**Politecnico di Milano**

DIPARTIMENTO DI ELETTRONICA ED INFORMAZIONE
Corso di Laurea Specialistica in Ingegneria delle Telecomunicazioni

Tesi di Laurea Specialistica

# Distribuzione ottima dei contenuti in reti veicolari con infrastruttura

Candidato:
**Fulvio Saviori**
**Matricola 750481**

Relatore:
**Ing.Matteo Cesana**

**Anno Accademico 2010–2011**

# Indice

**Introduzione** **xi**

**1 Reti Veicolari** **1**
1.1 VANET . . . . . 1
1.2 Gli scenari applicativi . . . . . 1
1.3 Caratteristiche . . . . . 5
1.4 Problematiche e sfide tecnologiche . . . . . 7

**2 Soluzioni di *caching*** **13**
2.1 La *cache* e il suo funzionamento . . . . . 13
2.2 Tecnologie *cache-based* . . . . . 15
2.2.1 *Web caching* . . . . . 15
2.2.2 *Caching* nelle reti ad hoc . . . . . 16
2.2.3 Altri ambiti di utilizzo del *caching* . . . . . 17
2.3 Il *caching* nelle infrastrutture . . . . . 18
2.4 *Cache Management* . . . . . 22
2.4.1 *Replacement Policy* . . . . . 22
2.4.2 *Scheduling* . . . . . 24
2.5 Distribuzione contenuti in VANET: altri approcci . . . . . 26

**3 Un Modello per la distribuzione dei contenuti** **29**
3.1 Scenario . . . . . 29
3.2 Il modello . . . . . 31
3.2.1 Indici e Parametri . . . . . 32
3.2.2 Variabili . . . . . 33
3.2.3 Vincoli . . . . . 35
3.2.4 Variante . . . . . 37
3.3 Contributi innovativi . . . . . 37

**4 Valutazione delle prestazioni** **43**
4.1 Scenario Simulativo . . . . . 43
4.2 Analisi . . . . . 43
4.2.1 *BandaTot* e *Cap*: due parametri limitanti . . . . . 43
4.2.2 Local Dependent Vs Local Independent . . . . . 46
4.2.3 Pochi Contatti Vs Molti Contatti . . . . . 47
4.2.4 Distribuzioni *Query* e *Size* . . . . . 50
4.2.5 Distribuzioni della frequenza dei Percorsi . . . . . 58
4.2.6 Interferenze nei *File Server* . . . . . 60
4.2.7 Sul modello alternativo . . . . . 64

**5 Conclusioni e sviluppi futuri** **69**

**Ringraziamenti** **71**

**Bibliografia** **73**

# Elenco delle figure


1.1 I protagonisti in una rete veicolare. . . . . 2
1.2 *Cluster-based* routing. . . . . 8
1.3 Differenti scenari di comunicazione. . . . . 9

2.1 Motivazione della nascita del *Web caching*. . . . . 15
2.2 *Cooperative caching* in ad-hoc *networks*. . . . . 16
2.3 Recupero file. . . . . 18
2.4 File download. . . . . 20
2.5 *Content Manager*. . . . . 21
2.6 *Directional Weight*. . . . . 26

3.1 Le tipologie di architetture *V.A.Nets*. . . . . 29
3.2 Concetto di *cache* in *V.A.Nets*. . . . . 30

4.1 Probabilità di recupero al variare del numero di utenti $N$ per diversi valori di banda totale disponibile, $BandaTot$. . . . . 45
4.2 Probabilità di recupero al variare del numero di utenti $N$ per diversi valori della capacità di *storage* dei *file server*, $Cap$. . . 46
4.3 Probabilità di recupero al variare del numero di utenti $N$, casi *Local Dependent* e *Local Independent*. . . . . 47
4.4 Schemi matrice contatti con diverso numero di contatti, uguali per ogni percorso. . . . . 48
4.5 Probabilità di recupero al variare del numero di contatti, uguali per ogni percorso. . . . . 49
4.6 Probabilità di richiesta *query* dei file per tre diversi casi in base alla dimensione *size* dei file, primo approccio. . . . . 50
4.7 Probabilità di recupero al variare della capacità di *storage* dei *file server* per i tre casi precedenti e per due diverse configurazioni di *size* dei file, primo approccio. . . . . 51

4.8 Percentuali medie di un file immagazzinato nei *file server* al variare della capacità di *storage* per ogni combinazione (caso-*size*), primo approccio. . . . . 52
4.9 Percentuali, in base alla *size* dei file, del contenuto del *file server* al variare della sua capacità di *storage* per i tre casi, primo approccio. . . . . 53
4.10 Numero di file presenti nel *file server* al variare della sua capacità di *storage* per i tre casi e per le *size* dei file, primo approccio. . . . . 54
4.11 Tre diverse distribuzioni (casi) di probabilità di richiesta *query* dei file uguali per ogni *size*, secondo approccio. . . . . 55
4.12 Probabilità di recupero al variare della capacità di *storage* dei *file server* per i tre casi precedenti e per due diverse configurazioni di *size* dei file, secondo approccio. . . . . 56
4.13 Percentuali medie di un file immagazzinato nei *file server* al variare della capacità di *storage* per ogni combinazione (caso-*size*), secondo approccio. . . . . 57
4.14 Percentuali, in base alla *size* dei file, del contenuto del *file server* al variare della sua capacità di *storage* per i tre casi, secondo approccio. . . . . 58
4.15 Numero di file presenti nel *file server* al variare della sua capacità di *storage* per i tre casi e per le *size* dei file, secondo approccio. . . . . 59
4.16 Cinque diverse distribuzioni (casi) di frequenze dei percorsi $P(v)$. 60
4.17 Probabilità di recupero al variare della capacità di *storage* dei *file server* per le cinque distribuzioni. . . . . 61
4.18 Probabilità di recupero al variare della capacità di *storage* dei *file server* per le cinque distribuzioni, con banda più alta. . . . 61
4.19 Schema matrice contatti con numero di contatti per *file server* crescente. . . . . 62
4.20 Percentuali medie di un file immagazzinato nel *file server* al variare della capacità di *storage* per i cinque *file server* e per le *size* dei file. . . . . 63
4.21 Numero dei file presenti nel *file server* al variare della capacità di *storage* per i cinque *file server* e per le *size* dei file. . . . . 64
4.22 Contenuto dei cinque *file server*, diviso in base alla *size* dei file presenti, al variare della capacità di *storage*. . . . . 65
4.23 Somma di tutte le capacità di *storage* dei *file server*, al variare del numero di utenti $N$, necessarie per garantire diversi livelli di efficienza. . . . . 66

4.24 Distribuzione in base alla *size* della quantità necessaria di file immagazzinati nei cinque *file server* al variare del numero di utenti $N$ per garantire un livello fissato di efficienza. . . . . 67
4.25 Capacità di *storage* dei cinque *file server*, in relazione alla somma *Cap Tot*, al variare del numero di utenti $N$, necessarie per garantire un livello fissato di efficienza. . . . . 67

# Elenco delle tabelle


1.1 Applicazioni di Sicurezza attiva. . . . . 3
1.2 Applicazioni di Pubblica sicurezza. . . . . 4
1.3 Applicazioni di Ausilio alla guida. . . . . 4
1.4 Applicazioni di Business/ *Entertainment*. . . . . 4

2.1 Politiche di *Cache Management*. . . . . 24

3.1 Indici del modello. . . . . 32
3.2 Parametri del modello. . . . . 33
3.3 Variabili del modello. . . . . 34
3.4 Il modello matematico. . . . . 36
3.5 Il modello alternativo. . . . . 38

4.1 *BandaTot* - Configurazioni. . . . . 44
4.2 *Cap* - Configurazioni. . . . . 45

# Introduzione

Oggigiorno l'automobile rappresenta il principale mezzo di trasporto, utilizzato in tutto il mondo da milioni di persone nella loro quotidianità. La grande diffusione di questo mezzo, però, porta spesso ad avere delle situazioni di traffico intenso sulle strade e di conseguenza si vengono a creare delle condizioni di pericolo a causa di fattori quali imprevisti, alte velocità, errori umani, che rischiano di aumentare in modo considerevole la probabilità di incidenti stradali.

Per queste ragioni, uno dei principali obiettivi della ricerca del settore è stato fin da subito quello di provare ad assistere il conducente di un veicolo attraverso la realizzazione di tecnologie che possano incrementare il livello di sicurezza degli occupanti dell'auto durante gli spostamenti. Con la diffusione poi di sistemi di comunicazione wireless e di localizzazione geografica come il GPS lo scenario applicativo si è ulteriormente ampliato. Nasce allora l'idea di sfruttare la cooperazione tra veicoli per creare delle vere e proprie reti veicolari. Questa tecnologia si basa su collegamenti wireless e permette ad un veicolo di comunicare sia con altri veicoli (V2V) che con delle infrastrutture fisse (V2I). Ovviamente, la questione non è cosi diretta ed immediata, in quanto le reti veicolari hanno delle caratteristiche e dei requisiti unici, che creano non solo delle sfide, ma anche delle opportunità del tutto nuove, spingendo la ricerca a formare delle comunità con un focus esclusivo sull'argomento. Ulteriori fattori, come le difficoltà di creare uno standard uguale per tutti o la mancanza di una vera applicazione "killer", non hanno ancora permesso la proliferazione di questa tecnologia.

Si è pensato allora di utilizzare la rete per la distribuzione di contenuti perché quest'ultima funzionalità può rappresentare il business per gli operatori delle telecomunicazioni e quindi l'investimento iniziale per la diffusione della tecnologia. Queste applicazioni, l'accesso a internet in primis, richiedono però il collegamento con infrastrutture fisse (*Road Side Unit*), dunque bisognerà pensare a creare una buona copertura di rete ed un sistema di distribuzione dei contenuti, unico nel suo genere, per garantire una buona qualità del servizio.

Il presente lavoro di tesi analizza il problema della distribuzione dei

contenuti in reti veicolari. In particolare, la tesi studia il posizionamento ottimo (*caching*) dei contenuti presso *Road Side Unit*, che cioè massimizzi la disponibilità dei contenuti stessi all'utente finale.

Il problema del *caching* ottimo è rappresentato con un modello di programmazione matematica. A partire da tale modello, la tesi propone un'analisi delle prestazioni del *caching* ottimo al variare della mobilità dei veicoli, della distribuzione di interesse per i contenuti e della capacità di *storage* degli RSU.

Il **Capitolo 1** introduce le reti veicolari dandone una descrizione generale al fine di familiarizzare con i concetti di base. Vengono presentati i possibili scenari applicativi, le caratteristiche peculiari e alcune delle principali sfide tecnologiche su problematiche che ancora oggi sono alla ricerca della miglior soluzione.

Nel **Capitolo 2** si approfondisce l'argomento cardine della tesi. Viene mostrato il panorama attuale delle soluzioni al problema della distribuzione dei contenuti nelle reti veicolari.

Nel **Capitolo 3** viene descritto il modello di ottimizzazione proposto mettendo in evidenza le differenze rispetto allo stato dell'arte.

Nel **Capitolo 4** vengono mostrate nostre ulteriori analisi allo scopo di trovare le caratteristiche e le migliori impostazioni per una possibile attuazione in uno scenario realistico.

Il **Capitolo 5** conclude la tesi commentando i risultati ottenuti e i possibili sviluppi futuri.

# Capitolo 1

# Reti Veicolari

In questo capitolo viene fornita una descrizione generale delle reti veicolari, indicando alcune problematiche più comuni a questo tipo di tecnologia. Si concentra l'attenzione sui possibili scenari applicativi e sulle attuali sfide tecnologiche.

## 1.1 VANET

Il termine VANET sta per *Vehicular Ad-hoc NETwork* e indica una recente tipologia di rete wireless che sfrutta i veicoli come nodi occasionali. La topologia della rete non è statica ma è in continua evoluzione, a causa della mobilità dei suoi nodi. Un veicolo per essere considerato un nodo della rete, deve essere dotato di una OBU (*On Board Unit*) in grado di comunicare con altre OBU o con infrastrutture statiche. Le antenne fisse, dislocate lungo il reticolo stradale sono dette RSU (*Road Side Unit*). Oltre a queste, i veicoli sono in grado di sfruttare nel caso altre tecnologie wireless, ad esempio le attuali reti cellulari. Esistono quindi due tipologie di VANET che spesso interagiscono tra loro: V2V e V2I. V2V sta per *Vehicle-to-Vehicle* e riguarda la comunicazione diretta tra veicoli. V2I sta per *Vehicle-to-Infrastructure* e riguarda il collegamento tra veicoli e infrastruttura fissa (Figura 1.1). Il libro [19], l'articolo di *survey* [20] e il seminario [21] offrono una completa descrizione di questa tecnologia.

## 1.2 Gli scenari applicativi

Lo scopo primario delle VANET è stato inizialmente lo sviluppo di applicazioni distribuite e pubbliche orientate alla sicurezza stradale, allo scopo di salvare vite umane, e migliorare le condizioni del traffico, riducendo di

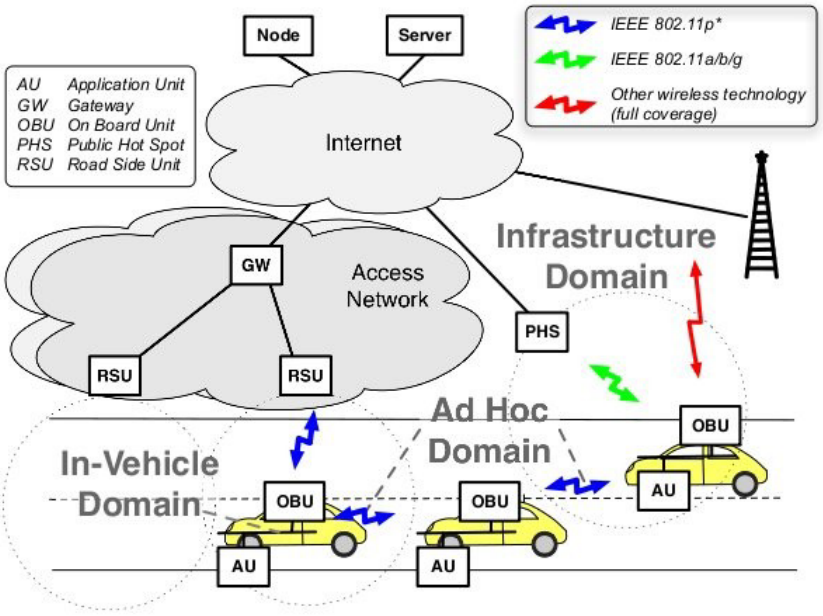



Figura 1.1: I protagonisti in una rete veicolare.

conseguenza anche l'impatto ambientale. Spesso le cose vanno di pari passo perché evitando incidenti si evitano ingorghi e di conseguenza si riduce l'inquinamento. Inoltre una migliore efficienza della mobilità si tramuta in risparmio energetico e di tempo, ossia in una migliore qualità della vita. Allo stesso tempo deve essere possibile però lo sviluppo di servizi privati in grado di finanziarne i costi di sviluppo, e/o incoraggiare la loro rapida adozione grazie al valore aggiunto percepito. Si sono aggiunte quindi applicazioni per la distribuzione di informazioni e contenuti di intrattenimento. Possiamo allora distinguere le applicazioni in 4 macro categorie: Sicurezza attiva, Servizi di pubblica sicurezza, Ausilio alla guida, Business/ *Entertainment*.

Nelle prime di solito viene inviato in *broadcast* da un veicolo a tutti i veicoli vicini un messaggio d'allerta che notifica un determinato evento, come una collisione o cattive condizioni stradali, in modo da diminuire il numero di incidenti e migliorare il controllo del flusso stradale. Le situazioni tipo possono essere quindi: allarmi per condizioni stradali pericolose, allarmi per condizioni stradali anomale, allarmi di pericolo di collisione, incidente imminente, incidente avvenuto (Tabella 1.1). Nelle seconde, molto simili, abbiamo invece: allarmi di risposta all'emergenza, supporto ai controlli stradali (Tabella 1.2).

Le terze poi sono applicazioni per l'efficienza del trasporto e troviamo ad esempio: navigazione che tenga conto di ingorghi o incidenti, segnalazione di

parcheggi liberi nelle vicinanze, calcolo della velocità ottimale per prendere la cosiddetta "onda verde" (Tabella 1.3). Lo scambio di informazioni, a seconda dell'applicazione, avviene solamente trai veicoli oppure necessita del collegamento con un'infrastruttura fissa che funge da coordinatore.

Infine l'ultima categoria che riguarda la distribuzione di informazioni e contenuti di intrattenimento. Gli esempi possono essere numerosissimi. Si spazia da applicazioni molto utili quali la diagnosi remota del veicolo in tempo reale o la notifica di punti di interesse (benzinai, officine, ristoranti, ecc.), ad applicazioni di puro intrattenimento quali collegamento a internet o download di file a pagamento (Tabella 1.4). Queste possibili funzionalità sono quelle più appetibili per gli utenti e quindi, come già accennato, possono rappresentare un business per gli operatori nel campo delle telecomunicazioni, oltre che la leva per la proliferazione delle VANET. La maggior parte di queste applicazioni, a differenza di quelle per la sicurezza, richiede il collegamento con un RSU ed è quindi relativamente facile per un operatore creare un business per la distribuzione di contenuti, come ad esempio file *mp3*. E' sufficiente dislocare sul territorio un buon numero di RSU, creando un'area di copertura abbastanza estesa, creare un database dotato di numerosi titoli di diversi generi musicali raggiungibile dai vari RSU e studiare un metodo di pagamento per il download dei file. Per ora il grosso ostacolo è rappresentato dal bassissimo numero di veicoli dotati di OBU. Se in futuro tale problema verrà risolto, gli operatori che cercheranno di entrare in questo nuovo business saranno molti e per essi sarà fondamentale studiare attentamente i parametri di traffico e di interferenza per avviare l'attività in maniera efficace. E' in questo ultimo ambito che si inserisce il lavoro della nostra tesi.

| Situazione | Esempi |
|---|---|
| Allarmi condizioni stradali pericolose | Limite di velocità in curva; passaggio con il rosso; limite altezza sottopasso; mancato rispetto dello stop. |
| Allarmi condizioni stradali anomale | Segnalazione di strada dissestata; visibilità limitata; lavori in corso. |
| Allarmi pericolo collisione | Zona cieca del retrovisore; cambio di corsia; incrocio; veicolo lento anteriore; segnalazione di frenata; intersezione tram; attraversamento pedonale. |
| Incidente imminente | Predisposizione alla riduzione del danno. |
| Incidente avvenuto | Avviso e SOS automatico. |

Tabella 1.1: Applicazioni di Sicurezza attiva.

| Situazione | Esempi |
|---|---|
| Allarmi risposta emergenza | Veicolo in servizio di emergenza in arrivo; richiesta di precedenza per veicolo di servizio; veicolo di servizio sul posto. |
| Supporto controlli stradali | Patente e libretto elettronici; verifica operatività impianto di sicurezza; tracciamento veicoli rubati. |

Tabella 1.2: Applicazioni di Pubblica sicurezza.

| Situazione | Esempi |
|---|---|
| Guida facilitata | Assistenza presso rampe autostradali; assistenza svolta a sinistra; controllo crociera cooperativo ed adattativo; controllo cooperativo degli abbaglianti; ripetizione segnaletica a bordo; gestione adattiva della trazione. |
| Gestione traffico | Notifica di incidente o condizioni stradali ad un centro operativo; controllo intelligente del flusso di traffico; ausilio al tragitto ed al percorso; aggiornamento mappe; servizio individuazione parcheggio. |

Tabella 1.3: Applicazioni di Ausilio alla guida.

| Situazione | Esempi |
|---|---|
| Manutenzione veicolo | Diagnostica remota; aggiornamenti software; promemoria controlli di sicurezza; notifiche di guasto imminente. |
| Servizi turistici e mobilità | Accesso ad Internet; messaggistica istantanea; scaricamento mappe e audioguide; notifica di punti di interesse; pianificazione di itinerari. |
| Soluzioni per l'impresa | Gestione flotte; gestione auto a nolo; controllo di accesso geografico; tracciamento di materiale pericoloso. |
| Pagamento elettronico | Esazione di tariffe; pagamento parcheggio; pagamento rifornimento benzina. |

Tabella 1.4: Applicazioni di Business/ *Entertainment*.

## 1.3 Caratteristiche

Di seguito vengono mostrate le proprietà che caratterizzano una rete veicolare. Ricordiamo infatti che è una tecnologia completamente indipendente con delle caratteristiche uniche e necessita quindi di uno studio specifico.

### Capacità di *processing*, energetica e di comunicazione

Diversamente dall'ambito "mobile" in cui il vincolo dell'energia rappresenta uno dei limiti più rilevanti (pensiamo ad esempio ai piccoli dispositivi palmari), i veicoli nelle VANET non presentano alcuna restrizione, non solo nella capacità di *storage* dell'energia (possono usare infatti le batterie del veicolo stesso), ma anche in quella di calcolo e *processing* delle informazioni, supportando numerose interfacce di comunicazione. L'assenza di problematiche energetiche dei nodi e la possibilità di immagazzinare dati in dispositivi di massa installati sui veicoli quindi fanno sì che si possono creare algoritmi focalizzati alle performance e non al consumo energetico.

### Diversa mobilità in diversi ambienti

Il movimento dei veicoli risulta legato all'infrastruttura stradale, che può andare dalla rete autostradale sino ad un qualsiasi centro urbano. Diversi scenari hanno diverse caratteristiche di mobilità:

- rete cittadina caratterizzata da bassa velocità e un'elevata densità di veicoli;
- rete autostradale che si distingue per un movimento lineare e unidirezionale e una topologia semi-stabile;
- rete rurale che ha tragitti errabondi e una bassissima densità di veicoli.

Inoltre ostacoli strutturali come costruzioni abitative, ponti, tralicci, tutti elementi parte di questi tipi di ambienti, impongono vincoli sulla propagazione radio, in quanto causano effetti come il *multipath* o il *fading*, che incidono considerevolmente sul modello di mobilità e sulla qualità della trasmissione radio.

### Topologia altamente dinamica

Grazie alla elevata velocità di movimento tra veicoli, la topologia nelle VANET è sempre in evoluzione. Un elemento, infatti, può associarsi alla rete o abbandonarla in un intervallo di tempo molto ristretto, generando rapidi

mutamenti nella topologia di rete. Per esempio, supponiamo che il raggio di trasmissione wireless di ciascun veicolo sia 250 metri, in modo che vi è un collegamento tra due vetture solo se la distanza tra loro è inferiore a 250 metri. Nel caso peggiore di due auto con una velocità di circa 90 km/h in direzioni opposte, il collegamento durerà solo fino a 10 secondi. La connessione tra i veicoli rappresenta quindi un parametro chiave nella progettazione di una rete veicolare, soprattutto nel design dei protocolli di routing.

### Disconnessioni frequenti

Oltre alla topologia, anche la connettività nelle VANET potrebbe cambiare spesso. Specialmente quando la densità dei veicoli è bassa, si ha una maggiore probabilità di disconnessioni dalla rete. In alcune applicazioni, come l'accesso a Internet, il problema deve essere risolto. Una possibile soluzione è quella di installare diversi nodi di ripetizione o *Access Point* lungo la strada per mantenere la connettività.

### Modellazione della mobilità

A causa del movimento altamente mobile dei nodi e della topologia dinamica, un modello e una previsione della mobilità svolgono un ruolo importante nel design del protocollo di rete. Inoltre, i nodi veicolari vengono solitamente vincolati dalle strade. Così, data la velocità e la mappa stradale, la futura posizione del veicolo può essere prevista.

### Rigidi vincoli di ritardo

Nelle applicazioni di sicurezza, la rete non richiede bande elevate ma ha dei rigidi vincoli di ritardo. Ad esempio in uno scenario autostradale, quando avviene una frenata improvvisa, il messaggio deve arrivare alle auto vicine in un tempo breve per evitare un incidente automobilistico. In questo tipo di applicazioni primarie, invece del ritardo medio, sarà cruciale quindi il ritardo massimo.

### Comunicazione geografica

I veicoli, grazie all'uso di sistemi di posizionamento come il GPS e grazie all'uso di mappe dettagliate dei territori, hanno una buona conoscenza del territorio circostante. Rispetto ad altre reti che utilizzano *unicast* o *multicast*, in cui gli *end point* sono definiti da ID o gruppo ID, si sfrutta spesso un nuovo tipo di comunicazione che indirizza direttamente ad aree geografiche.

## 1.4 Problematiche e sfide tecnologiche

**Scarsa proliferazione**

Molti studi e molte aree di ricerca gravitano intorno alle reti veicolari e molti risultati soddisfacenti sono stati raggiunti. Ad oggi il più grande ostacolo rimane la scarsa penetrazione. Infatti il servizio ricevuto da un utente migliora all'aumentare del numero di veicoli equipaggiati di una OBU. La sfida è cercare di attirare gli utenti verso questa nuova tecnologia. Da questo punto di vista, possono ricoprire un ruolo chiave le applicazioni per la distribuzione di informazioni e di contenuti di intrattenimento. L'utente infatti può essere molto attratto da un'applicazione che permetta di conoscere in tempo reale e in maniera del tutto automatica informazioni su ristoranti nelle vicinanze o che permetta di scaricare file *mp3* da ascoltare durante il viaggio. Il concetto è quindi sfruttare queste applicazioni per attirare più utenti possibile, in maniera tale da aumentare la penetrazione delle VANET e quindi favorire anche la fruizione di applicazioni con finalità non ludiche quali la sicurezza e l'efficienza del trasporto. E' in questo senso che la ricerca in questo ambito è da ritenersi di fondamentale importanza.

**Routing**

Oltre a questo importante aspetto socio-economico, ci sono questioni di carattere puramente tecnico da studiare. In particolare la topologia dinamica della rete dovuta alla mobilità dei veicoli apre diverse questioni da affrontare. Una di queste riguarda i protocolli di routing che devono essere in grado di supportare la mobilità dei diversi nodi e disseminare l'informazione in maniera efficiente a seconda dell'applicazione. La maggior parte di questi infatti sono stati progettati per gestire una condizione speciale o un particolare problema. Ad esempio servizi per la sicurezza stradale sono basati su comunicazioni di tipo *one-hop broadcast* con messaggi sia periodici, di aggiornamento sulla posizione dei veicoli vicini, sia occasionali per la segnalazione di pericolo. In [8] vengono presentati diversi protocolli di routing classificati in cinque categorie: *ad-hoc*, *position-based*, *cluster-based*, *broadcast* e *geocast* routing (Figura 1.3). Nei primi si è provato a partire dall'idea di adattare protocolli come AODV (*Ad-hoc On demand Distance Vector*) e DSR (*Source Routing Dinamico*), utilizzati nelle reti ad-hoc mobili (MANET), per caratteristiche molto simili, come l'auto-organizzazione, l'autogestione senza l'aiuto di infrastrutture fisse. A differenza di AODV e DSR, che utilizzano informazioni sui link esistenti tra i nodi della rete per effettuare il routing (*topology based*), nei secondi abbiamo strategie di routing, come GPSR (*Greedy Perimeter Stateless Routing*) e GPCR (*Greedy Perimeter Coordinator Routing*), che decidono in base a

informazioni sulla posizione geografica, ricavate dalle mappe delle strade, dai modelli di traffico o più semplicemente dai sistemi di navigazione a bordo. Nei terzi ci sono invece tentativi di creare un'infrastruttura di rete virtuale attraverso il raggruppamento di nodi per fornire scalabilità. Ciascun *cluster* può avere un *head-cluster* responsabile dell'intra ed inter coordinamento per la gestione della rete (Figura 1.2). Nella quarta categoria troviamo poi quei protocolli utilizzati sia per condividere le informazioni, come ad esempio le condizioni stradali, che per trovare il miglior percorso verso la destinazione (fase di *routing discovery*). Infine nell'ultima categoria abbiamo tutti quei protocolli in grado di recapitare informazioni da un nodo sorgente ad un insieme di nodi in una specifica regione geografica, ZOR (*Zone Of Relevance*). Concludendo sull'argomento possiamo dire che l'area di ricerca è molto attiva e nuovi protocolli di routing vengono creati in continuazione.

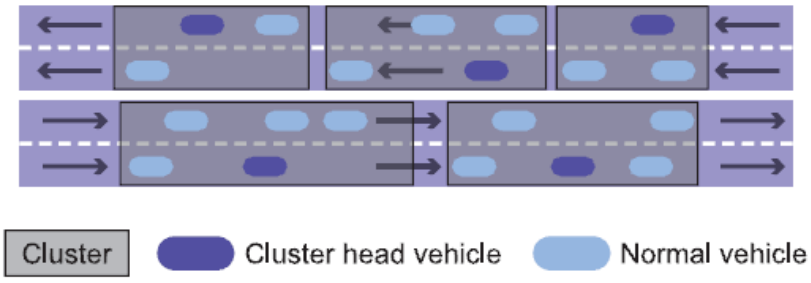



Figura 1.2: *Cluster-based* routing.

### Accesso

Un altro aspetto problematico è rappresentato dalla mancanza di un coordinatore centrale per quanto riguarda l'accesso nelle reti V2V. La natura *broadcast* della maggior parte dei messaggi e la topologia dinamica della rete rendono necessario l'utilizzo di un canale di controllo condiviso. Per questo motivo il protocollo MAC utilizzato è un altro concetto chiave nella progettazione delle reti veicolari. In particolare sono di fondamentale importanza i seguenti parametri: probabilità di corretta ricezione, tempo di accesso al canale, controllo di congestione, robustezza rispetto al *fading*. Un buon protocollo MAC deve offrire garanzie per quanto riguarda tutti questi parametri. In [22] viene descritto in maniera approfondita 802.11p (Livelli Fisico e MAC). Questo sembra essere ormai lo standard definitivo per questo tipo di reti. Descriviamo quindi le sue principali caratteristiche.

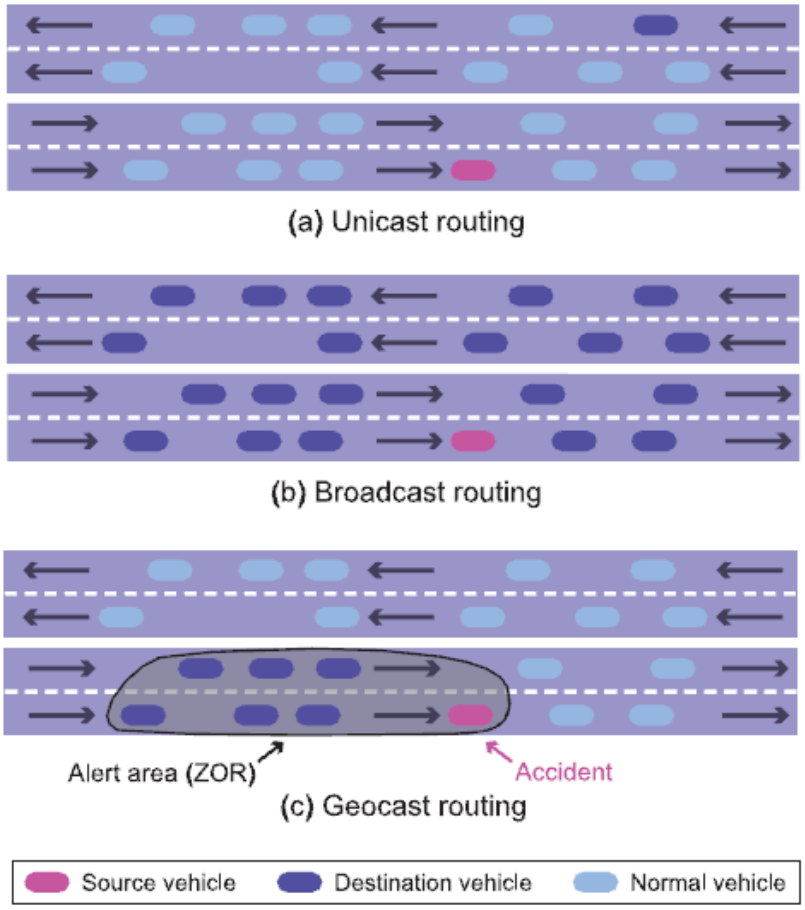



Figura 1.3: Differenti scenari di comunicazione.

A livello fisico la filosofia di 802.11p è stata quella di apportare cambiamenti minimi rispetto alla già esistente 802.11 PHY in modo da ottimizzare quest'ultima per l'utilizzo nelle reti veicolari. La banda utilizzata è 5.9 Ghz e il sistema è essenzialmente basato sulla tecnica di modulazione OFDM con una larghezza di banda del canale di 10 MHz anziché i 20 MHz di 802.11a, per aumentare la tolleranza dell'effetto di propagazione del segnale *multi-path*. Per poter poi supportare un raggio di comunicazione più ampio possibile, sono stati autorizzati valori di EIRP fino ad un massimo di 44.8 dBm (30W).

Per quanto riguarda invece il livello MAC l'idea proposta da IEEE802.11p è di riuscire a creare una comunicazione molto efficiente senza il pesante *overhead* tipicamente presente in 802.11 MAC, a vantaggio della velocità di trasmissione. Le operazioni di autenticazione e associazione infatti sono troppo onerose per permettere alle comunicazioni tra veicoli, soprattutto in caso di comunicazioni di sicurezza, di scambiare dati rapidamente. Pertanto è essenziale per tutti i dispositivi essere settati sullo stesso canale e configurati con lo stesso indirizzo MAC (BSSID).

La novità introdotta da IEEE802.11p è una diversa modalità di configurazione: WAVE. L'idea principale è che una stazione in modalità WAVE può inviare e ricevere pacchetti dati con indirizzo speciale BSSID indipendentemente dal fatto di essere o meno membro della stessa WBSS. Una WBSS è un tipo di BSS che consiste in una serie di stazioni (veicoli) che cooperano tra loro in modalità WAVE e che comunicano utilizzando uno stesso indirizzo BSSID.

Una WBSS è inizializzata quando una stazione in modalità WAVE invia un WAVE *beacon* nel quale include tutte le informazioni necessarie perché il ricevitore si configuri alla WBSS. Un stazione entra a far parte di una WBSS quando è configurata per inviare e ricevere pacchetti con indirizzo BSSID definiti per quella WBSS. Viceversa cessa di far parte di una WBSS quando il livello MAC interrompe la ricezione e l'invio di pacchetti dati contenenti l'indirizzo BSSID di quella WBSS. Una stazione non può essere membro di più di una WBSS contemporaneamente.

Una WBSS cessa di esistere quando non ci sono membri che la compongono. Inoltre la prima unità che richiede la creazione della WBSS invia un messaggio che è uguale a quello che utilizzano poi tutti gli altri veicoli, quindi, quando questa unità si dissocia, la WBSS continua ad esistere. Nelle applicazioni per la sicurezza è supportato da tutte le stazioni l'utilizzo dell'indirizzo speciale BSSID (*wildcard* BSSID). Questo significa che se un veicolo facente parte di una WBSS deve trasmettere una segnalazione per la sicurezza, può inviare il pacchetto con l'indirizzo speciale in modo che questo sia ricevuto da tutti i veicoli nei dintorni, anche se non compresi nella sua WBSS. Allo stesso modo veicoli che ricevono pacchetti con l'indirizzo speciale BSSID possono

utilizzarlo anche se non facenti parte delle WBSS della stazione trasmittente. I pacchetti che possono essere trasmessi anche alle unità al di fuori della propria WBSS vengono ricevuti solo se contengono l'indirizzo speciale BSSID perché, in genere, ogni stazione a livello MAC filtra i pacchetti ricevuti conservando solamente quelli provenienti dalla stessa WBSS a cui appartiene la stazione ricevente. Notiamo quindi che questa procedura di interazione è complessivamente molto veloce, perché permette di creare una connessione stabile tra più unità semplicemente con lo scambio di un pacchetto (*beacon*) di richiesta, e allo stesso tempo garantisce il servizio di applicazioni con diversi requisiti.

### Qualità del servizio

Protocolli di routing e protocolli MAC utilizzati influenzano in maniera diretta la QoS, la qualità del servizio percepita dall'utente. Non si devono quindi dimenticare tutti quegli studi che più in generale si interessano dell'impatto complessivo delle prestazioni. Ad esempio in [23] vengono condotte delle simulazioni in scenario urbano e autostradale al fine di analizzare le quattro metriche più indicative per valutare la QoS: durata della connessione, ritardo, variazione del ritardo (*jitter*) e PDR (*Packet Delivery Ratio*). I risultati hanno indicato che la durata della connessione e il PDR sono i parametri più critici, che possono creare problemi a seconda dello scenario considerato, della densità e della velocità dei veicoli presenti.

# Capitolo 2

# Distribuzione dei contenuti: soluzioni di *caching*

In questo capitolo viene proposto l'attuale panorama delle tecnologie basate su *cache*. Vogliamo capirne differenze e caratteristiche principali per poi mostrare come queste hanno contribuito a suggerirne una che, secondo noi, si adatta meglio per la *Content Distribution* nelle reti veicolari. Successivamente, invece, spaziamo su ogni tipo di tentativo recente volto ad affrontare più in generale lo stesso problema: la distribuzione dei contenuti in reti veicolari.

## 2.1 La *cache* e il suo funzionamento

Prima di partire con una carrellata delle diverse tecnologie spieghiamo inizialmente cosa le accomuna. La *cache*, o meglio la memoria *cache*, è in generale una memoria temporanea, non visibile al software, che memorizza un insieme di dati che successivamente possono essere velocemente recuperati su richiesta [10]. L'origine del nome deriva appunto dal fatto che la memoria *cache* ed il suo utilizzo sono trasparenti al programmatore, quindi "nascosti" allo stesso. Una *cache* è associata ad una memoria "principale", in cui risiedono i dati. Essa è tipicamente di capienza inferiore rispetto alla memoria principale, ma il suo utilizzo è più conveniente in termini di tempo di accesso e/o carico sul sistema. Quando è necessario l'accesso ad un dato, questo dato viene prima cercato nella *cache*. Se è presente e valido, viene utilizzata la copia presente. Viceversa, viene recuperato dalla memoria principale, e memorizzato nella *cache*, nel caso possa servire successivamente.

La memoria principale può essere qualcosa di semplice come un disco rigido, ma anche un complesso database distribuito, come il DNS o il web. In questi casi, la memoria principale può essere modificata senza passare

dalla *cache*, il che comporta problemi di coerenza tra i dati "originali" e quelli nella *cache*. In alcuni casi è possibile validare i dati contenuti nella *cache* interrogando la memoria principale per verificare se sono ancora attuali. Questo è quello che fanno i server *proxy*: chiedono al server HTTP se la pagina che posseggono è stata modificata dopo la sua memorizzazione, e se non lo è evitano di trasferirla e la ripropongono direttamente al *client*. In altri casi, si utilizza un meccanismo di scadenza a tempo dei dati memorizzati, e fino a quando un dato presente nella *cache* non è scaduto questo viene utilizzato, anche se non corrisponde a quanto presente nella memoria principale. Questo è il meccanismo adottato dal DNS.

Una *cache* riduce il carico di richieste che deve essere smaltito dalla memoria principale, e dal collegamento tra questa e l'utilizzatore dei dati. Anche questo può contribuire a migliorare le prestazioni del sistema. Si pensi per esempio ad un server *proxy* utilizzato da molti utenti: quando un utente richiede una pagina che era già stata richiesta da un altro, il *proxy* potrà rispondere senza doversi collegare al sito originale, ed eviterà così di caricare sia il sito originale che la rete, migliorando così le prestazioni del sistema anche per le richieste che devono essere inoltrate ai siti originali. Una *cache* utilizza un algoritmo per decidere quali dati mantenere e quali scartare, che tiene conto delle pagine utilizzate più di recente, della contiguità delle pagine, o di diversi altri fattori. Una *cache* può indicizzare i dati memorizzati sulla base del loro indirizzo (un blocco di memoria o di dati su disco fisso) o del loro "nome" (*cache* associativa, ad esempio una pagina web o un nome DNS).

In alcuni casi la memoria *cache* supporta anche la modifica dei dati. Questo è di implementazione semplice se la *cache* è l'unico percorso di accesso alla memoria principale, come nel caso della *cache* della memoria RAM presente nei processori: la *cache* "accetta" una operazione di scrittura verso la RAM, permettendo al processore di proseguire l'elaborazione, presenta subito al processore i dati aggiornati se questo ne chiede nuovamente la lettura, e si prende carico di scriverli sulla RAM prima di eliminare la pagina. In questo modo, se un dato in memoria è modificato frequentemente dal processore, è possibile mantenere le modifiche nella *cache* ed evitare continui trasferimenti verso la RAM. Nel prossimo paragrafo mostriamo le principali tipologie di *cache*, che si distinguono tra loro per il contesto dove vengono utilizzate.

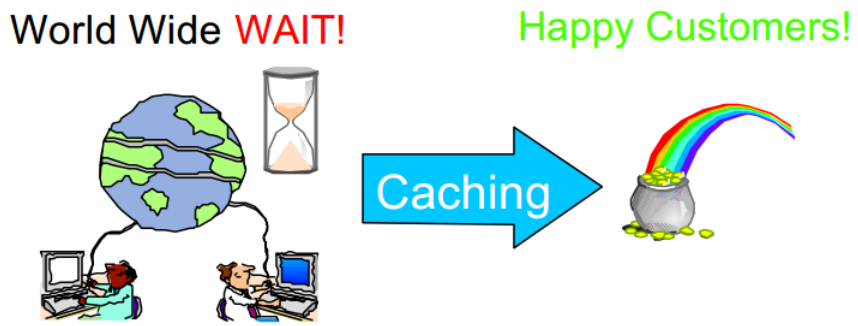



Figura 2.1: Motivazione della nascita del *Web caching*.

## 2.2 Tecnologie *cache-based*

### 2.2.1 *Web caching*

Il *Web caching* è la *caching* di documenti web (pagine HTML, immagini, ecc.) per permettere di ridurre l'uso della banda e il tempo di accesso ad un Sito web (Figura 2.1) [10]. Una *web cache* memorizza copie di documenti richiesti dagli utenti, successive richieste possono essere soddisfatte dalla *cache* se si presentano certe condizioni. Le *Web cache* di solito raggiungono picchi d'efficienza nell'ordine del 30% - 50%, e migliorano la loro efficienza al crescere del numero di utenti. I protocolli di *Web caching* hanno quindi dimostrato di essere una buona soluzione per i problemi della rete come la latenza e la congestione del traffico [2]. Il *Web cooperative caching* è un popolare protocollo di *Web caching* basato su un approccio gerarchico. I sistemi di *cooperative caching* in sostanza introducono una *cache* su ogni nodo della rete. Ogni *cache* deve poi richiedere i contenuti mancanti ad un livello gerarchico superiore. Questo processo viene ripetuto finché o il contenuto richiesto è trovato o la richiesta raggiunge la *cache* radice. Alla fine il contenuto viene memorizzato in tutte le *cache* intermedie mentre ritorna al richiedente. Il controllo del *Web caching* è un tema di ricerca importante nel concetto attuale di Web. Uno degli strumenti di controllo di *cache* più utilizzati è l'*Hypertext Transfer Protocol* (HTTP) *header*. Gli *header* HTTP offrono un controllo su entrambe le *cache* e *proxy* del browser. L' HTTP *header* di risposta contiene diversi campi e direttive che definiscono informazioni importanti come la data di richiesta o risposta, l' identificazione del server, campi di *cache-control*, tempo di scadenza, ultima data di modifica, tipo di contenuto e lunghezza.

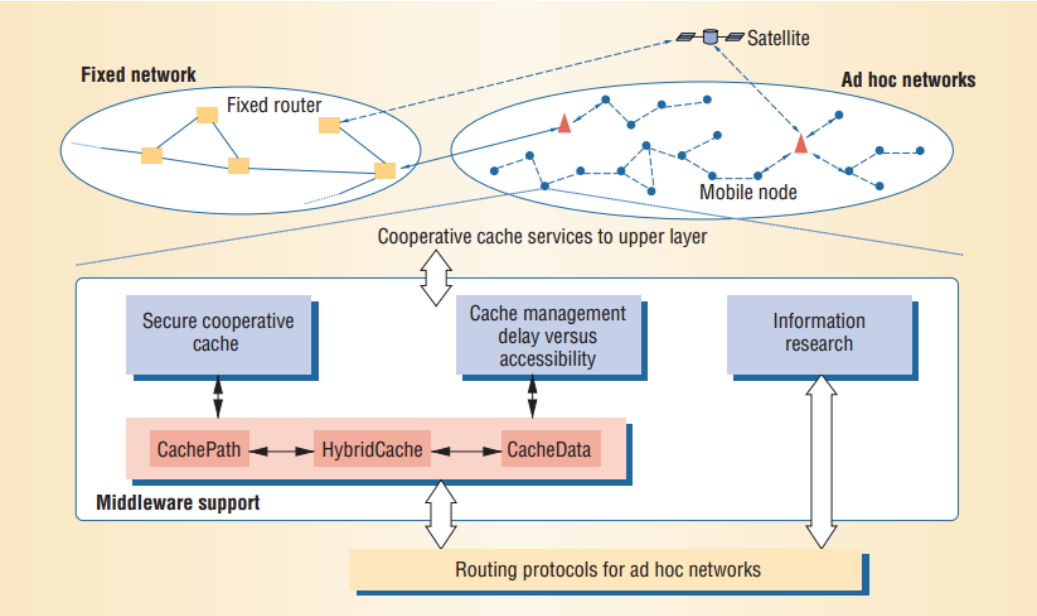



Figura 2.2: *Cooperative caching* in ad-hoc *networks*.

### 2.2.2 *Caching* nelle reti ad hoc

Le reti ad-hoc si basano sulla cooperazione wireless dei nodi della rete, dotati di una *cache* locale, per inoltrare il traffico tra loro [2]. Il *caching* cooperativo per reti ad-hoc affronta i vincoli energetici e i problemi di congestione tipici di questa rete. Esso presenta tre distinti protocolli di *caching*: *CacheData*, *CachePath* e *HybridCache* (Figura 2.2). Nel *CachePath*, quando due nodi comunicano, si scambiano l'identificativo del nodo richiedente relativo ai dati di cui sono portatori. Inoltre memorizzano la distanza con la sorgente e la destinazione (percorso) dei dati richiesti. Pertanto, quando un altro nodo emette la stessa richiesta, la richiesta sarà reindirizzata al nodo con l'identificativo memorizzato in precedenza. Nel *CacheData*, invece dei percorsi, vengono memorizzati i dati richiesti di più frequente per richieste future. L'*HybridCache* usa invece entrambi i protocolli.

I problemi nel *caching* cooperativo in sostanza sono due: la risoluzione della *cache* e il *cache management*. La prima riguarda i meccanismi di decisione del dispositivo mobile per trovare i dati richiesti dall'utente. L'algoritmo di inoltro dei nodi e i profili storici sono una buona soluzione utilizzata tra l'altro per diminuire il costo della comunicazione. Il secondo determina invece quali dati verranno messi/ eliminati nella *cache* locale. Questa gestione riduce significativamente il numero di copie *cache* tra i nodi, consentendo un più grande spazio di memorizzazione dati e migliorando le prestazioni della rete.

### 2.2.3 Altri ambiti di utilizzo del *caching*

In questa sezione vogliamo citare per completezza altre tecnologie *cache-based* che però nel nostro caso hanno influito in modo minore [10].

La *CPU cache* viene utilizzata per accelerare l'accesso alle posizioni di memoria RAM usate più frequentemente. Si tratta di una piccola quantità di memoria veloce installata direttamente sul processore o nelle sue immediate vicinanze. Viene utilizzata memoria di tipo SRAM, grazie al suo ridotto consumo e alla notevole velocità.

La *Page cache* consiste nell'avere una parte della RAM usata dal sistema operativo in cui si copiano dall'*hard disk* i dati correntemente in uso. In questo caso, l'accesso alla RAM è più veloce dell'accesso al disco. Poiché la memoria disponibile è generalmente limitata, il sistema operativo cerca di mantenere il più possibile in memoria una pagina, mantenendo una tabella delle pagine che non sono usate correntemente ma lo sono state in passato. Quando occorre caricare una pagina nuova, verrà sovrascritta la più vecchia non ancora in uso.

La *Disk cache* consiste nell'avere un hard disk che ha al suo interno una parte di RAM, dove possono venire caricati i settori del disco logicamente contigui a quello richiesto. Quando si accede in lettura al disco, nel caso i dati richiesti siano presenti nella *cache* si evita lo spostamento della testina di lettura del disco stesso, velocizzando il reperimento dell'informazione e contribuendo al ridurre l'usura del disco stesso.

La *cache DNS* è un server DNS che non possiede informazioni autoritative, ma è in grado di chiederle ai server autoritativi e memorizzare le risposte. I server DNS utilizzati dagli utenti di Internet sono normalmente dei server *cache*. Il DNS usa un meccanismo di scadenza, per cui ogni record recuperato da un server autoritativo è valido per un certo tempo, dopo il quale deve essere scartato.

La *Google cache*, o più propriamente *cache* dei motori di ricerca, consiste in copie delle pagine web salvate presso i server di un motore di ricerca. Queste vengono utilizzate per due motivi: eseguire ricerche locali all'interno delle pagine ed offrire la possibilità di vedere una copia, per quanto non aggiornata, di una pagina non disponibile, per problemi momentanei o perché è stata rimossa dal server originale.

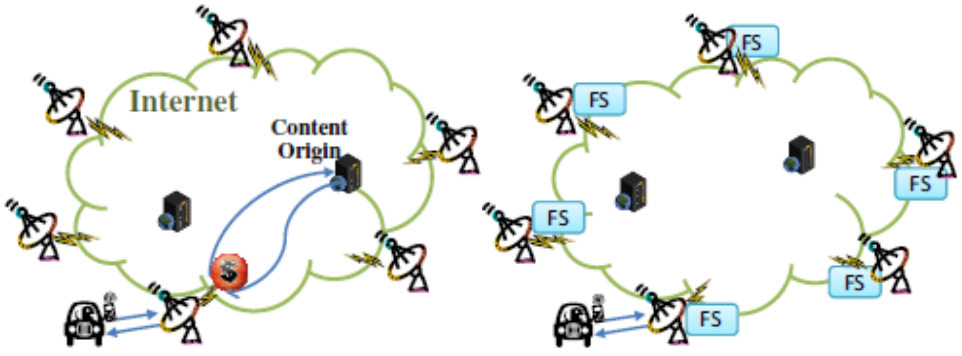
## 2.3 Il *caching* nelle infrastrutture di distribuzione di contenuti per VANET

Una volta descritta la maggior parte delle tecnologie *cache-based* ci chiediamo a questo punto a cosa effettivamente possa servire una simile struttura. Ricordiamo infatti che il nostro obiettivo è quello di trovare il miglior modo per distribuire contenuti in scenari molto particolari attraverso reti veicolari.

Nei progetti di reti veicolari tradizionali, come mostrato in Figura 2.3a, gli *Access Point* non hanno una *cache* [1]. Se un utente mobile vuole scaricare file di origine remota deve attenderne il recupero attraverso collegamenti wireless e fissi con altri *Access Point* e/o Internet. Questo schema funziona bene però solo per dispositivi che si muovono lentamente perché hanno connessioni stabili con gli *Access Point*. Diventa invece problematico quando viene applicato alle reti veicolari per le proprietà uniche delle connessioni WiFi in uno scenario altamente mobile.

Le connessioni WiFi con gli *Access Point* infatti risultano essere:

- transitorie (solitamente solo un po' di secondi considerando una velocità urbana);
- intermittenti (passano alcuni minuti prima di una nuova connessione);
- con un alto tasso di perdita (spesso del 20%).




(a) Reti veicolari tradizionali (b) Content Distribution Community

Figura 2.3: Recupero file.

Il tradizionale processo di recupero ha i seguenti inconvenienti (Figura 2.3a). In primo luogo, i collegamenti verso l'origine dei contenuti possono diventare costosi in termini di banda e ritardo nelle connessioni WiFi. In

secondo luogo, i sistemi all'origine dei contenuti sono progettati presupponendo che gli utenti richiedenti abbiano una stabile connessione *end-to-end*. Complessivamente c'è bisogno quindi di conciliare le reali condizioni delle connessioni wireless veicolari con la falsa ipotesi di collegamenti stabili degli utenti.

In [1] viene proposta una *Content Distribution Community Infrastructure* (CDCI) basata su WiFi simile alla ben più nota *P2P Content Distribution Community*. L'idea di base (vedi Figura 2.3b) è quella di collegare ogni *Access Point* con un *File Server* locale, che cerca di soddisfare istantaneamente le richieste di download degli utenti attraverso una *cache* interna, senza affacciarsi all'origine dei contenuti via Internet. Possiamo già notare come questa idea risulti essere incompatibile con metodi di *cache* più classici come il *Web caching*. Non è infatti pensabile un'impostazione gerarchica proprio per la durata limitata delle connessioni.

CDCI ha tuttavia diversi vantaggi. Principalmente si riesce a utilizzare a pieno il breve collegamento a disposizione con l'*Access Point* per recuperare i file attraverso il *File Server* locale. In più si riduce il traffico verso Internet, risparmiando banda per il recupero di quei file non presenti nel *File Server*.

La *WiFi-based Content Distribution Community* si basa sul fatto che gli utenti all'interno di una comunità condividono interessi simili. Se non ci fossero per esempio due utenti a richiedere lo stesso file, non vi sarebbe alcun beneficio nell'immagazzinare quel file nella *cache*. Per fortuna, in una comunità, è normale che molti utenti siano interessati ad un piccolo insieme di contenuti, come ad esempio notizie sul traffico e video della comunità. In CDCI, una questione importante è come memorizzare nella *cache* in modo ottimale file nei *File Server*, in modo da massimizzare la probabilità di recupero dei file da parte degli utenti mobili, vincolati da un tempo e da un deposito di ogni *File Server* limitato. Nel prossimo capitolo viene proposta l'architettura di gestione dei contenuti e formalizzato il problema di gestione dei contenuti come un problema di ottimizzazione lineare, considerando fattori come l'archiviazione negli *Access Point*, i modelli di mobilità, la popolarità dei file, la dimensione dei file, ecc.

In [1] si esclude la comunicazione *node-to-node* (V2V) presente solitamente nelle reti ad-hoc veicolari (VANETs). Risulta infatti essere più realistico e pratico quando:

- gli utenti non hanno alcun incentivo a collaborare (sprechi di energia e mancanza di fiducia);
- gli utenti vogliono un livello alto di privacy e segretezza;
- il tempo di contatto per una coppia di nodi è imprevedibile.

C'è comunque da dire che questa ristretta comunicazione con solo gli *Access Point* semplifica ma allo stesso tempo riduce notevolmente il download dei file. Si possono difatti stimare prestazioni peggiori rispetto al caso in cui è invece possibile completare il recupero dei contenuti, scambiandosi pezzi di file anche tra utenti mobili.

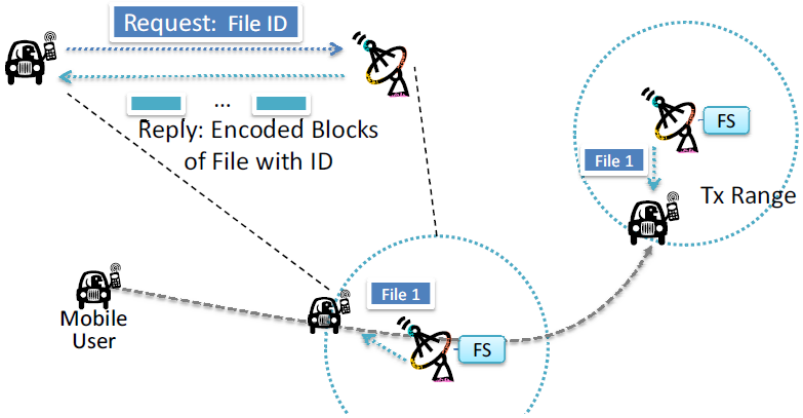



Figura 2.4: File download.

Gli utenti si muovono nella rete e avviano le richieste di download di volta in volta. Quando un utente si trova all'interno del raggio di trasmissione di un *Access Point*, invia la richiesta di download con l'identificativo del file. Il *File Server* contattato prende quindi i blocchi codificati di questo file e li restituisce all'utente. Gli scambi dei messaggi sono mostrati in Figura 2.4.

La qualità del sistema è misurata dalla probabilità che gli utenti ottengano il file di interesse. Le eccezioni avvengono quando la comunità nel suo insieme (ovvero tutti gli *Access Point*):

- non ha i contenuti richiesti e allora il *File Server* contattato può recuperare il file attraverso Internet;
- ha il file, ma il tempo stimato di recupero tra *File Server* supera il tempo massimo e allora il *File Server* può richiedere (in parte) il file tramite Internet.

In tutti gli altri casi i collegamenti tra *Access Point* e Internet non vengono utilizzati. C'è anche da tenere conto che gli utenti tollererebbero un certo tipo di ritardo nel download dei file solo nel momento in cui il costo presunto del servizio di recupero non fosse eccessivamente alto. Questo ritardo in linea di massima è paragonabile alla durata media del veicolo nel raggio di trasmissione dell'*Access Point*.

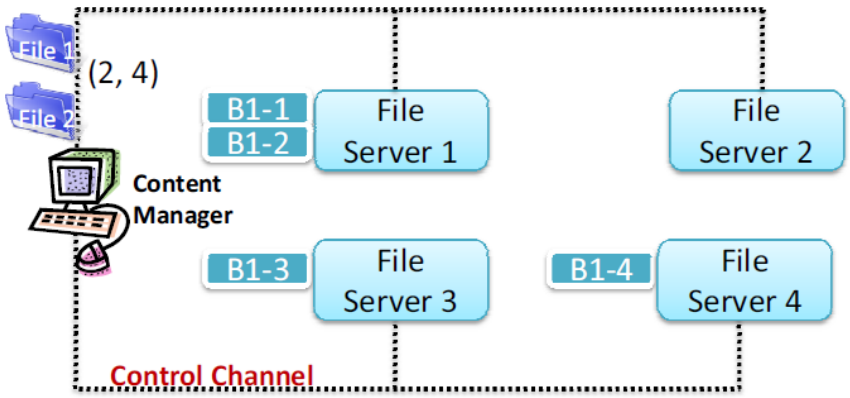



Figura 2.5: *Content Manager.*

Al fine di ottimizzare le prestazioni di distribuzione dei contenuti, c'è bisogno quindi di un *Content Manager* (CM) come mostrato in Figura 2.5. Esso ha canali di controllo sicuri e affidabili con tutti i *File Server* via Internet. Date le varie informazioni, come la popolarità dei file e i record di accesso degli *Access Point* (monitorati individualmente da questi), il CM decide un modo ottimale per distribuire i contenuti fissando un set di file specifico per ogni *File Server*. Questi blocchi di file codificati vengono quindi inviati ai *File Server* attraverso i canali di controllo. Il CM ricalcola periodicamente la strategia ottimale e ridistribuisce i file, adattandosi all'evoluzione delle preferenze di contenuti e dei modelli di movimento all'interno della comunità. La strategia di gestione dei contenuti del CM sarà quella che massimizza la probabilità di recupero dei file entro un tempo fissato.

Se lo spazio di archiviazione di ciascun *File Server* fosse illimitato, il miglior modo sarebbe quello di mettere una copia di ogni file in ogni *Access Point*. Ogni visita al generico *Access Point* garantirebbe una qualsiasi parte di qualsiasi file. Tuttavia, a causa di una grossa quantità di file e di uno spazio di archiviazione limitato, mettere ogni copia di file in ogni *Access Point* sarebbe fisicamente infattibile. Il CM deve allora decidere il grado di ridondanza, non necessariamente un numero intero, per ogni file e il corrispondente blocco di allocazione. A questo scopo i codici a cancellazione offrono una maggiore flessibilità e prestazioni migliori. Questa tecnica è infatti già usata con successo nel *peer-to-peer content distribution* e nelle DTN. Pertanto memorizzeremo distinti blocchi codificati di file nei *File Server*. La codifica garantisce che ad ogni visita ad un *Access Point* si possa scaricare una porzione utile di file e diminuire la perdita di pacchetti. Molti fattori influenzano la strategia di gestione dei contenuti. Ad esempio un file popolare di piccole dimensioni, intuitivamente, avrà una grande ridondanza negli *Access Point* più frequentati.

Nel prossimo capitolo analizzeremo questi fattori.

## 2.4 *Cache Management*

### 2.4.1 *Replacement Policy*

Nel paragrafo precedente abbiamo detto che il CM decide, in base ad alcune informazioni, la strategia ottimale. In questa sezione ci chiediamo allora in che modo queste informazioni possano incidere sulla strategia, descrivendo, più in generale, la maggior parte delle politiche di *cache replacement* al momento conosciute. Con *cache replacement* intendiamo tutti quegli algoritmi volti a ricambiare nel tempo il contenuto della *cache*, per soddisfare, di volta in volta, il maggior numero di richieste di download di file.

Prima di pensare a quelle più complesse e specifiche come nel nostro caso, vediamo, giusto per dare un'idea, le politiche base [3]. Possiamo dividerle in due categorie: politiche basate su *temporal locality* e *spatial locality*. Nella prima, basata su un fattore temporale, troviamo FIFO (*First In First Out*), LRU (*Least Recently Used*) e LFU (*Least Frequently Used*). FIFO: è la scelta più classica e meno sofisticata. I primi contenuti inseriti nella *cache* saranno poi i primi ad essere eliminati. LRU: la strategia è principalmente quella di ordinare i contenuti nella *cache* in base al tempo passato dall'ultima richiesta e di eliminare quelli con il tempo maggiore, ovvero quei contenuti meno richiesti di recente. LFU: è molto simile a quella precedente, difatti la politica è quella di tenere il conto delle richieste dei contenuti presenti e di eliminare quelli col minor numero, ovvero quelli richiesti meno frequentemente.

Nella seconda categoria, caratterizzata invece da un fattore spaziale, troviamo FAR-E (*Furthest Away Replacement - Euclidean distance*) e FAR-N (*Furthest Away Replacement - Network distance*). FAR-E: la scelta è quella di ordinare i contenuti in base alla distanza dalla loro reale origine e di quindi eliminare quelli più lontani. Il criterio si basa sul fatto che risulta meno probabile che un utente richieda contenuti al di fuori della sua area. FAR-N: sostanzialmente non differisce da quella precedente, ma è più facilmente attuabile. La distanza in questo caso si basa sul numero degli *hop* dei nodi necessari per raggiungere il server della reale origine del contenuto. Queste sono le *policy* principali che è sempre possibile trovare in qualsiasi sistema *cache-based*. Esistono poi soluzioni più complicate che tengono conto di più fattori, come ad esempio la *Greedy-Dual-Size* [11]. In questo caso la strategia è di rimpiazzare i contenuti in base ad una combinazione di popolarità (LRU o LFU) e dimensione.

Mano a mano che aumenta il grado di complessità possiamo trovare vere e proprie funzioni-costo che si adattano, sempre più, al caso specifico. Vediamo ora, come esempio, le considerazioni che sono state fatte per una generica rete ad-hoc. In [4] gli algoritmi di *cache replacement* sono stati ampiamente studiati nei sistemi operativi, nella gestione della memoria virtuale e dei buffer dei database. Nonostante ciò, potrebbero essere inadatti per le reti ad-hoc per diverse ragioni:

I la dimensione dei file non è fissata in ambienti wireless. La politica LRU deve essere quindi estesa per gestire contenuti di varie dimensioni;

II il tempo di trasferimento del singolo dato potrebbe dipendere dalla sua dimensione e dalla distanza tra il nodo richiedente e l'origine del dato;

III l'algoritmo dovrebbe prendere in considerazione anche la coerenza della *cache*, cioè, contenuti che tendono ad essere incoerenti prima devono essere i primi ad essere sostituiti.

Ad esempio, per l'ultima ragione, può capitare che un primo file venga richiesto l'1% del tempo e aggiornato l'1% del tempo mentre un secondo file venga richiesto lo 0,5% del tempo ma aggiornato solo lo 0,1% del tempo. L'algoritmo LRU dovrebbe sostituire il secondo e mantenere il primo. Tuttavia, mantenendo il secondo elemento, si potrebbe avere una migliore performance.

Nella maggior parte degli algoritmi di ricambio della *cache* per il Web ci sono funzioni basate su fattori temporali come: il tempo passato dall'ultima richiesta (LRU), il tempo di comparsa nella *cache* (FIFO), il tempo di trasferimento e la scadenza del file. Le più sono però troppo generiche e funzionano relativamente bene solo per un certo obiettivo, come può essere quello classico del Web, ovvero il minor tempo di risposta. Quando l'obiettivo cambia c'è bisogno di funzioni più specifiche basate sull'esperienza. Ambienti in cui la preoccupazione principale è l'accessibilità dei dati richiedono una particolare politica di ricambio della *cache*. Qui consideriamo due fattori nella scelta dei dati (vittime) da sostituire. Il primo fattore è la distanza $\delta$, ovvero il numero di *hop* dalla sorgente dati o nodo di *caching*. $\delta$ è strettamente correlata con la latenza, pertanto si eliminano quei contenuti con $\delta$ grande. Il secondo fattore è la frequenza d'accesso a questi dati. Sapendo che la topologia della rete ad-hoc può sempre cambiare, il valore della distanza $\delta$ potrebbe diventare obsoleto. Utilizziamo allora un altro parametro, $\tau$, che cattura il tempo trascorso dall'ultimo aggiornamento di $\delta$. Otteniamo $\tau$ da $1/(t_{cur} - t_{up})$, dove $t_{cur}$ e $t_{up}$ sono rispettivamente il tempo corrente e il tempo dell'ultimo aggiornamento di $\delta$. Se $\tau$ è vicino a 1, allora $\delta$ è stata recentemente aggiornata. Se invece $\tau$ è vicino a 0, allora è passato tanto tempo dall'ultimo aggiornamento. In questo modo utilizziamo $\tau$ come indicatore di $\delta$ per

selezionare una vittima. Utilizzando questi due fattori, possiamo progettare diversi algoritmi di ricambio della *cache*. Ad esempio, la vittima può essere l'elemento con il valore di $\delta \times \tau$ più piccolo. Alcuni risultati preliminari mostrano che valori diversi di questi parametri influenzano il *tradeoff* tra accessibilità dei dati e ritardo di richiesta. Un ulteriore passo avanti potrebbe essere poi quello di incrementare l'accessibilità dei contenuti considerando quelli già presenti nelle *cache* dei nodi vicini.

Con questa ultima idea ci si avvicina ulteriormente alla strategia del nostro CM. Nel nostro caso infatti, a differenza delle reti ad-hoc, siamo a conoscenza di tutti i nodi (gli *Access Point*) e la strategia sarà centralizzata, considerando, per l'appunto, i contenuti presenti in ogni *File Server*. Oltre ai soliti fattori come la popolarità, la dimensione del file, già viste nelle politiche più semplici, ne si considerano ulteriori come la banda garantita di ogni *Access Point* e i modelli di mobilità. Ad ognuna di queste viene associato un peso e ad ogni aggiornamento starà al CM decidere quali contenuti tenere e quali rimpiazzare nelle *cache*.

| Acronimo | Significato |
|---|---|
| FIFO | First In First Out |
| LRU | Least Recently Used |
| LFU | Least Frequently Used |
| FAR-E | Furthest Away Replacement - Euclidean distance |
| FAR-N | Furthest Away Replacement - Network distance |
| GDS | Greedy - Dual - Size |
| DW | Directional Weight |
| FCFS | First Come First Service |

Tabella 2.1: Politiche di *Cache Management*.

### 2.4.2 *Scheduling*

Oltre alle politiche di *replacement* appena viste, un buon sistema a *cache* deve anche tenere conto dello *scheduling* ossia, del problema dell'accesso multiplo tra veicoli attestati sotto lo stesso RSU. Nel nostro caso il problema non viene preso in considerazione perché, come vedremo nel capito successivo, supponiamo che ogni *Access Point* è in grado di servire istantaneamente in parallelo qualsiasi numero di utenti richiedenti. Un maggior traffico quindi non creerà lunghe code d'attesa ma avrà il difetto di diminuire sempre più la banda garantita. E' realistico comunque pensare ad un servizio gestito in

serie. A tal proposito vediamo in questa sezione le strategie di *scheduling* al momento conosciute, mostrando, principalmente, gli studi di *Junghoon Lee* sull'argomento.

L'efficacia di un meccanismo di *cache* dipende sia dalla architettura di rete che dalle caratteristiche del servizio offerto. In [3], si considerano solo servizi *local-dependent* come, per esempio, un servizio di informazioni sul traffico nelle aree circostanti. Si ipotizza quindi che informazioni di aree lontane siano richieste con basse probabilità. Si è poi a conoscenza che le richieste, in generale, possono raggiungere grandi numeri in specifici intervalli di tempo nell'arco della giornata, soprattutto quando si è in zone solitamente trafficate. Come conseguenza, in poco tempo, aumentano notevolmente la lunghezza delle code di richiesta all'*Access Point* di riferimento e, contemporaneamente, i tempi di elaborazione della richiesta. Quando ciò accade, sia la percentuale di successi (recuperi andati a buon fine) che il tempo di risposta possono essere migliori attraverso una adeguata politica di *scheduling*. E' necessario infatti che ci sia il giusto criterio nel gestire la coda. Però, per progettare nelle reti veicolari un buono *scheduling* su misura, deve esserci prima una dettagliata analisi sul pattern di accesso alle informazioni. Questa analisi può rivelare infatti molte caratteristiche utili come gli intervalli di tempo con più traffico, le distanze tra *Access Point* e le zone a cui si riferiscono le informazioni, e la tendenza direzionale delle informazioni richieste.

Lo *scheduling* è generalmente combinato con il resto del programma di *cache management* ed ha gli stessi obiettivi come la percentuale di successi, l'equità, e il tempo di risposta. Per esempio, in un ambiente multi-cella il fattore critico delle prestazioni del sistema è garantire l'equità ai clienti. A tal proposito nel lavoro di *Zheng* si sceglie di dare priorità alle richieste di quei clienti che stanno per lasciare la cella, in modo da far ricevere i contenuti prima della loro uscita [5]. Questo fa capire come ogni scenario può avere una diversa politica ideale e che quindi è sempre desiderabile sfruttare il pattern di accesso ai contenuti il più realisticamente possibile, per valutare bene le prestazioni del *cache management*.

Tornando a [3], viene proposto DW (*Directional Weight*), uno schema valevole per il *cache management* in generale. Si comincia dalla stessa idea di FAR, vista precedentemente, per poi migliorare ulteriormente la percentuale di successo, tenendo conto, appunto, del peso direzionale (DW). Il criterio, per quanto riguarda il *cache replacement*, è che se l'informazione si riferisce a una zona in cui molti veicoli tendono a dirigersi, allora è il caso di tenerla nella *cache*. Per quanto riguarda lo *scheduling*, invece della classica politica FCFS (*First Come First Service*), l'ordinamento segue lo stesso criterio del *cache replacement*. Si decide di dare precedenza di servizio a quegli utenti richiedenti informazioni che, come prima, si riferiscono ad aree, in cui

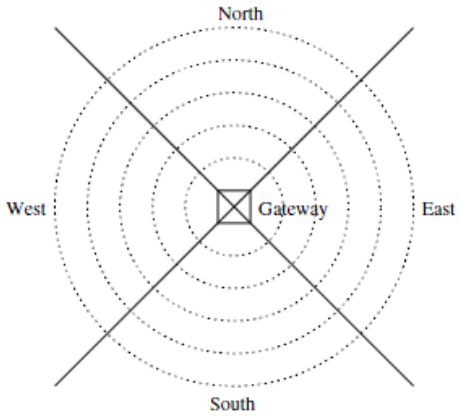



Figura 2.6: *Directional Weight*.

molti veicoli tendono a dirigersi. In questo caso la posizione angolare della destinazione diventa importante (vedi Figura 2.6). Così, quando la richiesta viene inserita nella coda d'attesa, lo *scheduling* ordina gli elementi secondo queste destinazioni più "frequenti". Nel lavoro [13] è possibile trovare un altro tentativo per uno *scheduling* ottimale. Nella Tabella 2.1 ricapitoliamo, invece, le principali politiche di *cache management*.

## 2.5 Distribuzione contenuti in VANET: altri approcci

In letteratura si trovano diversi lavori che trattano l'argomento della *content distribution* nelle reti veicolari. In questo paragrafo vogliamo proporre quelli più recenti, che non è stato possibile citare nei paragrafi precedenti, ma non per questo meno importanti. La maggior parte di questi sceglie di affrontare il problema sfruttando la cooperazione tra veicoli, che è possibile trovare nelle reti veicolari ad-hoc. L'idea di base, come già detto in precedenza, offre sicuramente buone prestazioni.

Ad esempio in [7] viene ideato un algoritmo, *Amleto*, in cui i nodi decidono in modo indipendente se memorizzare o no alcuni contenuti e per quanto tempo. Ogni nodo prende questa decisione in base alla percezione di ciò che può essere memorizzato nei nodi vicini con l'obiettivo di differenziare il proprio contenuto da quello dei vicini. In [9] viene invece presentato *CodeOn*, uno schema *push-based* per la distribuzione dei contenuti più popolari nelle VANET. Questi vengono messi in rete dagli *Access Point* e distribuiti trai veicoli attraverso la loro cooperazione. Viene utilizzata una particolare codifica

(SLNC) per combattere la caratteristica perdita delle trasmissioni wireless. In [11] possiamo trovare un altro approccio simile. In [14] troviamo l'idea di un sistema ancora cooperativo per la *content distribution* tra veicoli (CCDSV). Si basa su una codifica di rete, sulla previsione di contatto veicolo-*Access Point* e sulle tecniche di *prefetching* dei dati. In [16] invece viene presentato il protocollo SPAWN che rappresenta una strategia di *content download* di tipo cooperativo, necessaria per lo scaricamento di file di grandi dimensioni (es: film). A causa della mobilità e della velocità dei veicoli, il collegamento con una infrastruttura fissa è necessariamente limitato nel tempo e non permette il download completo di tali file. L'idea proposta è di dividere il contenuto in N parti, ognuna delle quali viene etichettata. L'utente scarica dall'*Access Point* solo alcune porzioni del file, più una lista di altri veicoli che hanno effettuato la stessa operazione. Ognuno degli utenti invia periodicamente dei messaggi di tipo "gossip" in cui indica quali parti del file possiede. In questo modo lo scaricamento del file può essere completato tramite collegamenti tra veicoli. Il punto debole di questo protocollo è rappresentato dalla difficoltà di trovare veicoli in possesso delle parti mancanti, soprattutto in caso di file poco richiesti. Infine in [17] viene presentata una strategia basata sempre sullo stesso concetto di download cooperativo in cui, la codifica delle porzioni di file e la ricerca di *peers* solo tra i vicini fisici migliora, in alcuni casi, le performance.

In [18] si fa invece un passo indietro e vengono riprese in considerazione diverse strategie tenendo conto anche della topologia degli *Access Point*. Viene quindi studiata la questione del download di contenuti sotto forma di problema di massimo flusso in cui le decisioni da prendere sono due: dove installare gli *Access Point* scegliendo tra diversi siti candidati e quali paradigmi di rete utilizzare nell'ambito delle reti veicolari. I paradigmi presi in considerazione sono tre: collegamento diretto veicolo-infrastruttura, collegamento *multi-hop* tra infrastruttura e utente finale, collegamento di tipo *carry-and-forward* in cui l'utente intermedio memorizza il contenuto e lo consegna direttamente all'utente finale o a un suo vicino. I risultati hanno evidenziato performance migliori nel caso di una maggiore presenza di *Access Point* sul territorio. In [15] troviamo di nuovo una proposta più generale. Vengono definite quattro categorie di informazione: *security*, *traffic*, *services* e *contents*. Per ciascuna vengono indicati alcuni metodi utilizzati per la circolazione di tali informazioni tra i vari utenti.

In [6] [8] l'attenzione è sul protocollo di comunicazione. Nel primo viene proposto VITP, un protocollo di comunicazione a livello applicativo che supporta tecniche di *caching*. Viene effettuato quindi uno studio di valutazione per garantire un particolare set di servizi (VIS). I risultati ne identificano i parametri critici e la capacità del protocollo di minimizzare l'*overhead* nella

rete. Nel secondo viene presentata invece un'ottima panoramica di tutti i più recenti protocolli di routing possibili per le reti veicolari ad-hoc, considerando diversi modelli di mobilità.

In [2] viene proposta una soluzione completamente automatizzata per il recupero e l'archiviazione dei contenuti nelle *Vehicular Delay-Tolerant Networks* (VDTN). Essa consiste nell'allegare ad ogni pacchetto dati delle particolari etichette di controllo. Queste etichette definiscono praticamente quali contenuti mettere nella *cache*. Infine in [12] si indaga sulle performance dello standard IEEE 802.11 nelle reti veicolari altamente mobili con particolare riguardo al livello MAC.

# Capitolo 3

# Un Modello per la distribuzione dei contenuti in reti veicolari con infrastruttura

In questo capitolo si propone la strategia per il *caching* ottimale nelle reti veicolari. Viene descritto il modello di ottimizzazione proposto mettendo in evidenza le differenze rispetto allo stato dell'arte.

## 3.1 Scenario

Il nostro scenario è quello classico di una piccola area, come può essere un quartiere di una città, con possibilità per le strade di *V.A.Nets*. Attraverso queste ultime, come detto in precedenza, gli utenti mobili possono comunicare sia con gli utenti nei veicoli vicini (*Vehicle To Vehicle* - Figura 3.1b) sia scaricare e rilasciare contenuti con gli *Access Point* nella loro area (*Vehicle To Infrastructure* - Figura 3.1a).

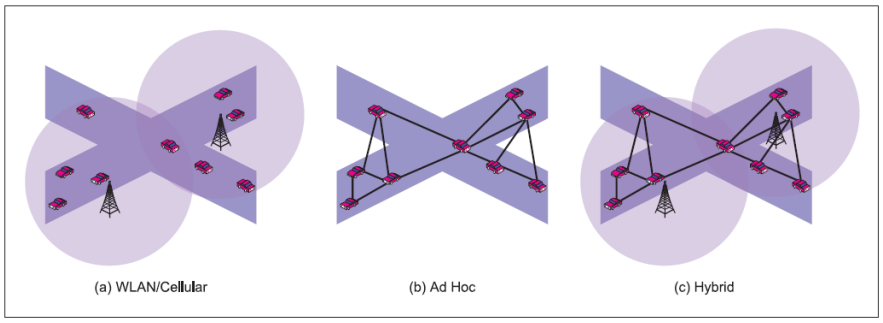



Figura 3.1: Le tipologie di architetture *V.A.Nets*.

La nostra attenzione è concentrata sulle problematiche dell'architettura V2I (Figura 3.1a) dove, come abbiamo già visto, un buon sistema a *cache* può venirci in aiuto. Ricordiamo infatti che spesso molti utenti in una comunità sono interessati ad un limitato set di contenuti che riguardano informazioni e notizie locali, come possono essere quelle sul traffico o sulle stazioni di servizio più vicine. In tutti gli altri casi invece, è comunque possibile notare una distinzione, probabilmente meno netta, tra contenuti più e meno richiesti. Questa informazione, che chiamiamo popolarità del contenuto, ci permette di stilare, di volta in volta, una classifica dei contenuti più richiesti e di prevedere e circoscrivere il range di domanda dell'utente. I meglio posizionati vengono inseriti nello spazio limitato della *cache* di ogni *Access Point*. Quello che ci aspettiamo è che, con buona probabilità, uno di questi contenuti verrà richiesto dai prossimi utenti. Questo principio è alla base della strategia di *cache* dei nostri *Access Point*. Gli altri fattori rilevanti, che vedremo in seguito, sono la topologia degli *Access Point* e le preferenze nei movimenti degli utenti.

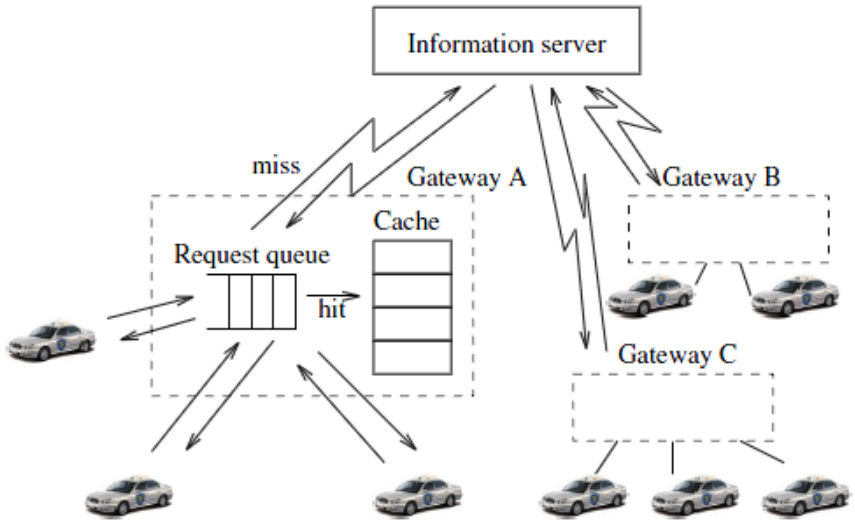



Figura 3.2: Concetto di *cache* in *V.A.Nets*.

Come mostrato in Figura 3.2, l'utente richiederà un contenuto all' *Access Point*. Quest'ultimo lo cercherà nella propria *cache* e, in caso positivo (*hit*), avverrà la trasmissione istantanea verso l'utente, in caso negativo (*miss*), dovrà richiederlo su Internet o agli altri *Access Point*, appoggiandosi a una architettura dedicata, per poi successivamente trasmetterlo all'utente. In caso di *miss*, oltre ai costi che ne derivano dovuti al tentativo di recupero

del contenuto da parte dell'*Access Point* (vedi Figura 2.3), potrà capitare a volte che, durante il tempo di recupero o il successivo di trasferimento, l'utente mobile sarà già uscito dalla zona dello stesso *Access Point* e non sarà più possibile completare il recupero del contenuto. Il nostro obiettivo sarà quello quindi di massimizzare le *hit* e minimizzare le *miss*, che potrebbero inoltre rallentare lo *scheduling* dell'*Access Point*. Quest'ultimo, infatti, oltre a rischiare di perdere tempo inutilmente, non garantirebbe il servizio ad altri utenti in coda. Nel nostro modello decidiamo allora di evitare la catena innescata dopo un *miss* e ritenere impossibile fin da subito l'ottenimento di tutti quei contenuti richiesti non presenti nella *cache*.

Riconsiderando la situazione nella sua totalità è molto più probabile invece che l'utente possa incontrare più di un *Access Point* durante un generico viaggio e che quindi sembrerebbe logico aver la possibilità di completare il recupero del contenuto attraverso collegamenti con più *Access Point*. E' a questo punto che entrano in gioco gli altri due fattori citati precedentemente, ovvero la topologia degli *Access Point* e le preferenze nei movimenti degli utenti. Dando già per assodata la migliore posizione degli *Access Point* nell'area, ottenuta attraverso uno studio sulle strade più e meno trafficate, ci concentreremo sui percorsi degli utenti. Infatti, attraverso uno studio più accurato, sarà possibile stimare i percorsi principali degli utenti. Con percorso intendiamo una partenza e un arrivo ma soprattutto l'informazione che più ci interessa, ovvero la serie di contatti con gli *Access Point* incontrati. Il contenuto richiesto verrà ritenuto recuperato con successo solamente se alla fine del viaggio l'utente sarà riuscito a scaricarne la sua totalità attraverso i contatti con gli *Access Point* incontrati nel percorso. Ogni *Access Point*, tra l'altro, garantirà una banda che verrà divisa tra tutti gli utenti passanti, quindi, più è trafficato l'*Access Point*, minore è, come è logico che sia, la banda a disposizione.

Tenendo conto di tutte queste considerazioni, il nostro compito finale sarà quello di massimizzare la media pesata della probabilità di recupero completo dei contenuti richiesti nei percorsi, operando sulla scelta dei contenuti da inserire nella *cache* di ogni *Access Point*. Contenuti più richiesti e percorsi più frequentati avranno peso maggiore, ovvero sarà ritenuto più importante garantire il recupero dei contenuti più richiesti nei percorsi più frequentati. Nel prossimo paragrafo vedremo più in dettaglio il modello adottato.

## 3.2 Il modello

In questo paragrafo viene presentato il modello matematico che prende spunto da quello base in [1]. A differenza di quest'ultimo, sostanzialmente si

è in grado di formalizzare il problema come un problema di ottimizzazione lineare e di risolverlo con *CPLEX* senza l'aiuto di euristiche.

### 3.2.1 Indici e Parametri

| Indice | Unità | Campo |
|---|---|---|
| $j$ | File | $\mathbb{J} = \{1, \ldots, J\}$ |
| $i$ | *File Server* | $\mathbb{I} = \{1, \ldots, I\}$ |
| $v$ | Percorso | $\mathbb{V} = \{1, \ldots, V\}$ |
| $(i, v)$ | Contatto | $\mathbb{C} = \mathbb{I} \times \mathbb{V}$ |

Tabella 3.1: Indici del modello.

Sono definiti il numero di $I$ *File Server* (*Access Point*) e il numero di $J$ File (contenuti). Ogni *File Server* può contenere fino a *Cap* [kbyte] e mette a disposizione, per il download dei File, una *Banda(i)* effettiva [kbyte/ contatto]. Questa è ricavata dalla capacità intrinseca (*BandaTot*) di ogni *File Server*. Successivamente vedremo in che modo, ma sappiamo già essere inversamente proporzionale alla quantità di utenti mobili passanti dallo $i^{\text{esimo}}$ *File Server*. Nel Contatto l'utente può scaricare non più di *Banda(i)* kbyte del File richiesto.

Ci sono $V$ Percorsi ed ognuno è composto da una diversa serie di Contatti con i *File Server*. Definiamo così una vera e propria matrice Contatti $(i, v)$ per lo $i^{\text{esimo}}$ *File Server* e il $v^{\text{esimo}}$ Percorso. $FS(i, v) \in [1/0]$ indica la presenza o l'assenza di un Contatto tra lo $i^{\text{esimo}}$ *File Server* e il $v^{\text{esimo}}$ Percorso.

*Size(j)* [kbyte] e *Query(j)* sono la dimensione e la probabilità di richiesta del $j^{\text{esimo}}$ File. Quest'ultima rispecchia la popolarità del File ed è impostata in modo tale che la somma delle probabilità di richiesta di tutti i File sia uguale a 1:

$$\sum_{j=1}^{J} Query(j) = 1. \tag{3.1}$$

Allo stesso modo *P(v)* è la frequenza del $v^{\text{esimo}}$ Percorso:

$$\sum_{v=1}^{V} P(v) = 1. \tag{3.2}$$

Dati $N$ utenti totali, il *Numero di utenti per Percorso*, per ogni Percorso, può essere calcolato come:

$$N_v = N \cdot P(v) \quad , \forall v \in \mathbb{V}. \tag{3.3}$$

Il *Numero di utenti per* File Server può quindi calcolarsi come:

$$N_i = \sum_{v=1}^{V} N_v \cdot FS(i,v) \quad , \forall i \in \mathbb{I}. \tag{3.4}$$

A questo punto troviamo, per ogni *File Server*, la sopracitata *Banda(i)*:

$$Banda(i) = \left( \frac{BandaTot}{N_i} \right) \quad , \forall i \in \mathbb{I}. \tag{3.5}$$

Le Tabelle 3.1, 3.2 definiscono, rispettivamente, gli indici e i parametri del modello.

| Parametro | Unità di misura | Descrizione |
|---|---|---|
| $Cap$ | kbyte | Capacità *cache* dei *File Server* |
| $Banda(i)$ | kbyte/contatto | Banda garantita all'utente passante per lo $i^{\text{esimo}}$ *File Server* |
| $FS(i,v)$ | contatti | Matrice Contatti per lo $i^{\text{esimo}}$ *File Server* e il $v^{\text{esimo}}$ Percorso |
| $Size(j)$ | kbyte | Dimensione del $j^{\text{esimo}}$ File |
| $Query(j)$ | - | Probabilità di richiesta del $j^{\text{esimo}}$ File |
| $P(v)$ | - | Frequenza del $v^{\text{esimo}}$ Percorso |
| $N_v$ | utenti | Numero di utenti del $v^{\text{esimo}}$ Percorso |
| $N_i$ | utenti | Numero di utenti passanti per lo $i^{\text{esimo}}$ *File Server* |
| $J$ | file | Numero di File |
| $I$ | *file server* | Numero di *File Server* |
| $V$ | percorsi | Numero di Percorsi |

Tabella 3.2: Parametri del modello.

### 3.2.2 Variabili

L'obiettivo è di massimizzare la media pesata della probabilità di recupero dei File nei Percorsi. Essendo lo spazio di archiviazione (*cache*) dei *File Server* limitato, la decisione sarà quella di scegliere la migliore distribuzione di File che andranno messi in ogni *File Server*. File immagazzinati che non saranno necessariamente interi poiché, grazie ai codici a cancellazione [1], ogni File è leggermente più grande ma diviso in blocchi più piccoli. Basterà scaricare, tra un Contatto e l'altro, entro la fine del Percorso, una quantità necessaria di blocchi qualsiasi (del File scelto) per considerare il File recuperato con

successo. La proprietà infatti, che sta alla base dei codici a cancellazione, è che una sequenza potenzialmente infinita di simboli codificati può essere generata a partire da un set fissato di simboli originale (il nostro File), in modo tale che quest'ultimo può essere riottenuto attraverso il recupero di un qualsiasi sottoinsieme, leggermente più grande del set di simboli originale, di simboli codificati. In altre parole è come se fosse sempre possibile recuperare qualsiasi pacchetto mancante del File attraverso l'utilizzo di pacchetti ridondanti inseriti nei blocchi.

| Variabile | Descrizione |
|---|---|
| $X(j,i)$ | Quantità del $j^{\text{esimo}}$ File presente nello $i^{\text{esimo}}$ *File Server* |
| $Y(i,v,j)$ | Quantità del $j^{\text{esimo}}$ File recuperabile nel Contatto $(i,v)$ |
| $B(v,j)$ | Indicatore recuperabilità del $j^{\text{esimo}}$ File nel $v^{\text{esimo}}$ Percorso |

Tabella 3.3: Variabili del modello.

Al nostro scopo ci servirà una variabile, $X(j,i)$, ad indicare la presenza del $j^{\text{esimo}}$ File nello $i^{\text{esimo}}$ *File Server* che, come spiegato precedentemente, non sarà binaria ma potrà essere invece una qualsiasi frazione del File. Assieme avremo bisogno di altre 2 variabili ausiliari.

Con la prima, $Y(i,v,j)$, potremo rappresentare la percentuale totale del $j^{\text{esimo}}$ File che un utente riuscirà a scaricare alla fine del $v^{\text{esimo}}$ Percorso:

$$Y_{j,v} = \sum_{i=1}^{I} Y(i,v,j) \quad ,\forall j \in \mathbb{J}, \forall v \in \mathbb{V}. \tag{3.6}$$

Ad ogni Contatto si potrà ottenere fino a $(Banda(i)/Size(j)) \cdot FS(i,v)$ del $j^{\text{esimo}}$ File ma, allo stesso tempo, si potrà scaricare solo fino a $X(j,i)$ dello stesso. Unendo i due vincoli arriviamo quindi ad un'unica espressione:

$$Y(i,v,j) = \min\left(X(j,i), \left(\frac{Banda(i)}{Size(j)}\right) \cdot FS(i,v)\right). \tag{3.7}$$

Il $j^{\text{esimo}}$ File verrà ritenuto recuperato con successo nel $v^{\text{esimo}}$ Percorso se:

$$Y_{j,v} \geq 1. \tag{3.8}$$

A questo punto abbiamo tutti gli "ingredienti" necessari per formalizzare la funzione obiettivo:

$$\max\left(\sum_{j=1}^{J}\sum_{v=1}^{V} Query(j) \cdot p\left(Y_{j,v} \geq 1\right)\right). \tag{3.9}$$

Volendo però linearizzare il modello bisogna scindere i 2 vincoli precedenti e trovare un metodo per calcolare concretamente la probabilità $p\left(Y_{j,v} \geq 1\right)$ e a questo scopo ci viene in aiuto la seconda variabile binaria: $B(v,j)$. Questa variabile binaria indicherà se nel $v^{\text{esimo}}$ Percorso sarà possibile (1) o impossibile (0) il completo recupero del $j^{\text{esimo}}$ File. In altre parole:

$$B(v,j) = \begin{cases} 1 & \text{, se } Y_{j,v} \geq 1 \\ 0 & \text{, se } Y_{j,v} < 1 \end{cases} \tag{3.10}$$

e possiamo quindi scrivere:

$$p\left(Y_{j,v} \geq 1\right) = P(v) \cdot B(v,j) \tag{3.11}$$

e riformulare definitivamente la funzione obiettivo nel seguente modo:

$$\max \left( \sum_{j=1}^{J} Query(j) \sum_{v=1}^{V} P(v) \cdot B(v,j) \right). \tag{3.12}$$

Nella Tabella 3.3 è presente un riepilogo delle variabili utilizzate nel modello.

### 3.2.3 Vincoli

1. Vincolo di Capacità:

$$Y(i,v,j) \leq \left( \frac{Banda(i)}{Size(j)} \right) \cdot FS(i,v) \quad , \forall j \in \mathbb{J}, \forall (i,v) \in \mathbb{C}. \tag{3.13}$$

   E' il primo dei due vincoli visti precedentemente. Indica che il download, ad ogni Contatto, del File scelto è limitato dalla banda messa a disposizione nei *File Server* incontrati.

2. Vincolo di Reperibilità:

$$Y(i,v,j) \leq X(j,i) \quad , \forall j \in \mathbb{J}, \forall (i,v) \in \mathbb{C}. \tag{3.14}$$

   E' il secondo dei due vincoli visti precedentemente. Collega le due variabili e mostra che il download, ad ogni Contatto, del File scelto è limitato anche dalla scelta della sua distribuzione nei *File Server*.

3. Vincolo di Riuscita:

| **Modello** | |
|---|---|
| **Parametri** | |
| $Cap$ | $\geq 0$ |
| $Banda(i)$ | $\geq 0$ |
| $FS(i,v)$ | $\{0,1\}$ |
| $Size(j)$ | $\geq 0$ |
| $Query(j)$ | $[0,1]$ |
| $P(v)$ | $[0,1]$ |
| **Variabili** | |
| $X(j,i)$ | $[0,1]$ |
| $B(v,j)$ | $\{0,1\}$ |
| $Y(i,v,j)$ | $\geq 0$ |
| **F.Obiettivo** | |
| $\max \sum_{j=1}^{J} Query(j) \sum_{v=1}^{V} P(v) \cdot B(v,j)$ | |
| **Vincoli** | |
| $Y(i,v,j) \leq \left(\frac{Banda(i)}{Size(j)}\right) \cdot FS(i,v)$ | $,\forall j \in \mathbb{J}, \forall (i,v) \in \mathbb{C}$ |
| $Y(i,v,j) \leq X(j,i)$ | $,\forall j \in \mathbb{J}, \forall (i,v) \in \mathbb{C}$ |
| $Y_{j,v} \geq B(v,j)$ | $,\forall v \in \mathbb{V}, \forall j \in \mathbb{J}$ |
| $\sum_{j=1}^{J} Size(j) \cdot X(j,i) \leq Cap$ | $,\forall i \in \mathbb{I}$ |

Tabella 3.4: Il modello matematico.

$$Y_{j,v} \geq B(v,j) \quad ,\forall v \in \mathbb{V}, \forall j \in \mathbb{J}. \tag{3.15}$$

Finisce di collegare l' ultima variabile con le altre due. Indica la possibilità o l' impossibilità del completo recupero del File scelto nel Percorso scelto.

4. Vincolo di Budget:

$$\sum_{j=1}^{J} Size(j) \cdot X(j,i) \leq Cap \quad ,\forall i \in \mathbb{I}. \tag{3.16}$$

Mostra che ogni *File Server* può contenere una quantità limitata di File (o parti di essi).

Nella Tabella 3.4 è possibile rivedere il modello matematico definitivo nel suo insieme.

### 3.2.4 Variante

In questa sezione vogliamo proporre un modello alternativo (Tabella 3.5) a quello visto fin qua per considerare il problema da un altro punto di vista e far emergere nuove considerazioni. Il nostro target sarà ora quello di voler garantire e mantenere un livello accettabile di Probabilità Recupero (*Eff*) cercando di utilizzare il minor spazio possibile nelle *cache* dei *File Server* (*CapTot*). Praticamente il vincolo di budget (Eq. 3.16) è diventato la nuova funzione obiettivo da minimizzare mentre la vecchia funzione obiettivo (Eq. 3.12) è diventata un nuovo vincolo di efficienza. In ultimo abbiamo sostituito il parametro *Cap* con il parametro *Eff*. Nel prossimo paragrafo vediamo invece come *Ying Huang e altri* hanno affrontato il problema, cercando di mostrare le principali differenze.

## 3.3 Contributi innovativi rispetto allo stato dell'arte

In questo paragrafo si vuole analizzare l'approccio in [1] per mettere in risalto le nostre diverse scelte. Come già è stato accennato, la principale differenza è stata quella di formalizzare il problema come un problema di ottimizzazione lineare e di risolverlo con *CPLEX* senza l'aiuto di euristiche. A questa se ne aggiungono però altre che è possibile leggere di seguito.

| **Modello** | |
|---|---|
| **Parametri** | |
| $Eff$ | $[0,1]$ |
| $Banda(i)$ | $\geq 0$ |
| $FS(i,v)$ | $\{0,1\}$ |
| $Size(j)$ | $\geq 0$ |
| $Query(j)$ | $[0,1]$ |
| $P(v)$ | $[0,1]$ |
| **Variabili** | |
| $X(j,i)$ | $[0,1]$ |
| $B(v,j)$ | $\{0,1\}$ |
| $Y(i,v,j)$ | $\geq 0$ |
| **F.Obiettivo** | |
| $\min \sum_{j=1}^{J} \sum_{i=1}^{I} Size(j) \cdot X(j,i)$ | |
| **Vincoli** | |
| $Y(i,v,j) \leq \left(\frac{Banda(i)}{Size(j)}\right) \cdot FS(i,v)$ | $,\forall j \in \mathbb{J}, \forall (i,v) \in \mathbb{C}$ |
| $Y(i,v,j) \leq X(j,i)$ | $,\forall j \in \mathbb{J}, \forall (i,v) \in \mathbb{C}$ |
| $Y_{j,v} \geq B(v,j)$ | $,\forall v \in \mathbb{V}, \forall j \in \mathbb{J}$ |
| $\sum_{j=1}^{J} Query(j) \sum_{v=1}^{V} P(v) \cdot B(v,j) \geq Eff$ | |

Tabella 3.5: Il modello alternativo.

In primo luogo si è scelto di correlare la *Banda(i)* con la quantità di utenti mobili passanti dallo $i^{\text{esimo}}$ *File Server*. Nel modello in [1] invece la *Banda(i)* era un unico valore uguale per tutti i Contatti. Questa semplificazione rispecchiava poco la realtà e allora si è deciso di optare per l'attuale scelta.

In secondo luogo si è deciso di utilizzare un modello di frequenze Percorso teorico al fine di permetterci poi di analizzare ogni possibile casistica. *Ying Huang e altri* propongono invece un algoritmo per il calcolo di queste frequenze basandosi comunque su un record dati di accessi agli *Access Point*. Vengono inoltre distinti due diversi tipi di Percorso: viaggio (una lunga serie di *Access Point* incontrati) e contatto canonico (una singola coppia di *Access Point* incontrati).

La modifica successiva è stata poi quella di stravolgere il modello. Sostanzialmente, "invertendo" il vincolo di budget (Eq. 3.16) con la funzione obiettivo (Eq. 3.12), è stato possibile far passare in secondo piano la qualità dell'intero sistema. Creando, a tutti gli effetti, un modello alternativo, ci si è potuti quindi concentrare su un altro problema, ovvero, la limitatezza dello spazio in memoria dei *File Server*.

Tornando invece alla principale differenza, cerchiamo di spiegare i motivi che hanno portato a tale scelta mostrando ora lo stato dell'arte attuale. Secondo *Ying Huang e altri* sono stati proposti molti algoritmi e strumenti efficienti per risolvere il problema di programmazione lineare. Tuttavia, vi sono ancora due sfide per risolvere il problema MIP proposto: le dimensioni del problema e il requisito di numero intero di $B(v, j)$. Innanzitutto, ci si aspetta un gran numero di variabili. Ad esempio, per una rete con decine di *File Server*, esisterebbero una centinaia di scelte di Percorso. Con un migliaio di File si arriverebbe poi ad un numero di variabili binarie $B(v, j)$ di oltre 100 000. Ciò sarebbe inefficiente e infattibile per problemi di memoria con gli strumenti di ottimizzazione correnti, come *MatLab* e *CPLEX*, al fine di trovare buone soluzioni. Poi, il vincolo di integralità di $B(v, j)$ impedisce l'utilizzo di tecniche standard di programmazione lineare. Questo infatti lo renderebbe un problema NP-difficile. Trovare l'esatta soluzione ottimale potrebbe richiedere un numero esponenziale di iterazioni. Pertanto, viene proposta una euristica per determinare $X(j, i)$, tenendo in considerazione il modello di mobilità. Viene quindi creato e utilizzato *MobaSsign*, un algoritmo di assegnazione blocchi secondo mobilità. Successivamente vengono studiate le sue prestazioni sia attraverso una simulazione che teoreticamente. Le prime analisi hanno l'obiettivo di conoscere le reali proporzioni tra le variabili in gioco. Le seconde invece mostrano come queste ultime interagiscono tra di loro.

In Algorithm 1 possiamo vedere dettagliatamente i passi dell'euristica, riadattata con le nostre notazioni. L'idea di base è di ordinare in modo decrescente tutti i possibili $B(v, j)$ in base a una funzione *utility* per cercare poi

**Algorithm 1:** *MobaSsign.*

**Output**: $X(j,i)$

$X(j,i) = 0 \quad , \forall j \in \mathbb{J}, \forall i \in \mathbb{I}$;

Calcola utility di $B(v,j)$ come:

$$u(B(v,j)) = \begin{cases} \frac{Query(j) \cdot P(v)}{Size(j)} & , \text{se } \left(\sum_{i=1}^{I} Banda(i) \cdot FS(i,v)\right) \geq Size(j); \\ 0 & , \text{altrimenti.} \end{cases}$$

Ordina tutti i $B(v,j)$ in modo decrescente secondo il valore di utility, ($B(v_k, j_k)$: $k = 1$ è il primo, $k = (V \cdot J)$ è l'ultimo);

**for** $k = 1$ *to* $(V \cdot J)$ **do**

  **if** $\left(\sum_{i=1}^{I} FS(i, v_k) \cdot X(j_k, i) \cdot Size(j_k)\right) \geq Size(j_k)$ **then**

    continue;

  **end**

  **for** $i = 1$ *to* $I$ **do**

    $nx(i) =$
    $\max\left(\left(Banda(i) \cdot FS(i, v_k) / \sum_{i=1}^{I} Banda(i) \cdot FS(i, v_k)\right), X(j_k, i)\right)$;

    **if** $Size(j_k) \cdot (nx(i) - X(j_k, i)) \leq Cap - \sum_{j=1}^{J} X(j,i) \cdot Size(j)$ **then**

      $X(j_k, i) = nx(i)$;

    **end**

  **end**

**end**

di allocare, uno alla volta, finché sia possibile, gli $X(j,i)$ necessari. La $k^{\text{esima}}$ $u(B(v_k, j_k))$ è positiva solo se il Percorso $v_k^{\text{esimo}}$ consente all'utente di scaricare il numero di blocchi necessari per ottenere il $j_k^{\text{esimo}}$ File ($B(v_k, j_k) = 1$). $u(B(v_k, j_k))$ rispecchia il relativo beneficio, ovvero la probabilità di recupero, proporzionale al prodotto tra $Query(j_k)$ e $P(v_k)$. Il costo è invece lo spazio di archiviazione occupato, ovvero $Size(j_k)$ (linea 3). Se $B(v_k, j_k)$ può essere soddisfatto attraverso allocazioni preesistenti di $X(j,i)$, non viene fatto nessun tipo di aggiustamento (linea 6). In caso contrario si cercherà di soddisfarlo. Complessivamente c'è bisogno di solo una copia del $j_k^{\text{esimo}}$ File e quindi vengono assegnate parti del File nei *File Server* contattati proporzionali alle bande garantite di quest'ultimi. I nuovi blocchi di allocazione, $nx(i)$, vengono calcolati come il massimo tra gli $X(j_k, i)$ preesistenti e l'assegnazione proporzionale precedente (linea 10). Solo se questo ulteriore spazio, necessario per soddisfare $B(v_k, j_k)$, non supera la memoria disponibile di ogni *File Server*, possiamo sostituire i vecchi $X(j_k, i)$ con $nx(i)$ (linea 11-12). La complessità dell'algoritmo sarà quindi un $\mathrm{O}(V \cdot J)$.

La nostra scelta è stata quindi quella di continuare a utilizzare il modello lineare base. Come vedremo, è stato sempre possibile raggiungere la soluzione ottima. Pensiamo poi di garantire risultati più precisi basandoci su degli ottimi e non su euristiche, comunque soddisfacenti. Nel prossimo capitolo è possibile vedere le nostre analisi che hanno l'obiettivo di approfondire ulteriormente gli studi svolti. Per non complicare troppo lo scenario è stato scelto un numero di variabili limitato ma comunque utile, ovvero seguendo grossomodo le proporzioni guida, al fine della valutazione dei risultati. Ribadiamo però che questo non significa che, aumentando le variabili in gioco, non si possa più ottenere la soluzione ottimale.

# Capitolo 4

# Valutazione delle prestazioni

In questo capitolo viene proposta la nostra analisi dettagliata ai modelli visti in precedenza. Lo scopo è quello di trovare, attraverso delle simulazioni, le caratteristiche e le migliori impostazioni per una possibile attuazione in uno scenario realistico.

## 4.1 Scenario Simulativo

Dopo aver scelto e modificato il modello a livello matematico, come si è ampiamente spiegato nel precedente capitolo, si è scelto ora di formalizzarlo con *AMPL*. Questo ci ha messo in grado di risolverlo attraverso *CPLEX*, al fine di effettuare poi una pratica e precisa analisi, al variare dei parametri. Per velocizzare allora il processo di simulazione, si è deciso di creare un motore per la generazione della configurazione di rete attraverso *MatLab*. Quest'ultimo ci ha permesso di creare facilmente moltissimi scenari con diverse configurazioni, da dare "in pasto" a *CPLEX*, in modo da far emergere risultati utili alla nostra analisi. Nel prossimo paragrafo vengono mostrati nel dettaglio gli aspetti più rilevanti che è stato possibile ricavare attraverso questi risultati.

## 4.2 Analisi

### 4.2.1 *BandaTot* e *Cap*: due parametri limitanti

Come prima cosa vogliamo capire come reagisce il sistema all'aumentare del traffico di utenti nell'area. Utilizziamo come indicatore della salute del sistema la media pesata della probabilità di recupero dei File nei Percorsi che da qui in avanti chiameremo semplicemente Probabilità Recupero. All'aumentare di

$N$ ($n^\circ$ utenti in tutta l'area) vedremo come influisce in un primo esempio la *BandaTot* e in un secondo la *Cap* dei *File Server*.

| Dati | |
|---|---|
| $J$ | 50 |
| $Size(j)$ | (600,500,400,300,200,100) |
| $Query(j)$ | $Zipf$, $a = 0.6$, (0,0995 - 0,0095) |
| $I$ | 5 |
| $Cap$ | $5{\cdot}10^3$ |
| $V$ | 16 |
| $P(v)$ | $Zipf$, $a = 0.6$, (0,1748 - 0,0331) |

Tabella 4.1: *BandaTot* - Configurazioni.

Nel primo caso (Tabella 4.1) abbiamo 5 *File Server*, 16 Percorsi e 50 possibili File da poter inserire nella *cache* di ogni *File Server*. La matrice dei Contatti è stata scelta casualmente. Il peso dei File varia da un massimo di 600 [kbyte] ad un minimo di 100 [kbyte]. La capacità di *cache* di ogni *File Server* è di $5{\cdot}10^3$ [kbyte]. La popolarità dei File e la frequenza dei Percorsi sono state generate con la distribuzione *Zipf*. La probabilità di richiesta del $j^{\text{esimo}}$ File è: $Query(j) = \frac{1}{j^a} / \sum_{j=1}^{50} \frac{1}{j^a}$ , dove $a$ è il parametro configurabile della distribuzione *Zipf*. Nello stesso modo la frequenza del $v^{\text{esimo}}$ Percorso è: $P(v) = \frac{1}{v^a} / \sum_{v=1}^{16} \frac{1}{v^a}$.

Per ogni curva abbiamo impostato un diverso valore di *BandaTot* [kbyte/ contatto], ovvero quella banda di ogni *File Server* che andrà poi divisa tra gli $N$ utenti. Notiamo dalla Figura 4.1 che con questa configurazione è impossibile soddisfare le richieste di tutti i File su tutti i Percorsi (Caso ideale: Probabilità Recupero = 1). Questa limitatezza è dovuta probabilmente ad un valore di *Cap* non grande a sufficienza. Il nostro massimo, Probabilità Recupero = 0,7241 , indica però una qualità accettabile del sistema. Questa vediamo che è garantita da tutte le curve, ad eccezione di una ($10^4$), quando il sistema è "scarico", ovvero quando ci sono utenti in tutta l'area sull'ordine delle centinaia. All'aumentare di $N$, e quindi all'aumentare dell'interferenza tra utenti, si può vedere che sistemi con *BandaTot* alta degradano di poco, garantendo comunque il servizio, mentre sistemi con *BandaTot* bassa crollano dopo poco, rendendo impossibile il recupero di qualsiasi File. Questo accade perché nei Contatti si ha sempre meno banda a disposizione e alla fine dei Percorsi difatti non si riesce a completare il recupero di buona parte dei File richiesti.

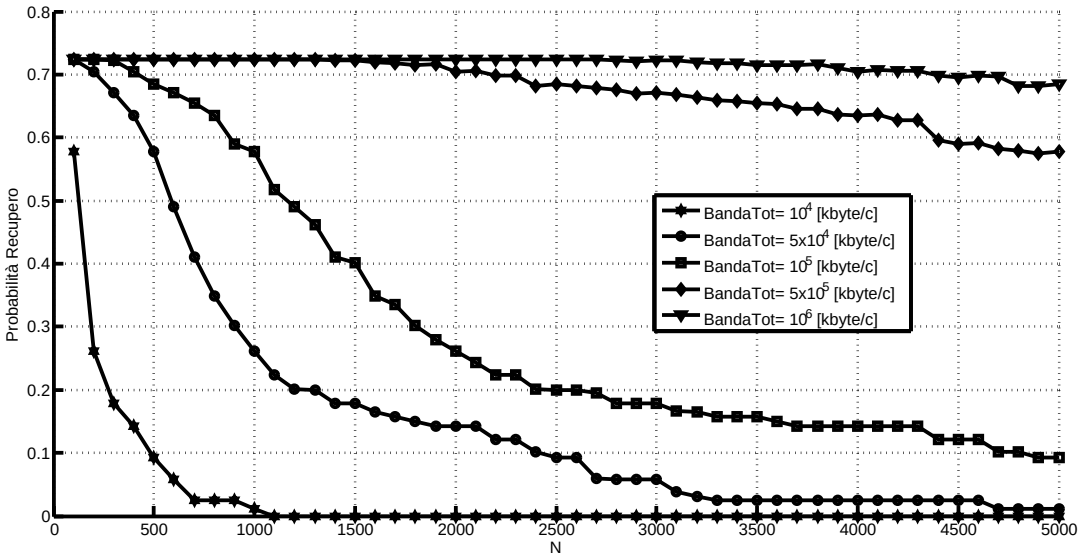



Figura 4.1: Probabilità di recupero al variare del numero di utenti $N$ per diversi valori di banda totale disponibile, $BandaTot$.

| Dati | |
|---|---|
| $J$ | 50 |
| $Size(j)$ | (600,500,400,300,200,100) |
| $Query(j)$ | $Zipf$, $a = 0.6$, (0,0995 - 0,0095) |
| $I$ | 5 |
| $BandaTot$ | $2{\cdot}10^5$ |
| $V$ | 16 |
| $P(v)$ | $Zipf$, $a = 0.6$, (0,1748 - 0,0331) |

Tabella 4.2: *Cap* - Configurazioni.

Nel secondo caso (Tabella 4.2) abbiamo tenuto principalmente le stesse impostazioni. Questa volta però fissiamo la *BandaTot* e mostriamo la situazione al variare di *Cap* [kbyte].

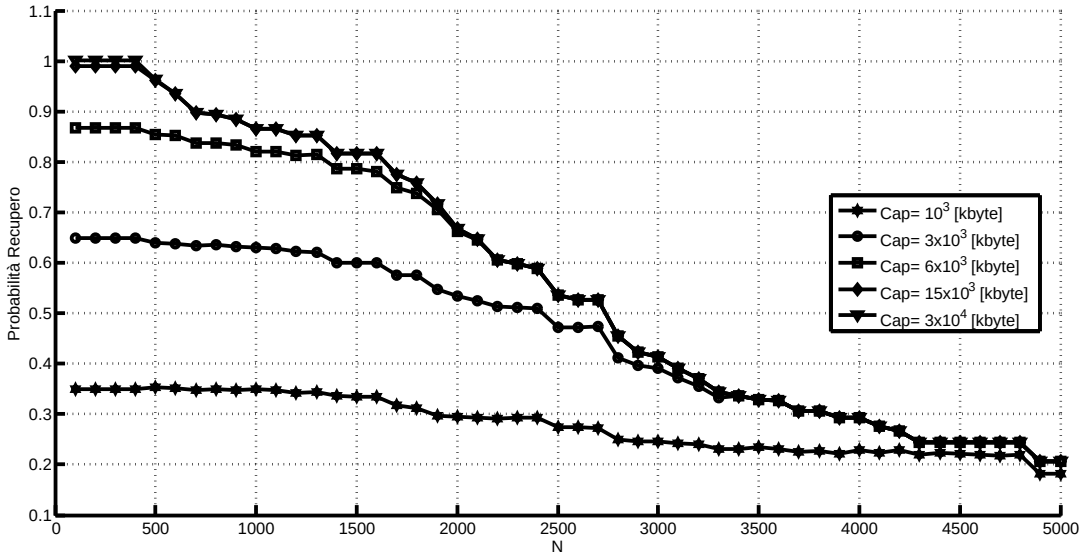



Figura 4.2: Probabilità di recupero al variare del numero di utenti $N$ per diversi valori della capacità di *storage* dei *file server*, *Cap*.

Dalla Figura 4.2 si può notare che all'aumentare di $N$ il sistema si degrada come nel caso precedente e che più grande è la variabile *Cap* più la soluzione è efficiente. Sarà infatti possibile immagazzinare più contenuti nei *File Server*. Questa dipendenza risulta rilevante quando abbiamo un $N$ piccolo. Al contrario, quando ci saranno parecchi utenti nell'area, avere una capacità di *cache* dei *File Server* più grande porta pochi benefici.

### 4.2.2 Local Dependent Vs Local Independent

In questa sezione vogliamo confrontare due ipotetici scenari diversi. Quello che li differenzia è una caratteristica della *Query*, ovvero della classifica dei File più richiesti. Il primo ha la particolarità di avere lo stesso ordine in classifica su ogni Percorso, quindi un'unica classifica per tutta l'area di interesse. Il secondo invece ha classifiche differenti nei Percorsi. Potrà capitare ad esempio che il File più popolare in un Percorso sia uno dei meno richiesti in un altro Percorso. Il primo caso, che potrebbe rappresentare un'area molto piccola in cui gli utenti richiedono le stesse informazioni sul traffico o più in generale

hanno gli stessi "gusti", lo etichettiamo come "Local Dependent". Mano a mano che consideriamo un'area sempre più vasta sarà invece logico pensare che questa dipendenza possa non esserci più. Questo è il nostro secondo caso ("Local Independent").

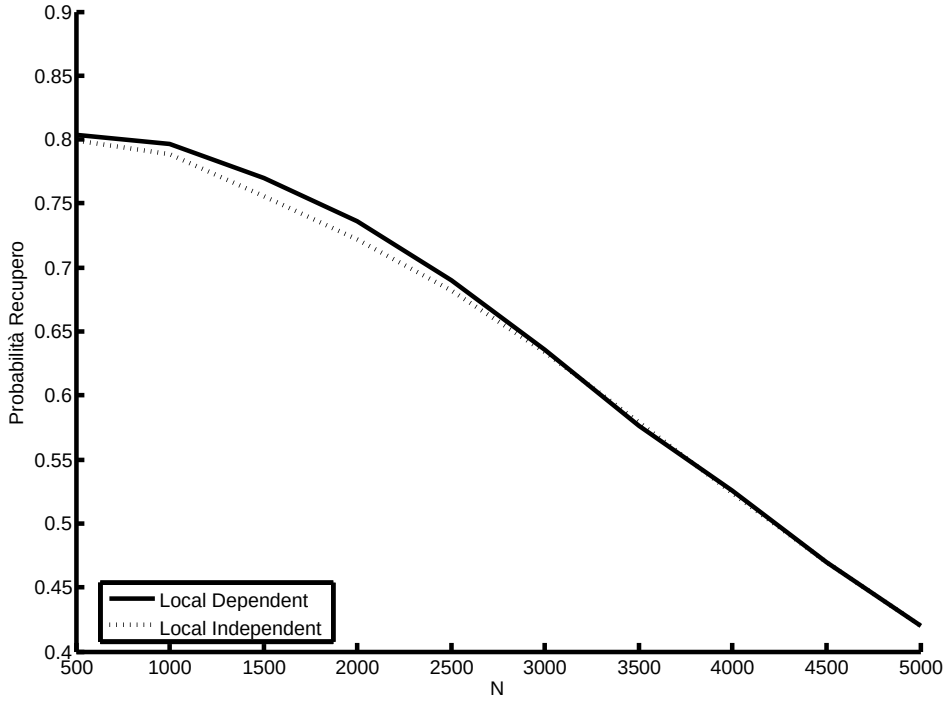


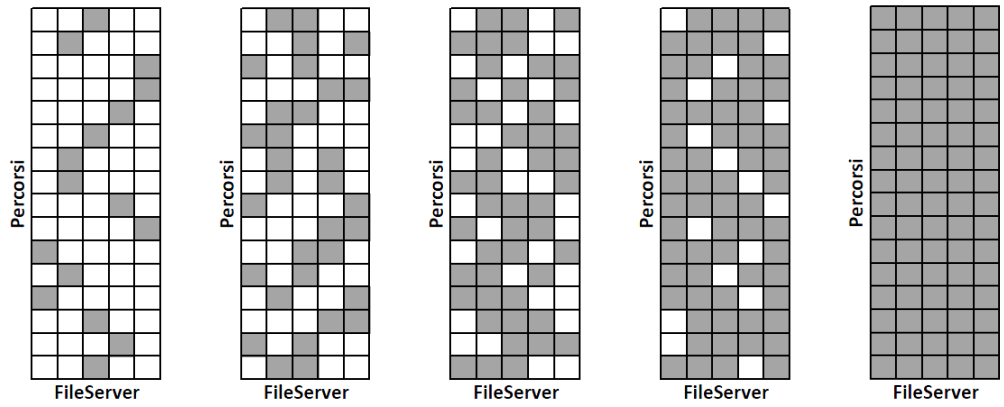

Figura 4.3: Probabilità di recupero al variare del numero di utenti $N$, casi *Local Dependent* e *Local Independent*.

Le configurazioni principali rimangono simili a quelle della sezione precedente (Tabelle 4.1, 4.2). Per non avere casi particolari si è preferito però mediare su 50 realizzazioni con diversi valori di $a$ (*Query*) e diverse matrici Contatto. Dalla Figura 4.3 abbiamo una conferma di quello che ci aspettavamo. Grazie a questa proprietà in più il caso "Local Dependent" risulta infatti più efficiente. Questo vantaggio, come si può notare, è però veramente limitato. Da $N$= 3000 in poi non riusciamo più ad individuare alcuna differenza trai due casi.

### 4.2.3 Pochi Contatti Vs Molti Contatti

In questa sezione vogliamo effettuare uno studio sulla matrice Contatti. Ci chiediamo quale sia la migliore configurazione, ovvero quella che massimizza

la Probabilità Recupero. Per intenderci, è meglio avere a disposizione tanti Contatti nei Percorsi per dare più possibilità di completare il recupero dei File o pochi Contatti per non intasare i *File Server* e garantire più banda nei Contatti? Questa è la domanda alla quale vogliamo rispondere. Quello che possiamo immaginare è che non ci sia una risposta che vada bene per ogni scenario. Proviamo quindi a osservare il caso scelto per poi cercare, con i risultati dei test, di generalizzare.




(a) 1 (b) 2 (c) 3 (d) 4 (e) 5

Figura 4.4: Schemi matrice contatti con diverso numero di contatti, uguali per ogni percorso.

Le configurazioni principali rimangono ancora simili a quelle di una sezione precedente (Tabelle 4.1, 4.2). Fissando anche $N = 1000$, vogliamo quindi far variare la struttura della matrice Contatti. Decidiamo di usare le cinque tipologie schematizzate in Figura 4.4. Ogni quadratino colorato indica la presenza di un Contatto e ogni tipologia, creata casualmente, ha l'unico vincolo di avere lo stesso numero di Contatti per ogni Percorso. Per la prima (4.4a) abbiamo un solo Contatto per ogni Percorso, per la seconda (4.4b) due, per la terza (4.4c) tre, per la quarta (4.4d) quattro e infine per l'ultima (4.4e) cinque Contatti ovvero una matrice Contatti piena.

Anche in questo caso si è deciso di mediare su 50 realizzazioni e quindi avere probabilmente 50 diverse matrici per ognuna delle cinque categorie. Questo è servito per evitare rischi di valutare configurazioni favorevoli di una categoria con quelle sfavorevoli di un'altra e di conseguenza di falsare i risultati.

Dal grafico in Figura 4.5 possiamo allora vedere che con un solo Contatto abbiamo le prestazioni peggiori, mentre per gli altri quattro la Probabilità di Recupero rimane pressapoco nello stesso intorno. La configurazione migliore

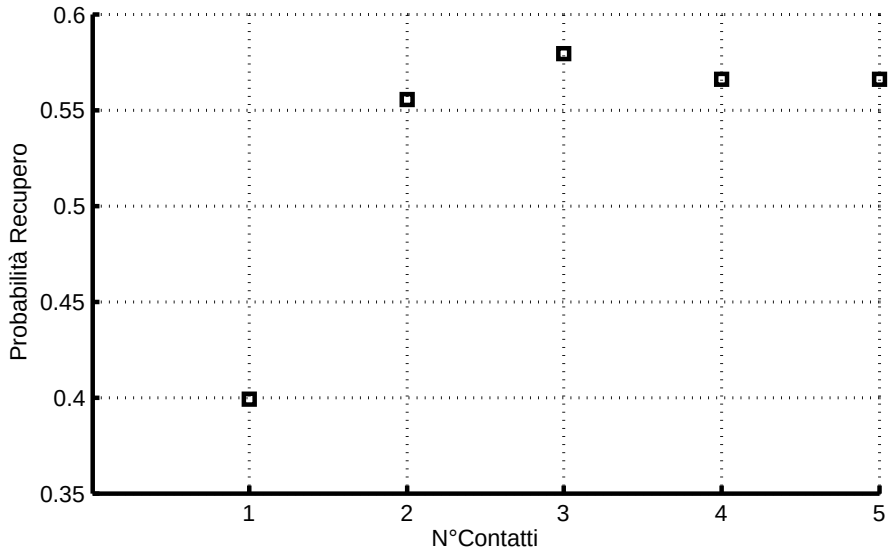



Figura 4.5: Probabilità di recupero al variare del numero di contatti, uguali per ogni percorso.

risulta essere di poco quella di mezzo con 3 Contatti per ogni Percorso. Con un po' più di attenzione potremmo allora dividere il grafico in due fasi, una crescente e una decrescente. Nella prima, da zero a 3 Contatti, incrementare sempre più i Contatti porta dei grossi benefici in termini di opportunità per il recupero dei File. Nella seconda, da 3 a 5 Contatti, invece, questi benefici sembrano non esserci, perché si è arrivati in una zona in cui aumentare i Contatti fa prevalere principalmente l'intasamento dei *File Server* e quindi la diminuzione della loro banda garantita. In altre parole si arriva sì ad avere 5 Contatti per ogni Percorso, ma nello stesso tempo, la somma dei download, più limitati rispetto a prima, nei Contatti non ci permette di completare il recupero di alcuni File, che nella configurazione migliore era possibile recuperare. Generalizzando possiamo dire che l'ottimo è un giusto compromesso tra questi due fattori correlati: il numero dei Contatti e la banda a disposizione dei *File Server*. Questi, messi a confronto con i valori degli altri parametri, determinano le due zone. Nel caso limite di *BandaTot* infinita sarà logico pensare invece ad un andamento ad un'unica fase crescente in cui il massimo si avrà con Matrice Contatti piena.

### 4.2.4 Distribuzioni *Query* e *Size*

In questa sezione vogliamo vedere quanto influiscono la *Query* e la *Size* sulle prestazioni e sulla scelta soprattutto di distribuzione dei File nella *cache* dei *File Server*. Decidiamo quindi di apportare modifiche alle configurazioni del nostro scenario per cercare di far emergere risultati utili. Le configurazioni, dove non specificato, rimangono quelle delle sezioni precedenti. Gli approcci scelti sono principalmente due.

#### Primo approccio

Consideriamo 30 File divisi in base alla *Size* in tre categorie (vedi Figura 4.6): 10 grandi (es. 600 [kbyte]), 10 medi (es. 400 [kbyte]) e 10 piccoli (es. 200 [kbyte]). Ad ogni categoria associamo, a seconda dei casi, un diverso livello di *Query*. Caso 1 (Uniforme): ogni File ha la stessa probabilità di esser richiesto. Caso 2 (Proporzionale alla *Size*): i File grandi sono più richiesti, i File medi sono mediamente richiesti e i File piccoli sono meno richiesti. Caso 3 (Inversamente proporzionale alla *Size*): i File grandi sono meno richiesti, i File medi sono mediamente richiesti e i File piccoli sono più richiesti.

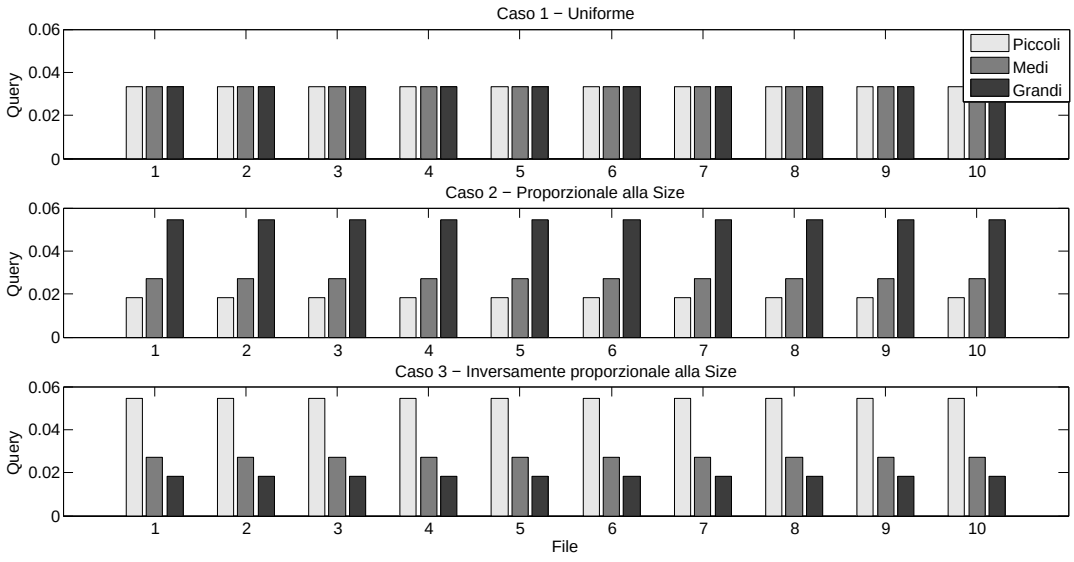



Figura 4.6: Probabilità di richiesta *query* dei file per tre diversi casi in base alla dimensione *size* dei file, primo approccio.

Questa volta si è preferito seguire l'andamento dei tre casi al variare del valore di *cache* dei *File Server* perché, oltre alle prestazioni, ci interessa poi

sapere, di volta in volta, il contenuto di queste *cache*. Nel grafico in Figura 4.7 abbiamo i risultati di due diverse configurazioni. Linea continua: *Size* = (300, 200, 100) [kbyte]. Linea tratteggiata: *Size* = (600, 400, 200) [kbyte]. La prima configurazione ha intuitivamente una migliore performance. Si può vedere che, sia in uno che nell'altro, il caso 3 è quello migliore, il caso 2 è quello più sfavorevole e il caso 1 risulta stare trai due. A pensarci bene, abbiamo poche sorprese nel vedere che nella prima fase, in cui siamo limitati da *Cap*, si verifichi ciò. E' logico pensare infatti che contenuti piccoli e più richiesti sono più facili da gestire rispetto a contenuti grandi e più richiesti. Quello che potrebbe sembrare strano è che a valori elevati di *Cap* ,quindi fuori dalla sua zona di dipendenza, le tre curve non si uniscano. Anche in questo caso però non c'è da stupirsi perché, pur non essendo più vincolati dalla scelta di distribuzione dei contenuti nella *cache* ( in ogni *cache* infatti avremo a disposizione tutti i File), ci possono essere sempre Percorsi in cui non è possibile recuperare File grandi. Questi mancati recuperi hanno differenti "pesi" ed è per questo che le curve tendono poi a mantenere quel costante divario. Come controprova se aumentassimo la *BandaTot* fino ad un valore, tale per cui poi sarebbe necessario un solo contatto per recuperare un File grande, verificheremmo certamente l'unione delle curve.

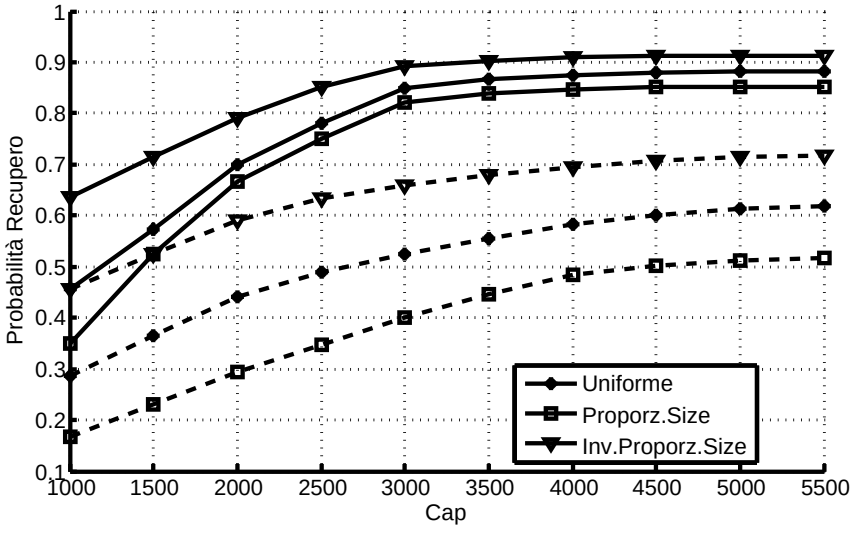



Figura 4.7: Probabilità di recupero al variare della capacità di *storage* dei *file server* per i tre casi precedenti e per due diverse configurazioni di *size* dei file, primo approccio.

Concentriamoci ora solo sulla configurazione *Size* = (300,200,100) [kbyte] e proviamo a scoprire a questo punto cosa troviamo nella *cache* dei *File Server*. Ricordando di aver mediato ancora su 50 realizzazioni con diverse matrici Contatti, per i soliti motivi spiegati nelle sezioni precedenti, possiamo vedere all'interno di uno qualsiasi, risultando simili, dei cinque *File Server*. Nel grafico in Figura 4.8 è rappresentato l'andamento delle percentuali medie dei File immagazzinati nei *File Server* divise per *Size* (G: grandi, M: medi, P: piccoli) e per categorie di *Query* (Uni: caso 1, Pro: caso 2, Inv: caso 3). Con piccoli valori di *Cap* lo scenario sembra abbastanza vario e difficile da comprendere. Già da *Cap* = 3000 kbyte circa possiamo invece notare un assestamento. Non c'è più dipendenza in base alla *Query* e le nove curve seguono grossomodo tre uniche linee guida in base solo alla dimensione. A questo punto con l'aumentare di *Cap* crescono queste percentuali e arriviamo ai seguenti valori: percentuale media di File grande = 61%, percentuale media di File medio = 90%, percentuale media di File piccolo = 100%. Ricordiamo ancora però che si tratta soltanto di una media su più realizzazioni e non ho detto quindi che nella singola realizzazione non si possano trovare percentuali che si discostino molto tra un *File Server* e un altro.

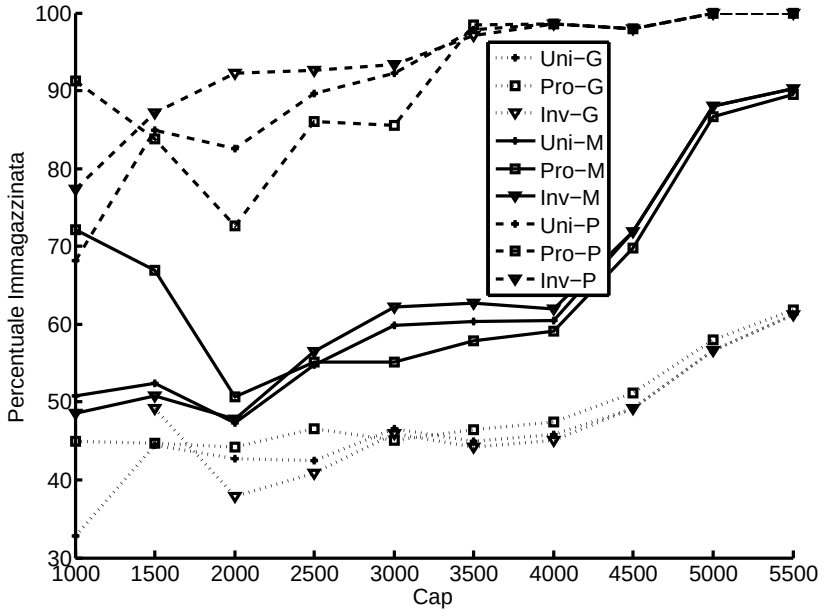



Figura 4.8: Percentuali medie di un file immagazzinato nei *file server* al variare della capacità di *storage* per ogni combinazione (caso-*size*), primo approccio.

Nel grafico in Figura 4.9 mostriamo invece la distribuzione della quantità delle tre categorie di File nel *File Server* per i tre casi al variare di *Cap*. In altre parole vogliamo sapere l'evoluzione di quanto spazio della *cache* viene riservato per i File piccoli, quanto per i File medi e quanto per quelli grandi. Si è deciso di trasformare tutto in percentuale sul totale dello spazio usato per un miglior impatto visivo. La cosa curiosa, che a questo punto però non si può vedere ma che comunque è stata appresa durante i test, è che lo spazio usato è uguale allo spazio disponibile solo inizialmente. Già a partire da *Cap* = 3000 [kbyte] abbiamo notato che il *File Server* non satura la sua capacità. Essendo una media su più realizzazioni può voler dire che non tutti i *File Server* saturano la capacità. Questo fatto non è per nulla scontato e ci fa capire che il nostro modello non fa inserire File nel generico *File Server*, anche se ha spazio libero, se è a conoscenza di non riuscire a completare il loro recupero nei percorsi passanti da questo. Ciò potrebbe essere un problema solo per quegli scenari in cui ci siano valori così alti di *Cap* che praticamente non vincolano il sistema.

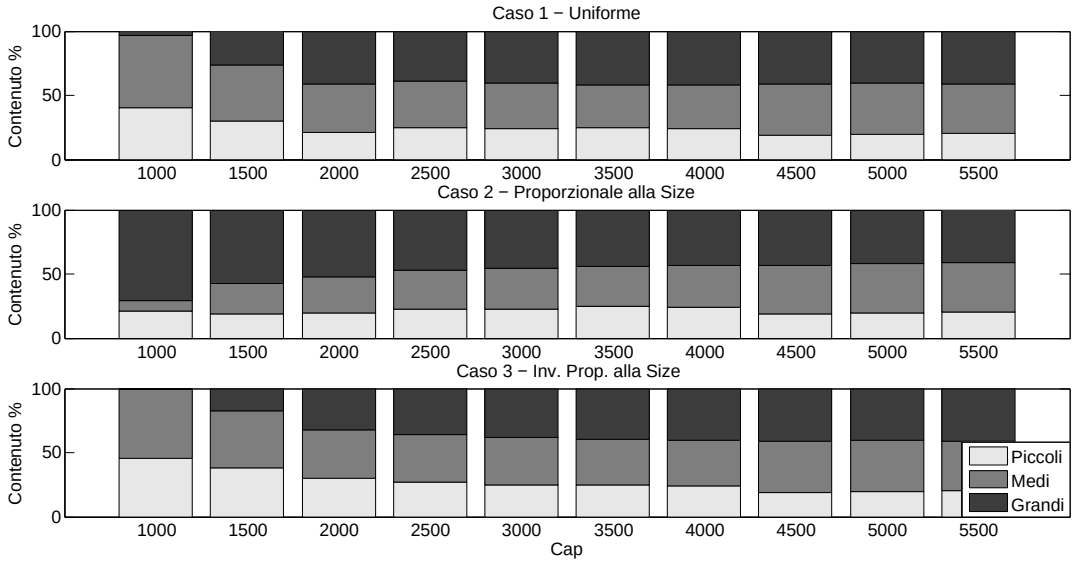



Figura 4.9: Percentuali, in base alla *size* dei file, del contenuto del *file server* al variare della sua capacità di *storage* per i tre casi, primo approccio.

Nell'altro grafico in Figura 4.10 invece si può vedere quanti File sono presenti nel *File Server* sempre al variare di *Cap*. Nel computo vengono considerati anche quei File con solo una piccola percentuale immagazzinata. Da questi due ultimi grafici emergono le stesse considerazioni. Notiamo infatti che caso 1 e caso 3 hanno un andamento molto simile sia in termini di quantità

che di numero di File. Logicamente il caso 3, che è più portato a preoccuparsi di completare il recupero dei File piccoli, all'inizio ha il maggior numero e la maggior quantità di File piccoli, poi però segue pressapoco l'andamento del caso 1. Per quanto riguarda il caso 2 saltano subito all'occhio nelle due figure gli andamenti iniziali completamente diversi rispetto agli altri due casi.

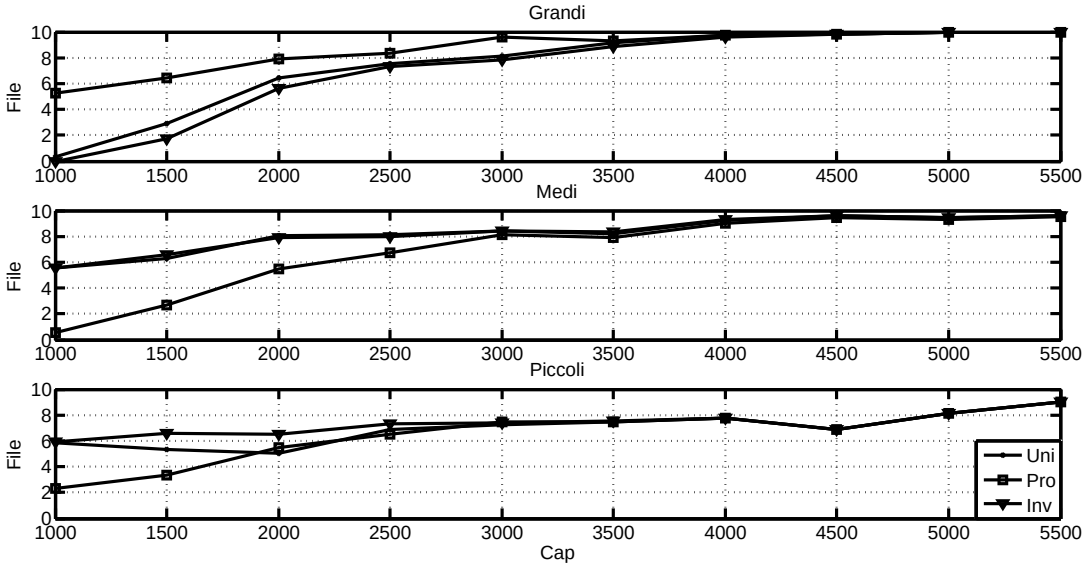



Figura 4.10: Numero di file presenti nel *file server* al variare della sua capacità di *storage* per i tre casi e per le *size* dei file, primo approccio.

### Secondo approccio

Consideriamo ancora 30 File divisi in base alla *Size* in tre categorie (vedi Figura 4.11): 10 grandi (es. 600 [kbyte]), 10 medi (es. 400 [kbyte]) e 10 piccoli (es. 200 [kbyte]). Questa volta ad ogni categoria associamo stesse distribuzioni di *Query*. Queste ultime le facciamo variare, a seconda dei casi, agendo sul parametro $a$ della distribuzione *Zipf*. Caso 1 (Uniforme - *Zipf*,$a = 0$): la probabilità di richiesta, per ogni categoria, del $j^{\text{esimo}}$ File è: *Query(j)* $= (\frac{1}{j^0} / \sum_{j=1}^{10} \frac{1}{j^0})/3$. Ogni File ha la stessa probabilità di esser richiesto. Caso 2 (*Zipf*,$a = 1$): la probabilità di richiesta, per ogni categoria, del $j^{\text{esimo}}$ File è: *Query(j)* $= (\frac{1}{j^1} / \sum_{j=1}^{10} \frac{1}{j^1})/3$. Caso 3 (*Zipf*,$a = 2$): la probabilità di richiesta, per ogni categoria, del $j^{\text{esimo}}$ File è: *Query(j)* $= (\frac{1}{j^2} / \sum_{j=1}^{10} \frac{1}{j^2})/3$. Praticamente più $a$ è grande più la differenza tra il più richiesto e il meno richiesto è grande, ovvero abbiamo una distribuzione di *Query* più "inclinata".

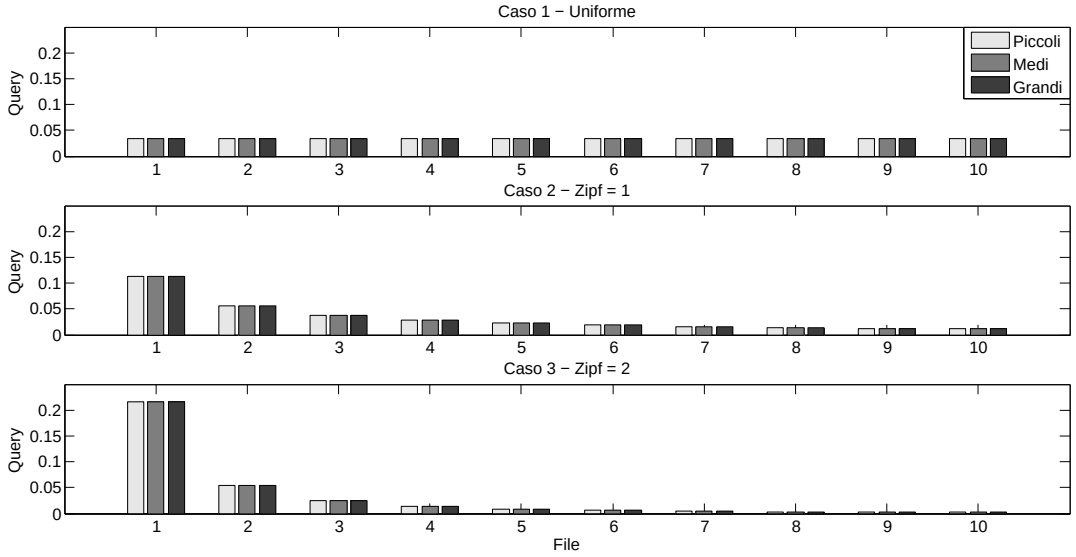



Figura 4.11: Tre diverse distribuzioni (casi) di probabilità di richiesta *query* dei file uguali per ogni *size*, secondo approccio.

Ancora una volta si è scelto di seguire l'andamento dei tre casi al variare del valore di *cache* dei *File Server* per poter poi esaminare il contenuto di queste *cache*. Nel grafico in Figura 4.12 abbiamo i risultati delle due configurazioni. Linea continua: *Size* = (300, 200, 100) [kbyte]. Linea tratteggiata: *Size* = (600, 400, 200) [kbyte]. La prima configurazione si conferma ancora come quella con migliore performance. In entrambi i casi abbiamo che più la distribuzione è sbilanciata più abbiamo dei vantaggi nella prima fase. Questo accade perché, avendo delle *cache* limitate, si possono inserire e recuperare solamente alcuni File. Logicamente vengono scelti i File più richiesti. Questi File posizionati meglio in classifica hanno però "pesi" diversi a seconda dei casi. Nel caso 3, il più sbilanciato dei tre, è proprio la garanzia di recupero di questi primi File che determina la maggior parte della qualità del sistema. Con l'aumentare di *Cap*, e quindi con la possibilità di recuperare altri File, si può vedere infatti che il caso 3, rispetto agli altri due, incrementa solo di poco la Probabilità Recupero perché garantisce il recupero di altri File con pesi già poco rilevanti. Diversamente dal primo approccio poi, tutti e tre, una volta usciti dalla fase in cui siamo limitati da *Cap*, raggiungeranno sempre lo stesso limite. Questa volta infatti, pur essendoci sempre Percorsi in cui non è possibile recuperare File grandi, il mancato recupero di questi penalizzerà in ugual modo i tre casi. Per convincercene basta verificare che la somma delle

*Query* dei File grandi è sempre uguale a $\frac{1}{3}$ in tutti i tre casi.

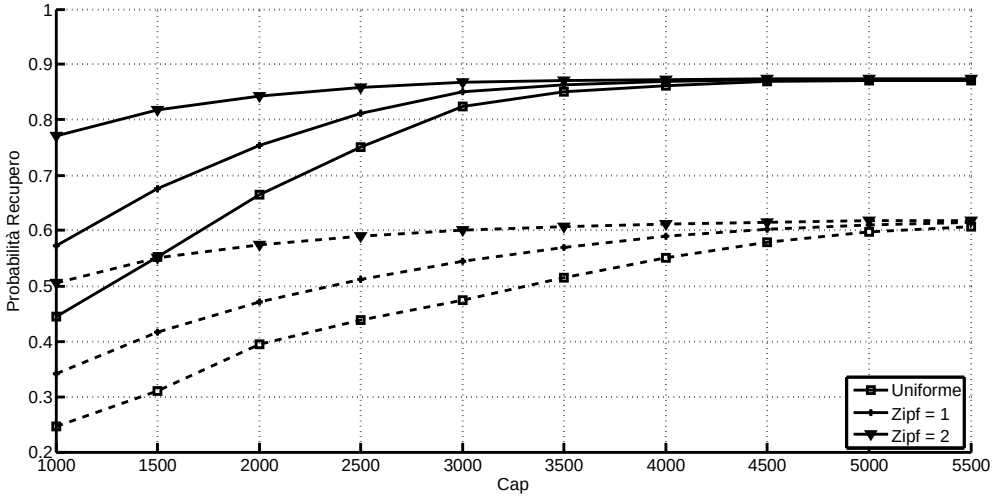



Figura 4.12: Probabilità di recupero al variare della capacità di *storage* dei *file server* per i tre casi precedenti e per due diverse configurazioni di *size* dei file, secondo approccio.

Consideriamo ora solo la configurazione *Size* = (300, 200, 100) [kbyte] come prima per provare a scoprire cosa troviamo nella *cache* dei *File Server*. Ricordando di aver mediato ancora su 50 realizzazioni con diverse matrici Contatti, per i soliti motivi spiegati nelle sezioni precedenti, possiamo vedere all'interno di uno qualsiasi, risultando simili, dei cinque *File Server*. Nel grafico in Figura 4.13 è rappresentato l'andamento delle percentuali medie dei File immagazzinati nei *File Server* divise per *Size* (G: grandi, M: medi, P: piccoli) e per categorie di *Query* (Z0: caso 1, Z1: caso 2, Z2: caso 3). Con piccoli valori di *Cap* lo scenario di nuovo sembra abbastanza vario e difficile da comprendere. Già da *Cap* = 3500 [kbyte] circa possiamo invece notare il solito assestamento. Non c'è più dipendenza in base alla *Query* e le nove curve seguono grossomodo tre uniche linee guida in base solo alla dimensione. A questo punto con l'aumentare di *Cap* crescono queste percentuali e arriviamo ai seguenti valori: percentuale media di File grande = 63%, percentuale media di File medio = 91%, percentuale media di File piccolo = 100%. Ricordiamo ancora però che si tratta soltanto di una media su più realizzazioni e non ho detto quindi che nella singola realizzazione non si possano trovare percentuali che si discostino molto tra un *File Server* e un altro.

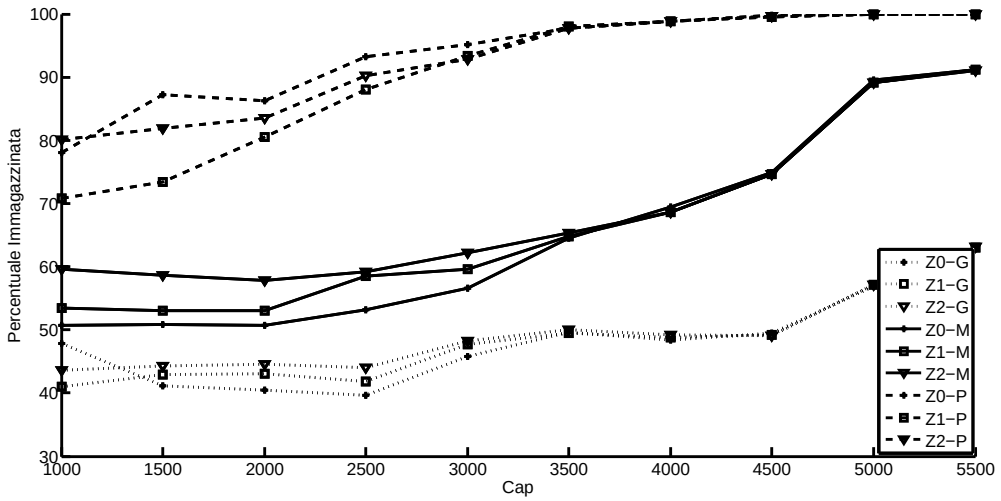



Figura 4.13: Percentuali medie di un file immagazzinato nei *file server* al variare della capacità di *storage* per ogni combinazione (caso-*size*), secondo approccio.

Nel grafico in Figura 4.14 mostriamo invece la distribuzione della quantità delle tre categorie di File nel *File Server* per i tre casi al variare di *Cap*. Come prima vogliamo sapere l'evoluzione di quanto spazio della *cache* viene riservato per i File piccoli, quanto per i File medi e quanto per quelli grandi. Si è deciso di trasformare tutto in percentuale sul totale dello spazio usato per un miglior impatto visivo. Ancora una volta lo spazio usato è uguale allo spazio disponibile solo inizialmente. A partire da $Cap = 3500$ [kbyte], per i motivi spiegati nel primo approccio, ci sono alcuni *File Server* che non saturano la capacità. Per quanto riguarda il grafico si può facilmente notare la diversa distribuzione iniziale. Abbiamo caso 1 (P:47%, M:47%, G:6%), caso 2 (P:38%, M:34%, G:28%), caso 3 (P:30,5%, M:33%, G:36,5%). Generalizzando abbiamo che nel caso uniforme ci si preoccupa di inserire più File possibili e quindi, avendo tutti lo stesso "peso", di privilegiare i File più piccoli. Più la *Query* è sbilanciata più si tenderà, come detto prima, ad inserire File nei *File Server* meglio in classifica indipendentemente dalla *Size* e quindi ad arrivare a un rapporto nel caso specifico di (100: 200: 300) ovvero (P:16,6%, M:33,3%, G:50%). C'è da dire che l'ultimo rapporto non è del tutto vero perché ci sarebbe da considerare anche la possibilità di poter mettere anche solo parti dei File. In tutti e tre i casi poi, con l'aumentare di *Cap*, arriviamo sempre ad unico rapporto (P:20%, M:39%, G:41%). Ricordiamo ancora però che si

tratta soltanto di una media su più realizzazioni e non ho detto quindi che nella singola realizzazione non si possano trovare percentuali che si discostino molto tra un *File Server* e un altro.

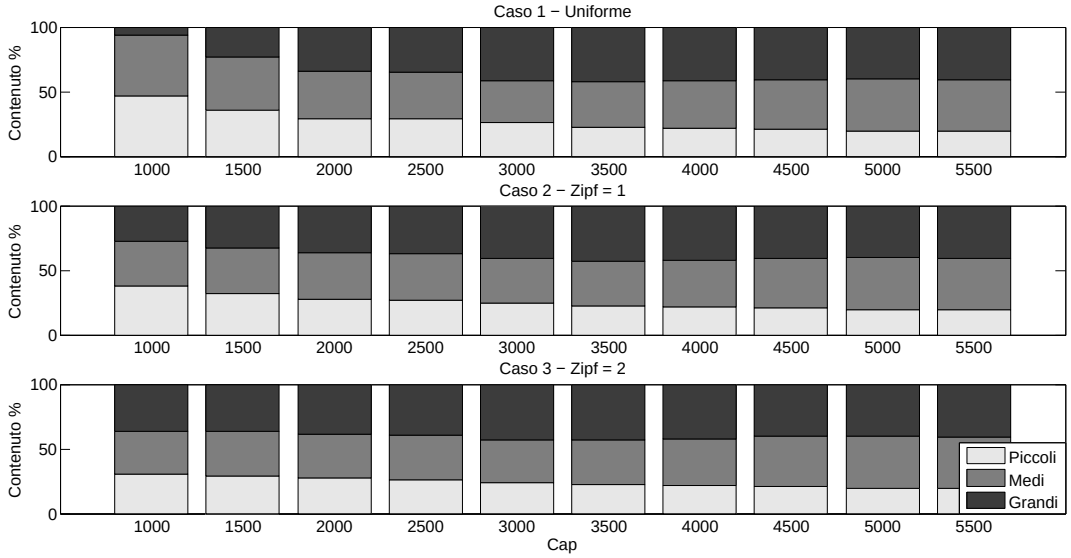



Figura 4.14: Percentuali, in base alla *size* dei file, del contenuto del *file server* al variare della sua capacità di *storage* per i tre casi, secondo approccio.

Nell'altro grafico in Figura 4.15 invece si può vedere quanti File sono presenti nel *File Server* sempre al variare di *Cap*. Nel computo ricordiamo che vengono considerati anche quei File con solo una piccola percentuale immagazzinata. Quello che possiamo notare è solo che tutti i tre casi hanno un andamento molto simile per tipo di File e che il numero di File piccoli, eccezion fatta per la prima fase, è a sorpresa quello più basso dei tre. Questo fatto ci fa pensare che molto probabilmente i File piccoli sono distribuiti solo in determinati *File Server* e che questa purtroppo è solamente una media.

### 4.2.5 Distribuzioni della frequenza dei Percorsi

Sulla falsa riga dell'approccio 2 della sezione precedente vogliamo ora conoscere le prestazioni con diverse distribuzioni di $P(v)$. Come prima si fa variare il parametro $a$ della distribuzione *Zipf* per distinguere, questa volta, cinque casi (vedi Figura 4.16). Caso 1 (Uniforme - *Zipf*,$a = 0$): la frequenza del $v^{\text{esimo}}$ Percorso è: $P(v) = \frac{1}{v^0} / \sum_{v=1}^{16} \frac{1}{v^0}$. Ogni Percorso ha la stessa frequenza. Caso 2 (*Zipf*,$a = 1$): la frequenza del $v^{\text{esimo}}$ Percorso è:

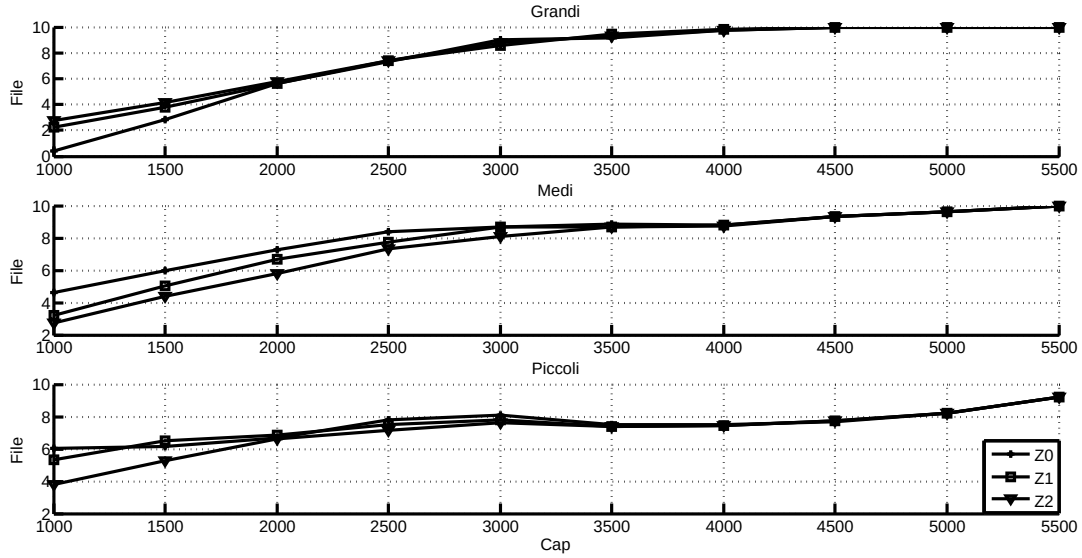



Figura 4.15: Numero di file presenti nel *file server* al variare della sua capacità di *storage* per i tre casi e per le *size* dei file, secondo approccio.

$P(v) = \frac{1}{v^1} / \sum_{v=1}^{16} \frac{1}{v^1}$. Caso 3 (*Zipf*,$a = 2$): la frequenza del $v^{\text{esimo}}$ Percorso è: $P(v) = \frac{1}{v^2} / \sum_{v=1}^{16} \frac{1}{v^2}$. Caso 4 (*Zipf*,$a = 3$): la frequenza del $v^{\text{esimo}}$ Percorso è: $P(v) = \frac{1}{v^3} / \sum_{v=1}^{16} \frac{1}{v^3}$. Caso 5 (*Zipf*,$a = 4$): la frequenza del $v^{\text{esimo}}$ Percorso è: $P(v) = \frac{1}{v^4} / \sum_{v=1}^{16} \frac{1}{v^4}$. Ricordiamo che praticamente più $a$ è grande più la differenza tra il più frequentato e il meno frequentato è grande, ovvero abbiamo una distribuzione di $P(v)$ più "inclinata".

Consideriamo ora una configurazione di *Size* precedente (600, 400, 200) con *Query* uniforme e guardiamo l'andamento dei cinque casi al variare del valore di *cache* dei *File Server*. Nel grafico in Figura 4.17 abbiamo i risultati. Il grafico può essere diviso in due parti. Più la distribuzione è sbilanciata più abbiamo dei vantaggi nella prima fase. Questo accade perché, avendo delle *cache* limitate, si possono inserire e recuperare solamente alcuni File. Logicamente vengono scelti i File più piccoli. Più la distribuzione è sbilanciata meno Percorsi, effettivamente frequentati, determinano la maggior parte della qualità del sistema. Ad esempio nel caso 5, il più sbilanciato dei cinque, basta garantire il recupero di questi File nel Percorso 1 per avere delle prestazioni migliori rispetto agli altri casi. Il caso uniforme infatti, per avere le sue stesse prestazioni, dovrebbe garantire il recupero di quei File sulla maggior parte dei Percorsi. Con l'aumentare di *Cap*, e quindi con la possibilità di immagazzinare altri File nei *File Server*, la situazione invece si ribalta perché

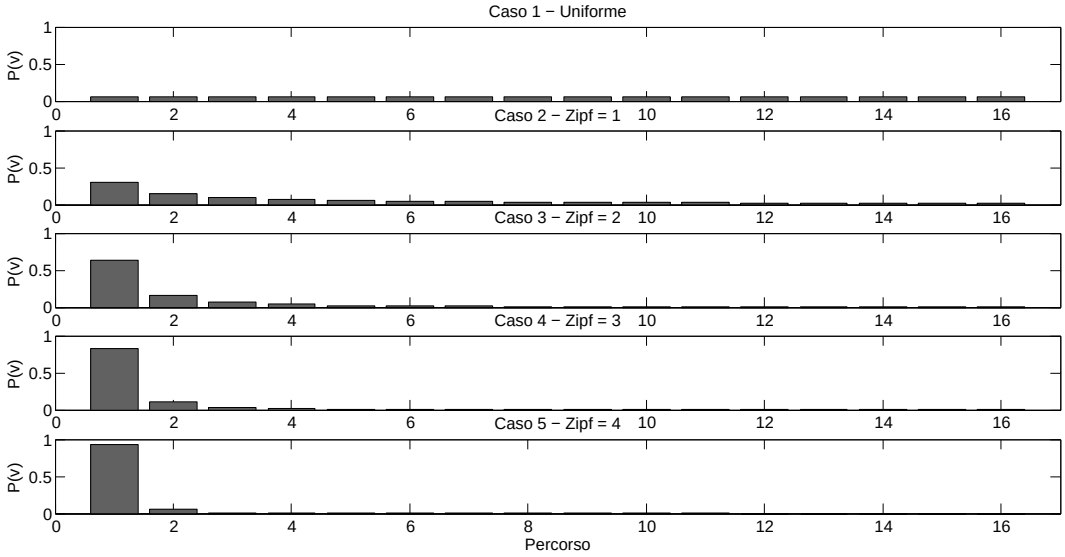



Figura 4.16: Cinque diverse distribuzioni (casi) di frequenze dei percorsi $P(v)$.

c'è da considerare un nuovo fattore dominante: la *Banda(i)*. Continuando con l'esempio del caso 5, sappiamo infatti che la maggior parte degli $N$ utenti effettua il Percorso 1 e come conseguenza che i *File Server* contattati, essendo quelli con più interferenza, garantiscono una *Banda(i)* sempre più limitata da non permettere agli utenti di recuperare gli altri File più grandi. Nel caso uniforme questo succede più difficilmente perché gli utenti sono distribuiti equamente su tutti i Percorsi e quindi i *File Server* con meno interferenza garantiscono ancora una *Banda(i)* che permette agli utenti di recuperare anche i File più grandi. Generalizzando, nella seconda fase possiamo dire che più la distribuzione è uniforme migliore è la performance alla quale possiamo arrivare. Come controprova proviamo ora ad usare un valore di *BandaTot* più grande tale per cui poi sarà possibile recuperare un File grande anche in un Percorso con interferenza massima ($N$ utenti passano in quel percorso). Nel grafico in Figura 4.18 mostriamo quindi il caso con $BandaTot = 10^6$ [kbyte/ contatto] (nel precedente $10^5$) e effettivamente possiamo dire che, non avendo più problemi di banda, non avviene nessun tipo di stravolgimento.

### 4.2.6 Interferenze nei *File Server*

In questa sezione vogliamo cercare, continuando lo studio precedente sulla matrice Contatti, di distinguere i *File Server* in base al numero di

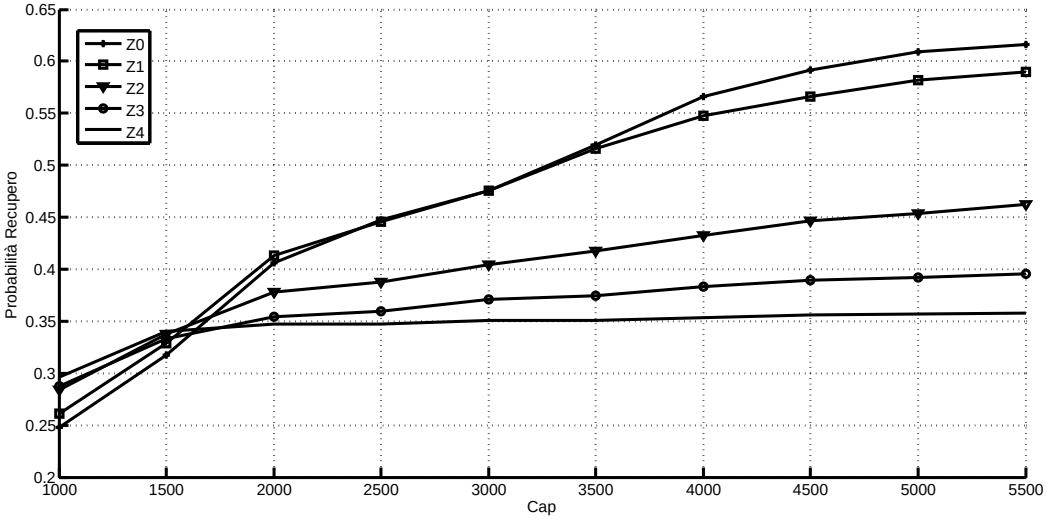



Figura 4.17: Probabilità di recupero al variare della capacità di *storage* dei *file server* per le cinque distribuzioni.

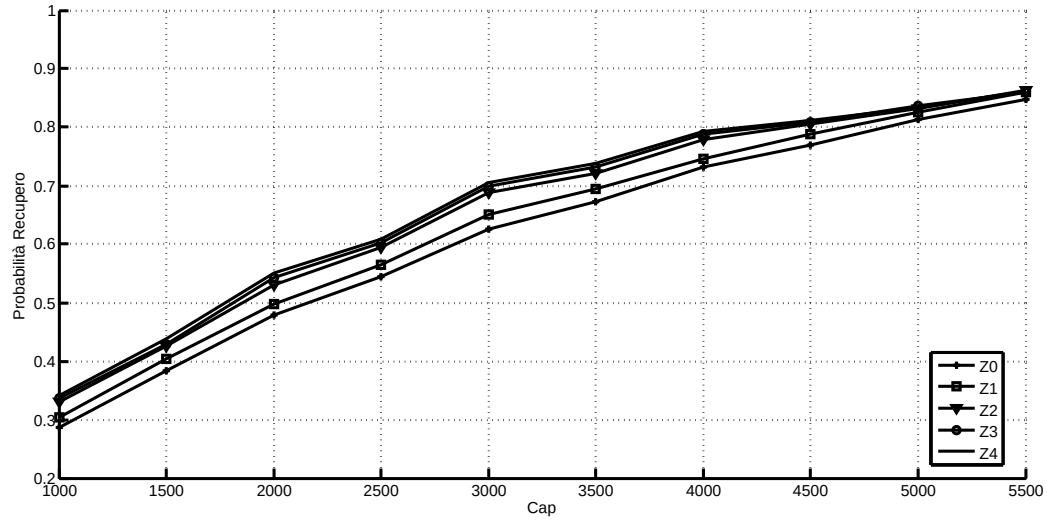



Figura 4.18: Probabilità di recupero al variare della capacità di *storage* dei *file server* per le cinque distribuzioni, con banda più alta.

Contatti. In tutte le analisi precedenti infatti non è stato possibile, mediando su più realizzazioni con diverse matrici Contatti, mantenere e dunque studiare caratteristiche diverse trai *File Server*. Per intenderci, come si comporteranno i *File Server* con diversi numeri di Contatti, e quindi livelli di interferenza, nello stesso scenario? Questa è la domanda alla quale vogliamo rispondere. Proviamo ancora una volta a osservare il caso scelto per poi cercare, con i risultati dei test, di generalizzare.

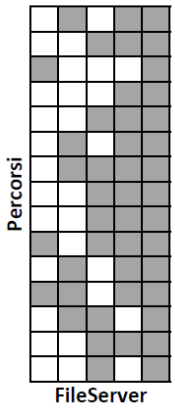



Figura 4.19: Schema matrice contatti con numero di contatti per *file server* crescente.

Le configurazioni principali sono: 15 Percorsi (per comodità di calcoli ne abbiamo tolto uno), 5 *File Server*, 30 File divisi ancora in base alla *Size* in tre categorie (10 grandi - 600 [kbyte], 10 medi - 400 [kbyte], 10 piccoli - 200 [kbyte]), *Query* e $P(v)$ uniformi. Per quanto riguarda la matrice Contatti decidiamo, come detto prima, di vincolare la struttura in modo tale da avere un diverso numero di contatti trai *File Server*. Abbiamo per ogni singolo scenario (vedi Figura 4.19) tre Percorsi passanti per il primo *File Server*, sei Percorsi per il secondo, nove Percorsi per il terzo, dodici Percorsi per il quarto e infine tutti e quindici i Percorsi passanti per il quinto e ultimo *File Server*. Fissati quindi i livelli di interferenza, i *File Server* saranno caratterizzati, in ogni realizzazione, dalle seguenti bande: *Banda(*1*)* = 500 [kbyte/contatto], *Banda(*2*)* = 250 [kbyte/contatto], *Banda(*3*)* = 167 [kbyte/contatto], *Banda(*4*)* = 125 [kbyte/contatto], *Banda(*5*)* = 100 [kbyte/contatto]. I risultati che vedremo saranno una media su 50 realizzazioni con probabilmente 50 diverse matrici vincolate nel modo appena descritto.

Nel grafico in Figura 4.20 troviamo l'andamento delle percentuali medie dei File, separati per *Size*, immagazzinati nei *File Server* (FS 1, FS 2, FS 3, FS 4, FS 5) al variare di *Cap*. Possiamo notare che non c'è una uniformità

trai *File Server* e in particolar caso nei File grandi è possibile vedere che meno Contatti ha il *File Server* e più è alta la percentuale immagazzinata di questi. In generale confermiamo quello che ipotizzavamo nelle analisi precedenti, ovvero di trovare percentuali che si discostano tra un *File Server* e un altro, principalmente dovute a diversi livelli di interferenza e quindi alle diverse *Banda(i)*.

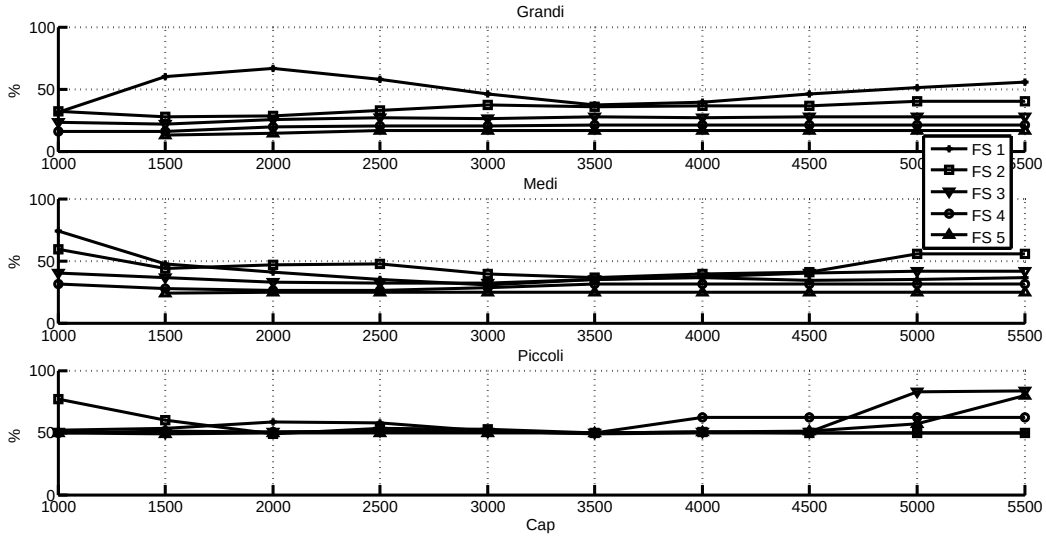



Figura 4.20: Percentuali medie di un file immagazzinato nel *file server* al variare della capacità di *storage* per i cinque *file server* e per le *size* dei file.

Nel grafico in Figura 4.21 invece si può vedere quanti File sono presenti nei diversi *File Server* sempre al variare di *Cap*. Nel computo ricordiamo ancora che vengono considerati anche quei File con solo una piccola percentuale immagazzinata. Grazie a questo grafico è possibile far emergere una caratteristica significativa dello scenario. Notiamo infatti che tutti i File piccoli son distribuiti fin da subito da i *File Server* 4 e 5 e che i *File Server* 1 e 2 ne contengono invece una minima parte. Proviamo a spiegarne allora il motivo. Logicamente con *Cap* limitato e *Query* uniforme, come abbiamo visto in analisi precedenti, vengono scelti i File piccoli perché è possibile garantire un recupero di più File. Tutti i Percorsi poi passano principalmente per i *File Server* 4 e 5 e quindi, posizionando qui i File, è possibile garantirne il recupero su tutti o quasi i Percorsi. Potendoli recuperare attraverso questi due contatti sarebbe allora inutile averli a disposizione negli altri *File Server*. Inoltre c'è da dire che una *Banda(i)* limitata in questi *File Server* è in grado

di garantire il solo recupero di File piccoli. Gli altri *File Server*, in accordo con le percentuali viste nel grafico precedente, avendo una *Banda(i)* maggiore, si preoccupano principalmente di garantire il recupero di File più grandi.

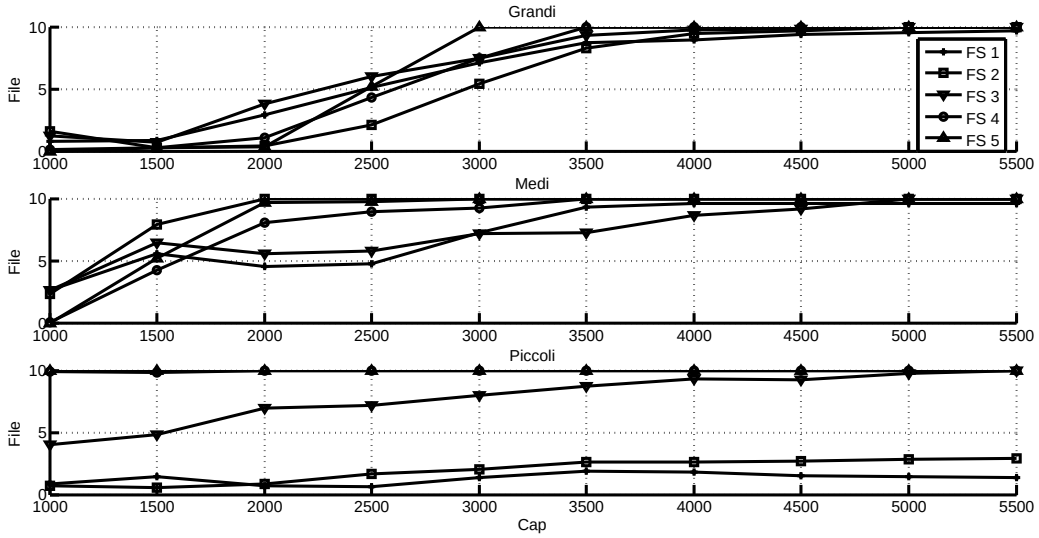



Figura 4.21: Numero dei file presenti nel *file server* al variare della capacità di *storage* per i cinque *file server* e per le *size* dei file.

Nel grafico in Figura 4.22 mostriamo invece la distribuzione della quantità delle tre categorie di File nei diversi *File Server* al variare di *Cap*. Questa volta si è deciso di non trasformare tutto in percentuale per mostrare quello che si era intuito ma che non era possibile raffigurare nelle analisi precedenti, ovvero che non tutti i *File Server* saturano la capacità. In questo ultimo grafico si racchiude un po' tutto quello emerso in precedenza. Salta subito all'occhio infatti la diversa distribuzione trai *File Server*, i primi, con prevalenza di colori scuri, incaricati al recupero di File medi e grandi, e gli ultimi, con presenza di colori chiari, incaricati al recupero di File piccoli.

### 4.2.7 Sul modello alternativo

Per quanto riguarda quest'ultima sezione vogliamo analizzare il modello alternativo (Tabella 3.5). Questo, come vedremo, ci ha permesso di fare delle utili considerazioni che non si sarebbero potute ottenere col modello precedente. Nel grafico in Figura 4.23 mostriamo inizialmente l'andamento di *CapTot* al variare di $N$. Per ogni curva abbiamo impostato un diverso

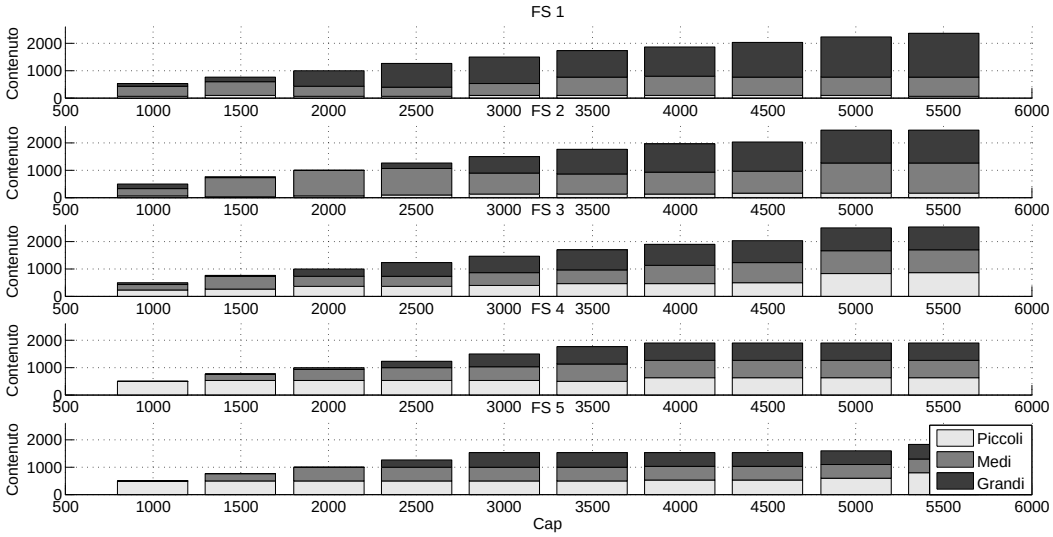



Figura 4.22: Contenuto dei cinque *file server*, diviso in base alla *size* dei file presenti, al variare della capacità di *storage*.

valore di efficienza da garantire. I risultati sono basati su un singolo caso e le altre configurazioni sono simili a quelle viste nelle sezioni precedenti. L'andamento delle curve è abbastanza scontato. All'aumentare di $N$, infatti, il livello di *CapTot* necessario cresce. Questo accade perché con l'aumentare dell'interferenza diminuiscono le *Banda(i)* e alcuni File inseriti nella *cache* diventano irrecuperabili. C'è bisogno quindi o di inserire parte di questi File in altri *File Server* o garantire il recupero di nuovi File per per mantenere la stessa qualità. Di conseguenza più vogliamo un livello alto di qualità del sistema e più abbiamo bisogno di un maggior spazio complessivo. Inoltre possiamo garantire un livello di *Eff* solo fino un certo numero di utenti nell'area e questo numero, che ricordiamo segue il livello di interferenza, è inversamente proporzionale al livello di efficienza.

Nell'analisi successiva abbiamo invece considerato scenario e configurazioni della sezione precedente (Interferenze nei *File Server*). Usando questo nuovo modello vogliamo mostrare la distribuzione della quantità delle tre categorie di File nei diversi *File Server* al variare di $N$ per mantenere una *Eff* = 0.8. Principalmente vogliamo sapere come e dove viene distribuito questo *CapTot* nei diversi *File Server*. Nel grafico in Figura 4.24 è messa in risalto la distribuzione delle tre categorie di File per ogni *File Server* mentre in Figura 4.25 troviamo i *Cap* di ogni *File Server* in relazione a quello totale.

I risultati mostrano che la maggior parte dei contenuti è immagazzinata

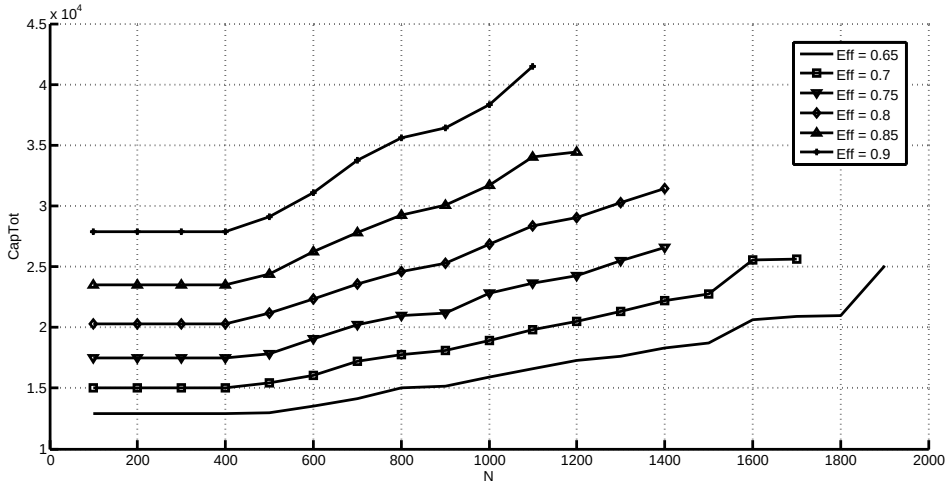



Figura 4.23: Somma di tutte le capacità di *storage* dei *file server*, al variare del numero di utenti $N$, necessarie per garantire diversi livelli di efficienza.

sempre nel *File Server* 5, che ricordiamo essere quello presente in tutti i percorsi. Fin quando è possibile mantenere il livello d'efficienza, recuperando con un solo contatto i File, conviene immagazzinare i contenuti solo in questo *File Server*. Con l'aumentare di $N$ diminuiscono i contenuti in quest'ultimo, perché la *Banda(i)* non garantisce più il recupero dei File più grandi con un solo contatto, e aumentano quindi i contenuti negli altri *File Server*. Logicamente si preferisce inserire più contenuti in quei *File Server*s presenti in più percorsi. I File piccoli tra l'altro li troviamo sempre solo nel *File Server* 5 perché la *Banda(i)*, con questi valori di $N$, ne garantisce ancora il recupero con un solo contatto.

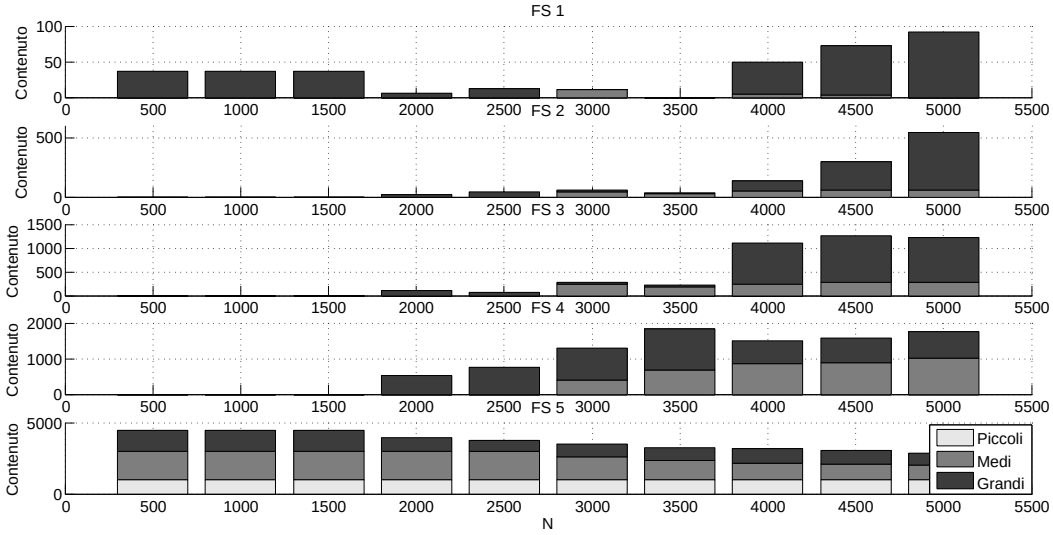



Figura 4.24: Distribuzione in base alla *size* della quantità necessaria di file immagazzinati nei cinque *file server* al variare del numero di utenti $N$ per garantire un livello fissato di efficienza.

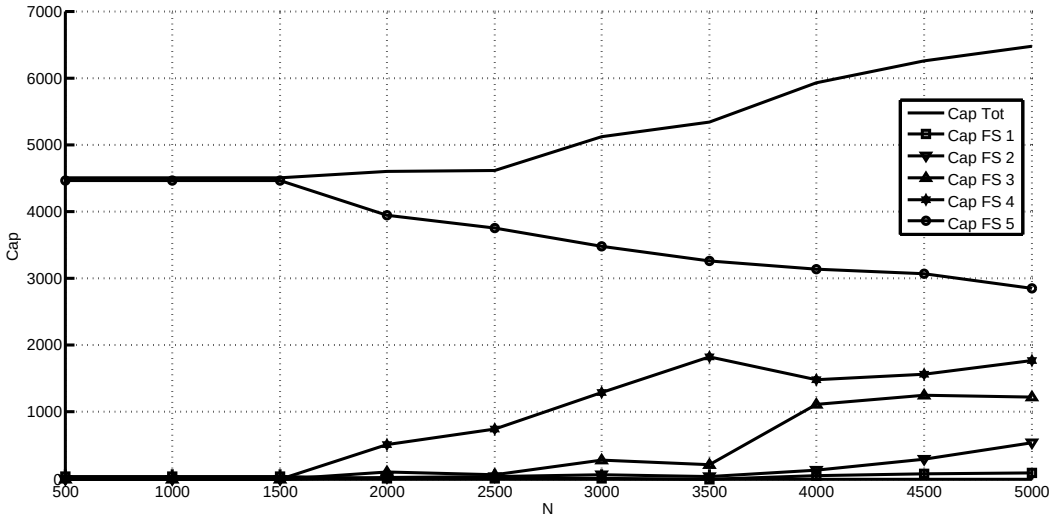



Figura 4.25: Capacità di *storage* dei cinque *file server*, in relazione alla somma *Cap Tot*, al variare del numero di utenti $N$, necessarie per garantire un livello fissato di efficienza.

# Capitolo 5

# Conclusioni e sviluppi futuri

Questa tesi nasce dalla curiosità nel voler scoprire il ruolo che avranno le reti veicolari nei prossimi anni. Una tecnologia dalle numerose potenzialità che ormai sembra essere molto più di una semplice scommessa. L'incognita rimane ancora però sullo standard da adottare circa l'architettura. Noi crediamo che l'idea di creare una buona rete V2I per la distribuzione dei contenuti sia la giusta strada da seguire per avviare la diffusione di questa tecnologia. Una volta che il sistema riuscirà a prendere piede, sarà possibile infatti garantire buone prestazioni a tutti quegli altri servizi di sicurezza che utilizzano la rete V2V. Di conseguenza ciò attirerà ulteriori utenti e potrà innescare una rapida distribuzione della tecnologia. La *WiFi-based Content Distribution Community Infrastructure* può essere quindi più di un punto di partenza. Da questo studio è emerso come sia di fondamentale importanza il *caching* negli RSU per la qualità delle prestazioni. Le caratteristiche della rete veicolare obbligano in sostanza ad accorciare i tempi di recupero dei contenuti e, se la strategia di *caching* è ben strutturata, si hanno, come si è visto, evidenti benefici. Il modello proposto consente l'ottimizzazione della distribuzione dei contenuti in reti veicolari. Trai tanti risultati è stato possibile:

- trovare la giusta proporzione tra il numero medio di Contatti per Percorso e la banda a disposizione dei *File Server*;
- scoprire la miglior scelta del contenuto dei *File Server*, con diversi valori della dimensione e della popolarità dei File, al variare della capacità massima;
- scoprire la miglior scelta del contenuto di *File Server* con diversi livelli di interferenza (numero di Percorsi passanti).

Un possibile sviluppo futuro potrebbe essere quello di pensare ad un sistema autonomo, non più gestito da un singolo *Content Manager*, per

permettere una buona scalabilità del sistema. Una simile struttura potrebbe essere come quella vista in [24]. Un'altra proposta poi potrebbe essere quella di abbandonare il concetto di Percorso, ritenendolo una scelta poco realistica, e di seguire l'idea presentata in [25]. Ricevuta una richiesta di contenuto da parte di un utente, il *File Server* contattato è ora in grado di:

- prevedere statisticamente il prossimo contatto futuro del veicolo;
- inviare contemporaneamente la richiesta al *File Server* previsto di recuperare il contenuto richiesto prima dell'arrivo dello stesso veicolo.

E' ovvio che questi ulteriori sviluppi potrebbero presentare sia dei pro che dei contro, starà quindi a noi valutare in un futuro quale sia veramente la scelta migliore.

# Ringraziamenti

[Spazio ringraziamenti]

# Bibliografia


[1] Ying Huang, Yan Gao, Klara Nahrstedt, Wenbo He. *"Optimizing File Retrieval in Delay-Tolerant Content Distribution Community"*, 29th IEEE International Conference on Distributed Computing Systems, Luglio 2009.

[2] Bruno M. Silva, Farid Farahmand, Joel J.P.C. Rodrigues. *"Performance Assessment of Caching and Forwarding Algorithms for Vehicular Delay Tolerant Networks"*, IEEE Globecom, 2010.

[3] Junghoon Lee. *"Access pattern analysis for a directional cache scheduling scheme on vehicular telematics networks"*, ECTI-CON, 2009.

[4] Guohong Cao, Liangzhong Yin, Chita R. Das. *"Cooperative Cache-Based Data Access in Ad Hoc Networks"*, Computer, 2004.

[5] B. Zheng, W. Lee, and D. Lee. *"On semantic caching and query scheduling for mobile nearest-neighbor search"*, Wireless Networks, Vol. 10, pp. 653-664, 2004.

[6] Nicholas Loulloudes, George Pallis, Marios D. Dikaiakos. *"On the Evaluation of Caching in Vehicular Information Systems"*, 11th International Conference on Mobile Data Management, 2010.

[7] Marco Fiore, Francesco Mininni, Claudio Casetti, Carla-Fabiana Chiasserini. *"To Cache or Not To Cache?"*, IEEE Infocom, 2009.

[8] Fan Li, Yu Wang. *"Routing in Vehicular Ad Hoc Networks: A Survey"*, IEEE Vehicular Technology Magazine, Giugno 2007.

[9] Ming Li, Zhenyu Yang, Wenjing Lou. *"CodeOn: Cooperative Popular Content Distribution for Vehicular Networks using Symbol Level Network Coding"*, IEEE Journal on Selected Areas in Communications, Vol. 29, N. 1, Gennaio 2011.

[10] Wikipedia. *"Cache"*, Website - *http://it.wikipedia.org/wiki/Cache*, Febbraio 2012. *"Web cache"*, Website - *http://it.wikipedia.org/wiki/Web_cache*, Ottobre 2011.

[11] Wei Gao, Guohong Cao, Arun Iyengar, Mudhakar Srivatsa. *"Supporting Cooperative Caching in Disruption Tolerant Networks"*, 31st International Conference on Distributed Computing Systems (ICDCS), 2011.

[12] Tom Hao Luan, Xinhua Ling, Xuemin Shen. *"MAC in Motion: Impact of Mobility on the MAC of Drive-Thru Internet"*, IEEE Transactions on Mobile Computing, Febbraio 2011.

[13] Hao Liang, Weihua Zhuang. *"Optimal Scheduling for Roadside WLANs with Pre-Downloaded Messages"*, IEEE ICC, 2011.

[14] Da Zhang, Chai Kiat Yeo. *"A Cooperative Content Distribution System For Vehicles"*, IEEE Globecom, 2011.

[15] Prashant Krishnamurthy. *"Information dissemination and information assurance in vehicular networks: a survey"*, iSchools Conference, pagine 1-6, Febbraio 2008.

[16] Alok Nandan, Shirshanka Das, Giovanni Pau, Mario Gerla, M. Y. Sanadidi. *"Cooperative downloading in Vehicular Ad-hoc Wireless Networks"*, Second IEEE annual conference on Wireless On-demand Network Systems and Services, pagine 32-41, Gennaio 2005.

[17] Uichin Lee, Joon-Sang Park, Joseph Yeh, Giovanni Pau, Mario Gerla. *"CodeTorrent: content distribution using network coding in VANET"*, MobiShare, 1st international workshop on Decentralized resource sharing in mobile computing and networking, pagine 1-5, Settembre 2006.

[18] Francesco Malandrino, Claudio Casetti, Carla-Fabiana Chiasserini, Marco Fiore. *"Content downloading in Vehicular Networks: what really matters"*, IEEE Infocom, pagine 426-430, Aprile 2011.

[19] Hannes Hartenstein, Kenneth P. Laberteaux. *"VANET, Vehicular Applications and Inter-Networking Technologies"*, Wiley, Dicembre 2009.

[20] Kamini, Rakesh Kumar. *"VANET Parameters and Applications: a Review"*, Global Journal of Computer Science and Technology, Vol 10, N. 7, pagine 72-77, Settembre 2010.

[21] Alessandro Falaschi. *"Vehicular Ad Hoc Networks: Scenari, Tecnologie, Sviluppo Applicazioni"*, Dicembre 2009.

[22] IEEE 802.11p Amendment. *"Wireless Access in Vehicular environments"*, available from IEEE standards, 15 luglio 2010.

[23] Mate Boban, Geoff Misek, Ozan K. Tonguz. *"What is the best achievable QoS for unicast routing in VANET?"*, IEEE Globecom workshops, pagine 1-10, Dicembre 2008.

[24] Tom H. Luan, Lin X. Cai, Jiming Chen, Xuemin (Sherman) Shen, Fan Bai. *"VTube: Towards the Media Rich City Life with Autonomous Vehicular Content Distribution"*, 8th Annual IEEE Communications Society Conference on Sensor, Mesh and Ad Hoc Communications and Networks (SECON), pagine 359 - 367, Giugno 2011.

[25] Da Zhang, Chai Kiat Yeo. *"A Cooperative Content Distribution System For Vehicles"*, IEEE Globecom, 2011.