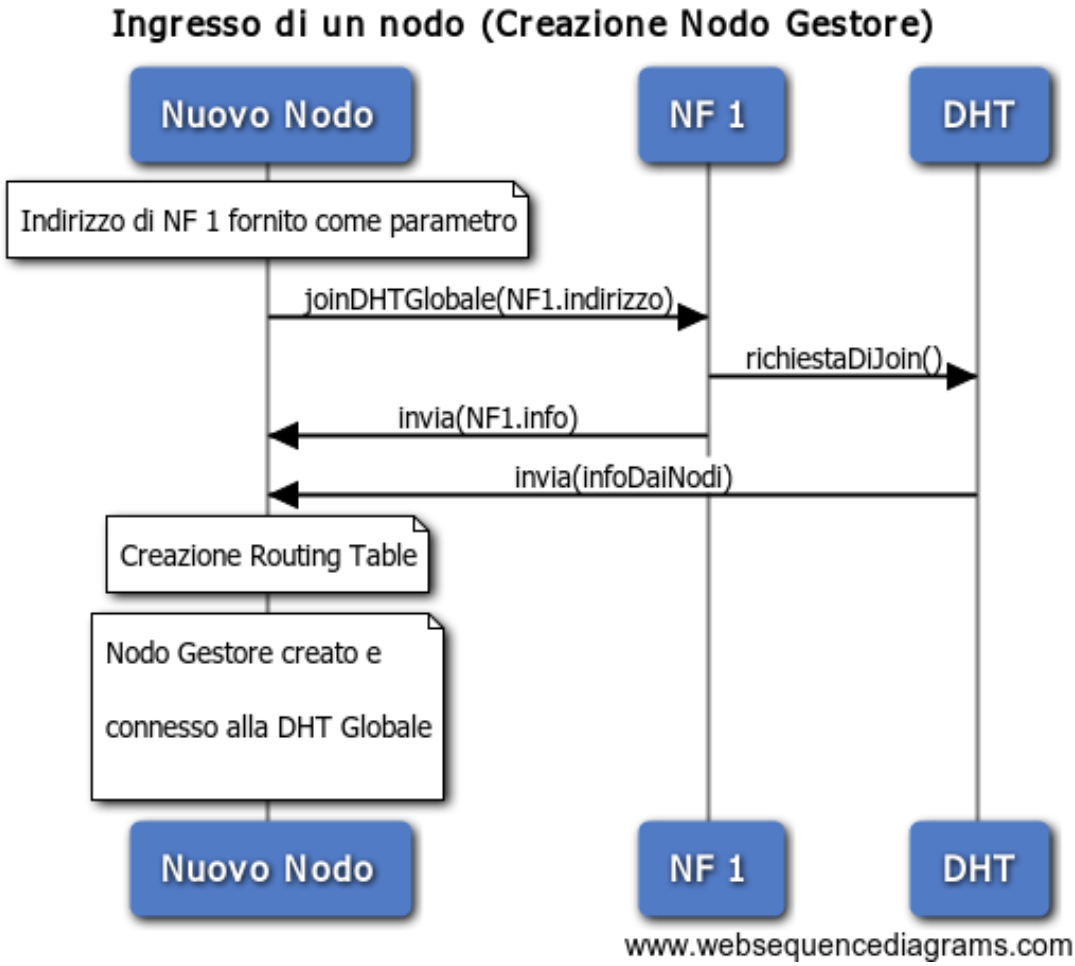

Figura 11. Ingresso di un nodo (Creazione Nodo Gestore)

Il nodo gestore del nuovo nodo si connette alla DHT Globale e diventa a tutti gli effetti uno dei nodi partecipanti ad essa.

Il passo successivo si divide in due casi precisi: il caso in cui tutti gli attributi gestiti dal nuovo nodo hanno già un loro cluster e il caso in cui almeno uno di essi è un nuovo attributo da memorizzare nella DHT.

Nel primo caso, presentato in Figura 11, tutti gli attributi sono già presenti nella DHT, andiamo quindi ad analizzare i singoli passaggi. Il Nodo Gestore appena creato e collegato alla DHT Globale invia un richiesta di ricerca per un nodo con identificatore uguale all'hash dell'attributo in cui inserire un nodo. Nel caso in esempio utilizziamo la "Temperatura", quindi il Nodo Gestore invierà una richiesta di ricerca all'interno della DHT Globale per un nodo con identificatore *hash("Temperatura").* Dato che abbiamo supposto che sono già presenti tutti gli attributi gestiti dal nostro nuovo nodo, la DHT troverà il nodo corrispondente e gli chiederà di

ritornare al nodo che ha effettuato la richiesta l'indirizzo del nodo interno al cluster dell'attributo "Temperatura".

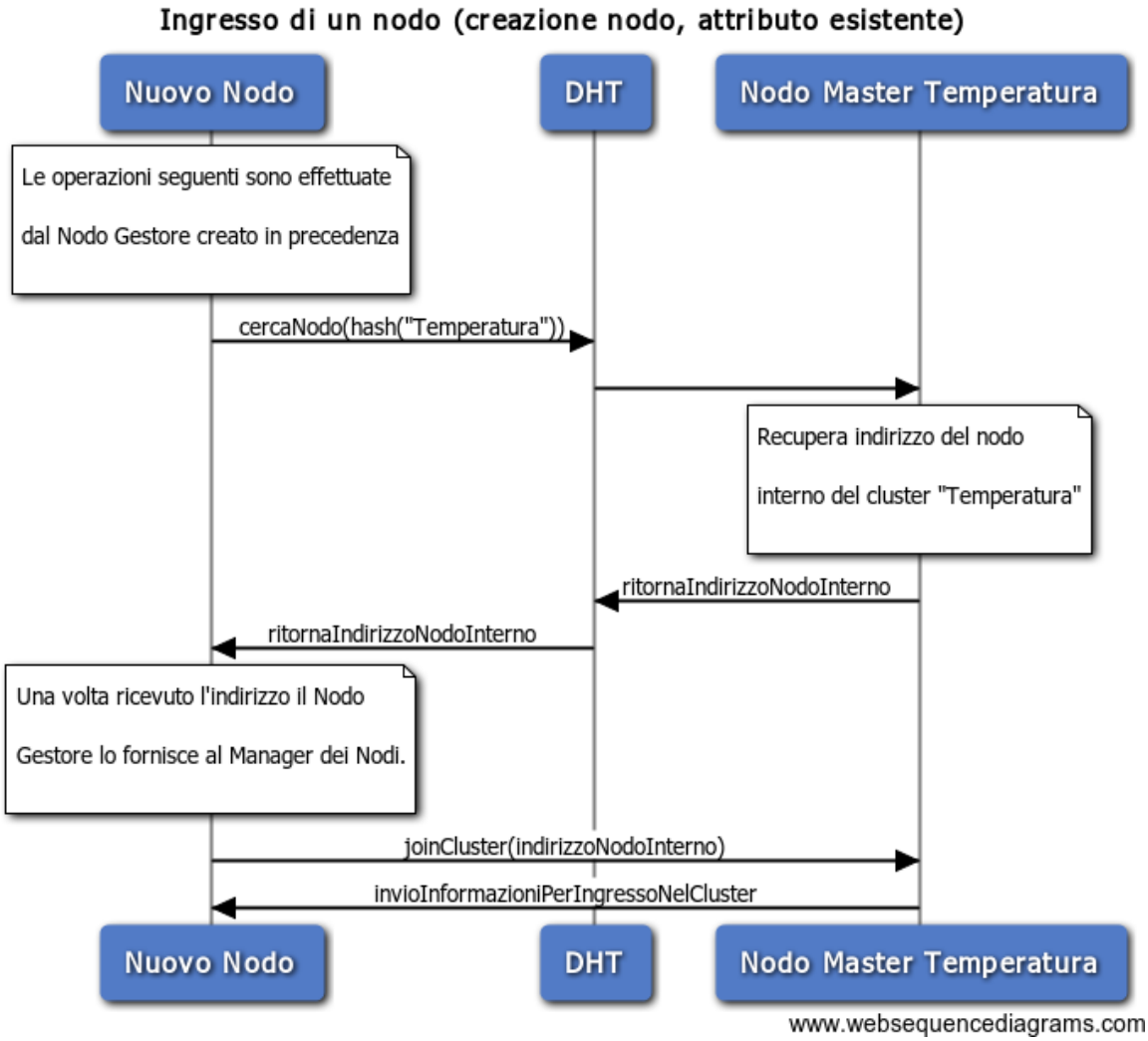

Figura 12. Ingresso di un nuovo nodo (creazione nodo in cluster esistente)

Una volta che il nuovo nodo riceve la risposta, questa viene passata al Manager dei nodi che crea un nuovo nodo virtuale ed utilizza l'indirizzo fornitogli dal Nodo Gestore per collegare il nuovo nodo al cluster dell'attributo voluto, in questo caso al cluster della "Temperatura". La creazione dei vari nodi virtuali gestiti dallo stesso Nodo Fisico all'interno dei rispettivi cluster avviene ovviamente in parallelo, una soluzione differente non sarebbe stata altrettanto performante.

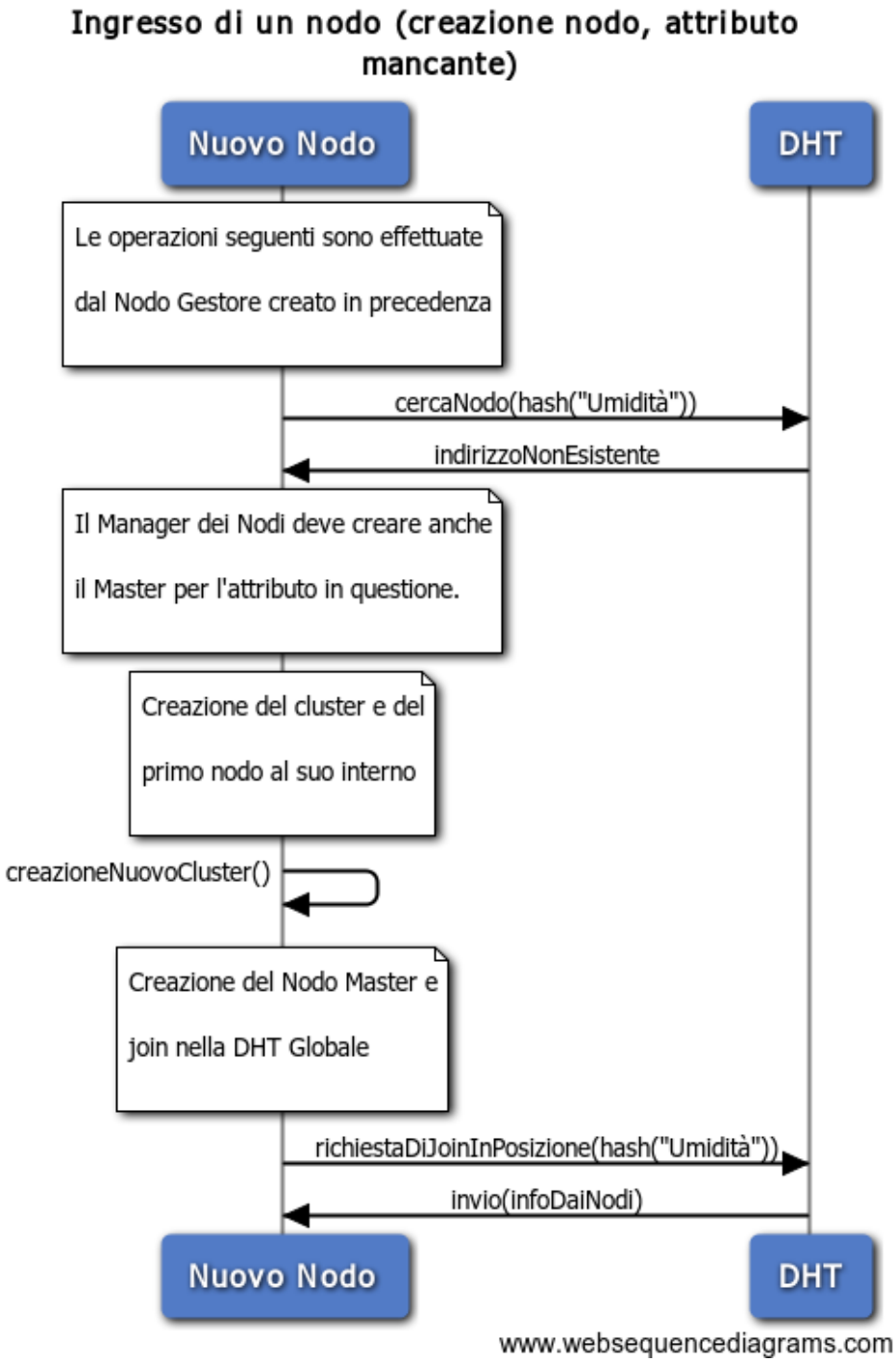

Figura 13. Ingresso di un nuovo nodo (creazione nodo e creazione nuovo cluster)

Nel caso in cui il cluster dell'attributo coinvolto non è presente, deve essere il nuovo nodo virtuale a crearlo. Dalla Figura 12 possiamo vedere che la DHT risponde negativamente alla richiesta di ricerca del nodo, il Manager dei nodi dovrà quindi creare non solo il nodo interno al cluster ma anche il Nodo Master, per quello specifico attributo, che dovrà essere collegato alla DHT Globale. Come vediamo dal diagramma di sequenza, per prima cosa verrà

creato il cluster grazie alla creazione del Nodo Virtuale interno e in secondo luogo verrà creato il Nodo Master per l'attributo, in questo caso *Umidità*, e inserito nella DHT Globale in posizione *hash("Umidità")*.

Pseudo-codice eseguito dal Manager dei Nodi di un Nodo Fisico che vuole accedere alla DHT

```
CreaNodi(lista di attributi da gestire) {
      creo il Nodo Gestore nella DHT Globale
      for (ogni attributo in lista) {
            verifico se è presente il Master
            if (MasterPresente) {
                  creo il nodo per l'attributo
            }
            else {
                  aggiungo il Master alla lista dei Master da creare
                  creo il cluster e il nodo per l'attributo
            }
      }
      for (ogni Master nella lista dei Master) {
            creo il Master dell'attributo corrispondente nella DHT Globale
      }
}
```

4.2.1.2. Inserimento di un dato

Per eseguire l'inserimento di un dato all'interno della DHT si devono differenziare due casi in cui si può trovare il nodo che vuole effettuare l'operazione. Nel primo caso il Nodo Fisico gestisce tra suoi Nodi Virtuali un nodo appartenente al cluster dell'attributo di cui si vuole inserire il valore, nel secondo caso il Nodo Fisico non gestisce questo tipo di nodo e deve adoperarsi per trovare un altro Nodo Fisico che possa finalizzare l'operazione di inserimento.

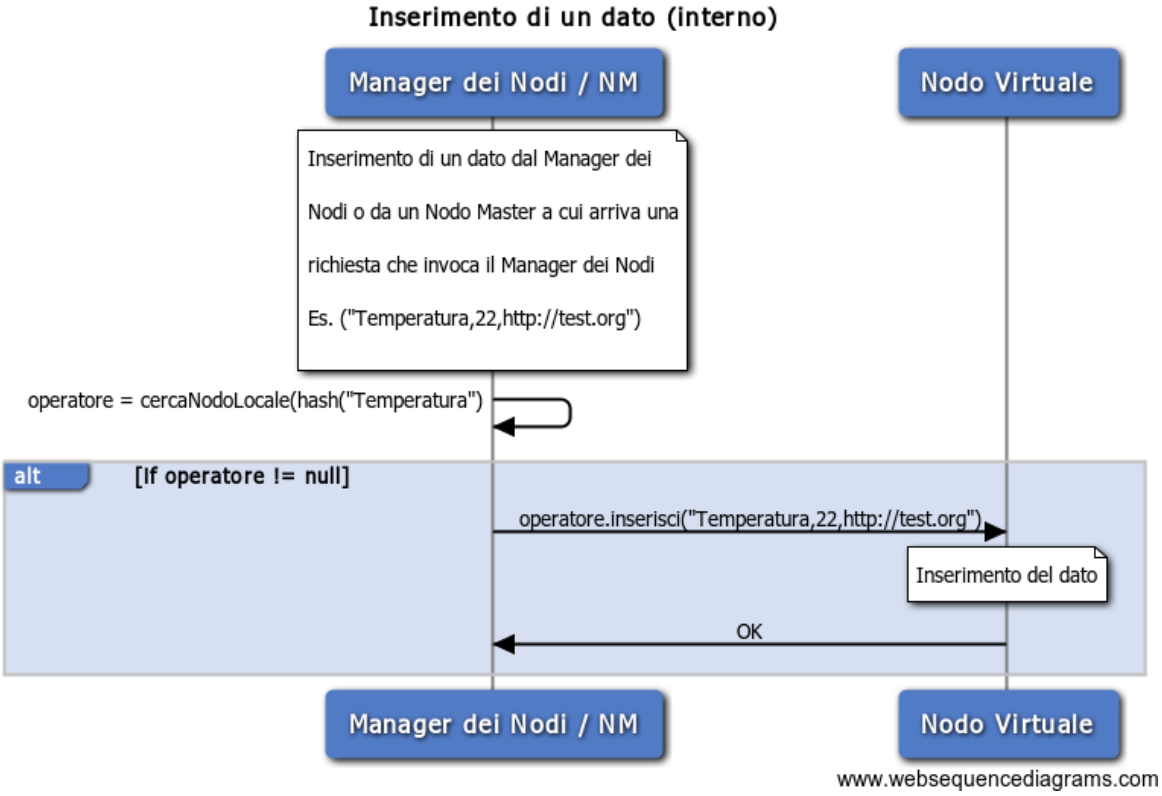

Figura 14. Inserimento di un dato (interno)

La Figura 13 presenta il diagramma di sequenza del primo caso. Il Manager dei nodi cerca se lui stesso ha nodi che posso effettuare operazioni su uno specifico attributo, nell'esempio si prende la "Temperatura". Se il Manager trova il nodo allora può utilizzarlo per effettuare l'inserimento del dato e delle informazioni relative ad esso (ad esempio un URL) .

La Figura 14 invece mostra le operazioni da eseguire nel caso in cui il nodo che effettua l'inserimento non abbia tra i suoi Nodi Virtuali un nodo facente parte del cluster dell'attributo di cui fa parte il dato. Una volta che il Manager dei Nodi ha verificato che non ci sono nodi locali in grado di effettuare l'inserimento invierà alla DHT una richiesta di inserimento al nodo in posizione *hash("Temperatura").* Il nodo che riceverà la richiesta potrà gestire o meno l'attributo, in caso non lo gestisca ritornerà indietro un errore di "attributo non esistente" altrimenti effettuerà l'operazione di inserimento interna come riportato in Figura 13 e ritornerà un messaggio di "dato inserito correttamente" al nodo che aveva richiesto l'operazione.

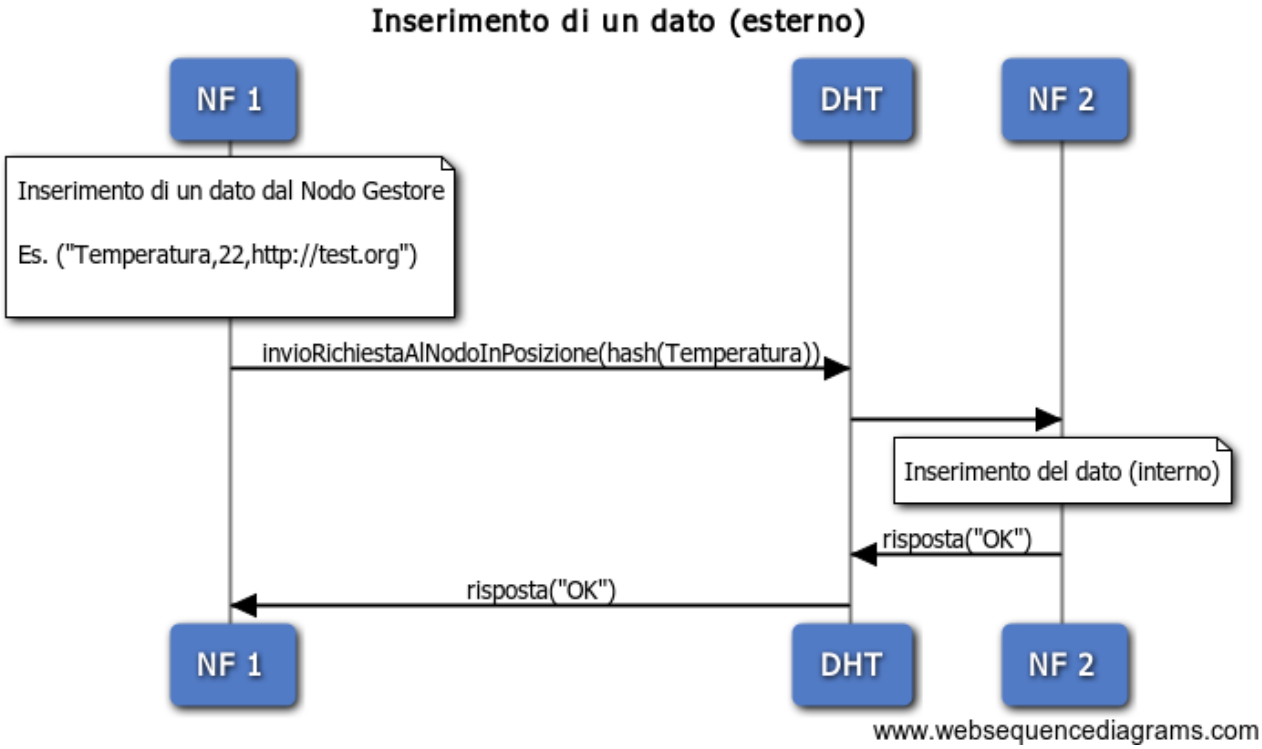

Figura 15. Inserimento di un dato (esterno)

Pseudo-codice di una operazioni di inserimento

```
InserisciElemento(attributo, valore, contenuto) {
      cerco un nodo locale facente parte del cluster 'attributo'
      if (nodoTrovato) {
            inserisco l'elemento in posizione 'localityPreservingHash(valore)'
      } else {
            invio il dato nella DHT Globale al nodo con identificatore
            hash(attributo)
            // in modo asincrono aspetto la risposta di inserimento effettuato
      }
}
```

4.2.1.3. Lookup e Range Lookup del dato

Come possiamo vedere dal diagramma di sequenza in Figura 15 l'operazione di lookup è praticamente identica all'operazione di inserimento. Il Manager dei Nodi controlla se tra i suoi Nodi Virtuali ce ne è uno che può effettuare l'operazione di lookup. Come per l'inserimento si possono verificare due casi: il Manager dei Nodi può eseguire il lookup

tramite un nodo da lui gestito oppure deve inviare la richiesta all'interno della DHT e aspettare, in modo asincrono, la risposta contenente i dati richiesti.

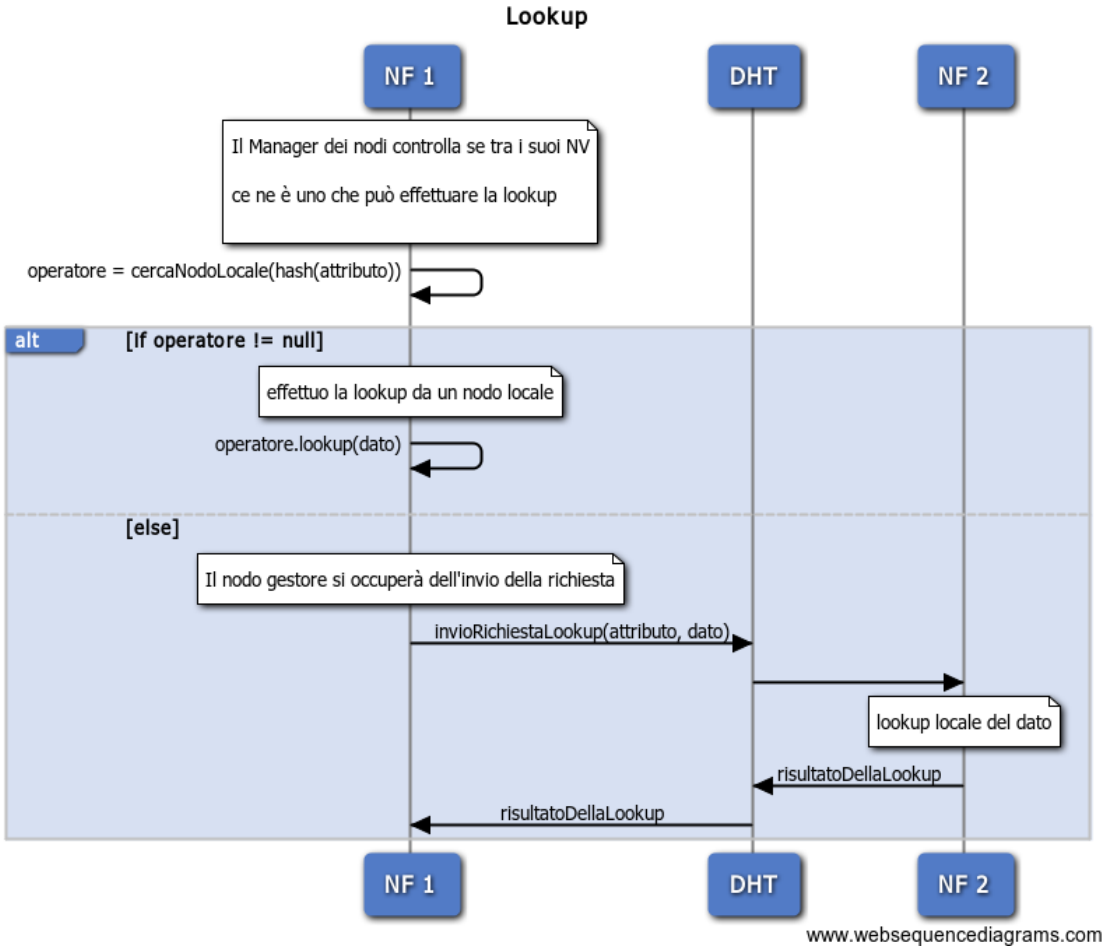

Figura 16. Lookup di un dato

Parliamo ora dell'operazione più importante che può essere effettuata dalla DHT nonché, obiettivo di questo lavoro, la Range Query. Il suo comportamento, nel caso in cui il nodo fisico non gestisca quel tipo di attributo, è identico a quello di una operazione di inserimento o di lookup, vale a dire: invio un messaggio all'interno della DHT andando a contattare chi può effettuare l'operazione per me. La differenza sta nel come vengono fatte le richieste all'interno dei singoli cluster. Facendo riferimento alla Figura 16 prendiamo un Nodo Virtuale (NV1), interno al Nodo Fisico (NF1), ed eseguiamo una range lookup che prende tutti i valori tra [minVal, maxVal]. NV1 invierà il messaggio al primo nodo interno al range, cioè il nodo che dovrebbe contenere *Lhash(minVal)*, con *Lhash* funzione hash locality preserving. Supponiamo che il primo nodo del range sia NVn (appartenente a NFn), quest'ultimo

risponderà al NV1 inviando i dati che soddisfano il range e la parte del suo leaf set[1] che include gli identificatori dei NV che potrebbero contenere dati interni al range.

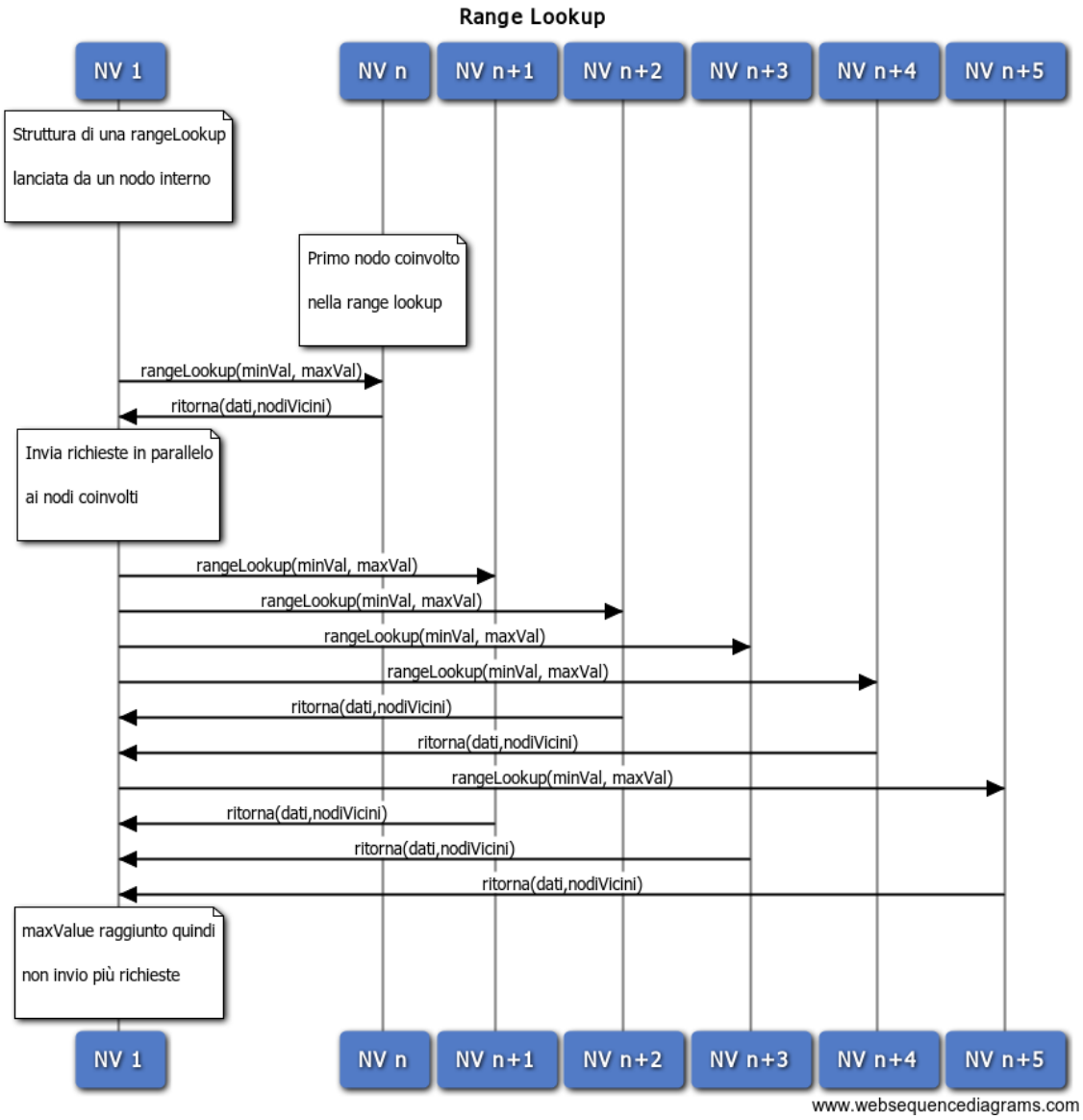

Figura 17. Range Lookup

Una volta ricevuta parte del leaf set di NVn che comprende i nodi interni al range, NV1 invierà in parallelo tante richieste di Range Lookup quanti sono i nodi della lista. Ogni nodo che riceverà il messaggio risponderà con i suoi dati e il suo leaf set filtrato, cioè contenente

[1] Leaf Set = Parte della Routing Table caratteristica di Pastry. Contiene una lista di nodi numericamente vicini sia in senso orario che in senso antiorario

solo gli identificatori dei nodi interni al range. NV1 analizzerà i nuovi leaf set e verificherà la presenza di nuovi nodi cui mandare la richiesta di range lookup, se sono presenti manderà tante altre richieste in parallelo per quanti sono, e così via ricorsivamente. L'algoritmo si fermerà se e solo se sono stati interrogati tutti i nodi all'interno del range più uno o più nodi successivi ad esso dato che potrebbero contenere dei valori interni al range stesso.

Una precisazione da fare per quanto riguarda la range query è che il client che compie le operazioni non saprà quanti nodi risponderanno alla range query quindi rimarrà in "ricezione" per un tempo predefinito (nel nostro caso 5 secondi) dopo il quale non accetterà più messaggi.

Pseudo-codice di una operazione di Lookup

```
CercaElemento(attributo, valore) {
      cerco un nodo locale facente parte del cluster 'attributo'
      if (nodoTrovato) {
            cerco il dato in posizione 'localityPreservingHash(valore)'
      } else {
            invio la richiesta di lookup nella DHT Globale al nodo con
            identificatore hash(attributo)
            // in modo asincrono aspetto la risposta contenente il contenuto
            // del dato
      }
```

Pseudo-codice di una operazione di Range Lookup

```
CercaElementiInRange(attributo, min, max) {
      cerco un nodo locale facente parte del cluster 'attributo'
      if (nodoTrovato) {
            invio richiesta di rangeLookup il al nodo in posizione
            'localityPreservingHash(min)'
            // avrò in risposta parte della sua Routing Table
            for (nodi nella Routing Table) {
                  Nuovo Thread: invio richiesta di rangeLookup al nodo[i]
                  // all'interno del thread il nodo aspetta le routing table
                  // di risposta e se trova nuovi nodi a cui non è stata
                  // ancora inviata la richiesta, apre nuovi thread per i
                  // nuovi nodi, e così via ricorsivamente
            }
      } else {
            invio la richiesta di rangeLookup nella DHT Globale al nodo con
            identificatore hash(attributo)
            // in modo asincrono aspetto le risposte contenenti i contenuti
            // dei dati
      }
}
```

4.2.3. Diagramma delle Classi

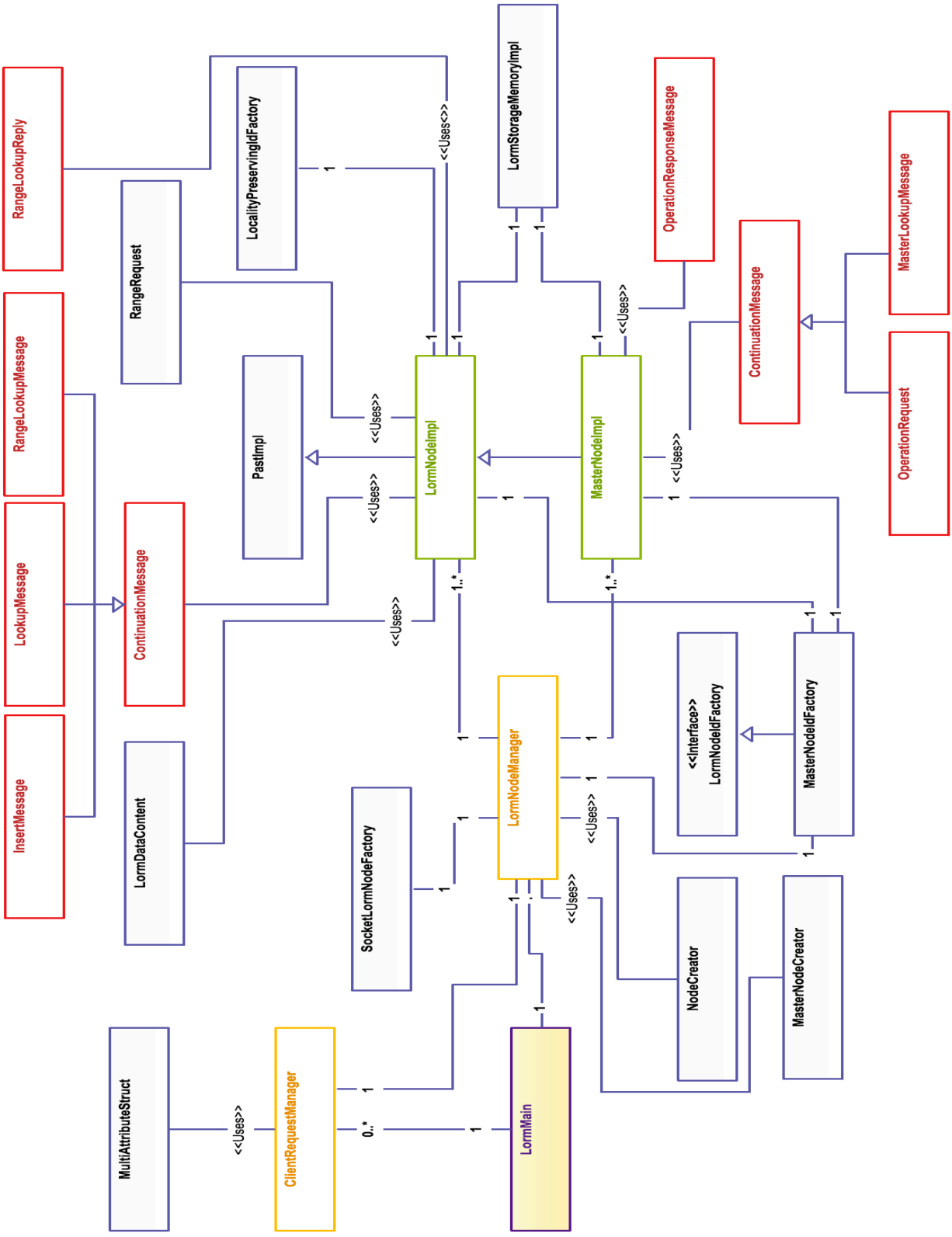

Figura 18. Diagramma delle Classi

In questo paragrafo presenteremo le classi principali che compongono il nostro software e le loro caratteristiche. Per facilitarne la comprensione abbiamo evidenziato con vari colori i diversi tipi di classi. In arancione le due classi che gestiscono tutto il Nodo Fisico, cioè la classe che gestisce le operazioni di comunicazione con il client e il Manager che gestisce tutti i nodi. In verde i due tipi di nodi: normale e master. Infine in rosso tutti i tipi di messaggi che girano all'interno della rete, sia i messaggi di richiesta sia i messaggi di risposta.

Andiamo ora a descrivere le singole classi cercando di farne comprendere la funzione per la quale sono state sviluppate:

- **LormMain:** È la classe principale del programma da cui vengono attivati i principali servizi, ossia il Manager dei Nodi e il SocketServer per la comunicazione con il client
- **ClientRequestManager:** Si occupa della gestione della comunicazione con il client e dell'invio delle operazioni da effettuare al LormNodeManager.
- **LormNodeManager:** Gestisce tutto il Nodo Fisico, crea e gestisce i nodi virtuali sia Master che non, invia ai giusti nodi virtuali le operazioni richieste. È il centro del Nodo Fisico.
- **NodeCreator e MasterNodeCreator:** Queste due classi si occupano della creazione di un nodo normale e di un nodo master, sono diverse perché utilizzano meccanismi di posizionamento diversi.
- **LormNodeIdFactory e MasterNodeIdFactory:** Si occupano del calcolo dell'ID del nodo e quindi del suo posizionamento all'interno del cluster o all'interno della DHT Globale.
- **LormNodeImpl:** È la classe che si riferisce ad un nodo virtuale non master, da qui partono messaggi di richiesta di operazione (ContinuationMessage) per tutte le operazioni possibili all'interno della DHT. Ovviamente questa classe gestisce anche il recupero e l'invio del dato se richiesto ad esempio da un **LookupMessage.**
- **MasterNodeImpl:** È la classe che si riferisce ad un nodo Master e al Nodo Gestore, gestisce l'invio delle richieste all'interno della DHT Globale, la ricerca dei cluster, l'invio di messaggi di risposta agli altri Master che hanno fatto delle richieste che lo coinvolgono, si occupa anche di richiedere ad un LormNodeImpl di eseguire una determinata operazione e restituirgli il risultato.
- **MasterNodeIdFactory:** È la classe per il calcolo dell'ID dei nodi, restituisce un ID casule se si tratta di un nodo normale o del Nodo Gestore, mentre, nel caso in cui si

tratti di un Nodo Master l'ID sarà uguale all'hash con l'algoritmo SHA1 della stringa che fa riferimento a quello specifico attributo.

- **RangeRequest:** Questa classe si occupa di inviare messaggi di range lookup ai nodi coinvolti, se ne creano tante istanze che inviano messaggi in parallelo quanti sono i nodi che potrebbero rispondere.
- **ContinuationMessage:** è il tipo padre dei messaggi di richiesta che circolano all'interno della rete, tutti i messaggi di richiesta derivano da questo.
- **InsertMessage:** Messaggio di richiesta di un'operazione di inserimento (solo all'interno del cluster).
- **LookupMessage:** Messaggio di richiesta di un'operazione di lookup (solo all'interno del cluster).
- **RangeLookupMessage:** Messaggio di richiesta di un'operazione di range lookup (solo all'interno del cluster).
- **MasterLookupMessage:** Messaggio che verifica l'esistenza di un cluster e quindi del suo Nodo Master (solo DHT Globale).
- **OperationRequest:** Messaggio di richiesta di un'operazione specificata al suo interno. Questo è l'unico messaggio di richiesta oltre al MasterLookupMessage che viaggia all'interno della DHT Globale.
- **RangeLookupReply:** Messaggio di risposta ad una range query.
- **OperationResponseMessage:** Messaggio di risposta ad una richiesta di operazione inviata da un Nodo Gestore ad un Nodo Master.
- **LocalityPreservingIdFactory:** Classe che implementa la locality preserving hash function utilizzata per calcolare gli ID dei dati che si riferiscono a valori degli attributi che richiedono il mantenimento della località.

UTILIZZO DELLA DHT MULTI-ATTRIBUTO

In questo capitolo andremo a vedere come utilizzare il software da noi creato e come un eventuale utente possa adattarlo alle sue esigenze. Andremo anche a vedere come utilizzare il client fornito con il software, per eseguire le operazioni di inserimento, lookup, range lookup e query multi-attributo. Infine, dato che il client presentato, è stato sviluppato solo per testare le performance della DHT e quindi è puramente per scopi valutativi, proporremo una soluzione più standard che utilizza l'interfaccia REST.

5.1. Creazione della DHT

Il software presenta due cartelle, una prima cartella denominata "clientLorm" che contiene appunto il client e una seconda cartella "nodeLorm" che contiene il codice da eseguire per il lancio del nodo fisico.

Iniziamo spiegando come creare un nodo fisico e creare una nuova DHT o semplicemente fare un'operazione di accesso ad una DHT già in esecuzione. Un utente che vuole iniziare ad utilizzare il nostro programma deve aver ben chiara una lista di attributi (o almeno uno), da far gestire al nodo fisico e le loro soglie, cioè i valori minimi e massimi per ogni attributo. Questo è molto importante perché la funzione hash locality preserving per poter funzionare deve attenersi, all'interno dello stesso cluster, sempre agli stessi valori degli estremi, questo vuol dire che ogni attributo una volta stabiliti gli estremi non potrà più cambiarli a meno di avviare una nuova DHT o un nuovo cluster, cambiando però il nome dell'attributo a cui si riferisce. Se un nodo, al momento dell'accesso alla DHT, vuole gestire un attributo già presente all'interno di essa, si farà inviare gli estremi già utilizzati nel cluster. Al contrario, se l'attributo non è presente, creerà un cluster con gli estremi specificati al momento del lancio del nodo.

Una volta entrati nella cartella nodeLorm dobbiamo per prima cosa compilare il programma. Per facilitare l'operazione abbiamo scritto uno script in Python che esegue la compilazione in modo automatico senza dover richiamare *Ant* e *Javac.* Il comando sarà semplicemente:

```
./compileDHT.py
```

Una volta compilato facciamo partire il nostro nodo. Se vogliamo creare una nuova DHT dovremmo utilizzare il nostro IP come IP del nodo di boot altrimenti dovremmo in qualche modo reperire l'indirizzo di un Nodo Master o di un Nodo Gestore già presenti all'interno della DHT.

```
java -cp .:pastry.jar:lib/xmlpull_1_1_3_4a.jar:lib/xpp3-1.1.3.4d_b2.jar
rice.tutorial.lesson3bis.LormMain <PortaLocale> <IndirizzoDiBoot>
<PortaDiBoot> <PortaDelServerSocket> <Attributo1,minVal1,maxVal1>
<Attributo2,minVal2,maxVal2> …
```

- <PortaLocale> si riferisce alla porta del primo nodo virtuale che andremo a creare, cioè il Nodo Gestore, tutti gli altri nodi virtuali andranno ad utilizzare le porte successive
- <IndirizzoDiBoot> è l'indirizzo del nodo che verrà utilizzato come nodo di accesso alla DHT, se stiamo creando una nuova DHT, invece, andremo ad utilizzare il nostro IP (non l'indirizzo di loopback).
- <PortaDiBoot> la porta utilizzata da uno dei Nodi Master o, ancora meglio, del Nodo Gestore del nodo che andremo ad utilizzare come nodo di accesso. Se stiamo creando una nuova DHT questo parametro sarà uguale alla <PortaLocale>.
- <PortaDelServerSocket> è la porta del server a cui si connette il client per effettuare le operazioni di inserimento e di ricerca dei dati.

Un esempio di esecuzione potrebbe essere:

```
java -cp .:pastry.jar:lib/xmlpull_1_1_3_4a.jar:lib/xpp3-1.1.3.4d_b2.jar
rice.tutorial.lesson3bis.LormMain 9001 10.0.1.1 9001 9000 Umidità,0,100
Temperatura,-100,100 MagnitudineAstrale,0,250
```

Il numero di nodi virtuali gestiti dal nodo fisico, dipenderà quindi da quanti attributi inseriremo nel comando di lancio. Supponiamo che l'IP della nostra macchina sia "10.0.1.1", una volta lanciato il comando dell'esempio abbiamo in risultato una nuova DHT, che gestisce tre attributi. Ipotizziamo ora di voler aggiungere altri nodi, il comando sarà praticamente lo stesso, dato che utilizzeremo 10.0.1.1 come nodo di boot e 9001 come porta di boot, se vogliamo possiamo cambiare la nostra porta locale e gli attributi che andremo a gestire. Supponiamo di voler gestire solo due attributi ed uno di questi non ancora inserito nella DHT, il comando sarà:

```
Java -cp .:pastry.jar:lib/xmlpull_1_1_3_4a.jar:lib/xpp3-1.1.3.4d_b2.jar
rice.tutorial.lesson3bis.LormMain 8900 10.0.1.1 9001 9000 Umidità,0,100
Gas,0,100
```

Con questo comando il nodo si connetterà alla DHT creata in precedenza, aggiungendo un nuovo cluster per l'attributo *Gas* e andando ad inserire un nuovo nodo nel cluster di *Umidità* già presente.

5.2. Utilizzo del Client

Introduciamo ora l'utilizzo del client. Il client prende in ingresso due parametri, il primo è l'indirizzo del socket server a cui ci connettiamo e il secondo è l'operazione o la lista di operazioni che andremo ad eseguire. Come introdotto precedentemente, non possiamo considerare questo client come un prodotto finito, proprio perché la sua implementazione non rispetta nessun protocollo standard dato che è servito solo per scopi di valutazione della rete. Più in avanti si potrà implementare un client che rispetti, ad esempio, il protocollo REST.

5.2.1. Operazione di Inserimento e Ricerca

Una volta lanciato il primo nodo della DHT possiamo procedere da subito alle prime operazioni di inserimento e ricerca dei dati. Introduciamo per primo, com'è ovvio, l'inserimento.

Entriamo nella cartella "clientLorm" e compiliamo il client che utilizzeremo per connetterci alla DHT ed effettuare le operazioni.

```
javac –classpath commons-cli-1.2.jar LormClient.java
```

Ora il client è pronto per compiere le operazioni e per mostrare la lista di quelle possibili basterà lanciare:

```
java –cp .:commons-cli-1.2.jar LormClient.java -h
```

Per quanto riguarda l'inserimento, il client dovrà specificare l'attributo di cui inserire il dato, il valore da inserire e un contenuto di riferimento che può servire ad identificare il nodo che ha inserito il dato, ad esempio la sua URL.

```
java –cp .:commons-cli-1.2.jar LormClient.java –c <Indirizzo:Porta> -i
<Attributo,Valore,Contenuto>
```

Mostriamo qui sotto un esempio di inserimento:

```
java –cp .:commons-cli-1.2.jar LormClient.java –c 10.0.1.1:9000 -i
Umidità,88,http://nodo1.org
```

È anche possibile effettuare l'inserimento di più dati contemporaneamente, semplicemente dividendoli con il ";" (Es. Umidità,88,http://nodo1.org;Temperatura,21,http://nodo1.org). Se tutto è andato a buon fine il nodo risponderà con un messaggio di "inserimento effettuato".

L'operazione di lookup si lancerà praticamente allo stesso modo andando a cambiare l'opzione –i con –l e inserendo Attributo e Valore, questa volta non inseriremo il Contenuto dato che è proprio quello che vogliamo ricavare

```
java –cp .:commons-cli-1.2.jar LormClient.java –c 10.0.1.1:9000 -l Gas,88
```

Il nodo eseguirà la query e ritornerà al client il risultato se questo è presente, altrimenti tornerà un errore di "dato non trovato" o "attributo non esistente" a seconda della motivazione. Anche in questo caso possiamo accodare le richieste da effettuare, basterà inserire un ";" proprio come viene fatto nell'inserimento.

L'operazione di range lookup è praticamente identica alla lookup appena spiegata, le uniche differenze riguardano l'opzione, invece di "–l" useremo "–r" e il fatto di dover inserire un intervallo invece che un solo valore, di seguito mostriamo un esempio di range query.

```
java –cp .:commons-cli-1.2.jar LormClient.java –c 10.0.1.1:9000 -r Gas,0,67
```

Il client aspetterà la ricezione di tutti i dati fin quando non scadrà il timeout della sua connessione al server. Questo è necessario proprio perché essendo un ambiente distribuito e inviando ricorsivamente richieste in parallelo (vedi capitolo 4.1.2), non sappiamo quanti nodi risponderanno alla nostra richiesta.

L'ultima operazione che è possibile eseguire è una query multi-attributo, questa racchiude in se sia la normale lookup sia la lookup di un intervallo. L'opzione da utilizzare sarà "-m" e tutti gli attributi coinvolti saranno separati da ";", sia che si riferiscano ad operazioni di normale lookup o di range lookup. Per far capire meglio all'utente includiamo un esempio di lancio di questa operazione:

```
java –cp .:commons-cli-1.2.jar LormClient.java –c 10.0.1.1:9000 -m
Gas,0,67;Umidità,76
```

Nell'esempio andremo a prendere tutti i dati per l'attributo *Gas* inclusi nell'intervallo e tutti i dati che presentano *Umidità* uguale a 76. Verranno poi intersecati i contenuti ricavati dalle due query e quindi mostrati all'utente. Questo tipo di query è uno strumento molto potente ed è quello che veramente mancava all'interno dell'internet degli oggetti

5.2.2. Implementazione del Client

In questo paragrafo presentiamo come è stato implementato il client. Come introdotto precedentemente, questa implementazione non è stata pensata per il prodotto finale ma, semplicemente, per avere un client con cui effettuare delle valutazioni sulle prestazioni della nostra rete nel più breve tempo possibile.

Una volta lanciato il comando da terminale, il client ne fa il parse ricavandone tutte le parti di suo interesse, prediamo, ad esempio, l'operazione di lookup vista prima:

```
java –cp .:commons-cli-1.2.jar LormClient.java –c 10.0.1.1:9000 -l Gas,88
```

Da questo comando il client ricaverà l'indirizzo del server a cui connettersi e l'operazione da eseguire, in questo caso una lookup.

Come vediamo dal diagramma di sequenza in Figura 18 il client invierà al server una stringa contenente il tipo di operazione da effettuare, il server quindi si disporrà in modo da ricevere dei dati in un certo formato ed eseguire la giusta operazione. Una volta inviati i dati il client si mette inattesa della risposta da parte del server, nel caso in cui il client aspetti il risultato di un inserimento o di una lookup appena arriva la risposta la connessione col server viene chiusa. Al contrario, nel caso in cui il client abbia richiesto un'operazione di range lookup o query multi-attributo la connessione viene chiusa solo dopo un tempo prestabilito, in cui il client non riceve dati, che l'utente può impostare andando ad editare il codice sorgente. Noi abbiamo deciso di impostarlo a 5 sec dato che per i nostri test ci sembrava più che sufficiente, essendo stati fatti in una rete locale.

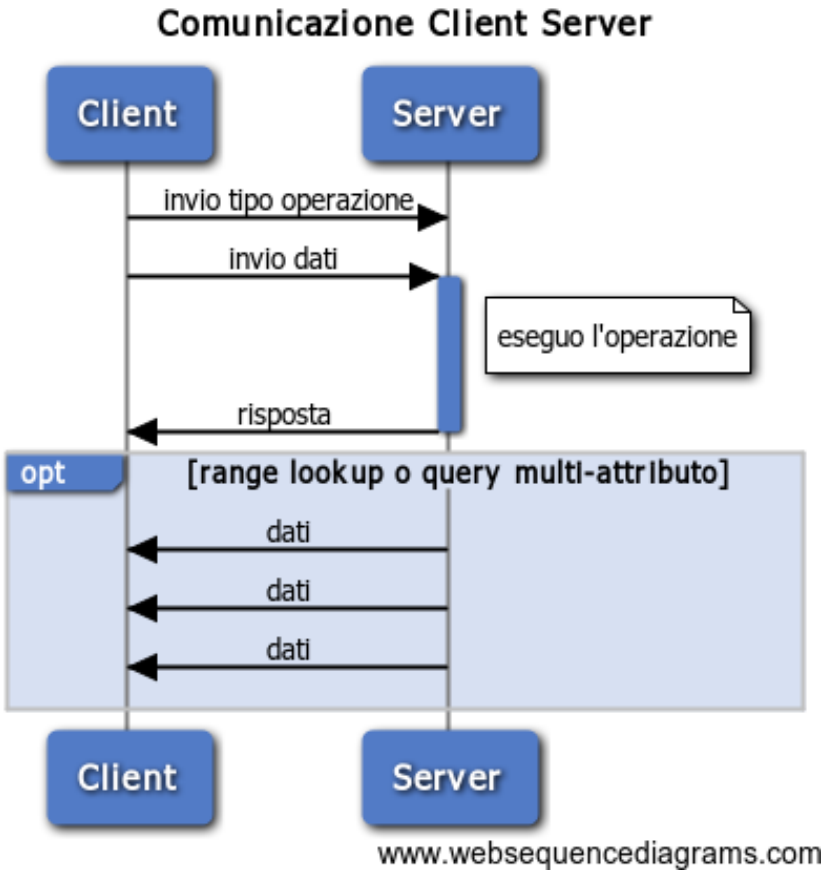

Figura 19. Comunicazione Client-Server del nodo

5.2.2.1. Implementazione con Interfaccia REST

In futuro si dovrà, ovviamente, modificare e standardizzare l'implementazione del client, una buona proposta potrebbe essere l'utilizzo dell'interfaccia REST, un'interfaccia leggera che secondo noi si può ben applicare all'internet degli oggetti. Si fa riferimento a REST quando viene applicato ai web services utilizzando le API dette RESTful[38]. Queste API definiscono quattro metodi HTTP standard (GET, PUT, POST, REMOVE) che prendono come parametro una URI, ad esempio "GET http://test.org/risorsa".

Tabella 3. Interfaccia RESTful

GET	Recupera una rappresentazione dell'elemento indirizzato nella raccolta.
PUT	Inserisce l'elemento indirizzato della raccolta, o se non esiste, la crea.
POST	Non è generalmente usata.
REMOVE	Elimina l'elemento indirizzato.

Analizzando queste quattro operazioni ci rendiamo conto che alla nostra DHT ne basterebbero due: la GET e la PUT e in futuro anche la REMOVE ma per ora non verrà trattata. I parametri da passare all'operazione verranno tutti inclusi all'interno della URI in uno specifico formato, di modo che una volta arrivati al server questo li possa facilmente dividere con un'operazione di parse sulla stringa rappresentante l'URI.

Le due operazioni che abbiamo implementato nel nostro sistema sono l'inserimento e le varie operazioni di lookup. Possiamo far corrispondere la PUT all'inserimento e la GET alla lookup, in questo modo il server saprà subito quale delle due operazioni dovrà eseguire.

Mentre nel caso della PUT la cosa è semplice dato che non ci sono ambiguità, nel caso della GET dobbiamo differenziare i vari tipi di operazioni che recuperano gli elementi. Per fare ciò possiamo discriminare le operazioni inserendo un riferimento a quale operazione di lookup si vuole eseguire dall'interno della URI.

Come detto prima questa standardizzazione sarà oggetto di un lavoro futuro, abbiamo però voluto introdurre e sottolineare la necessità di utilizzare dei protocolli standard leggeri e flessibili che calzino bene con i paradigmi dell'internet degli oggetti.

VALUTAZIONE DELLE PRESTAZIONI

Per verificare le prestazioni della nostra DHT abbiamo effettuato una serie di test andando prima a porre la rete in uno stato consistente, cioè con un numero di nodi precisi, che poi varieremo, e con i dati già inseriti, per poi andare a testare i tempi di risposta per operazioni di lookup e range lookup a rete scarica o considerando più nodi che effettuano richieste contemporaneamente (rete carica). I vari nodi fisici sono stati emulati grazie alla creazione di interfacce virtuali connesse tramite un bridge, anch'esso virtuale, in modo da poter far girare più nodi fisici, con routing table indipendenti, all'interno della stessa macchina. Questo metodo ci ha permesso di arrivare fino alla costruzione di 40 nodi fisici dopodiché la macchina ha dato segni di pesanti rallentamenti dovuti non alla memoria ma, crediamo, al kernel di Ubuntu 13.10.

Il meccanismo di creazione di interfacce di rete virtuali(veth) che ci ha permesso di poter testare il nostro sistema si chiama Linux Network Namespaces. Cerchiamo di spiegare cosa sono e a cosa ci sono servite queste interfacce virtuali. In generale, un sistema Linux condivide un unico insieme di interfacce di rete e di voci all'interno della tabella di routing. È possibile modificare le voci della tabella di routing e la politica di routing ma, cioè che non cambia è sostanzialmente il fatto che l'insieme delle interfacce di rete e le tabelle di routing sono condivise tra l'intero sistema operativo. I Network Namespaces cambiano questo presupposto. Con i Network Namespaces, si possono avere diverse e separate istanze di interfacce di rete e tabelle di routing che operano indipendenti l'uno dall'altra. Questo ci ha permesso di lanciare ogni nodo su una diversa interfaccia virtuale, avendoli tutti sulla stessa macchina. Possiamo fare un breve esempio di utilizzo dei Network Namespaces.

Creare un Network Namespaces è abbastanza semplice, basta utilizzare questo comando:

```
ip netns add <nome del namespace>
```

Supponiamo di voler creare due Network Namespace chiamati lorm-ns1 e lorm-ns2.

```
ip netns add lorm-ns1
ip netns add lorm-ns2
```

Creare i network namespaces è solo l'inizio; la parte successiva è quella di assegnare le interfacce ai namespaces, e quindi configurare le interfacce per la connettività di rete. È possibile assegnare solo interfacce ethernet virtuali (Veth) ai network namespaces. Le interfacce Ethernet virtuali vengono create sempre in coppia e sono collegate tra loro come un tubo, cioè ciò che entra nella prima interfaccia Veth uscirà dall'altra interfaccia. Come risultato, è possibile utilizzare le interfacce Veth per collegare i network namespaces al mondo esterno o anche solamente tra loro. Per fare questo necessitiamo della creazione di un bridge che raccolga tutte le "prime" interfacce Veth.

```
brctl addbr lorm-br
```

Creiamo ora le interfacce Veth:

```
ip link add veth1 type veth peer name veth1-ns
ip link add veth2 type veth peer name veth2-ns
```

Come scritto prima raccogliamo tutte le prime interfacce veth all'interno del bridge:

```
brctl addif lorm-br veth1
brctl addif lorm-br veth2
```

Il prossimo passo è quello di inserire le altre interfacce Veth all'interno dei network namespaces creati.

```
ip link set veth1-ns netns lorm-ns1
ip link set veth2-ns netns lorm-ns2
```

Ora per poter far comunicare le nostre veth dobbiamo assegnargli un'IP sia a quelle nel bridge sia a quelle all'interno dei diversi network namespaces

```
ifconfig veth1 10.0.0.1
ifconfig veth2 10.0.0.2
ip netns exec lorm-ns1 ifconfig veth1-ns 10.0.1.1
ip netns exec lorm-ns2 ifconfig veth2-ns 10.0.1.2
```

`ip netns exec` serve ad eseguire un comando su una specifico network namespaces, infatti, come vediamo subito dopo è riportato il nome del namespace e il comando da eseguire.
Per completezza andiamo ad attivare l'interfaccia di loopback all'interno dei nostri network namespaces con questi due semplici comandi.

```
ip netns exec lorm-ns1 ifconfig lo up
ip netns exec lorm-ns2 ifconfig lo up
```

Per finire attiviamo il bridge altrimenti non funzionerà nulla

```
ip link set lorm-br up
```

Ora se proveremo ad eseguire un ping da un network namespace all'altro vedremo che le due interfacce possono comunicare.

Grazie a questo meccanismo abbiamo potuto emulare il comportamento di una rete reale senza dover andare a scomodare pesanti macchine virtuali.

Un altro passaggio necessario per la valutazione delle prestazioni è stata la scrittura degli script in Python. Grazie agli script abbiamo potuto automatizzare il processo di creazione dei network namespaces, creazione dei nodi, inserimento dei dati, aggiunta del ritardo con Dummynet e la valutazione di tutti i test che saranno mostrati nel prossimo paragrafo.

6.1. Lookup

La prima cosa che siamo andati a valutare sono stati i tempi di una normale operazione di lookup. Inizialmente abbiamo dovuto stabilire i vari parametri del sistema da mantenere fissi e quelli da far variare per eseguire i test, tenendo conto anche dei limiti hardware e adattandoci ad essi.

I seguenti test sono fatti in una DHT con 20 attributi, 100 valori per attributo e 5 attributi da gestire per ogni Nodo Fisico. Quello che andremo a variare sarà il numero di nodi fisici all'interno della rete, partiremo con 10 NF aumentando di 10 fino ad arrivare a 40 NF che è il limite massimo consentito dal nostro hardware. Proprio a causa di questo limite abbiamo deciso di diminuire la dimensione del leaf set da 24, valore standard di Pastry, a 10, in questo modo vengono memorizzati 5 nodi in ordine orario e 5 in ordine antiorario, in modo da riuscire mostrare un comportamento della rete più reale possibile. Possiamo calcolare i Nodi Virtuali per ogni test all'interno della DHT in questo modo $(n \times 6) + 20$ dove n è il numero dei nodi fisici coinvolti, 6 è il numero degli attributi che gestisce ogni nodo più il Nodo Gestore ed infine 20 è il numero di Nodi Master che ovviamente è pari al numero degli attributi. Ad esempio con 40 nodi fisici avremo 260 nodi virtuali all'interno della rete.

Nella Figura 17 possiamo vedere l'andamento del tempo di lookup a rete scarica all'aumentare dei nodi connessi alla rete. L'aumentare del tempo è dovuto, ovviamente, all'aumentare dei salti che la richiesta deve fare all'interno dei cluster e della DHT globale per andare a ricavare l'informazione voluta.

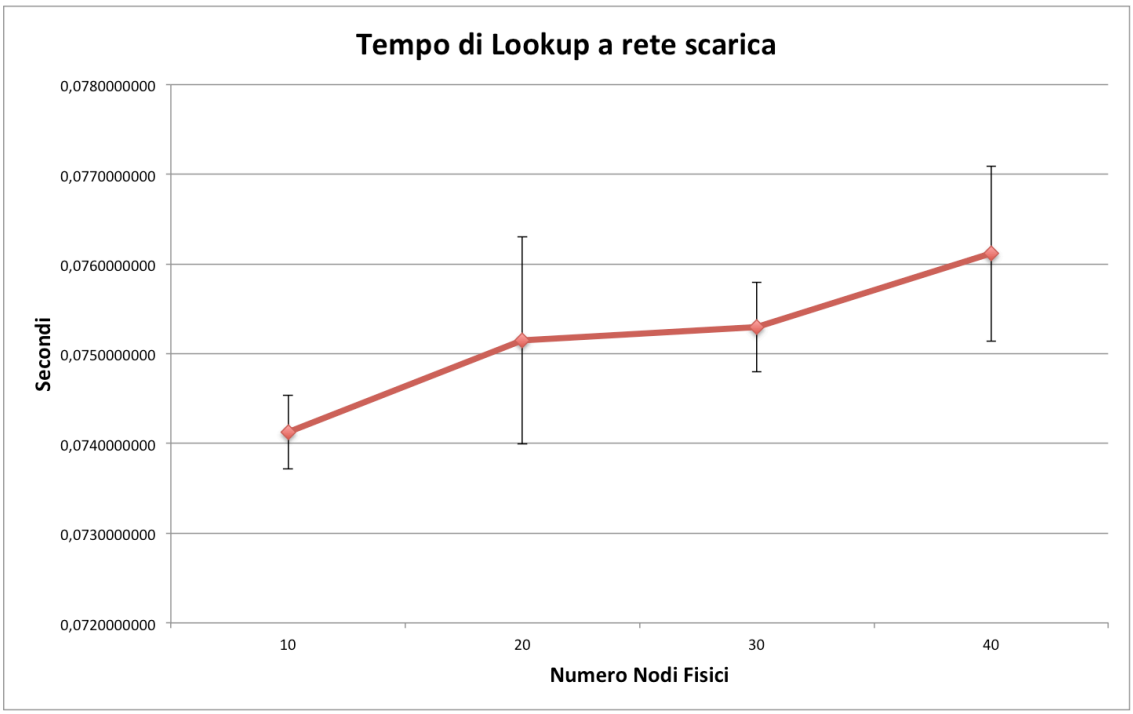

Figura 20. Tempo di Lookup a rete scarica

Nella Figura 18 abbiamo analizzato come risponde la DHT quando ci sono più richieste che girano in parallelo all'interno di essa. In questo specifico test abbiamo utilizzato cinque nodi che fanno richieste di lookup intervallate dalla distribuzione di Poisson con un rate di 200ms. Abbiamo poi variato il rate della distribuzione basandoci sul fatto che un rate di 200ms(blu) è ottimale per il funzionamento della stessa, quindi abbiamo analizzato la rete anche con un rate di 100ms(rosso) e 400ms(verde) per analizzare il comportamento con una rete più affollata ed una più sgombra. Come ci aspettavamo la rete risponde più lentamente per quanto riguarda 100ms di rate e più velocemente per un rate di 400ms.

Entrambi questi test sono stati effettuati senza inserire il ritardo tra le connessioni. Abbiamo pensato, quindi, di inserire il ritardo grazie all'utilizzo di Dummynet[22] ed compiere nuovamente gli stessi test per verificare il corretto funzionamento della rete simulando un ambiente più vicino alla realtà. Il ritardo inserito è pari a 40ms per nodo fisico, quindi un ping tra interfacce sarà maggiore uguale a 80ms.

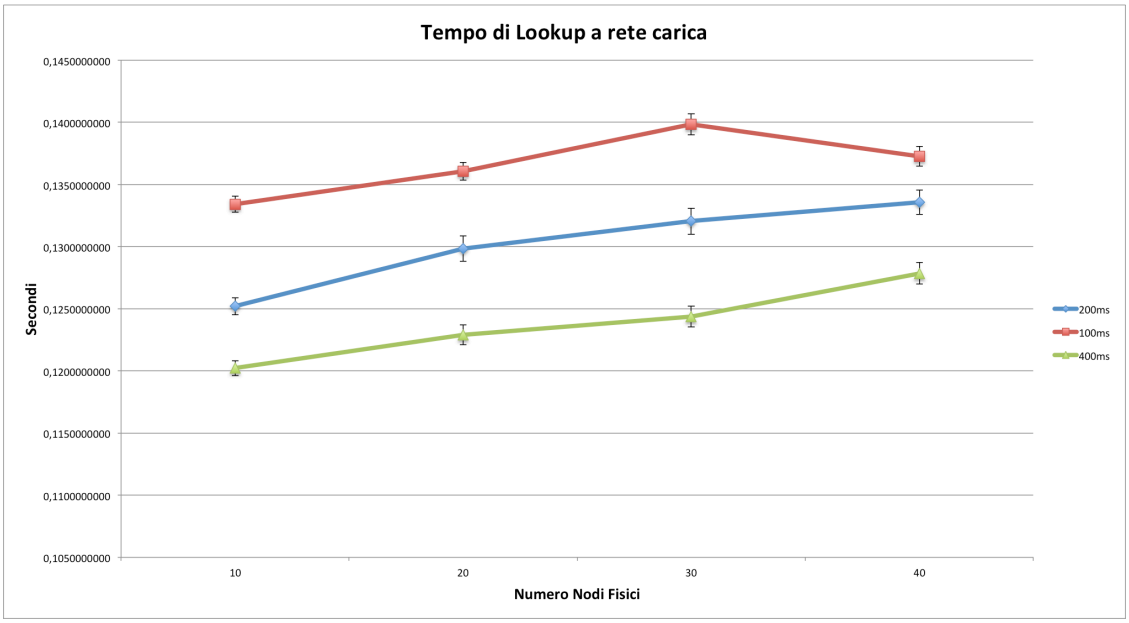

Figura 21. Tempo di Lookup a rete carica

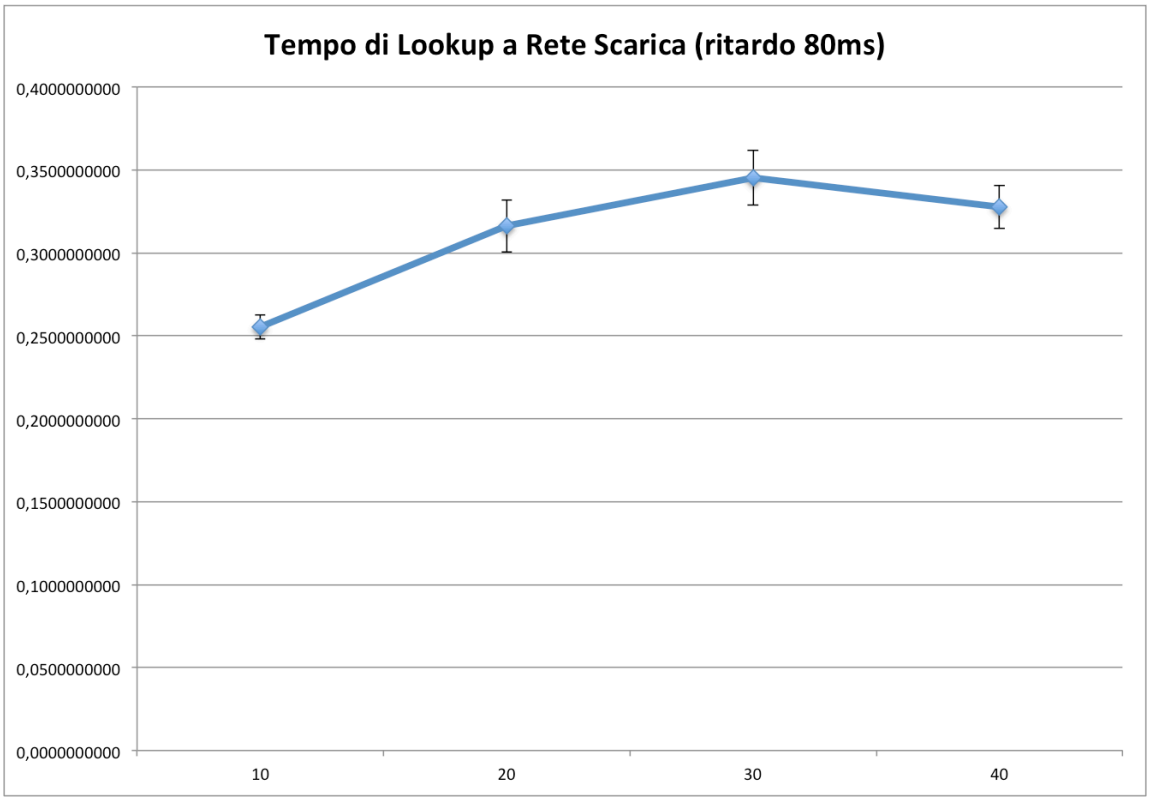

Figura 22. Tempo di Lookup a rete scarica con ritardo

Come vediamo in Figura 19 e in Figura 20 la rete si comporta come ci aspettavamo. Nel grafico a rete carica tutte le righe sono molto più ravvicinate ma mantengono la giusta

posizione condizionata dal rate utilizzato, la 100ms(blu) è quella con più ritardo mentre la 400ms(verde) è quella con meno ritardo.

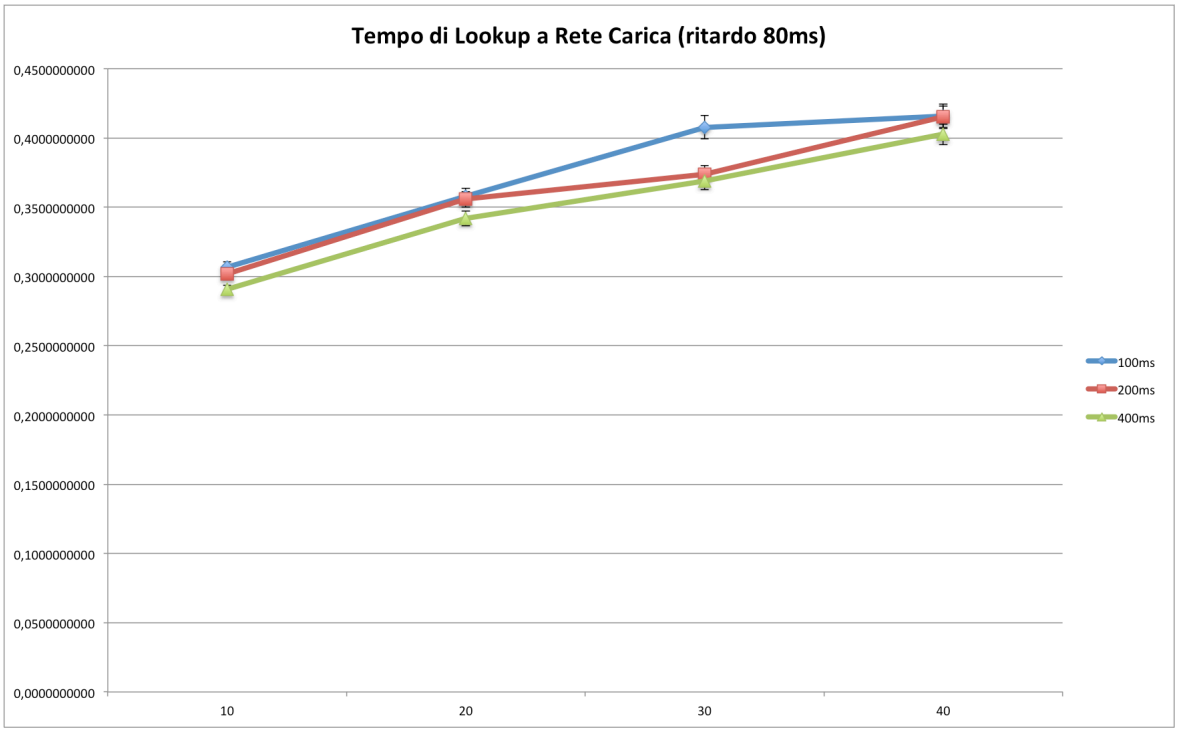

Figura 23. Tempo di Lookup a rete carica

Possiamo cercare di stimare la complessità dell'algoritmo facendo riferimento alla complessità di Pastry DHT $O(\log_B N)$ dove $B = 2^b$ con $b = 4$ secondo lo standard (praticamente sarebbe la dimensione della routing table). Abbiamo quindi due casi: il caso in cui il Nodo Fisico ha tra i suoi nodi virtuali un referente per l'attributo cercato e il caso in cui il Nodo Fisico deve far passare la richiesta per la DHT Globale dato che non gestisce internamente quel tipo di attributo. Nel primo caso la complessità sarà uguale a quella di Pastry $O(\log_B N)$, nel secondo caso sarà pari a 2 volte la complessità di Pastry dato che dovrà attraversare anche la DHT Globale $O(2\times\log_B N)$.

6.2. Range Lookup

Per quanto riguarda la range lookup abbiamo analizzato il ritardo della rete all'aumentare dell'intervallo cercato. Abbiamo deciso di fare i test con la rete configurata con 40 nodi in modo da mostrare il comportamento con il numero massimo di nodi fisici connessi alla DHT. Nella Figura 21 viene mostrato il tempo medio di una operazione di ricerca un solo elemento, per poi passare a una range di 5 elementi, 10 e così via aumentando di 10 fino a 100 elementi, cioè tutti gli elementi inserito per un singolo attributo. In questo modo la richiesta

di lookup con 100 elementi coinvolgerà di sicuro tutti nodi virtuali del cluster di quello specifico attributo. Possiamo notare dal grafico come il tempo di ricerca aumenti in base a quanti valori vogliamo ricercare, questo perché più aumentiamo l'intervallo di valori, più salti dovrà fare la nostra richiesta.

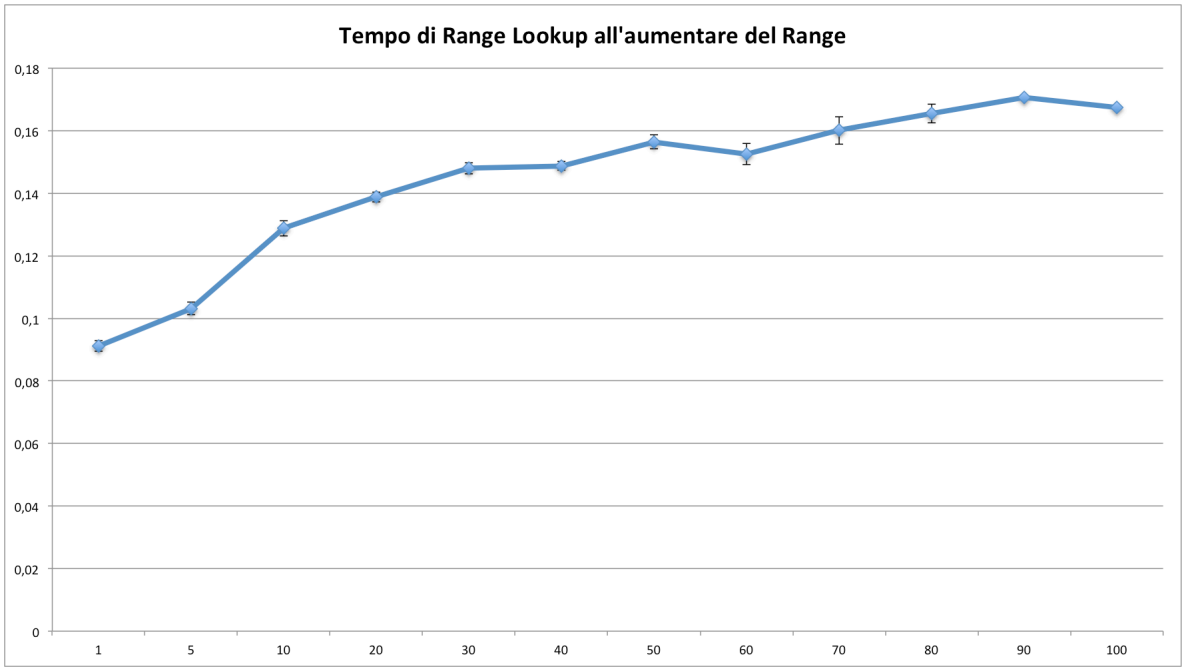

Figura 24. Tempo di Range Lookup all'aumentare del Range ricercato

Anche per la Range Lookup possiamo cercare di stimare la complessità dell'operazione che sarà praticamente identica a quella della lookup con la differenza che nel caso della range dobbiamo sommare gli n salti per il recupero di tutti i dati inclusi nell'intervallo. In formule $O(\log_B N) + n$ per quanto riguarda la lookup da un nodo locale mentre $O(2 \times \log_B N) + n$ per quanto riguarda una richiesta di lookup passante per la DHT Globale.

CONCLUSIONI E LAVORI FUTURI

Internet ha cambiato radicalmente il nostro modo di vivere, di muoversi e di interagire con le persone in vari contesti, che vanno dalla vita professionale ai rapporti sociali. L'Internet degli Oggetti ha il potenziale per aggiungere una nuova dimensione a questo processo, consentendo la comunicazione con e tra oggetti intelligenti.

A questo scopo, l'internet degli oggetti dovrebbe essere percepito quale parte integrante dell'Internet del futuro, che sarà notevolmente diverso dall'Internet che usiamo oggi. Infatti, è evidente che l'Internet attuale è stato costruito intorno ad un paradigma di comunicazione host-to-host, questo è ora un fattore limitante per lo sviluppo dell'Internet del futuro. Ormai, Internet è usato soprattutto per la pubblicazione e il recupero di informazioni (indipendentemente dell'host su cui tale informazione viene pubblicata o recuperata) e, quindi, al centro delle soluzioni di comunicazione e di rete dovrebbero esserci le informazioni stesse.

La ricerca di informazioni è quindi un punto cruciale nell'Internet delle cose, senza meccanismi di ricerca la visione che abbiamo dell'internet del futuro non sarebbe realizzabile. A differenza dei meccanismi di ricerca analizzati nel capitolo 3, il sistema sviluppato in questa tesi implementa una soluzione multi-attributo con possibilità di effettuare richieste di informazioni all'interno di uno specifico range. Tutto ciò mantenendo delle performance comparabili a DHT mono-attributo e un alto livello di scalabilità, affidabilità, robustezza e facilità di manutenzione del sistema complessivo propria dei sistemi DHT.

Possiamo pensare, il lavoro svolto in questa tesi, come un ulteriore passo verso lo sviluppo dell'internet degli oggetti, verso un mondo in cui la vita dell'utente verrà ulteriormente facilitata grazie alle informazioni che riceverà in tempo reale sul suo smartphone, sul suo pc o, perché no, su altri oggetti collegati alla rete. In questi anni la ricerca sull'internet delle cose sta avendo una crescita esponenziale e questo lavoro può essere considerato un altro gradino verso il futuro dell'internet moderno.

In futuro sarebbe interessante testare il nostro sistema con migliaia di nodi che inviano e richiedono informazioni per verificare la scalabilità del sistema in un ambiente reale. Si potrebbe pensare di aggiungere dei vincoli ai singoli nodi in base alle prestazioni di ognuno, ad esempio nodi con prestazioni limitate non dovrebbero mai essere usati come Nodi Master,

oppure nodi con poca memoria dovrebbero contenere meno dati dei nodi con molta memoria. Questo tipo di modifiche, eseguite in base alle caratteristiche di ogni partecipante alla DHT, sarebbero molto interessanti da sviluppare, per il fatto che anche un semplice sensore con una interfaccia di rete potrebbe partecipare alla DHT e inserire le sue informazioni ma non potrebbe mai gestire un carico di informazioni pari ad un grosso server con un'enorme capacità computazionale e di memoria. Un altro perfezionamento sarebbe il poter inserire delle politiche all'interno dei nodi che stabiliscano quali nodi possono richiedere informazioni e quali no, quali enti hanno accesso a tutte le informazioni e quali no. Altra cosa, introdotta già all'interno della tesi, sarebbe la standardizzazione del protocollo di comunicazione utilizzato per lo scambio di messaggi tra client e server sul nodo fisico.

RINGRAZIAMENTI

In questo capitolo cercherò di ringraziare tutti quelli che mi sono stati vicino in questi otto anni di profondi cambiamenti, di grandissime soddisfazioni ma anche di piccole delusioni, otto anni che rimarranno indelebili nella mia mente. Spero di non dimenticare nessuno.

In primis i miei genitori e mia sorella Ilaria, senza i loro, tanti forse tantissimi, sacrifici e il loro continuo supporto non sareste qui a leggere questi ringraziamenti, quindi a loro va il primo e più grande grazie, spero un giorno di potervi ripagare in pieno. Ringrazio anche il Prof. Enzo Mingozzi insieme con Carlo e Giacomo che mi hanno aiutato molto nello sviluppo di questa tesi e che a volte mi hanno letteralmente tolto le patate bollenti dal fuoco.

Un ringraziamento speciale va ad Armando, un fratello, un punto di confronto e una sicurezza fin dal primo giorno di Università, con l'augurio che, prima o poi, finiremo a lavorare insieme e a coronare i sogni che ci accomunano. Un grande grazie ad Alessandra, la sorpresa inaspettata di questo ultimo anno, lei mi ha sopportato, supportato e strigliato quando ce ne era bisogno, soprattutto nelle ultime due settimane prima della discussione, grazie. Grazie anche a Francesco, anche lui un coinquilino/fratello con cui ho condiviso ogni giornata pisana prima che partisse per la terra dei canguri.

Un grazie a chi c'è sempre stato: mio cugino Federico, Andrea, Massimiliano e Federico, che nonostante la mia fuga da Terni mi sono sempre stati vicino. Ai colleghi conosciuti nei primi anni, alcuni persi per strada, altri sempre presenti: Alessandro, Antonio, Andrea, Alessio, Angelo. Alle amicizie fatte in questi ultimi anni con cui ho condiviso tutta la mia vita mondana pisana e ternana: Elisa, Selene Daniele, Santina, Chiara, Federica, Anna ecc. A tutti i pazzi dei G.O. 77 Passi, senza di voi questi ultimi anni non sarebbero stati così divertenti. Agli irriducibili del PerLab, ai traguardi raggiunti al CUS con Tremanza e con le Fere e a tutti voi che state leggendo questi ringraziamenti.

E per ultimo grazie a me che finalmente ho raggiunto il traguardo che sognavo fin da quando avevo 1010 anni.

BIBLIOGRAFIA

[1] Evdokimov, S., Fabian, B., Kunz, S., & Schoenemann, N. (2010). Comparison of Discovery Service Architectures for the Internet of Things . IEEE International Conference on Sensor Networks, Ubiquitous, and Trustworthy Computing , 8.

[2] Shen, H., & Xu, C.-Z. (2012). Leveraging a Compound Graph-Based DHT for Multi-Attribute Range Queries with Performance Analysis. IEEE TRANSACTIONS ON COMPUTERS , 61 (4), 15.

[3] EPCglobal, "EPCglobal Object Naming Service (ONS) 1.0.1," EPCglobal, May 2008. [Online]. Available: http://www.epcglobalinc.org/standards/ons

[4] EPCglobal, "The EPCglobal Architecture Framework – Version 1.0," March 2005. [Online]. http://www.epcglobalinc.org/standards/architecture/architecture_1_0-framework- 20050701.pdf

[5] EPCglobal, "The EPCglobal Architecture Framework – Version 1.3," March 2009. [Online]. Available: http://www.epcglobalinc.org/ standards/architecture/

[6] BRIDGE, "BRIDGE WP02 – High Level Design for Discovery Services," August 2007.

[7] ——, "BRIDGE WP02 – Working Prototype of Serial-level Lookup Service," February 2008.

[8] A. Rezafard, "Extensible Supply-chain Discovery Service Problem Statement," IETF Internet-Draft, Nov. 17 2008, draft-rezafard-esds- problem-statement-03. [Online]. Available: http://tools.ietf.org/html/draft- rezafard- esds- problem- statement- 03

[9] K. Främling, M. Harrison, and J. Brusey, "Globally Unique Product Identifiers – Requirements and Solutions to Product Lifecycle Management," in Proceedings of 12th IFAC Symposium on Information Control Problems in Manufacturing (INCOM), 2006, pp. 17–19.

[10] H. Balakrishnan, M. F. Kaashoek, D. R. Karger, R. Morris, and I. Stoica, "Looking up Data in P2P Systems," Communications of the ACM, vol. 46, no. 2, pp. 43–48, 2003.

[11] B. Fabian, "Implementing Secure P2P-ONS," in Proceedings IEEE In- ternational Conference on Communications (IEEE ICC 2009), Dresden, 2009.

[12] N. Scho¨nemann, K. Fischbach, and D. Schoder, “P2P Architecture for Ubiquitous Supply Chains,” in 17th European Conference on Informa- tion Systems (ECIS’09), Verona, Italy, 2009.

[13] ISO, “ISO/IEC TR 9126-1:2001 – Software Engineering – Product Quality,” 2001.

[14] M. Cai, M. Frank, J. Chen, and P. Szekely, “MAAN: A Multi-Attribute Addressable Network for grid information services”, Journal of Grid Computing, vol. 2, no. 1, pp. 3 – 14, 2004.

[15] I. Stoica, R. Morris, D. Karger, F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. In Proceedings of ACM SIGCOMM 2001, 2001.

[16] A. Bharambe, M. Agrawal, and S. Seshan. Mercury: Supporting Scalable Multi-Attribute Range Queries. In SIGCOMM, Portland, OR, August 2004.

[17] Macedonia, M. R., Zyda, M. J., Pratt, D. R., Brutzman, D. P., and Braham, P. T. Exploiting reality with multicast groups: A network architecture for large-scale virtual environments. In Proc. of the 1995 IEEE Virtual Reality Symposium (VRAIS95) (Mar. 1995).

[18] H. Shen, C. Xu, and G. Chen, “Cycloid: A Scalable Constant- Degree P2P Overlay Network,” Performance Evaluation, vol. 63, no. 3, pp. 195-216, 2006.

[19] Shen H, Xu C. Leveraging a compound graph-based dht for multi-attribute range queries with performance analysis. IEEE Transactions on Computers, 2012, 61(4): 433–447

[20] Rowstron, Antony, and Peter Druschel. "Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems." Middleware 2001. Springer Berlin Heidelberg, 2001.

[21] Stoica, Ion, et al. "Chord: A scalable peer-to-peer lookup service for internet applications." ACM SIGCOMM Computer Communication Review 31.4 (2001): 149-160.

[22] L.Rizzo, Dummynet: a simple approach to the evaluation of network protocols, ACM SIGCOMM Computer Communication Review 27 (1), 31-41, 1997

[23] Atzori, Luigi, Antonio Iera, and Giacomo Morabito. "The internet of things: A survey." Computer networks 54.15 (2010): 2787-2805.

[24] M. Presser, A. Gluhak, The Internet of Things: Connecting the Real World with the Digital World, EURESCOM mess@ge – The Magazine for Telecom Insiders, vol. 2, 2009, <http://www.eurescom.eu/message>.

[25] I. Toma, E. Simperl, Graham Hench, A joint roadmap for semantic technologies and the internet of things, in: Proceedings of the Third STI Roadmapping Workshop, Crete, Greece, June 2009.

[26] A. Katasonov, O. Kaykova, O. Khriyenko, S. Nikitin, V. Terziyan, Smart semantic middleware for the internet of things, in: Proceedings of the Fifth International Conference on Informatics in Control, Automation and Robotics, Funchal, Madeira, Portugal, May 2008.

[27] W. Wahlster, Web 3.0: Semantic Technologies for the Internet of Services and of Things, Lecture at the 2008 Dresden Future Forum, June 2008.

[28] I. Vázquez, Social Devices: Semantic Technology for the Internet of Things, Week@ESI, Zamudio, Spain, June 2009.

[29] Y.-W. Ma, C.-F. Lai, Y.-M. Huang, J.-L. Chen, Mobile RFID with IPv6 for phone services, in: Proceedings of IEEE ISCE 2009, Kyoto, Japan, May 2009.

[30] S.-D. Lee, M.-K. Shin, H.-J. Kim, EPC vs. IPv6 mapping mechanism, in: Proceedings of Ninth International Conference on Advanced Communication Technology, Phoenix Park, South Korea, February 2007.

[31] I.F. Akyildiz, J. Xie, S. Mohanty, A survey on mobility management in next generation All-IP based wireless systems, IEEE Wireless Communications Magazine 11 (4) (2004) 16–28.

[32] M. Mealling, Auto-ID Object Name Service (ONS) v1.0, Auto-ID Center Working Draft, August 2003.

[33] V. Krylov, A. Logvinov, D. Ponomarev, EPC Object Code Mapping Service Software Architecture: Web Approach, MERA Networks Publications, 2008.

[34] L. Eschenauer, V.D. Gligor, A key-management scheme for distributed sensor networks, in: Proceedings of the Ninth ACM Conference on Computer and Communications Security, Washington, DC, USA, November 2002.

[35] R. Acharya, K. Asha, Data integrity and intrusion detection in wireless sensor networks, in: Proceedings of IEEE ICON 2008, New Delhi, India, December 2008.

[36] H. Krawczyk, M. Bellare, R. Canetti, HMAC: Keyed-Hashing for Message Authentication, IETF RFC 2104, February 1997.

[37] G.V. Lioudakis, E.A. Koutsoloukas, N. Dellas, S. Kapellaki, G.N. Prezerakos, D.I. Kaklamani, I.S. Venieris, A proxy for privacy: the discreet box, in: EUROCON 2007, Warsaw, Poland, September 2007.

[38] Richardson, Leonard, and Sam Ruby. RESTful web services. " O'Reilly Media, Inc.", 2008.

[39]