UNIVERSITA' DEGLI STUDI DI FIRENZE
Facoltà di Scienze Matematiche, Fisiche e Naturali
Corso di Laurea in Informatica

Valutazione quantitativa della QoS dei servizi applicativi su rete ATN (Aeronautical Telecommunications Network)

Candidato :
LorenzoViti

Relatore :
Prof. Andrea Bondavalli

Correlatori :
Ing. Pierluigi Fantappié – OTE Spa
Ing. Paolo Maltese – OTE Spa
Dott. Lorenzo Falai

Anno Accademico 2004-05

Prefazione

Ormai da alcuni anni si osserva una crescente diffusione di architetture distribuite in molteplici campi di applicazione. Tale diffusione è motivata principalmente da ragioni economiche come la riduzione del costo di processori e della connettività. L'introduzione di architetture distribuite basate sul partizionamento e l'allocazione di applicazioni su più risorse computazionali, ha notevoli benefici rispetto alle tradizionali architetture monolitiche: la riduzione dei costi complessivi, la maggiore scalabilità e l'opportunità di raggiungere un alto livello di affidabilità. Un sistema distribuito può essere capace di tollerare i guasti, ridistribuire il carico applicativo su risorse non compromesse e permettere quindi la continuità del servizio del sistema. Quest'ultimo aspetto riveste fondamentale importanza in tutte le applicazioni critiche come ad esempio il controllo del traffico aereo. I sistemi per il controllo del traffico aereo hanno l'obbligo di rispettare stringenti requisiti di qualità del servizio (QoS) relativi a sicurezza, affidabilità e prestazioni, imposti da autorità internazionali come ICAO (International Civil Aviation Organization) per evitare che il malfunzionamento di tali sistemi abbia conseguenze dannose per l'uomo, l'ambiente ed economiche. Nel campo dell'aeronautica civile vale la regola che nessun nuovo sistema è introdotto se non rappresenta un miglioramento, in termini di qualità del servizio, rispetto al sistema attuale. Questa politica frena ogni cambiamento radicale, accettando solo cambiamenti lenti e graduali. Ciò è giustificato anche da ragioni economiche, in quanto gli ingenti investimenti sugli apparati attualmente operativi, rendono improbabile una loro sostituzione in blocco.

Ogni anno però si registra una crescita del volume del traffico aereo, delle simulazioni di Eurocontrol, l'ente europeo preposto alla gestione del traffico aereo, intravedono l'inadeguatezza delle strutture di controllo attuali per soddisfare i futuri scenari. Da qui l'esigenza di promuovere un nuovo programma di sviluppo denominato *Link2000+* che prevede di integrare nuovi servizi su di una architettura distribuita. Mentre l'attuale sistema è quasi per intero basato su trasmissioni analogiche (vocali), la nuova architettura utilizzerà invece la rete ATN (Aeronautical Telecommunications Network) basata su trasmissione digitale. Come sottolineato il nuovo sistema non potrà sostituire immediatamente quello esistente.
In quest'ottica si pone il nuovo servizio CPDLC (Controller Pilot Data Link Communication) che affiancandosi alla tradizionale comunicazione vocale, consentirà l'automatizzazione di certe procedure di controllo e lo scambio messaggi tra controllore e pilota attraverso la trasmissione digitale.
La scelta di introdurre un nuovo sistema in un contesto critico, comporta tra le altre, anche una valutazione quantitativa per stabilire se il sistema raggiunge o meno i livelli attesi di qualità del servizio. Questa tesi svolta in collaborazione con l'azienda OTE di Firenze, ha come obiettivo primario la valutazione quantitativa di un sistema CPDLC per riuscire a stimare i tempi di risposta del sistema percepiti dall'utente.
Le tecniche possibili per un'analisi quantitativa sono tre: analitiche, simulative e sperimentali. Le prime due tecniche si basano su un modello del sistema, mentre l'ultima utilizza direttamente il sistema reale. La scelta di una tecnica rispetto ad un'altra, dipende dalla fase del sistema in cui tale tecnica viene impiegata.

Mentre le funzionalità del servizio CPDLC sono già state ben specificate, ancora non è disponibile un sistema CPDLC in versione definitiva, eccetto qualche prototipo come quello utilizzato da Eurocontrol nell'area di Maastricht. Perciò l'impiego di una tecnica puramente sperimentale è pressoché impossibile, così come l'utilizzo di una tecnica analitica basata su un modello matematico molto astratto non è utilizzabile in quanto si rischia di trascurare troppi dettagli del sistema. Così anche l'impiego di una tecnica completamente simulativa non è ritenuta la scelta più appropriata, perché richiede di modellizzare oltre all'applicazione CPDLC anche il comportamento dei protocolli di comunicazione della rete ATN. In definitiva, in questa tesi è stato seguito un approccio misto che si colloca tra la simulazione e le misure sperimentali, allo scopo di realizzare un prototipo di applicazione CPDLC usando componenti reali della rete ATN, quest'ultimi forniti dall'azienda irlandese Airtel (partner OTE) specializzata in router ATN. Questo approccio ha il vantaggio di essere ancora basato su un modello facilitando la rappresentazione di diverse configurazioni del sistema.

La tesi è strutturata nel modo seguente:

Nel primo capitolo sono introdotti i concetti di sistema distribuito, dependability e performance, ed inoltre sono presentate le tecniche e gli strumenti utilizzabili per la valutazione quantitativa. Nel secondo ed nel terzo capitolo sono descritte in maniera specifica l'architettura di rete ATN e l'applicazione CPDLC. Nel quarto capitolo sono ripresi i concetti e le metodologie presentate nel primo capitolo, per applicarle al contesto specifico della valutazione quantitativa del servizio CPDLC. In particolare, sono

definiti l'obiettivo di studio, le quantità di interesse e la tecnica di valutazione utilizzata; è anche presentato lo strumento Neko per l'analisi e la prototipizzazione di algoritmi distribuiti. Nel quinto capitolo è descritto il modello del sistema CPDLC formato dall'applicazione realizzata in Neko e dalla rete ATN di test. Infine, nell'ultimo capitolo sono descritti gli esperimenti per la valutazione delle quantità di interesse, analizzati e commentati i risultati ottenuti.

Indice

Elenco delle figure

Capitolo 1

Valutazione di sistemi distribuiti

1.1 Sistemi distribuiti

Esistono in letteratura diverse definizioni di sistema distribuito, ognuna delle quali è influenzata in modo più o meno forte dalla area di ricerca di provenienza. Tanembaum, esperto di sistemi operativi, definisce un sistema distribuito come una collezione di computers indipendenti che appaiono agli utenti del sistema come un singolo computer [1]. Una studiosa di algoritmi distribuiti come Lynch, fornisce una definizione ancora più specifica di quella di Tanembaum, introducendo il concetto di correttezza [2]. Altro esperto come Coulouris, enfatizza l'aspetto che i computer siano collegati tra loro attraverso una rete di comunicazione [3]. Mentre Lamport fornisce la seguente definizione:

> *Un sistema distribuito è quello che ti impedisce di lavorare a causa del guasto di una macchina di cui non avevi mai sentito parlare.*

Lo sviluppo del software di un sistema distribuito è un compito non banale ed è facile commettere errori. Ad esempio, i malfunzionamen-

ti sono difficili da prevedere perché possono manifestarsi durante particolari interazioni tra le componenti del sistema dovute alla sovrapposizione (interleaving) degli eventi. Ciò non accade in sistemi centralizzati, dove gli eventi seguono un ordine sequenziale e una logica deterministica. Questo problema del comportamento dei sistemi distribuiti, talvolta non deterministico, ha portato nel corso degli ultimi anni ad un'intensa attività di ricerca con lo scopo di fornire delle metodologie e degli strumenti per la verifica della correttezza di questi sistemi. Nelle prime fasi dello sviluppo di un sistema, cioé quando vengono definite le sue funzionalità, l'aspetto della correttezza gioca un ruolo fondamentale. Successivamente, si pongono altri interrogativi per capire quanto il sistema funziona bene sia in situazioni normali che in presenza di guasti. Infatti, in tanti contesti, in particolare quelli critici, i sistemi hanno il requisito di garantire il loro funzionamento anche in presenza di guasti. In linea generale si può dire che i sistemi sono caratterizzati da quattro proprietà fondamentali: Funzionalità, Dependability, Performance e Costo.

1.2 Dependability

La **dependability** è definita come *la capacità di fornire un servizio su cui si può fare affidamento in maniera giustificata* [4]. La dependability è un proprietà dei sistemi che comprende altri concetti importanti come l'affidabilità, la disponibilità, la safety, etc. Questo termine nonostante sia sinonimo di affidabilità non è da confondersi con la reliability che ha invece un preciso significato matematico; inoltre, esso aggiunge la nozione di dipendenza per evidenziare la crescente dipendenza della nostra società dai sistemi informatici.

Il *servizio fornito* è una caratteristica dinamica che rappresenta il comportamento del sistema percepito dall'utente tramite l'interfaccia del servizio, ed è diverso dalla funzione, caratteristica statica, descritta invece nella specifica del sistema.
Una trattazione sistematica della dependability consiste di tre parti fondamentali: gli impedimenti, gli attributi e i mezzi per ottenerla [5].

1.2.1 Impedimenti

Un servizio è **proprio** quando il servizio implementa la specifica funzionale del sistema. Un **fallimento** del sistema è un evento che occorre quando il servizio fornito devia dal servizio proprio, diventando così **improprio**. Un sistema può fallire sia perché non corrisponde alla sua specifica, sia perché la sua funzione non è adeguatamente descritta. Ad alto livello, cioé dal punto di vista dell'utente, il servizio di un sistema può essere rappresentato attraverso un semplice diagramma di stato:

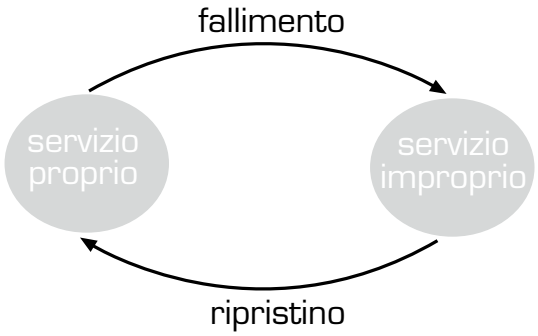

Figura 1.1: Gli stati di un servizio

La transizione da servizio proprio a servizio improprio indica un fallimento, l'altra, da servizio improprio a servizio proprio, indica il ripristino

del servizio. L'intervallo di tempo durante il quale il sistema è improprio è chiamato *periodo di outage* del servizio. In realtà, un fallimento è la conseguenza percepita dall'utente di altri eventi come i guasti e gli errori.
Un **guasto** rimane nascosto fino alla sua attivazione, dopo di che, genera un errore coinvolgendo una o più componenti del sistema. Successivamente, l'errore può propagarsi e generare nuovi errori. Quando questi errori arrivano fino all'interfaccia del servizio e provocano effetti sul servizio offerto, in quel momento si verifica un fallimento.

Una possibile classificazione dei guasti è in base alla loro causa fenomenologica. Si hanno allora guasti *fisici* oppure *umani*. Un guasto fisico è dovuto a cause interne come i cortocircuiti, oppure esterne come le perturbazioni elettromagnetiche e le temperature. Un guasto umano è provocato da un sbaglio nel design del sistema, sia in fase iniziale, cioé passando dai requisiti all'implementazione, sia durante la definizione di procedure operative e di manutenzione. Un guasto umano è anche causato da un' interazione dell'utente che viola le modalità di funzionamento previste dal sistema.
In base alla loro natura i guasti possono essere accidentali, di intento malizioso e non malizioso.
I guasti sono anche classificati in base alla loro persistenza. Quando essi sono continui e stabili sono detti *permanenti* e sono provocati da difetti fisici del sistema o da design inadeguato. Oppure sono *intermittenti* se si presentano occasionalmente, ad esempio per colpa di hardware instabile. Infine, sono chiamati *transienti* quei guasti dovuti a temporanee condizioni ambientali.
In generale, un sistema non fallisce mai allo stesso modo. Anche i fallimenti sono classificati in base a diversi criteri. Uno dei più importanti

è quello rispetto alla conseguenza provocata dal fallimento. I fallimenti sono così distinti in *benigni*, se la loro conseguenza ha lo stesso ordine di grandezza, spesso inteso in termini di costo, del servizio offerto in assenza di fallimento; e in *catastrofici* se la loro conseguenza non è ritenuta accettabile perché può comportare gravi danni a persone e ambiente, oppure avere costi troppo elevati.

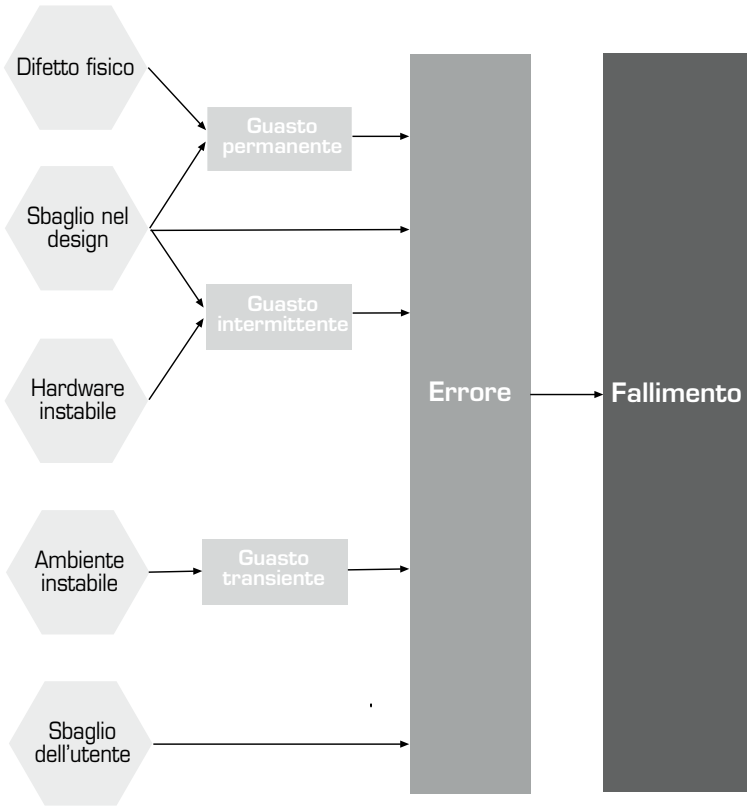

Figura 1.2: La catena di impedimenti

1.2.2 Attributi

Gli attributi che compongono il concetto generale di dependability sono i seguenti:

- **Affidabilità** (Reliability): indica la continuità del servizio offerto.
- **Disponibilità** (Availability): esprime la capacità di fornire un servizio proprio.
- **Manutenibilità**: indica la capacità di ripristino del servizio.
- **Safety**: garantisce l'assenza di conseguenze catastrofiche per l'utente e l'ambiente.
- **Confidenzialità**: garantisce l'assenza di diffusione non autorizzata di informazioni.
- **Integrità**: garantisce l'assenza di alterazione impropria dello stato del sistema.

L'interpretazione del significato di questi attributi deve essere in senso relativo e probabilistico, e non in senso assoluto e deterministico. Quindi a questi attributi in particolare ad affidabilità e disponibilità, si associano delle *misure* in termini di probabilità [9]. L'affidabilità (Reliability) di un sistema è la capacità di funzionare correttamente durante un intervallo di tempo $[0, t]$. In altre parole, sia X la variabile casuale che rappresenta il tempo di vita di un sistema (tempo al fallimento) e sia F la funzione di distribuzione (CDF) della variabile X, allora l'affidabilità del sistema al tempo t è definita dalla probabilità che il sistema sopravviva fino al tempo t, dunque $R(t) = P(X > t) = 1 - F(t)$ assumendo che al tempo $t = 0$ il

sistema funzioni correttamente $R(0) = 1$.
La *disponibilità* è definita invece dalla funzione $A(t)$ uguale alla probabilità che il sistema funzioni correttamente nell'istante di tempo t, nonostante il numero di fallimenti e le riparazioni occorse nell'intervallo $[0, t]$. Talvolta risulta essere più intuitivo esprimere l'affidabilità e la disponibilità utilizzando le seguenti metriche. Per l'affidabilità si usa il tempo medio al fallimento $MTTF$ (Mean Time To Failure) che indica il tempo medio di funzionamento del sistema. Mentre per la disponibilità si usa la seguente formula:

$$\text{Disponibilità} = \frac{MTTF}{MTTF + MTTR}$$

Dove $MTTR$ (Mean Time To Repair) indica il tempo medio di riparazione a seguito di un fallimento. Inoltre, questa metrica rappresenta una stima di un altro attributo: la *manutenibilità*. Dalla formula si può notare quanto il fattore manutenibilità è importante al fine di ottenere una buona disponibilità: più basso è il tempo di riparazione e più aumenta la disponibilità.
Analogamente all'affidabilità, è possibile quantificare la sicurezza (safety) di un sistema con la misura del tempo al fallimento, considerando quest'ultimo di tipo catastrofico.

1.2.3 Mezzi

Lo sviluppo di un sistema *dependable* richiede l'applicazione combinata delle seguenti attività:

- **Prevenzione dei guasti** (Fault prevention): per prevenire l'occorrenza dei guasti.

- **Tolleranza ai guasti** (Fault tolerance): per riuscire a fornire un servizio proprio in presenza di guasti.
- **Rimozione dei guasti** (Fault removal): per ridurre il numero o la severità dei guasti.
- **Previsione dei guasti** (Fault forecasting): per prevedere la futura incidenza dei guasti.

Prevenzione dei guasti

L'attività di prevenzione dei guasti è svolta durante le fasi di definizione dei requisiti, di design e di implementazione, quindi per prevenire i guasti prima che il sistema sia messo in funzione. Sono impiegate tecniche per il raffinamento dei requisiti, la verifica formale, la programmazione strutturata e modulare (Object-Oriented).

Tolleranza ai guasti

L'attività di prevenzione aiuta a diminuire i guasti, ma non è sufficiente se non è supportata da un'attività di tolleranza ai guasti.
La tolleranza ai guasti permette di preservare la fornitura di un servizio proprio, magari degradato, anche in presenza di guasti attivi. Generalmente essa si compone di due attività: **il rilevamento dell'errore** (Error Detection) e il **ripristino del sistema** (System Recovery).
Il rilevamento di un errore genera un segnale oppure un messaggio nel sistema. Un errore che è presente, ma che ancora non è stato rilevato è detto *latente*. L'individuazione di un errore può avvenire in modalità *concorrente* se ha luogo durante la fornitura del servizio, oppure *preventiva* quando il

servizio viene sospeso. Quest'ultima consente un controllo più approfondito per determinare l'esistenza di errori latenti e guasti non ancora attivi. Il ripristino del sistema (recovery) trasforma uno stato del sistema contenente degli errori e dei guasti, in un altro stato che invece ne è privo. Ciò è possibile utilizzando delle tecniche per il trattamento degli errori (Error Handling) per eliminare gli errori dallo stato del sistema, e delle tecniche per il trattamento dei guasti (Fault Handling) per prevenire la ricomparsa degli errori già localizzati. Nel primo caso si hanno due scelte: o riportare il sistema ad un stato salvato prima del rilevamento dell'errore: **Rollback**, oppure trasformando lo stato attuale con un nuovo stato: **Rollforward**.
Il trattamento dei guasti comprende invece le seguenti tecniche:

- **Diagnosi del guasto** (Fault Diagnosis): per identificare e localizzare la causa dell'errore
- **Isolamento del guasto** (Fault Isolation): per escludere dei componenti guasti (logici e fisici) .
- **Riconfigurazione del sistema** (System Reconfiguration): per riassegnare tasks a componenti non guasti.
- **Reinizializzazione del sistema** (System Reinizialization): per controllare e definire la nuova configurazione del sistema.

L'impiego sufficientemente ridondato dei componenti, permette il ripristino del sistema senza un'attività esplicita di rilevamento dell'errore. Questa tecnica è chiamata **mascheramento dei guasti** (Fault Masking).
Le tecniche presentate sono strettamente collegate alle tipologie di guasto e di errore che coinvolgono il sistema. E' opportuno fare precise assunzioni su guasti, errori e fallimenti (utilizzando le classificazioni viste nel

paragrafo degli impedimenti) prima di scegliere ed implementare qualsiasi tecnica di tolleranza ai guasti [12].

Rimozione dei guasti

La rimozione dei guasti ha luogo sia in fase di sviluppo sia in fase operativa del sistema. Nella fase di sviluppo questa attività si compone di tre fasi: verifica, diagnosi e correzione. La **verifica** è il processo per controllare che il sistema rispetti certe condizioni predefinite. Se le condizioni non sono soddisfatte allora è necessaria una **diagnosi** di tale impedimento. Una volta diagnosticato il guasto è opportuno svolgere un'azione di **correzione**.
Durante la fase operativa del sistema, la rimozione dei guasti è intesa come manutenzione, la quale può essere correttiva oppure preventiva. La **manutenzione correttiva** rende possibile la rimozione di guasti che si sono manifestati esplicitamente sottoforma di errori. Mentre la **manutenzione preventiva** permette di scoprire e di rimuovere guasti prima che essi si manifestino.

Previsione dei guasti

La previsione dei guasti è svolta allo scopo di valutare il comportamento del sistema rispetto all'occorrenza di guasti. Questa attività è **qualitativa** se è intesa a identificare e classificare le modalità di fallimento del sistema. E' invece **quantitativa** se cerca di stimare in termini di probabilità, l'incidenza dei guasti rispetto alle misure di dependability.

1.3 Qualità del servizio (QoS), Performance e Performability

Spesso per indicare le proprietà del servizio offerto da un sistema è usata l'espressione generica **Qualità del Servizio (QoS)**. Questa espressione può includere diverse caratteristiche come l'affidabilità, la tolleranza ai guasti, il tempo di risposta, la velocità di esecuzione o la facilità di utilizzo. Includendo quindi aspetti qualitativi e quantitativi, ed in modo più specifico sia misure di dependability che di performance.
L'analisi della qualità del servizio coinvolge tutto il ciclo di vita di un sistema. Durante la fase di specifica del sistema mira a delineare i parametri che caratterizzano il servizio e a definirne i valori attesi. Quest'ultimi sono da ritenersi puramente indicativi, perchè nelle prime fasi di realizzazione non si ha una conoscenza approfondita del sistema. Nelle fasi successive, compatibilmente al budget previsto, sono condotti altri studi sulla qualità del servizio. Essi porteranno a risultati più attendibili grazie alla maggiore disponibilità di informazioni ed esperienza del sistema. In fase conclusiva si confronteranno i valori attesi con quelli effettivamente misurati per validare il sistema rispetto alla specifica.
Lo scopo di questa tesi è la valutazione di aspetti quantitativi, perciò dopo aver descritto la dependabilty nelle sue forme, è opportuno presentare l'altra misura quantitativa: la **performance**. E' presente anche una misura chiamata **Performability** che combina sia aspetti di affidabilità sia quelli di prestazione [9].

1.3.1 Metriche di performance

La performance può essere espressa in termini di *efficacia* riferendosi ad esempio, al tempo di risposta oppure al throughput, mentre in termini di *efficienza* quando si riferisce all'utilizzo delle risorse. Le misure di performance permettono di valutare le prestazioni del sistema durante le normali condizioni di funzionamento, quindi in assenza di fallimenti. Per la valutazione relativa ad un sistema oppure per confrontare due o più sistemi è opportuno scegliere dei criteri che vengono chiamati metriche. Metriche diverse possono portare a valori di performance totalmente diversi. E' difficile individuare le metriche giuste per una corretta valutazione delle prestazioni di un sistema. Spesso la selezione delle metriche dipende dal tipo di applicazione, perché una stessa metrica può avere un peso differente per un'applicazione rispetto ad un'altra.
Le metriche per la valutazione di performance rientrano in due categorie principali: *Orientate al sistema* (System-Oriented) ed *Orientate all'utente* (User-Oriented). Le metriche orientate al sistema si riferiscono al throughput e all'utilizzo di risorse.
Il **throughput** esprime quante occorrenze di transazioni, processi, jobs, pacchetti sono osservate in unità di tempo, è definito dal rapporto $\frac{n}{\Delta}$ dove n sono occorrenze e Δ è un determinato intervallo di tempo. Un caso particolare di throughput legato alle reti di comunicazione, è la larghezza di banda *bandwidth* che quantifica il numero di bits trasmessi in un secondo. Altre metriche riferite alle reti di comunicazione sono il ritardo dei pacchetti sia in un verso **one-way** che in entrambi **roundtrip**, il **jitter** che esprime la variazione del ritardo e il **loss-rate** che rappresenta la frazione dei pacchetti che possono essere persi o erronei. L'utilizzo di una risorsa è invece rappresentato dalla frazione di tempo in cui la risorsa è occupata.

Le metriche orientate all'utente tipicamente includono il tempo di risposta oppure di richiesta-risposta *turn-around*. Queste metriche si riferiscono al tempo trascorso dall'inizio di una richiesta da un utente o da un'applicazione, fino alla ricezione della risposta dal sistema.
Inoltre possono essere definite altre metriche ad-hoc in risposta a specifiche necessità legate al problema o al sistema in esame.
Un passo ugualmente importante alla selezione delle metriche è la selezione del carico di lavoro **workload**.

1.3.2 Carico di lavoro (Workload)

Un sistema è progettato per funzionare in un particolare ambiente e per uno specifico carico di lavoro. Per questa ragione un'analisi di performance di un sistema deve considerare il carico di lavoro, il quale è definito come l'insieme delle attività, delle transazioni e dei dati, che vengono processati in un dato periodo di tempo; oppure più semplicemente, può essere visto come la domanda degli utenti del sistema [7]. Il carico di lavoro che realmente interessa il sistema è spesso complesso ed irripetibile, perciò è opportuno fare delle assunzioni che permettono di definirne un modello.
Tale modello deve essere:

- **Rappresentativo**: deve catturare accuratamente gli aspetti statici e dinamici del carico di lavoro reale.
- **Riproducibile**: deve essere facilmente riproducibile e modificabile, in maniera tale da essere usato ripetutamente in differenti studi.
- **Compatto**: deve essere compatto in modo tale da essere facilmente

utilizzato su sistemi diversi, ciò è utile se si vogliono confrontare sistemi differenti progettati per lo stesso scopo.

1.4 Tecniche e strumenti di valutazione

Una volta identificate le quantità del sistema da valutare, è necessario scegliere la tecnica di valutazione più adatta. Ogni tecnica ha vantaggi e svantaggi ed è difficile poter affermare che una tecnica sia migliore di un'altra. Per la selezione di una tecnica, molti sono i fattori da prendere in considerazione, principalmente bisogna aver chiaro l'obiettivo da raggiungere e la fase in cui tale tecnica deve essere impiegata. Alcuni autori classificano le tecniche di valutazione in due categorie: *basate sul modello* che utilizzano un modello del sistema e *basate sul sistema* che agiscono direttamente sul sistema reale. Le tecniche basate su modello si distinguono poi in analitiche e simulative. Durante le fasi iniziali di progettazione del sistema, cioé quando ancora il sistema non è stato realizzato e le misurazioni dirette sono ovviamente impossibili, allora può risultare conveniente usare un semplice modello analitico. Avanzando nella fase di progettazione, aumentano anche le informazioni e i dettagli del sistema, così è più opportuno utilizzare una simulazione oppure una tecnica basata su un modello analitico più complesso. Infine, quando il sistema reale è sviluppato e disponibile, le misurazioni diventano possibili.
E' ritenuta una buona pratica confrontare i risultati ottenuti utilizzando tecniche differenti, in questo modo i risultati acquistano maggiore attendibilità e credibilità.

1.4.1 Definizione del modello

Per definire un buon modello che rappresenti il sistema oggetto di analisi, occorre molta cura ed attenzione. Un buon modello deve descrivere il comportamento del sistema nel modo più accurato possibile, ma allo stesso tempo deve essere semplice. Queste due caratteristiche sono tra loro contrastanti, ma entrambe da tenere in considerazione. Infatti, maggiori sono i dettagli catturati del comportamento del sistema, tanto più la rappresentazione è accurata, ma anche più complessa. D'altra parte però, un modello complesso richiede molti parametri, diventando così difficile da trattare per costi, tempi e risorse. Quindi in base all'obiettivo di analisi, è oppurtuno includere nel modello i parametri che si riferiscono ai dettagli del sistema più importanti e trascurare invece quelli meno significativi.

1.4.2 Tecniche analitiche

Esistono molteplici formalismi che permettono la valutazione di dependability e performance, i quali sono classificati in due categorie: *combinatori* e *basati sullo spazio degli stati*. I formalismi combinatori hanno il vantaggio di essere molto semplici da trattare, putroppo però non riescono a rappresentare certe dipendenze dei sistemi, poiché necessitano di assunzioni troppo restrittive come l'indipendenza dei fallimenti e l'indipendenza delle riparazioni. Tra questi formalismi si trovano i *Fault Trees*, i *Reliability Block Diagram* e i *Reliability Graphs*. Mentre i formalismi basati sullo spazio degli stati sono più sofisticati, ma anche più difficili da trattare, poiché soffrono del problema dell'esplosione degli stati. Tali formalismi sono le *Catene di Markov*, le *Reti di Code*, le *Reti di Guadagno* (SRN), le *Reti di Petri* e le loro varianti (SPN, GSPN, SAN). Per la definizione e la

risoluzione di modelli con questi formalismi sono disponibili diversi strumenti automatici, tra i quali si ricordano [9, 21, 22, 23]: **SHARPE**, **SURF-2**, **SPNP**, **UltraSAN**, **GreatSPN**, **DEEM**, **MÖBIUS**.

1.4.3 Tecniche simulative

Talvolta il modello del sistema risulta troppo complicato per essere trattato con tecniche analitiche, è necessario allora ricorrere ad altre tecniche. Quando il sistema reale è già realizzato e disponibile, anziché adottare un modello è preferibile utilizzarlo direttamente, a condizione che questo non sia distruttivo per il sistema ed abbia costi ragionevoli. Nelle situazioni in cui tale approccio non è praticabile, è usata la simulazione [10, 15].
Le tecniche simulative sono classificate in *statiche* se si basano su un modello del sistema indipendente dal tempo, e *dinamiche* se invece il modello varia nel tempo. Le tecniche statiche includono modelli non parametrici come **Montecarlo** e guidati da tracce **Trace-Driven**. Quest'ultimi modelli sono caratterizzati dall'utilizzo di sequenze di eventi dette *tracce*, che provengono da osservazioni sperimentali.
Le tecniche dinamiche sono invece parametriche e sono distinte a seconda della variazione del modello rispetto al tempo che può essere *discreta* oppure *continua*. Più precisamente, queste tecniche utilizzano un modello formato da variabili che permettono di rappresentare lo stato del sistema. Se la modifica delle variabili di stato, a seguito dell'occorrenza di un evento, avviene in maniera discontinua nel tempo la tecnica che ne risulta è a **Eventi Discreti**; altrimenti se questa è continua, la tecnica diventa ad **Eventi Continui**. Quest'ultima tecnica comprende modelli definiti da equazioni differenziali oppure alle differenze. Esistono anche tecniche

miste che combinano sia eventi continui sia discreti. Per lo scopo di questa tesi si utilizzerano solo eventi discreti.
La simulazione ad eventi discreti necessita di un riferimento temporale, solitamente diverso dal tempo reale, detto *tempo simulato* ed un meccanismo che possa incrementarlo. Sono due i meccanismi principali: *l'avanzamento al prossimo evento* ed *l'avanzamento ad intervallo fisso*. Utilizzando il primo dei due, quello più diffuso, una simulazione avanza nel seguente modo: il tempo simulato parte da zero e viene incrementato ogni volta della quantità di tempo necessaria al verificarsi del prossimo evento. Durante la simulazione, le variabili di stato variano per effetto degli eventi occorsi, permettendo così di creare una storia dell'esecuzione. Quest'ultima servirà per ricavare attraverso metodi statistici, delle stime di quantità di interesse (dependability, performance e performability) del sistema rappresentato.
Gli strumenti per la simulazione, detti simulatori, possono essere per scopi generici come **Arena** e **SIMULA**, oppure finalizzati ad applicazioni specifiche come **NS-2** [24] e **SSF** [25], orientati allo studio di reti e protocolli. Per quanto riguarda gli algoritmi e le applicazioni distribuite è presente il framework **Neko** [19], utilizzato in questa tesi e presentato nel dettaglio più avanti, il quale permette sia la simulazione su reti virtuali sia il testing su reti reali.

1.4.4 Tecniche sperimentali

A differenza delle tecniche precedenti, le tecniche sperimentali operano direttamente sul sistema reale, come già accennato esse non sono praticabili in assenza del sistema oppure di un suo prototipo. L' osservazione

sul campo del comportamento del sistema con il carico di lavoro reale, ha il vantaggio di raccogliere dei dati con un livello di accuratezza nettamente maggiore rispetto a quelli ricavati da qualsiasi modello del sistema. Purtroppo in contesti critici è difficile, se nonché impossibile, effettuare delle misurazioni dirette del sistema, per ragioni di sicurezza (safety) perché potrebbero alterarne il loro funzionamento normale, e di costo perché richiedono tempi lunghi e strumentazione apposita. Nel caso di valutazione di dependability, le misure sperimentali potrebbero risultare inopportune perché gli eventi da osservare, come ad esempio i fallimenti su sistemi ad alta affidabilità, possono essere così rari da comportare delle sessioni di misurazione molto lunghe e talvolta insufficienti.
Comunque nello studio di dependability e performance, i dati ottenuti da prove sul sistema reale sono molto importanti, perché servono alla validazione dei modelli (analitici e simulativi) rispetto al sistema reale.
In fase prototipale, una tecnica di **Fault-injection** [16, 17] può essere utile per valutare il comportamento del sistema in presenza di guasti, in particolare per determinare la copertura (coverage) di rilevamento dei guasti ed la capacità di ripristino e riconfigurazione del sistema, dei meccanismi fault-tolerance impiegati. Una tecnica di Fault-Injection permette di innescare nel sistema guasti (virtuali o reali) utilizzando degli strumenti hardware oppure software.

Capitolo 2

ATN (Aeronautical Telecommunications Network)

2.1 Introduzione

Nei primi anni ottanta, l'**ICAO** (International Civil Aviation Organization) ha evidenziato i limiti dei sistemi esistenti per riuscire a soddisfare i futuri volumi di traffico aero previsti, ed ha istituito una commissione speciale denominata **FANS** (Future Air Navigation Systems) al fine di studiare, identificare e promuovere nuovi concetti e tecnologie per il controllo del traffico aereo. Ciò ha portato alla definizione del concetto di sistema globale **CNS/ATM** per la comunicazione, la navigazione e la sorveglianza (CNS) e la gestione del traffico aereo (ATM), identificando nella comunicazione dati e nei sistemi basati su satelliti (es. GPS) i principali mezzi per il miglioramento dei sistemi esistenti. Successivamente, tale concetto di sistema globale CNS/ATM è stato ulteriormente sviluppato e raffinato.
La rete ATN (Aeronautical Telecommunications Network) [44, 45, 46] fa

parte del sistema globale CNS/ATM ed include sia entità applicative sia servizi di comunicazione che consentono l'interoperabilità tra le parti coinvolte nella gestione del traffico aereo come le unità di controllo, i velivoli e le compagnie aeree. Dalla fine degli anni novanta anche a livello europeo si contribuisce allo sviluppo di ATN, in particolare, Eurocontrol ha promosso il programma Link2000+ [40] per pianificare e coordinare l'implementazione del servizio di collegamento dati aria/terra.
I benefici attesi dalla comunicazione dati attraverso la rete ATN sono l'incremento dell'automazione delle attuali procedure per il controllo del traffico aereo, con la conseguente diminuzione del carico di lavoro degli operatori; e la riduzione delle incomprensioni tra i piloti e controllori dovute alla trasmissione sul canale vocale (via radio). La rete ATN intende offrire un servizio di comunicazione affidabile, robusta e ad alta integrità tra le unità di controllo (Air Traffic Services ATS), le compagnie aree ed altre organizzazioni aeronautiche (Aeronautical Operational Control Services AOC) ed i velivoli.

Le caratteristiche principali della rete ATN sono le seguenti:

- E' un sistema di comunicazione dati specifico ed esclusivo per le organizzazioni e l'entità aeronautiche.
- Offre servizi di comunicazione sia tra più sistemi di terra sia tra sistemi di terra ed avionici.
- Fornisce un servizio di comunicazione finalizzato al raggiungimento dei requisiti di qualità del servizio delle applicazioni.
- Utilizza ed integra diverse reti di comunicazioni dati: aeronautiche,

commerciali e pubbliche formando un'infrastruttura di comunicazione aeronautica globale.

2.2 Architettura

Il paradigma di comunicazione usato dalla rete ATN è il modello OSI (Open Systems Interconnections) costituito dai livelli (dal più alto al più basso): applicazione, presentazione, sessione, trasporto, rete, collegamenti dati e fisico. Il primo livello riguarda le entità applicative che comprendono, oltre alla parte applicativa dell'utente, anche un'interfaccia di comunicazione con il livello sottostante denominata ASE (Application Service Entity). I livelli di presentazione e sessione, sono conoscuti come *Upper Layer Communication Services* (ULCS) ed hanno la funzione di codificare e decodificare le informazioni nel formato atteso dal livello applicativo e dal livello di trasporto. Quest'ultimo e il livello di rete, identificati con l'espressione *Internet Communication Services* (ICS), permettono la comunicazione tra sistemi diversi.

L'architettura della rete ATN è composta da **sottoreti**, **Intermediate Systems** IS (routers) ed **End Systems** ES. Una sottorete rappresenta il livello di collegamento dati, è basata su una specifica tecnologia come ad esempio: Ethernet, ATM (Asynchronous Transfer Mode), X.25 o VDL2 (VHF Data Link) e permette la trasmissione delle informazioni tra i sistemi. Un sistema IS implementa il livello di rete: integra più sottoreti distinte ed instrada i pacchetti dati. Infine, un sistema ES comprende le applicazioni, i servizi di comunicazione di alto livello (ULCS) e di interconnessione (ICS) necessari per lo scambio di messaggi applicativi tra i sistemi ATN.

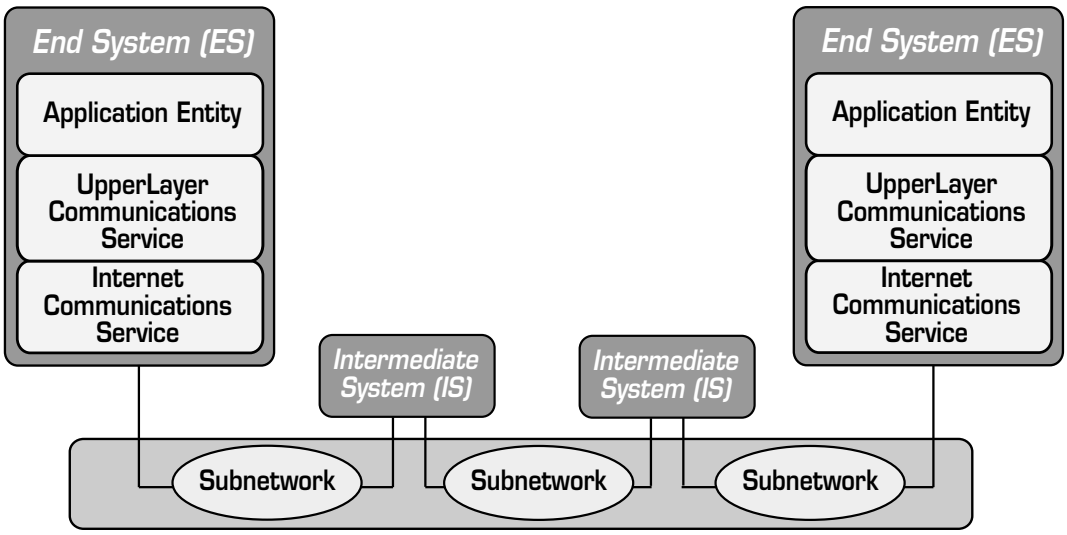

Figura 2.1: Le componenti di ATN

2.2.1 Identificazione e Indirizzamento

Lo schema di indirizzamento ATN ha una struttura gerarchica. La prima parte dell'indirizzo è utilizzata dalle organizzazioni come ISO e ICAO per l'amministrazione a livello internazionale, mentre la parte restante è assegnata invece dalle entità con ruoli amministrativi via via più limitati (le autorità regionali, nazionali e locali).

La dimensione di indirizzo ATN completo può variare da 21 a 22 bytes a seconda della lunghezza del campo TSEL (TSAP Selector) ovvero il selettore del servizio di trasporto che può essere di uno oppure due bytes. I primi due campi AFI (Authority Format Identifier) e IDI (Initial Domain Identifier) sono assegnati e fissati da ISO, mentre i successivi tre: VER (Version), ADM (Administration) e RDF (Routing Domain Format) da ICAO

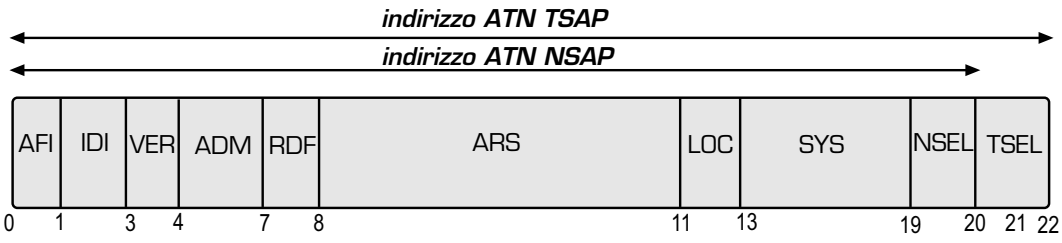

Figura 2.2: Indirizzo ATN

e IATA. I campi ARS (Administrative Regional Selector), LOC (Location), SYS (System Identifier) ed il selettore del servizio di rete NSEL (NSAP Selector) sono assegnati dalle autorità regionali e nazionali, mentre l'ultimo campo TSEL è assegnato localmente.
Dall'indirizzo completo ATN si ricavano altri indirizzi per l'identificazione delle componenti di ATN. Tra questi troviamo:

- **NSAP** (Network Service Access Point): un indirizzo formato dai primi 20 byte (compreso il campo N-SEL) che identifica un utente del servizio di rete a livello globale.
- **TSAP** (Transport Service Access Point): un indirizzo formato dal precedente e da 1 o 2 byte relativi al campo T-SEL (il selettore di trasporto assegnato localmente) che identifica un utente del servizio di trasporto.

2.3 Classi di servizio e priorità

La rete ATN differenzia il traffico dati in due categorie principali: ATSC (Air Traffic Service Communications) e AOC (Airline Operational Com-

munications). Poiché le applicazioni ATN generano prevalentemente traffico dati ATSC, allora per poter facilitare il raggiungimento dei requisiti applicativi, questo è suddiviso in varie *classi di servizio* identificate da lettere alfabetiche dalla A (la migliore) alla H (la peggiore). La metrica di classificazione usata è il tempo massimo di ritardo di comunicazione (oneway).

Classe ATSC	Ritardo massimo oneway (in secondi)
A	riservata
B	4.5
C	7.2
D	13.5
E	18
F	27
G	50
H	100

Figura 2.3: Classi di servizio ATSC

Ad ogni messaggio applicativo viene fatto corrispondere una classe di servizio, per ovvie ragioni ai messaggi urgenti è associata la migliore classe disponibile. Dal livello applicativo le informazioni sulla classe di servizio sono propagate ai livelli più bassi utilizzando una struttura detta **ATN security label**, contenuta nell'intestazione di ogni pacchetto dei protocolli di trasporto e di rete. Contrariamente al nome, l'ATN security label non offre alcun supporto per la sicurezza delle applicazioni a livello di trasporto o di rete (es. autenticazione, firma digitale, ..) poiché il compito di garantirla è affidato ai livelli superiori (Upper Layer).

Un altro aspetto importante di ATN è l'uso di **priorità** per garantire la precedenza dei dati con priorità alta rispetto a quelli con priorità più bassa.

2.4 Livello di trasporto

Il livello di trasporto ATN fornisce un servizio di trasferimento dati end-to-end trasparente agli utenti (TSUser). Il protocollo di trasporto opera soltanto tra End System (ES) ed è implementato solo su tali sistemi. Nel rispetto del paradigma di comunicazione OSI, le entità di livello di trasporto ATN comunicano utilizzando sempre il servizio offerto dal livello rete. Le tipologie di servizio previste sono due:

- **Connection mode Transport Service (COTS):** che permette lo scambio di dati attraverso un canale di comunicazione logico (connessione) tra due TSUser garantendo l'ordine di consegna e l'integrità dei dati.
- **Connectionless mode Transport Service (CLTS):** che permette lo scambio di dati senza stabilire una connessione tra due TSUser, non garantendo però una consegna affidabile.

Per il supporto della prima modalità è usato il protocollo TP4 (standard ISO 8073) simile al protocollo TCP, mentre per l'altra è usato il protocollo CLTP (standard ISO 8602) simile al protocollo UDP. Entrambi i protocolli permettono il trasferimento di pacchetti dati TSDU (Transport Service Data Units) sfruttando il servizio offerto dal livello di rete ATN.

2.4.1 Protocollo orientato alla connessione TP4

Il protocollo TP4 attraverso l'uso combinato di meccanismi basati su *handshake* e su *timer*, permette di stabilire una connessione tra due sistemi, trasferire i dati e chiudere la connessione. Il meccanismo di handshake è utilizzato per creare un accordo tra due parti su un'azione da compiere, come ad esempio, l'apertura e la chiusura di connessione. L'impiego di timer garantisce invece la validità dei dati permettendo di scartare dati che si riferiscono ad esempio a connessioni già terminate.
La tipica procedura di connessione del protocollo TP4 tra due utenti (TSUser) A e B è la seguente (fig. 2.4).

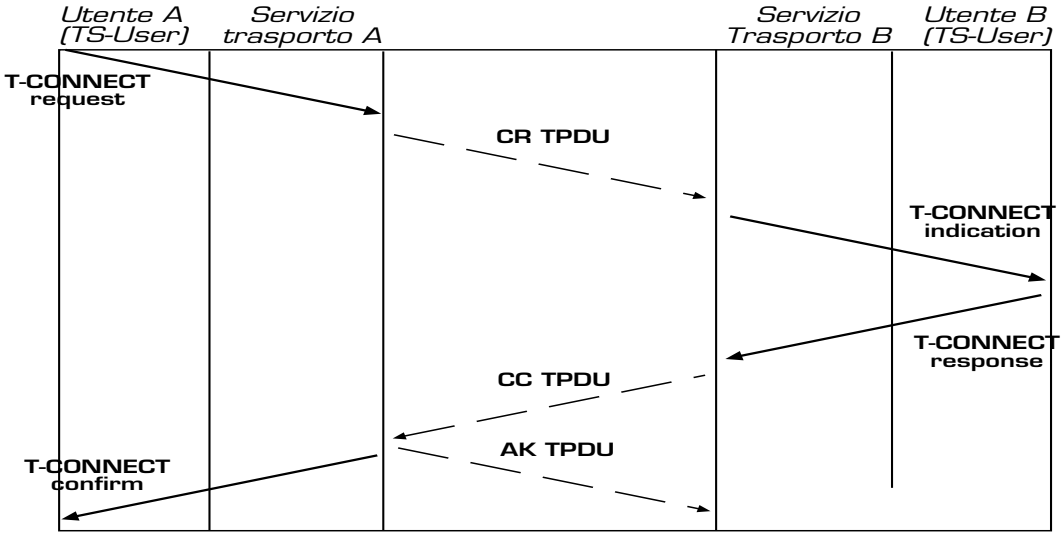

Figura 2.4: Connessione TP4

L'utente A invia una richiesta **T-CONNECT request** al suo livello di trasporto specificando i parametri necessari alla connessione: l'indirizzo

(ATN NSAP e selettore TSAP) del chiamante e del destinatario, la classe di servizio desiderata (ATN Security Label) e la priorità. Inoltre, è possibile specificare il tipo di checksum (nessuno, a 16 bit, a 32 bit esteso) per garantire l'integrità dei dati. Tali parametri servono al livello di trasporto di A per effettuare una richiesta di connessione (CR TPDU) al livello di trasporto di B. Dopo che B ha ricevuto la richiesta di connessione mediante **T-CONNECT Indication**, esso può accettarla e inviare una risposta **T-CONNECT response** al suo livello di trasporto. Quest'ultimo conferma la connessione (CC TPDU) al livello di trasporto di A, che avvisa l'utente A attraverso **T-CONNECT confirm**. Il livello di trasporto A invia poi un'ulteriore conferma dell'avvenuta connessione (ACK).
Durante questa fase i due utenti si accordano sulla qualità del servizio che possono mantenere per la tutta la durata della connessione, se tale accordo non è trovato, la connessione non viene stabilita.
Una volta instaurata la connessione gli utenti possono trasferire i dati attraverso la primitiva **T-DATA request** del livello di trasporto, che suddivide i dati in più unità TPDU e li trasmette al destinatario mediante le primitive **N-UNITDATA** del protocollo di rete.

Il protocollo TP4 implementa un meccanismo a finestra scorrevole per il controllo del flusso dei dati, capace di preservare l'ordine dei TPDU e garantire che se un TPDU è ricevuto allora tutti i TPDU precedenti sono stati consegnati. La finestra può essere ridimensionata dinamicamente per prevenire il congestionamento delle rete.
Il rilascio della connessione può avvenire per iniziativa dell'utente oppure del fornitore del servizio. Quest'ultimo caso può verificarsi in qualsiasi momento per problemi alla risorse (locali o remote) oppure anche in

fase di connessione, come ad esempio il mancato accordo sulla qualità del servizio.

2.4.2 Protocollo senza connessione CLTP

Il protocollo senza connessione CLTP permette di trasferire i dati in una sola direzione (one-way) senza la necessità di stabilire una connessione tra gli utenti. Ciò riduce l'overhead relativo alla connessione ed aumenta la velocità di trasferimento, pagando però in termini di affidabilità della comunicazione. Il protocollo CLTP utilizza solo le primitive **N-UNITDATA request** e **N-UNITDATA indication** offerte dal servizio di rete. A differenza del protocollo orientato alla connessione, non esiste alcuna forma di negoziazione della classe di servizio tra le parti, per questa ragione un'unità TSDU di dati è indipendente dall'altra ed possibile associargli differenti classi di servizio.

2.5 Livello di rete

Il livello di rete ATN, come quello di trasporto, è specificato per fornire un servizio nelle due modalità: **orientato alla connessione** e **senza connessione**. Queste due modalità appaiono giustificate nel livello di trasporto per poter supportare diverse tipologie di applicazione, mentre non è così per il livello di rete dove l'omogeneità favorisce l'interoperabilità tra i sistemi. A tale proposito è scelta la modalità più semplice ovvero quella senza connessione.

Il livello di rete si compone dei seguenti sottolivelli o ruoli:

- **Subnetwork Independent Convergence Role:** E' responsabile nel fornire un servizio di rete comune a tutte le sottoreti.
- **Subnetwork Dependent Convergence Role:** Disaccoppia le funzioni del sottolivello precendente a seconda delle specifiche caratteristiche delle differenti sottoreti.
- **Subnetwork Access Role:** E' complementare al sottolivello precedente e serve per l'accesso alle specifiche sottoreti.

Il Subnetwork Independent Convergence Role in un ES è responsabile nel fornire un servizio di rete indipendente dalla sottorete a cui l'ES è collegato. Mentre in un IS, esso è responsabile dell'instradamento e del passaggio dei dati lungo il percorso (route) tra due utenti. Tale sottolivello contiene anche i protocolli che supportano lo scambio di informazioni di routing (ES-IS, IS-IS, IDRP).

2.5.1 Protocollo CLNP

Il protocollo CLNP (Connectionless Network Protocol) specificato dallo standard ISO 8473 supporta il servizio di rete senza connessione basandosi sulle primitive: **N-UNITDATA request** e **N-UNITDATA indication**. La procedura di trasferimento dati del protocollo CLNP è la seguente (fig. 2.5):
Un utente del servizio di rete (NS-User) richiama la primitiva **N-UNITDATA request** passando come parametri: i dati utente, l'indirizzo di rete (NSAP) della destinazione e della sorgente, ed altri valori corrispondenti alla classe di servizio (ATN security label) ed alla priorità da assegnare ai dati; inoltre è consentito di specificare una o più sottoreti di preferenza. Queste infor-

mazioni sono utilizzate per formare i pacchetti dati NPDU da inoltrare ad un ES o un IS adiaciente (next-hop) scelto in base alle informazioni di routing locali ottenute dal protocollo ES-IS. Se la dimensione del pacchetto dati supera quella prevista dalla sottorete, il pacchetto è frammentato in parti più piccole e poi trasmesso. Tale procedura è realizzata dal sottolivello *Subnetwork Dependent Convergence Role* poiché è dipendente dal tipo di sottorete impiegata.

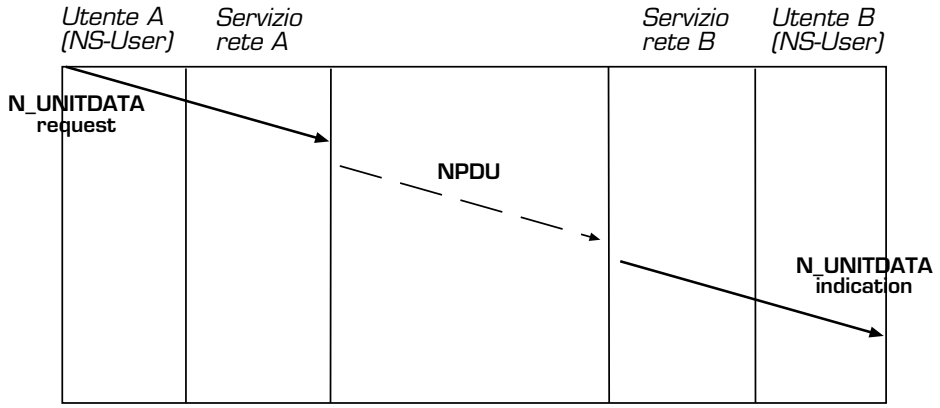

Figura 2.5: Trasferimento dati CLNP

Il pacchetto dati (NPDU) passa da un sistema IS ad un altro adiacente, grazie alle informazioni di routing in loro possesso ottenute dai protocolli: ES-IS, IS-IS, IDRP. Come vedremo nel paragrafo successivo, queste informazioni sono più complesse rispetto a quelle mantenute nei sistemi ES.
Quando i dati arrivano all'ES di destinazione, sono riassemblati per poter ricostruire l'intero NPDU da recapitare al destinatario (NS-User) attraverso la primitiva **N-UNITDATA indication**. Ovviamente, l'operazione di deframmentazione e riassemblaggio dei NPDU ha luogo in tutte le entità

di rete attraversate per raggiungere la destinazione.
Allo scopo di garantire la priorità dei dati, il livello di rete differenzia i flussi di informazione associando delle code distinte ai dati in uscita in base alla loro priorità.

2.5.2 Routing

L'architettura di routing riflette la natura gerarchica degli indirizzi ATN, essa è giustificata sia da ragioni amministrative come la delegazione delle responsabilità alle autorità locali, sia da quelle tecniche come la riduzione del traffico di rete legato alla diffusione delle informazioni di routing. I sistemi ES e IS interconnessi possono essere raggruppati in aree di routing, le quali a loro volta formano un dominio di routing (RD). L'interconnessione tra domini differenti è ottenuta grazie a dei sistemi IS posti al confine dei domini, chiamati appunto BIS (Boundary Intermediate Systems); raggruppando ancora i domini si ottengono delle confederazioni di domini (RDC). Una confederazione di domini appare ai domini esterni come un unico dominio di routing di cui però non ne conoscono la topologia. Le confederazioni di domini possono essere organizzate in isole ATN e continuando in maniera ricorsiva si possono raggruppare isole ATN in altre isole ancora.
All'interno di un dominio sono impiegati due protocolli: **ES-IS** (standard ISO 9542) tra End Systems e Intermediate Systems e **IS-IS** (standard ISO 10589) tra Intermediate Systems. Per la comunicazione tra domini differenti è utilizzato invece il protocollo IDRP (Inter-Domain Routing Protocol) specificato dallo standard ISO 10747.

Protocollo ES-IS

Il protocollo ES-IS fornisce un meccanismo per lo scambio di informazioni relative alla connettività all'interno di una sottorete locale. Questo protocollo permette in maniera dinamica sia il processo di **configurazione** dove i sistemi ES e IS appartenenti alla stessa sottorete possono conoscersi a vicenda, sia quello di **redirezione** in cui gli IS informano gli ES riguardo ai migliori percorsi (routes) da seguire. Affinché tale protocollo possa svolgere al meglio il proprio compito è necessario che la sottorete supporti l'invio di messaggi in modalità broadcast. Nel processo di configurazione gli ES inviano ad intervalli regolari dei messaggi ESH (End System Hello) contenenti il proprio indirizzo di rete (NSAP) a tutti gli IS della sottorete, in modo analogo ogni IS invia a tutti gli ES un messaggio ISH (Intermediate System Hello). Gli ES hanno una capacità limitata riguardo alle decisioni di routing, se essi non conoscono la locazione della destinazione di un pacchetto semplicemente lo inviano all'IS scoperto durante la configurazione.
La redirezione ha luogo in due casi: il primo, quando la destinazione appartiene alla stessa sottorete della sorgente, e il secondo, se un IS intende proporre un percorso alternativo migliore. Un messaggio di redirezione è inviato dall'IS per informare l'ES della sottorete più appropriata per raggiungere l'indirizzo di destinazione.

Protocollo IS-IS

Il protocollo IS-IS prevede lo scambio di informazioni di routing tra IS appartenenti ad uno stesso dominio. Ciascun IS invia ad ogni altro IS tutte le informazioni di routing in suo possesso, permettendo ad un IS di

costruire la mappa topologica completa del dominio. Periodicamente e ad ogni cambiamento della topologia del dominio, ciascun IS forma un *Link State Protocol Data Unit* (LSP) che identifica i collegamenti attivi tra IS adiacenti, i collegamenti scoperti con il protocollo ES-IS tra ES e IS e la qualità di servizio associata ad ogni link. Questo LSP è inviato a tutti gli IS adiacienti. Quando un IS ha ricevuto tutti gli LSP dagli IS attivi, può costruire la mappa completa del dominio che servirà a calcolare i percorsi (route) ottimali. La procedura descritta è chiamata *Link State Routing*.

Protocollo IDRP

Il protocollo IDRP consente lo scambio delle informazioni di routing tra domini differenti che si riferiscono al minimo necessario per l'indicazione di percorsi, perciò non rivelano la topologia interna dei domini come accade invece nei protocolli precedenti. In particolare, il protocollo IDRP permette le seguenti azioni:

- la comunicazione ai router esterni di un dominio dei percorsi alle destinazioni locali.
- la comunicazione ai router esterni di un dominio dei percorsi provenienti da routers di altri domini.
- l'interpretazione delle informazioni ricevute da altri routers e la scelta del percorso migliore ad una stessa destinazione tra più alternative in base a politiche di routing.

Il protocollo IDRP è implementato nei sistemi BIS, cioé nei sistemi IS posti al confine di un dominio di routing. Un sistema BIS può comunicare simultaneamente con altri BIS appartenenti sia allo stesso dominio che a

domini differenti, utilizzando un protocollo BIS-BIS basato su connessione che garantisce uno scambio affidabile delle informazioni di routing. Il BIS-BIS è una versione semplificata del protocollo TP4 ed utilizza anch'esso il servizio di rete offerto da CLNP. Le informazioni scambiate tra BIS adiacenti riguardano i percorsi per le possibili destinazioni (routes), che il protocollo IDRP usa sia per la comunicazione interna ad un dominio sia esterna.

Supporto della mobilità

La rete ATN incorpora più sottoreti per consentire la comunicazione tra i sistemi fissi e mobili, tra queste ricordiamo: SSR mode S (Secondary Surveillance Radar), AMSS (Aeronautical Mobile Satellite Service) e VDL 2 (VHF Data Link). Se un velivolo fosse collegato sempre alla stessa sottorete e mai ad un'altra, potrebbe essere considerato come un sistema fisso, senza aver bisogno di uno specifico supporto per la mobilità. Nella realtà invece è opportuno considerare i velivoli come sistemi mobili, perché essi eseguono più volte la procedura di *hand-off* per passare da una sottorete ad un'altra adiacente. Un sistema mobile è considerato come un dominio a sè, distinto da quelli di terra, che richiede l'impiego di un router ATN con il supporto delle funzionalità BIS. L'esistenza di sistemi mobili ha un impatto significante sulla rete ATN, perché i collegamenti terra/aria a differenza di quelli di terra, cambiano rapidamente e contribuiscono ad accrescere le informazioni di routing scambiate. La soluzione a questo problema è l'introduzione dei concetti di *isola ATN* e *dominio Home*.
Un'isola ATN, come abbiamo già accennato, è una regione ATN che comprende un certo numero di domini di routing, alcuni dei quali offrono il

collegamento dati aria/terra (air/ground datalink). All'interno di un'isola ATN, almeno un dominio identificato come *backbone* ha il compito di fornire agli altri domini dell'isola di appartenenza, il *percorso di default* verso tutti i velivoli. Inoltre, un backbone è mantenuto continuamente informato sui *percorsi attuali* per raggiungere i velivoli attraverso sottoreti interne all'isola ATN, diventando così il fornitore dei percorsi predefinito. I domini interni all'isola ATN, ma fuori dal backbone, possono raggiungere un aereo in due modi: il primo, preferibile, con un percorso esplicito attraverso la loro sottorete aria/terra, il secondo, mediante un percorso comunicato dal backbone. La gestione dei percorsi verso gli aerei è affidata principalmente ai router che formano il backbone, gli altri router dell'isola ATN hanno a questo proposito un ruolo molto più limitato. La scelta di introdurre il concetto di isola ATN e di backbone rappresenta il primo passo per ottenere un'architettura di routing scalabile. La scalabilità può migliorare ancora se ai velivoli si associa un *dominio Home* cioé un'isola ATN che ha il compito di indicare il percorso di default verso quei velivoli ad altre isole ATN.

Capitolo 3

L'applicazione CPDLC (Controller Pilot Data Link Communication)

La rete ATN rappresenta un'infrastruttura per la comunicazione dati consentendo di supportare nuove applicazioni per la comunicazione, la navigazione e la sorveglianza nell'ambito della gestione del traffico aereo (CNS/ATM). Queste applicazioni si suddividono in due tipologie: *aria/terra* tra i sistemi mobili (i velivoli) e fissi, e *terra/terra* tra sistemi fissi (i centri di controllo). In questo capitolo è descritta una delle principali applicazioni *aria/terra* previste dal programma Link2000+ di Eurocontrol: l'applicazione CPDLC (Controller Pilot Data Link Communication) [40].

3.1 Generalità e principi operativi

L'applicazione CPDLC permette la comunicazione ATC tra un controllore ed un pilota sfruttando la trasmissione digitale di dati (data link). I messaggi scambiati corrispondono alla fraseologia vocale correntemente uti-

lizzata nelle procedure di controllo del traffico aereo. Un controllore può inviare ai velivoli le cosiddette *Clearance* per assegnare, ad esempio: il livello di volo, la velocità, il cambiamento di rotta oppure la frequenza radio; inoltre può assistere il pilota durante il trasferimento del controllo da un'autorità ad un'altra. Mentre un pilota può rispondere ai messaggi, richiedere delle *Clearance* ed altre informazioni.

L'implementazione dell'applicazione CPDLC è motivata dalla necessità di risolvere alcuni problemi che caratterizzano l'attuale tecnologia come il congestionamento del canale vocale per la crescita del volume di traffico, le incomprensioni tra pilota e controllore dovute alla bassa qualità vocale oppure a sbagliate interpretazioni del linguaggio, e la compromissione del segnale dovuto a trasmissioni simultanee (collisioni).
I benefici attesi dall'introduzione di CPDLC sono il miglioramento della gestione e del coordinamento del traffico aereo grazie a collegamenti più diretti ed efficienti tra sistemi di terra e avionici, ma anche tra diversi sistemi di terra rendendo più automatico il trasferimento di comunicazione tra le autorità di controllo. La migrazione verso un sistema più automatico di quello attuale, potrà ridurre il carico di lavoro dei controllori permettendo di gestire maggiori volumi di traffico.
Il programma Link 2000+ prevede l'implementazione delle funzionalità CPDLC in maniera progressiva, permettendo la definizione, la validazione e il raffinamento di concetti, requisiti e procedure operative [41]. Per minimizzare l'impatto sull'operatività di controllori e piloti sono stabiliti i seguenti principi operativi [38]:

1. La coesistenza del canale vocale e quello dati, perciò la comunicazione

dati mediante CDPLC è intesa come complementare alla comunicazione vocale.

2. L'uso di CPDLC è previsto nelle comunicazioni ATC non ritenute critiche, dove la criticità dipende dai tempi di risposta legati al contesto operativo e alla situazione del traffico areo.

3. La scelta dell'uso della voce o di CPDLC rimane a discrezione dell'utente (controllore o pilota).

Oltre a questi principi specifici, CPDLC deve rispettare il principio per cui un velivolo non possa essere contemporaneamente sotto il controllo di più di un'unità ATSU.

3.2 Formato dei messaggi CPDLC

I messaggi CPDLC sono formati da un'intestazione (*Message Header*) e da un corpo composto da uno o al massimo due elementi (*Message Elements*) [38, 40]. L'intestazione ha i seguenti campi:

- *Identificativo*: è un numero che identifica un messaggio ed è assegnato sequenzialmente dal mittente.

- *Riferimento*: è un numero utilizzato nei soli messaggi di risposta e contiene il numero di identificazione del messaggio a cui si riferisce la risposta.

- *Timestamp*: è la marca temporale corrispondente all'invio del messaggio.

- *Richiesta LACK*: indica se è richiesta la conferma automatica dei messaggi (Logical Acknowledgment LACK), quest'ultima è obbligatoria in risposta a tutti i messaggi CPDLC esclusi gli stessi LACK e i messaggi di errore.

Gli elementi che formano il corpo di un messaggio CPDLC sono presi da un insieme di elementi standard ICAO, quest'ultimo include sia elementi per i messaggi provenienti da terra verso i velivoli (Uplink) che viceversa (Downlink).

3.3 Descrizione dei servizi

I servizi previsti per l'applicazione CPDLC [38, 40] si riferiscono a molteplici contesti operativi: *En Route* (in rotta), *TMA* (in prossimità di un aeroporto), *Aerodrome* (all'interno di un aeroporto) ed *Approach/Departure* (decollo o atterraggio). In questa prima fase di CPDLC è considerato solo il contesto di *En Route*, che corrisponde alla parte più lunga di un volo (successiva al decollo e precedente l'atterraggio) ed allo stesso tempo, meno critica delle altre riguardo al tipo comunicazione.

I servizi forniti dall'applicazione CPDLC sono i seguenti:

- **DLIC** (Data Link Initiation Capability) per inizializzare il collegamento dati (data link).
- **ACM** (ATC Communications Management) per attivare e disattivare il canale dati CPDLC e trasferire la comunicazione dati e vocale.
- **ACL** (ATC Clearances) per scambiare messaggi operativi tra controllore e pilota.

- **AMC** (ATC Microphone Check) per verificare la frequenza radio del canale vocale.

3.3.1 Transazioni CPDLC e i parametri relativi

Le transazioni CPDLC sono rappresentate attraverso un diagramma temporale che permette di ben definire i parametri utilizzati per la specifica dei requisiti di performance CPDLC [47]. Alcuni di questi parametri, nel capitolo successivo saranno scelti come le metriche oggetto di studio in questa tesi.

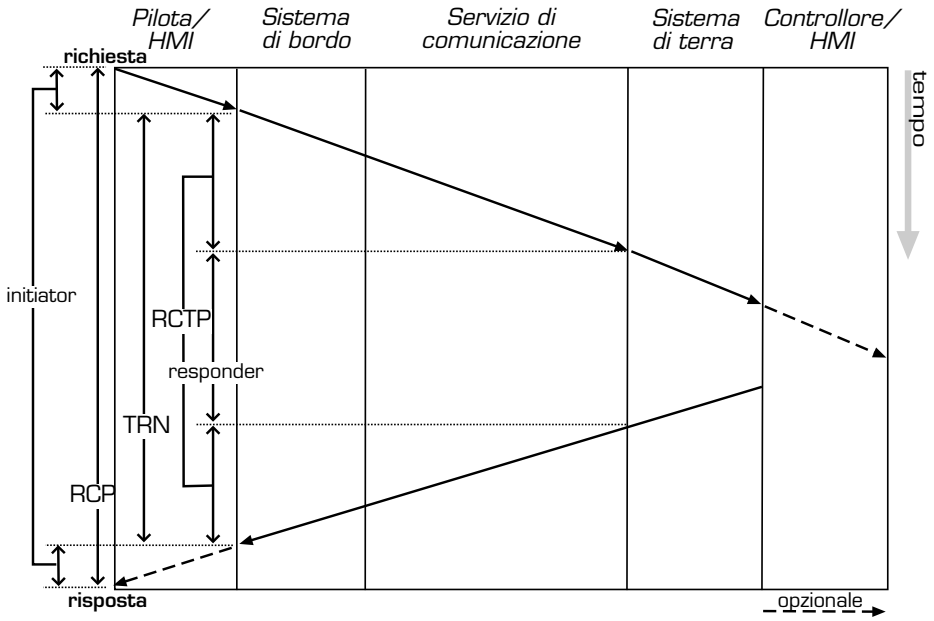

Figura 3.1: Diagramma temporale e definizione di RCP, TRN e RCTP

Il parametro RCP (Required Communication Performance) è definito come il tempo totale intercorso tra una richiesta e la risposta corrispon-

dente. Questo parametro include la parte relativa a chi inizia una richiesta ($Initiator$), ovvero la somma del tempo dalla composizione al rilascio di un messaggio e quello dalla notifica della risposta alla lettura del messaggio. Il parametro RCP comprende anche il tempo attribuito a chi risponde ($Responder$) cioé quello dalla notifica della ricezione della richiesta al rilascio della risposta.
Il parametro TRN (Transaction) è definito come il tempo tra una richiesta e la risposta corrispondente, sottraendo però il contributo dell'Initiator: $TRN = RCP - Initiator$, ciò esclude il tempo di interazione utente/sistema per la composizione e il riconoscimento del messaggio.
Il parametro $RCTP$ (Required Communication Technical Performance) è definito dalla differenza tra il parametro TRN e il contributo del Responder: $RCTP = TRN - Responder$. Questo parametro è chiamato *tempo tecnico di comunicazione* e rappresenta la somma del tempo da attribuire al sistema di terra, a quello di bordo e alla rete di comunicazione.

3.3.2 Servizio DLIC

Il servizio DLIC è utilizzato prima di ogni altro servizio applicativo allo scopo di fornire le informazioni necessarie all'attivazione del collegamento tra un'unità di controllo (ATSU) ed un velivolo.
Questo servizio attraverso la procedura di **Logon** iniziata da un aereo, permette l'associazione delle informazioni di volo provenienti dall'aereo (identificativo aereo, ora e data di partenza, aeroporto di partenza e arrivo) al piano di volo conosciuto dall'ATSU.
E' prevista comunque un'altra procedura *Contact* iniziata da terra, per

richiedere ad un velivolo di effettuare il Logon ad uno specifico ATSU.

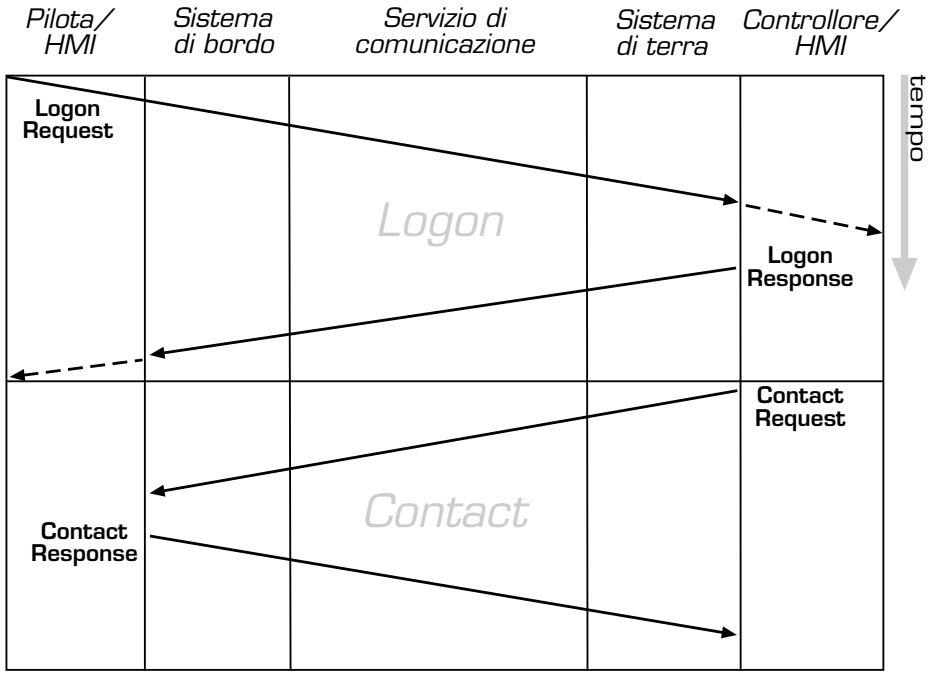

Figura 3.2: Servizio DLIC: Logon e Contact

3.3.3 Servizio ACM

Il servizio ACM, utilizzabile solo dopo il servizio DLIC, consente attraverso le procedure iniziate da terra *CPDLC Start* e *CPDLC End*, l'attivazione e la disattivazione del collegamento per lo scambio dei messaggi CPDLC tra controllore e pilota.

Il servizio ACM prevede altre procedure che consentono di facilitare le operazioni di controllori e piloti durante il trasferimento della comunicazione sia dati che vocale da un'autorità di controllo ad un'altra.

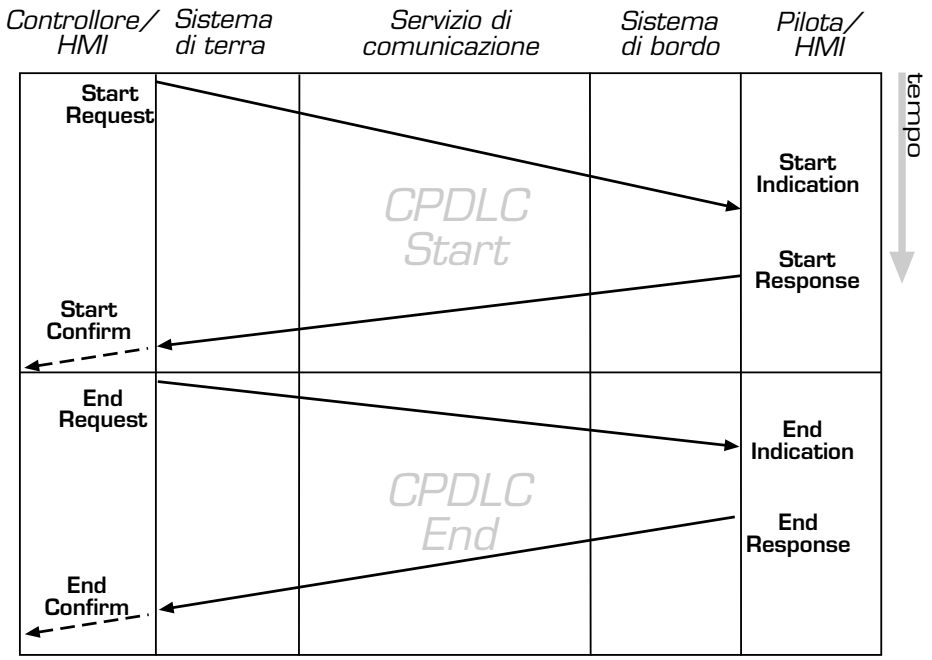

Figura 3.3: Servizio ACM: CPDLC Start e CPDLC End

3.3.4 Servizio ACL

Il servizio ACL permette la comunicazione di informazioni tra aereo e controllore di volo riguardanti: la richiesta e risposta di clearance, la sorveglianza dei livelli di volo ed altre informazioni generali. Il presente servizio è disponibile solo dopo aver attivato, mediante il servizio ACM, il collegamento dati tra il sistema di terra e l'aereo. Le modalità per lo scambio di messaggi sono due *iniziata da terra* (Ground-Initiated) e *iniziata dall'aereo* (Air-Initiated).

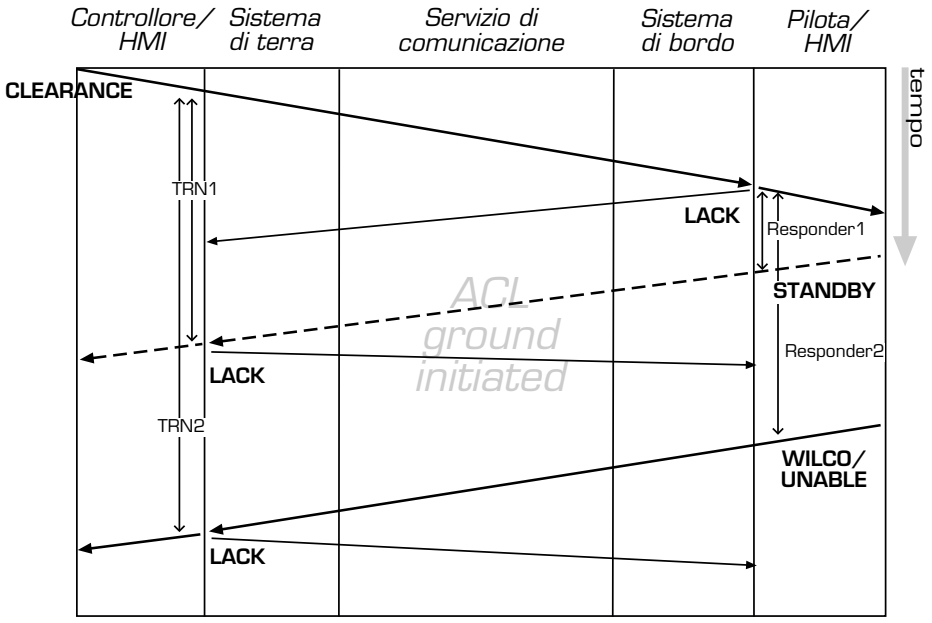

Figura 3.4: Servizio ACL Ground-Initiated

Modalità Ground-Initiated

Un controllore può iniziare una transazione ACL inviando ad un aereo un messaggio contenente un'azione da compiere (Clearance). Il sistema di bordo, dopo aver ricevuto e visualizzato il messaggio al pilota, risponde automaticamente un **LACK**. Se il pilota è impegnato in altre operazioni e non può rispondere, invia un messaggio di **STANDBY** al controllore comunicandogli di rimanere in attesa della risposta, e il sistema di terra conferma la ricezione con un **LACK**. Dopodiché se il pilota può compiere l'azione richiesta, invia al controllore un messaggio di risposta **WILCO**, altrimenti invia il messaggio **UNABLE**. Anche in questo caso il sistema di terra invia automaticamente un **LACK** di conferma. Il messaggio WILCO (Will Comply) è inteso come la promessa di compiere una certa azione.

Modalità Air-Initiated

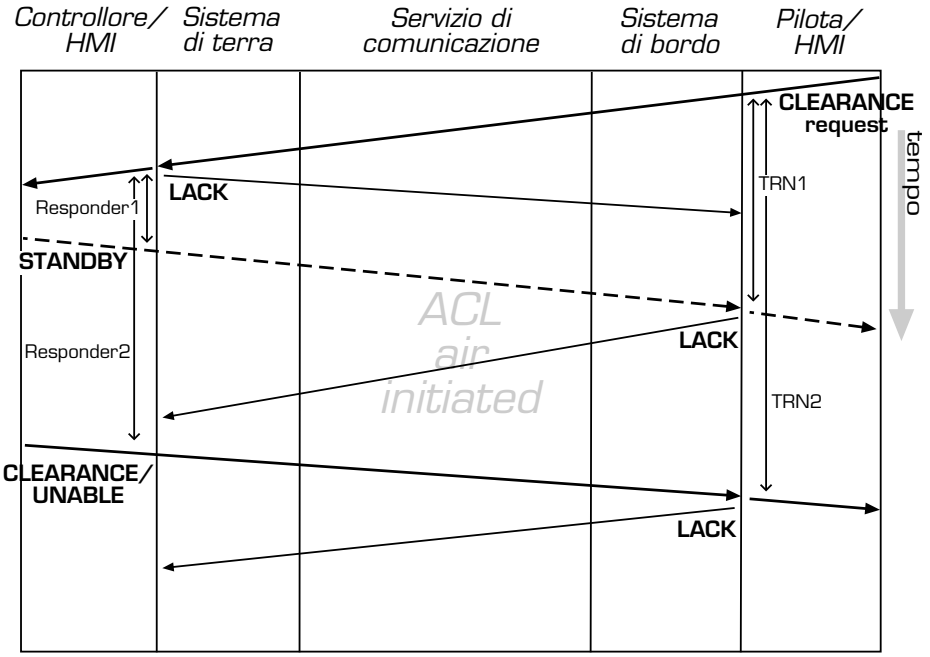

Figura 3.5: Servizio ACL Air-Initiated

Un pilota invia una richiesta di Clearance al controllore. Quando il sistema di terra riceve la richiesta e può avvisare il controllore, allora trasmette automaticamente un **LACK** all'aereo. Se il controllore è occupato in altre transazioni invia un messaggio **STANDBY** per indicare all'aereo che la Clearance sarà ritardata, è necessario comunque che l'aereo confermi con un **LACK** il messaggio **STANDBY**. Quando il controllore è pronto a rispondere all'aereo, può trasmettere la Clearance oppure il messaggio **UNABLE**. L'invio del messaggio Clearance fa continuare il dialogo tra

pilota e controllore come una nuova transazione iniziata dal controllore (modalità Ground-Initiated), altrimenti l'invio di **UNABLE** fa terminare la transazione.

Per entrambe le modalità nel contesto di *EnRoute*, sono associati dei requisiti di performance [42, 47] ai parametri di una transazione CPDLC definiti precedentemente: TRN, $Responder$ e $RCTP$. Per ciascun parametro è stabilito il massimo valore atteso ET (Expiration Time) e il valore relativo al 95% del totale delle transazioni TT_{95} (Transaction Time).

	ET	TT_{95}
$TRN_{EnRoute}$	120	60
$Responder_{EnRoute}$	100	44
$RCTP$	20	16

Figura 3.6: Servizio ACL Ground-Initiated: Requisiti di performance

	ET	TT_{95}
$TRN_{EnRoute}$	270	60
$Responder_{EnRoute}$	250	44
$RCTP$	20	16

Figura 3.7: Servizio ACL Air-Initiated: Requisiti di performance

3.3.5 Servizio AMC

Il servizio AMC è iniziato da terra e consente ad un controllore di inviare delle istruzioni ad un velivolo per verificare che l'equipaggiamento di bor-

do per la comunicazione vocale, non blocchi una certa frequenza radio. Affinché questo servizio possa essere utilizzato è necessaria l'attivazione del canale dati CPDLC con il servizio ACM.

3.3.6 Timers

L'applicazione CPDLC utilizza un meccanismo per determinare l'inattività del canale di comunicazione, in particolare, usa dei timers [38] per rilevare se entro accettabili periodi di tempo sono ricevute le risposte ai messaggi. Il *Technical Response Timer* denotato **TR** rileva la non ricezione di LACK per la conferma automatica di un messaggio CPDLC inviato. Questo timer è attivato all'invio un messaggio CPDLC, mentre è disattivato se è ricevuto il LACK corrispondente prima dello scadere del timer; se il timer scade è prevista una notifica al mittente del messaggio e la comunicazione tra controllore e pilota passa al canale vocale. Il valore specificato per il timer TR è di 20 secondi (lo stesso valore Expiration Time ET del parametro $RCTP$) nei servizi ACM, ACL e AMC per entrambi i sistemi di terra e di bordo in tutti i contesti operativi.

Sono utilizzati altri due timer **TTR** *Termination Timer Receiver* e **TTS** *Termination Timer Sender*. Il primo (TTR), consente all'applicazione che riceve (Receiver) un messaggio CPDLC di stabilire se l'utente o il sistema rispondono al messaggio. Mentre l'altro (TTS), permette all'applicazione che invia un messaggio (Sender) di rilevare l'assenza di una risposta operativa (WILCO o UNABLE). Quando le risposte operative sono posticipate attraverso l'uso del messaggio STANDBY, i timer TTR e TTS se non ancora scaduti, vengono riattivati rispettivamente all'invio ed alla ricezione di un

messaggio di STANDBY. Il raggiungimento del tempo limite per i timer TTR e TTS in un sistema, comporta la notifica dell'evento all'altro sistema e la chiusura del canale CPDLC aperto, in particolare quando scade un timer TTS è previsto di completare la transazione corrente attraverso il canale vocale.

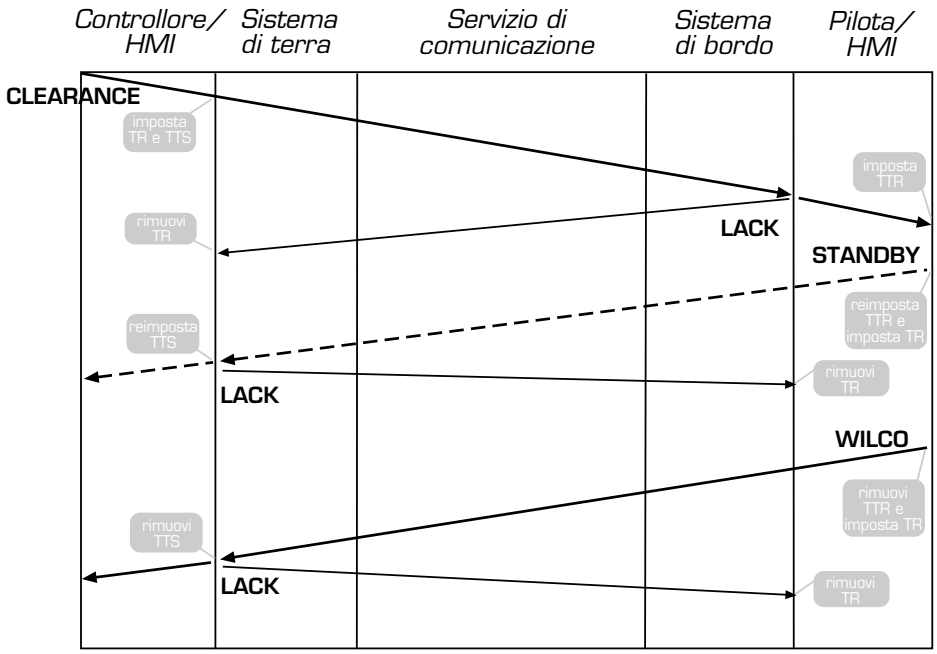

Figura 3.8: Timers TR, TTS, TTR in una transazione Ground-Initiated

I valori definiti per questi timers sono gli stessi valori di Expiration Time ET di TRN e $Responder$. Quindi nel contesto di *En Route* per una transazione relativa al servizio ACL iniziata da terra, il valore del timer TTS è 120 secondi per il sistema di terra e 100 secondi per il sistema di bordo, mentre per quella iniziata dall'aereo i valori dei timer TTS e TTR sono rispettivamente 270 e 250 secondi.

Capitolo 4

L'approccio simulativo/sperimentale e il framework Neko

La valutazione e l'analisi delle qualità del servizio offerte da un sistema rivestono un ruolo fondamentale soprattutto quando interessa lo sviluppo di quei sistemi caratterizzati da stringenti requisiti di disponibilità, affidabilità e sicurezza. Nel primo capitolo sono state presentate le tecniche utilizzabili per la valutazione quantitativa del servizio offerto da un sistema, con particolare riferimento alla valutazione di performance e dependability. In questo capitolo, si riprendono tali concetti per applicarli al contesto specifico di questa tesi: la valutazione quantitativa delle performance del servizio CPDLC.
Prima di iniziare qualsiasi attività di analisi e valutazione, è opportuno definire in maniera precisa l'obiettivo di studio e le quantità da valutare, successivamente occorre stabilire la tecnica e lo strumento più appropriati

per raggiungere lo scopo preposto. E' ritenuta buona pratica confrontare i risultati ottenuti con tecniche differenti per rendere la valutazione delle quantità più accurata e credibile.

4.1 Obiettivo di studio

Nella presente tesi ci proponiamo di riuscire a valutare le performance del servizio CPDLC, considerando solo il sottoservizio ACL (descritto nel capitolo 3) che consente a piloti e controllori l'invio e la ricezione di messaggi operativi (es. Clearance). In particolare, l'attenzione è rivolta ai parametri TRN e $RCTP$ definiti nel paragrafo 3.3.1. Il parametro TRN rappresenta la durata di una transazione CPDLC ovvero l'intervallo di tempo tra una richiesta e la risposta corrispondente; tale parametro fornisce un'indicazione importante delle prestazioni dell'applicazione CPDLC dal punto di vista dell'utente. Il parametro TRN comprende i tempi del livello applicativo CPDLC ed i livelli ATN sottostanti (protocollo di trasporto, rete e datalink) sia del sistema di terra che di bordo; ma include anche il tempo necessario all'utente per rispondere (Responder). Il tempo del risponditore può dipendere da molti fattori come l'esperienza, la capacità cognitiva e il carico di lavoro dell'utente, sottraendo questo contributo al tempo di transazione TRN si ottiene il parametro $RCTP$. Il parametro $RCTP$ è detto tempo tecnico di comunicazione e rappresenta un buon indicatore delle prestazioni del servizio di comunicazione. Entrambi i parametri TRN e $RCTP$ sono scelti come metriche di interesse per questo studio.

4.2 La tecnica: simulativa/sperimentale

La scelta della tecnica più opportuna è fortemente legata al tipo di studio da effettuare, ma anche alla fase di sviluppo ed alla conoscenza del sistema. Diversi studi e test sull'applicazione CPDLC e la sua infrastruttura sono già stati condotti a livello internazionale, ed hanno portato dapprima alla definizione di concetti, poi al loro raffinamento, ed infine allo sviluppo di prototipi per i primi test. Questo comporta un'ampia disponibilità di informazioni del sistema in esame. Tale considerazione unita all'analisi da effettuare, ci porta a scegliere una tecnica capace di considerare i molteplici dettagli del sistema. Come si è già accennato nel primo capitolo, una tecnica analitica non è adeguata per la valutazione delle quantità in questione e la complessità del sistema; inoltre, una tecnica puramente simulativa rende difficile il compito di rappresentare il comportamento dei protocolli di comunicazione ATN, che giocano un ruolo importante nella prestazioni del sistema. Così intendiamo scegliere una tecnica basata su un modello di applicazione CPDLC che permetta sia la simulazione che il testing, utilizzando per quest'ultimo delle componenti reali dell'infrastruttura ATN. Nel prossimo paragrafo è presentato lo strumento Neko e il pacchetto NekoStat utili per il nostro scopo, mentre nel prossimo capitolo verrà descritto dettagliatamente l'applicazione CPDLC realizzata.
La valutazione di performance è riferita solitamente ad un sistema in assenza di guasti, cioé quando esso fornisce un servizio proprio. Se il sistema è composto da parti replicate, come nel caso della infrastruttura ATN considerata per questo studio con router ridondati (capitolo 5); esso ha la possibilità di tollerare dei guasti e continuare comunque ad offrire il proprio servizio, magari in forma degradata. In tal caso è importante

la valutazione congiunta di performance e dependability (performability) per capire quanto le prestazioni del sistema sono affette dai guasti ai componenti.

4.3 Lo strumento: il framework Neko e il pacchetto Nekostat

Il framework Neko [19] è una piattaforma di comunicazione per l'analisi di algoritmi distribuiti che permette di utilizzare la stessa implementazione di un algoritmo in linguaggio Java sia per prove simulative che sperimentali su reti reali. Questo aspetto lo rende uno strumento capace di ridurre i tempi di sviluppo per un algoritmo, se paragonato all'impiego di implementazioni in linguaggi differenti per la simulazione e le misure (ad esempio SIMULA per la simulazione e C++ per l'implementazione reale). Inoltre, Neko è altamente portabile poiché scritto interamente in Java.

4.3.1 Architettura

L'architettura di Neko consiste di due parti principali: *l'applicazione* e *le reti*.
A livello applicativo, una collezione di processi numerati da 0 a $n-1$, comunicano utilizzando un'interfaccia per il passaggio di messaggi: il processo sorgente richiama la primitiva asincrona **Send** per inviare un messaggio sulla rete di comunicazione, che poi consegna il messaggio al processo di destinazione attraverso la primitiva **Deliver**.

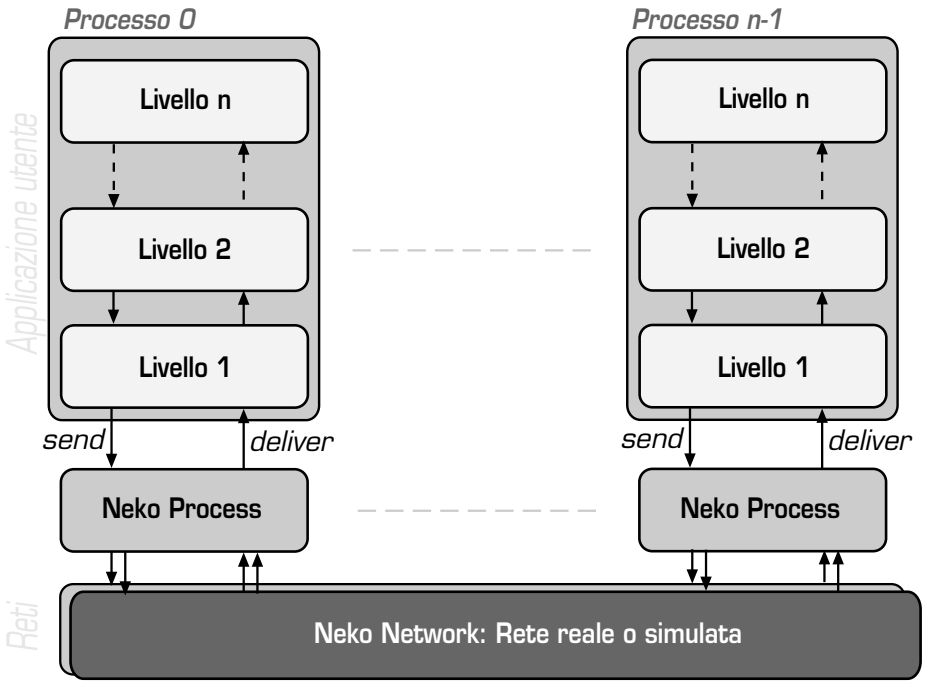

Figura 4.1: Architettura Neko

I livelli Applicativi

Le applicazioni Neko sono strutturate come gerarchie di livelli (layer). I messaggi sono passati da un livello ad un altro utilizzando la stessa interfaccia descritta prima, cioé mediante le primitive **Send** e **Deliver**. **Send** trasporta un messaggio da un livello più alto ad uno di livello più basso della gerarchia, viceversa **Deliver** trasporta un messaggio verso i livelli più alti.
I livelli possono essere *attivi* oppure *passivi*. Il livelli *passivi* non posseggono un proprio thread di controllo, contenuto invece, nei livelli *attivi*. Quest'ultimi infatti dispongono di una propria coda FIFO usata come buffer per i messaggi provenienti dal livello sottostante. Il thread di controllo dei

livelli attivi rimane in attesa della ricezione di un messaggio grazie alla primitiva **Receive** che ha semantica bloccante: se la coda di messaggi è vuota blocca il thread, altrimenti preleva un messaggio. Per questa primitiva è possibile specificare un timeout di attesa, se viene impostato un timeout nullo ne risulta una semantica non bloccante.

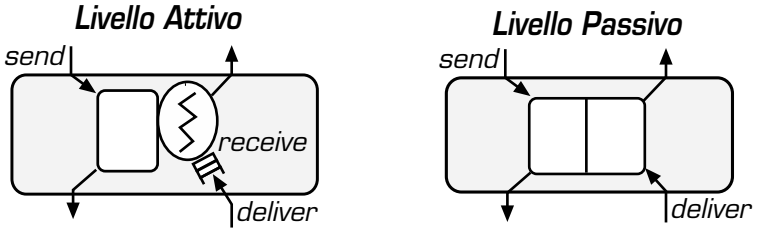

Figura 4.2: Livelli attivi e passivi

NekoProcess

Un componente importante di un'applicazione Neko è il **NekoProcess**, esso è posto tra i livelli applicativi e la rete. I principali ruoli del **NekoProcess** sono:

- Mantiene le informazioni comuni a tutti i livelli superiori, ad esempio l'identificatore associato al processo.
- Implementa alcuni servizi di utilità generale, come la registrazione (logging) dei messaggi inviati e ricevuti dal processo.
- Se l'applicazione supporta più di una rete in parallelo, è il NekoProcess ad inviare e ricevere i messaggi sulla rete appropriata.

NekoMessage

Le primitive di comunicazione **Send**, **Receive** e **Deliver** trasmettono messaggi detti **NekoMessage**s. Ciascun messaggio è composto da una *intestazione* e un *corpo*. L'intestazione è formata dalle seguenti informazioni:

- *Indirizzi*: Gli indirizzi sono relativi agli identificatori del processo mittente e del destinatario, quest'ultimo può essere di tipo *multicast* cioé riferito a più destinatari.
- *Rete*: Se Neko utilizza più reti in parallelo, ogni messaggio contiene un identificatore per la rete da utilizzare per la trasmissione.
- *Tipo di messaggio*: Ogni messaggio ha un identificatore (un valore intero definito dall'utente) per distinguerne la tipologia, ad esempio per messaggi relativi a differenti protocolli.

Il corpo può essere un qualsiasi oggetto Java definito dall'utente.

Reti

Il livello più basso dell'architettura Neko è rappresentato dalle reti (Networks), che possono essere *reali* oppure *simulate*. Un'applicazione può utilizzare una tipologia di rete anziché un'altra, senza dover alterare il codice sorgente.

Reti reali

Il framework Neko comprende già delle reti reali sviluppate sulla base di socket Java oppure su specifiche librerie per reti proprietarie. L'invio e la ricezione di un messaggio nelle reti reali avviene attraverso l'uso della

serializzazione del linguaggio Java, che consente di codificare oggetti Java in sequenze di bytes e viceversa (deserializzazione).

Le reti reali disponibili sono le seguenti:

- **TCPNetwork**: è sviluppata sullo stack TCP/IP e fornisce un servizio di consegna affidabile dei messaggi. Durante l'inizializzazione di un'applicazione distribuita è attivata una connessione TCP per ogni coppia di processi.
- **UDPNetwork**: è sviluppata sullo stack UDP/IP, ed a differenza della precedente non riesce a garantire un servizio di una consegna affidabile, ma può risultare adeguata per gli scambi di messaggi di heartbeats.
- MulticastNetwork: permette la spedizione di messaggi di multicast su UDP/IP (multicast inaffidabile).
- **PMNetwork**: utilizza la libreria PM [49] per la comunicazione a bassa latenza su architetture cluster, permette la comunicazione aggirando lo stack TCP/IP.
- **EnsembleNetwork**: integra il framework Ensemble [50] che permette di offrire un servizio di multicast affidabile su IP.

Reti simulate

Neko contiene alcune reti simulate predefinite come Ethernet, FDDI, CSMA-DCR[51]; esse sono dei modelli semplificati delle reti originali e non considerano alcuni fenomeni complessi come le collisioni nelle reti Ethernet.

Inoltre, esiste una rete che simula il ritardo di trasmissione con tempi casuali distribuiti esponenzialmente. Tale rete è molto utile in fase di debugging dell'algoritmo distribuito, perché riesce a mettere maggiormente alla prova l'algoritmo rispetto ad una rete dal comportamento più deterministico. L'introduzione di nuove reti simulate può essere realizzata facilmente, si definisce dapprima un modello del comportamento della rete e poi lo si implementa utilizzando le primitive **Send** e **Deliver**.

4.3.2 Configurazione ed esecuzione

La configurazione e l'inizializzazione di un'applicazione distribuita sono compiti tutt'altro che banali, a tale proposito Neko offre un supporto specifico. Tutte le parti di un applicazione Neko sono configurate attraverso un singolo file che fornisce i parametri necessari alla procedura di inizializzazione. L'inizializzazione di una applicazione Neko è diversa nei casi di simulazione ed esecuzione reale, questo comporta anche delle differenze nel file di configurazione da utilizzare. Per una simulazione è semplice: è necessaria solamente una istanza di Java Virtual Machine (JVM), sulla quale Neko crea ed avvia tutti i processi. Per l'esecuzioni distribuite invece, è opportuno utilizzare una JVM per ogni processo. In questo caso i processi sono distinti in *Master*, l'unico processo che ha il compito di coordinare l'inizializzazione e l'esecuzione distribuita dell'applicazione, e in *Slave* gli altri processi. Il Master legge il file di configurazione che contiene gli indirizzi IP di tutti gli Slaves, questi sono utilizzati per creare attraverso connessioni TCP una *rete di controllo temporanea* (non è attiva in fase di esecuzione), con la quale distribuire le informazioni di configurazione ai vari processi. Neko predispone delle *slave factories* che avviano automati-

camente le JVMs per i processi Slave su richiesta del Master, evitando così il noioso compito di avviare le JVMs manualmente.
Ciascun processo è inizializzato attraverso una classe Initializer specificata nel file di configurazione, che costruisce la gerarchia dei livelli applicativi sulla base del NekoProcess, aggiungendo i livelli applicativi dal basso verso l'alto. L'implementazione di questa classe è lasciata allo sviluppatore perché dipende dall'applicazione. Una volta terminata la procedura di inizializzazione, ha inizio l'esecuzione distribuita dell'applicazione.
Per la terminazione dell'applicazione, Neko fornisce una funzione *shutdown* che ogni processo può richiamare per provocare la chiusura di tutti i processi. Anche la procedura di terminazione dipende dal tipo di applicazione distribuita, per questo la funzione *shutdown* può essere implementata differentemente a seconda delle esigenze applicative.
Uno dei principali obiettivi di Neko è permettere la simulazione e l'esecuzione su rete reale di una stessa applicazione. Queste due modalità sono fondamentalmente diverse l'una dall'altra, ma seguendo due regole nell'implementazione Java dell'applicazione è possibile raggiungere questo obiettivo. La prima prevede di evitare l'uso di variabili globali (Static), perché esse possono essere condivise solo tra processi appartenenti ad una stessa JVM. Ciò sarebbe possibile nelle simulazioni, ma non nelle esecuzioni reali dove ai processi corrispondono JVMs differenti. La seconda regola si riferisce all'uso dei thread, poiché l'esecuzioni reali utilizzano la classe java.lang.Thread, mentre i pacchetti di simulazione integrati in Neko (uno di questi è SimJava [52]) usano specifici thread che evolvono in accordo al tempo simulato. Per nascondere questa differenza allo sviluppatore, Neko fornisce una classe NekoThread che raccoglie le funzionalità comuni alle due tipologie di thread. NekoThread è una versione semplifi-

cata del thread java che supporta anche i metodi *wait*, *notify*, *notify all* per la sincronizzazione dei threads. Esso si distingue dal thread java per due ragioni: la prima, inizia l'esecuzione solo quando l'intera applicazione Neko è avviata, e la seconda, sfrutta una variabile di tipo double per il tempo, permettendo di avere una granularità dei millisecondi.

4.3.3 Il pacchetto NekoStat

La valutazione quantitiva di un'applicazione distribuita in Neko è possibile attraverso l'intepretazione dei file di log oppure mediante l'uso del pacchetto NekoStat [20] presente nel framework Neko. Per entrambi gli approcci occorre inserire delle chiamate alla funzione *log* nei punti opportuni dell'implementazione dell'applicazione, le quali consentono la registrazione degli eventi di interesse in fase di esecuzione. Il primo approccio permette un'analisi solo al termine della simulazione o dell'esecuzione reale, ricavando le quantità di interesse attraverso l'intepretazione dei files di log. Questa procedura non è agevole, perché opera in modalità off-line ed è difficile ricostruire la storia di esecuzione dell'applicazione quando sono tanti i log prodotti (ogni processo scrive il proprio file di log), ma soprattutto perché richiede allo sviluppatore Neko l'implementazione di meccanismi appositi per il calcolo delle misure. L'altro approccio invece, offre un supporto sia per la raccolta degli eventi che per il loro trattamento. Nekostat è uno strumento integrato in Neko dalla versione 0.9, che permette di effettuare in maniera automatica delle valutazioni quantitative per le simulazioni e l'esecuzioni reali. Esso opera in modalità *on-line* permettendo analisi in parallelo alle simulazioni; mentre conserva quella *off-line* per l'esecuzioni reali. Il pacchetto Nekostat ricalca la struttura di

Neko, infatti predispone una interfaccia comune sia alle simulazioni sia all'esecuzioni reali che ne consente un utilizzo trasparente, cioé senza che lo sviluppatore si preoccupi dei dettagli implementativi di uno o dell'altro caso. E' previsto inoltre un specifico supporto per l'analisi statistica dei dati basato sulla libreria matematica Colt sviluppata dal Cern di Ginevra [26], che offre funzionalità più avanzate rispetto alla libreria standard Java (java.Math).

Architettura

L'architettura di Nekostat è formata principalmente dai seguenti componenti:

- **StatLogger**:
  Nel caso di simulazione: lo Statlogger è unico per tutti i processi, ha il compito di ricevere tutti gli eventi, attivare lo StatHandler per la gestione degli eventi e verificare attraverso un opportuno predicato di terminazione, se la simulazione può essere interrotta. Mentre nell'esecuzione reale, esiste uno Statlogger per ogni processo che raccoglie gli eventi da inviare al master al termine dell'esecuzione.

- **Event**:
  Event contiene le informazioni associate all'evento: l'identificatore del processo, il tempo (simulato oppure reale) in cui l'evento si verifica, la descrizione dell'evento e il contenuto. Quest'ultimo è un qualsiasi oggetto Java utilizzabile per passare informazioni da un componente ad un altro di Nekostat.

- **StatHandler**:
  Lo StatHandler ha il compito di gestire gli eventi per calcolare le misure di interesse. La definizione degli eventi da associare ad una o più quantità, dipende strettamente dal tipo di applicazione, perciò è richiesta l'implementazione di uno StatHandler specifico per una certa applicazione.

- **StatInitializer**:
  Lo StatInitializer inizializza i livelli ed altri oggetti necessari a NekoStat ed è richiamato dalla classe Inizializer dei processi.

- **StatLayer**:
  Lo StatLayer gestisce i messaggi di controllo di Nekostat al termine di una simulazione o una esecuzione. Nel primo caso: alla ricezione del messaggio di terminazione dell'analisi (NS_STOP), esso attiva la procedura di esportazione delle misure calcolate dallo StatHandler e conclude la simulazione. Mentre nel secondo caso, lo StatLayer agisce in maniera totalmente diversa: lo StatLayer del processo Master, alla ricezione del messaggio NS_STOP, invia un messaggio NS_SENDEVENTS agli StatLayer di tutti i processi Slave per richiedere l'invio di tutti gli eventi registrati. Gli StatLayer sui processi Slave non inviano gli eventi al Master tutti in blocco ma frammentati, evitando così il sovraccarico della rete e l'esaurimento della memoria del Master.

Supporto per l'analisi statistica

NekoStat mette a disposizione delle classi per il supporto all'analisi statistica, queste sono estensioni di classi della libreria matematica Colt.

- **QuantityOnlyStat**:
  E' un contenitore di valori con la possibilità di ricavare le misure statistiche di base: media, deviazione standard, minimo, massimo e dimensione.

- **QuantityDataOnFile**:
  E' come la classe precedente, ma permette di esportare i valori su file.

- **Quantity**:
  E' una classe ancora più sofisticata delle precedenti, che permette di analizzare oltre alle misure di base anche la covarianza rispetto ad un altro contenitore di valori, e la rimozione degli outlier (i valori di minimo e massimo).

Alle classi descritte è possibile associare dei predicati (condizioni di stop) che consentono la terminazione delle simulazioni dopo un numero predefinito di valori raccolti oppure al raggiungimento di un certo intervallo di confidenza.

Sincronizzazione Clock

In un'applicazione distribuita, la valutazione di una quantità può dipendere da eventi che si verificano localmente su uno stesso processo, ad esempio il tempo di richiesta e risposta (round trip); in tal caso è possibile effettuare le misure utilizzando il clock del processo, senza dover ricorrere ad alcuna forma di sincronizzazione tra i processi. Mentre è richiesta la presenza di un riferimento temporale comune a tutti i processi, quando la quantità è definita da eventi che coinvolgono più processi. La sincronizzazione dei processi è supportata da NekoStat attraverso uno scambio di

messaggi tra Master e Slave, che consentono di allineare ad una stessa origine temporale il clock dei processi al loro avvio. Questo può non essere sufficiente, ed è opportuno allora sincronizzare i clock Neko con un riferimento temporale esterno. A tale proposito è utilizzabile il protocollo NTP (Network Time Protocol) [53], largamente diffuso per la sincronizzazione attraverso la rete Internet.

Capitolo 5

Il modello del sistema CPDLC e l'implementazione in Neko

In questo capitolo è descritto in maniera dettagliata il modello del sistema CPDLC formato dall'applicazione CPDLC realizzata in Neko e dalla rete ATN di test. L'implementazione dell'applicazione CPDLC è strutturata a livelli come previsto dal framework Neko, ed è limitata alle funzionalità del servizio ACL per lo scambio di messaggi operativi tra pilota e controllore, nonché alle procedure *CPDLC Start* e *CPDLC End* del servizio ACM necessarie ad attivare e disattivare il trasferimento dei messaggi CPDLC tra le parti. E' implementato anche un apposito livello di rete Neko (*XtiNetwork*) per consentire ad un'applicazione Neko di utilizzare la rete ATN attraverso l'intefaccia XTI (X/Open Transport Interface).

5.1 Il livello applicativo

Il livello applicativo è caratterizzato da tre tipi di processi: **Air**, **Ground** e **Coordinator**. I primi due, entrambi processi *Slave* nella terminologia Neko, si riferiscono rispettivamente al lato applicativo del pilota e del controllore, mentre l'ultimo, il processo *Master* non corrisponde ad una figura realmente esistente, ma è qui impiegato per coordinare l'intera applicazione.

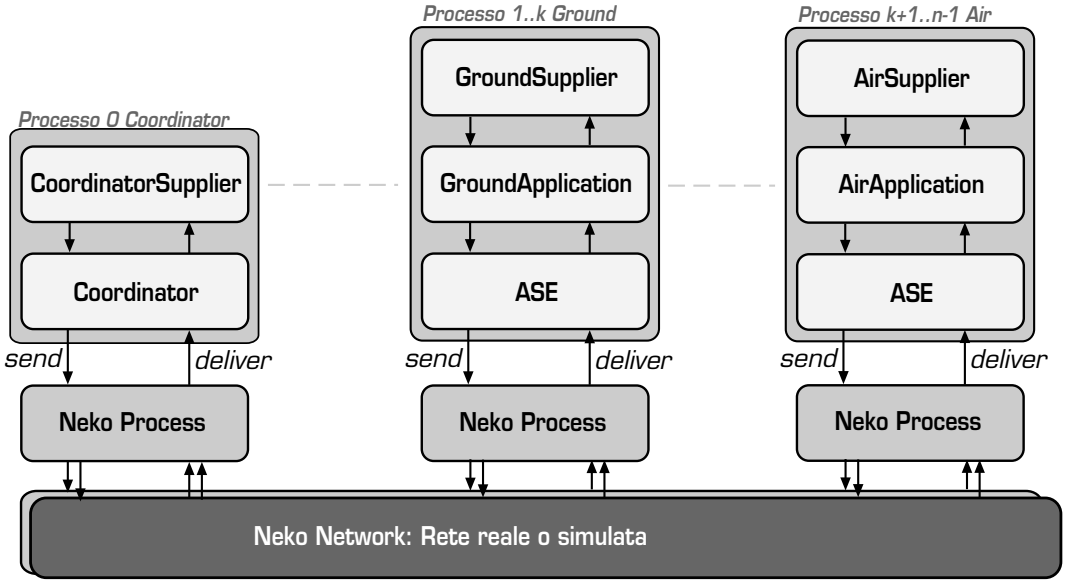

Figura 5.1: Architettura del livello applicativo

I processi Air e Ground si comportano in modo diverso, ma hanno una struttura comune composta da tre livelli (dall'alto verso il basso):

- **Supplier**: Supplier ha il compito di generare periodicamente richieste al livello sottostante UserApplication.

- **UserApplication**: UserApplication contiene la logica applicativa ACL CPDLC per iniziare transazioni CPDLC su richiesta del Supplier e rispondere ai messaggi CPDLC provenienti da altri UserApplication.
- **ASE**(Application Service Entity): il livello ASE implementa la procedura ACM per attivare e disattivare il trasferimento dei messaggi CPDLC, e dirige i messaggi provenienti dai livelli superiori verso le reti opportune.

5.1.1 Processo Air

AirSupplier

AirSupplier è un livello Neko attivo che genera richieste al livello applicativo del pilota. Questo livello rimane in attesa di ricevere il messaggio *GO* dal livello AirApplication sottostante, dopodiché attende ancora (un tempo casuale preso da una distribuzione esponenziale) ed invia un messaggio *START_ACL* al livello AirApplication per iniziare una transazione ACL CPDLC (fig. 5.2, 5.4). Se durante l'attesa è ricevuto un messaggio *STOP* dal livello sottostante, significa che il pilota è già impegnato in una transazione ACL iniziata da un controllore. Perciò è necessario bloccare AirSupplier fino alla ricezione di un nuovo messaggio GO che gli consente di continuare la propria esecuzione.

AirApplication

AirApplication implementa il lato aereo dell'applicazione CPDLC ed è anch'esso un livello di Neko attivo. Questo livello riceve i messaggi provenienti dal livello superiore (AirSupplier) oppure da quello inferiore (ASE).

AirApplication inizia la propria esecuzione alla ricezione di un messaggio *START_IND* inviato dal livello ASE per notificare la richiesta di attivazione del collegamento tra il velivolo e il controllore. Questo livello risponde automaticamente con *START_RES* ed invia il messaggio *GO* per attivare il suo Supplier. Da questo momento in poi sono possibili transazioni sia iniziate dal velivolo (fig. 5.2), richieste con il messaggio *START_ACL* da parte del Supplier, che iniziate da terra (fig. 5.4). Quando è ricevuta una richiesta iniziata da un controllore, si blocca il Supplier (*STOP*) che sarà poi riattivato al termine della transazione (*GO*).

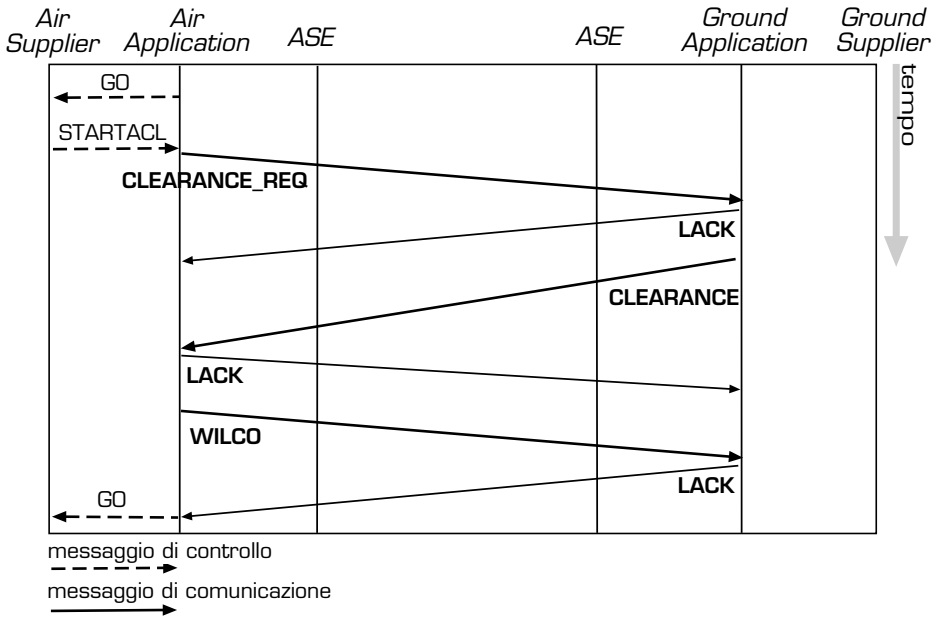

Figura 5.2: Transazione ACL iniziata dall'aereo

Per emulare il comportamento del pilota nella gestione di entrambe le tipologie di transazione (le transazioni sono gestite una alla volta), è

implementata una macchina a stati finiti FSM (fig. 5.3) che parte da uno stato iniziale $FIRSTSTATE$, evolve per effetto di certe sequenze di eventi come la ricezione ed l'invio di messaggi CPDLC, ritornando allo stato di partenza al termine di una transazione. Le risposte *WILCO*, *UNABLE* e *STANDBY* nella realtà sono a cura del pilota e non automatiche, mentre qui sono scelte ed inviate attraverso generazioni casuali.

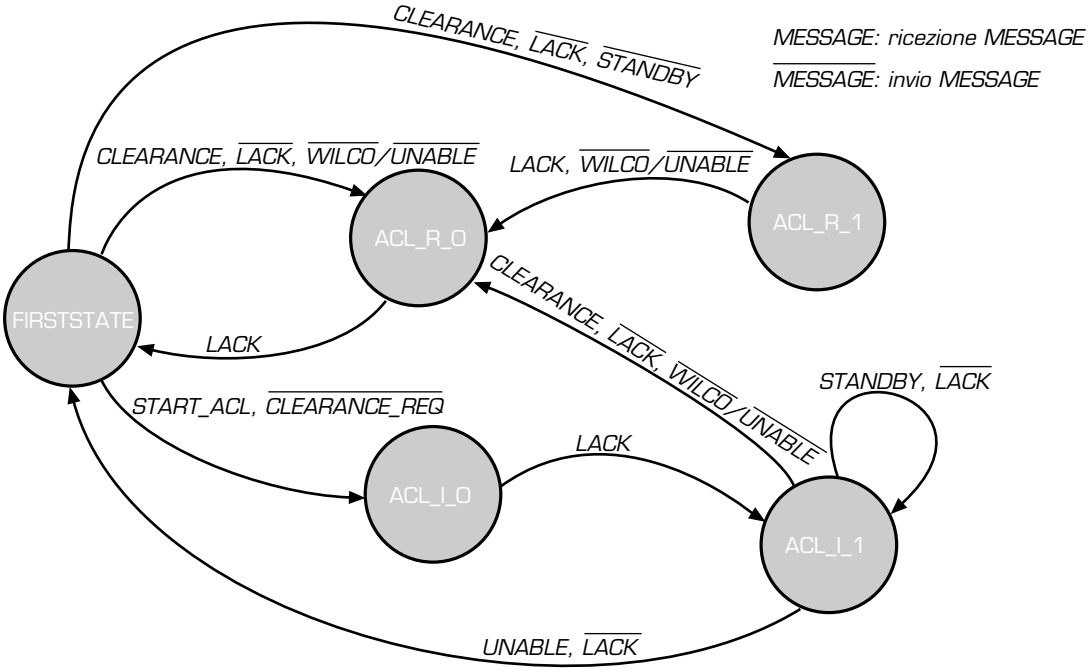

Figura 5.3: FSM implementata da AirApplication

AirApplication usa la classe *Timer* per la realizzazione dei timer **TR**, **TTR** e **TTS** previsti dall'applicazione CPDLC. Questo livello attiva gli oggetti Timer specificando la tipologia del timer e il timeout corrispondente espresso in secondi. Ad esempio, quando è iniziata una transazione

dal pilota (*CLEARANCE_REQ*) è attivato un timer **TTS** con un valore 270 sec, che viene poi disattivato alla ricezione della risposta operativa (*WILCO* oppure *UNABLE*), è comunque riattivato alla ricezione della risposta *STAND-BY*. Se un Timer raggiunge il suo timeout, viene comunicato ad AirApplication attraverso il metodo *timerExpired()* previsto dall'interfaccia *TimerInterface*. AirApplication è una classe Java che implementa l'interfaccia *TimerInterface* per garantire la comunicazione tra gli oggetti di questa classe e quelli Timer.

Durante situazioni di errore dovute allo scadere dei timeout oppure ad eventi non previsti dalla FSM, sono inviati dei messaggi destinati al processo Ground (*TIMEOUT* e *ERROR*), mentre se è ricevuto un messaggio di errore, non è inviato nessun altro messaggio. In entrambi i casi comunque, l'AirApplication rimuove tutti i timer ancora attivi e ritorna allo stato iniziale.

ASE (lato Air)

Il livello ASE è lo stesso per entrambi i processi Air e Ground, ma ha comportamenti differenti nell'uno e nell'altro caso. Il comportamento comune ai due processi è rivolto alla selezione della rete opportuna per i messaggi applicativi, più avanti sono descritte le due reti Neko utilizzate per la comunicazione dei messaggi CPDLC su rete ATN e per il controllo dell'applicazione distribuita. ASE agisce in maniera simile agli altri livelli (livello attivo), rimanendo in attesa di ricevere i messaggi. Nel caso del processo Air, ASE riceve un messaggio *MAPPING* (fig.5.6) inviato dal processo Coordinator contenente l'indirizzo Neko del processo Ground, permettendo l'associazione univoca tra il processo Air di un velivolo e il

processo Ground del controllore. Successivamente, è possibile attivare il trasferimento di messaggi CPDLC tra i due processi attraverso la procedura *CPDLC Start*.
La procedura Start implementata dal lato aereo è la seguente: Il livello ASE riceve una richiesta *START_REQ* proveniente dal processo Ground ed invia un messaggio di notifica *START_IND* al livello superiore. Quest'ultimo risponde automaticamente con *START_RES* e viene inviato un messaggio *START_CONF*. L'implementazione della procedura *CPDLC End* è analoga alla precedente.

5.1.2 Processo Ground

GroundSupplier

GroundSupplier è un livello attivo Neko che genera periodicamente richieste al livello applicativo del controllore. A differenza di AirSupplier non è mai bloccato esplicitamente ed invia (con tempo casuale preso da una distribuzione esponenziale) dei messaggi *IDLELIST_REQ* al GroundApplication, per richiedere la lista dei velivoli controllati non aventi transazioni attive (fig. 5.4). Quando riceve la lista richiesta, il GroundSupplier seleziona un velivolo in maniera casuale (distribuzione uniforme) se ci sono velivoli disponibili, ed invia un messaggio *START_ACL* (richiesta di transazione ACL) con l'indirizzo (Neko) corrispondente al velivolo scelto. Ovviamente se la lista è vuota non è inviata alcuna richiesta di transazione.

GroundApplication

GroundApplication è leggermente più complicato rispetto ad AirApplication, ha infatti la possibilità di gestire più transazioni CPDLC contem-

poraneamente, emulando così il reale comportamento di un controllore. Questo livello ha una struttura *AircraftList* basata sulla classe Java *Vector* che mantiene la lista degli aerei controllati. All'interno di questa struttura, gli aerei controllati sono rappresentati da oggetti *Aircraft* che hanno le seguenti informazioni: l'indirizzo Neko del processo Air corrispondente, lo stato, i timers (TR, TTS, TTR), il contatore di transazioni CPDLC e il tempo previsto per il controllo dell'aereo.
Dal livello superiore, GroundApplication riceve i messaggi *IDLELIST_REQ*, ai quali risponde inviando la lista degli aerei controllati che non hanno transazioni attive (*IDLELIST_RES*). Inoltre, riceve messaggi *START_ACL* che indicano il velivolo con cui iniziare una transazione ACL (fig. 5.4).

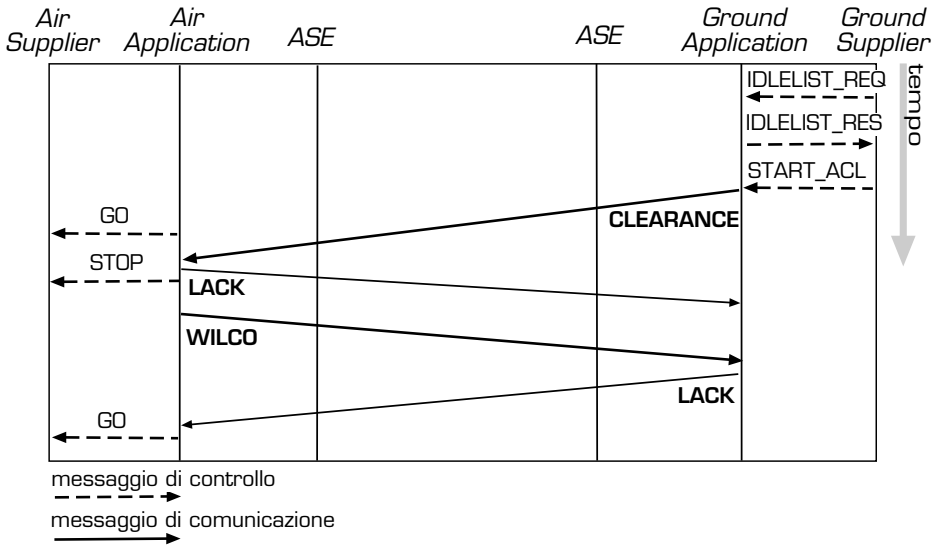

Figura 5.4: Transazione ACL iniziata da terra

Mentre dal livello inferiore (ASE), GroundApplication riceve l'indicazione dell'entrata di un nuovo aereo da controllare (*ENTRANCE*), alla

quale risponde con *START_REQ* per effettuare la procedura *CPDLC Start* (fig. 5.6). Dopo aver ricevuto la conferma di connessione con il velivolo (*START_CONF*), GroundApplication può associare a quel velivolo (processo Air) un oggetto *Aircraft*. Sempre dal livello sottostante, arrivano anche tutti i messaggi CPDLC relativi alle transazioni in corso con i processi Air.
Grazie all'impiego di una macchina a stati finiti FSM (fig. 5.5) e della struttura *AircraftList*, questo livello riesce a gestire più transazioni simultaneamente (al massimo una per ogni velivolo): Alla ricezione di messaggio CPDLC da un velivolo, GroundApplication seleziona l'oggetto Aircraft ad esso associato nella struttura *AircraftList*, preleva lo stato attuale del velivolo (lo stato è *FIRSTSTATE* se il velivolo non ha transazioni CPDLC attive) e decide quali azioni compiere per passare ad un nuovo stato.

GroundApplication gestisce in maniera analoga ad AirApplication i timer TR, TTS e TTR; l'unica differenza è che questi sono raccolti negli oggetti Aircraft. Allo scadere dei timers ed al verificarsi di eventi non previsti dalla FSM, sono inviati rispettivamente i messaggi *TIMEOUT* ed *ERROR*, inoltre è rimosso l'oggetto Aircraft associato al velivolo ed attivata la procedura *CPDLC End* (messaggio *END_REQ*). Quest'ultime operazioni hanno luogo anche allo scadere del tempo previsto per il controllo di un aereo.

Nella realtà ogni controllore può mantenere attive più transazioni con velivoli distinti, ma inviare loro una risposta (es. CLEARANCE) alla volta. GroundApplication si comporta allo stesso modo, utilizzando una semplice coda FIFO (nella realtà una coda con priorità è più opportuna [41])

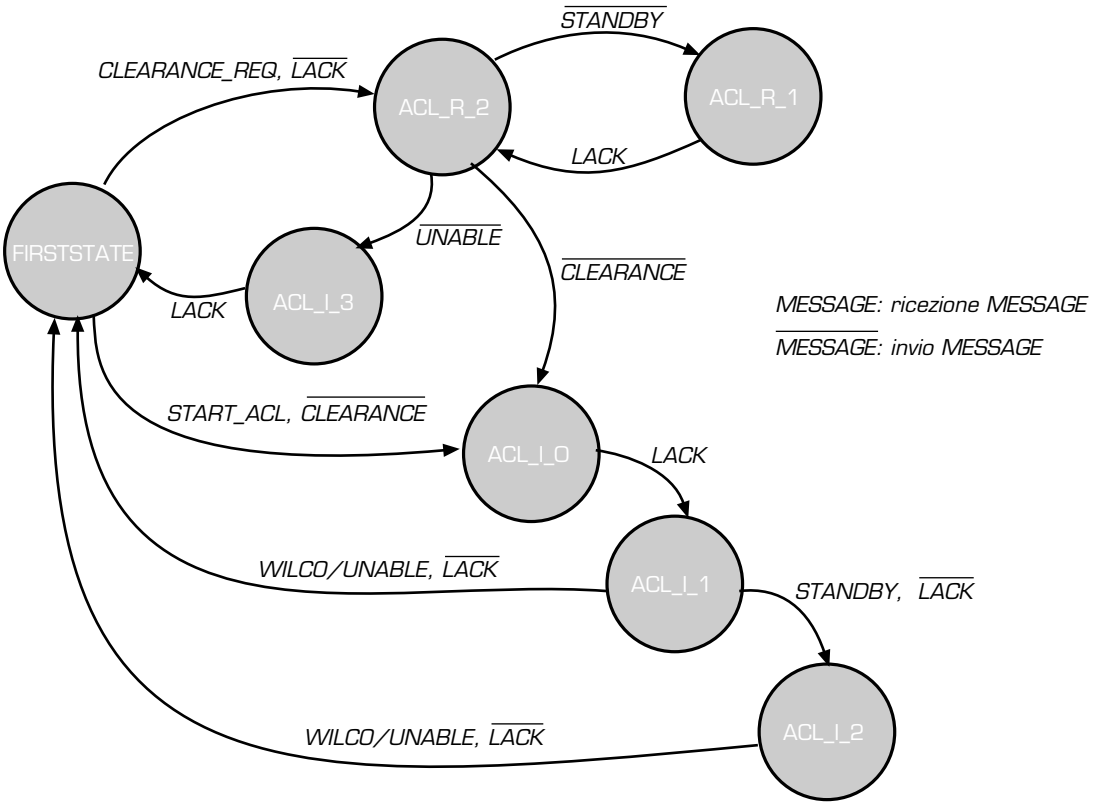

Figura 5.5: FSM implementata da GroundApplication

per gestire le richieste, ed un thread (*Responder*) per inviare le risposte. Alla ricezione di una richiesta da un velivolo (*CLEARANCE_REQ*): se il controllore è già impegnato a rispondere si inserisce la richiesta in coda, altrimenti quest'ultima può essere processata. In questo caso, Responder attende un tempo casuale (preso da una distribuzione Normale) e sceglie una risposta (in maniera uniforme) tra *CLEARANCE*, *STANDBY* ed *UNABLE*. Una volta inviata la risposta, il controllore può processare le richieste in coda.

ASE (lato Ground)

Alla ricezione di *MAPPING* da Coordinator con l'indirizzo Neko dell'aereo da controllare, ASE recapita un messaggio *ENTRANCE* al livello superiore. Questo procedimento si avvicina alla diffusione reale delle informazioni di Logon ottenute attraverso il servizio DLIC.
Durante la procedura CPDLC Start, ASE riceve il messaggio *START_RES* dal processo Air in risposta ad una richiesta *START_REQ* dal processo Ground, provvedendo ad inviare a quest'ultimo la conferma *START_CONF*.
Allo stesso modo funziona anche la procedura CPDLC END, l'unica eccezione è che al termine di questa procedura è indicata la terminazione del controllo del velivolo al coordinatore (messaggio *UNMAPPING*).

5.1.3 Processo Coordinator

CoordinatorSupplier

Il livello CoordinatorSupplier genera ripetutamente eventi relativi all'entrata di un velivolo da controllare che consentono di attivare i processi Air e Ground (fig. 5.6). CoordinatorSupplier attende un tempo casuale (preso da una distribuzione esponenziale), dopodiché seleziona l'indirizzo (Neko) del velivolo (processo Air) ed invia un messaggio *ENTRANCE* al livello Coordinator contenente il valore selezionato. CoordinatorSupplier preleva questo indirizzo da una struttura che mantiene gli indirizzi di tutti i velivoli non attivi.
CoordinatorSupplier ha inoltre il compito di generare il messaggio *END* dopo un tempo prefissato per la terminazione dell'intera applicazione (simulazione o esecuzione reale).

Coordinator

Coordinator è responsabile dell'attivazione dei processi Air e Ground (fig. 5.6). Quando riceve il messaggio *ENTRANCE* dal suo Supplier, il Coordinator associa al velivolo (il processo Air specificato nel messaggio) il controllore (processo Ground) meno occupato, ovvero quello che ha meno velivoli da controllare. Successivamente invia due messaggi (*MAPPING*): il primo al controllore contenente l'indirizzo del velivolo, viceversa il secondo. Tale associazione velivolo/controllore è mantenuta fino alla ricezione del messaggio *UNMAPPING* al termine della procedura *CPDLC End*.
Infine, Coordinator fa terminare l'esecuzione dell'applicazione distribuita inoltrando il messaggio *END*, proveniente dal livello superiore, a tutti i processi Air e Ground (i processi Slaves).

5.1.4 Generatori di numeri casuali e distribuzioni utilizzate

Nei precedenti paragrafi abbiamo accennato l'utilizzo di alcune distribuzioni di probabilità per la generazione di quantità casuali. Purtroppo la classe Java standard Random non offre un buon supporto a tale proposito, l'unica eccezione è la scelta di valori in un determinato insieme in maniera uniforme, perciò è utilizzata nei livelli AirApplication e GroundApplication. Per le altre quantità casuali è impiegata la libreria Colt [26] che fornisce oltre al supporto statistico visto per il pacchetto NekoStat, anche delle classi per le distribuzioni più importanti: Binomiale, Esponenziale, Gamma, Geometrica, Normale etc. L'uso di queste classi necessita di un generatore di numeri casuali, Colt mette a disposizione *MersenneTwister*

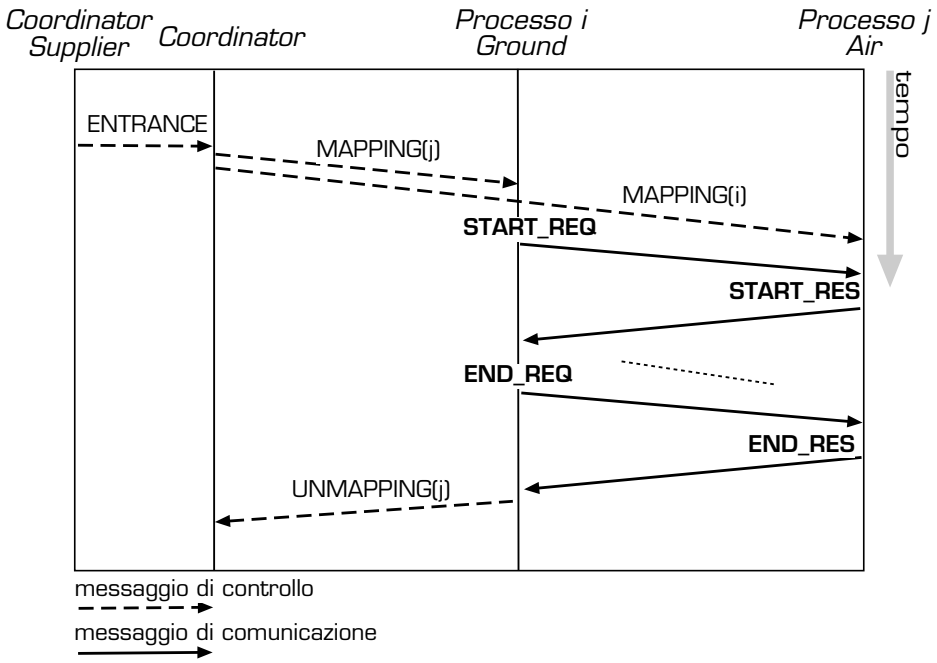

Figura 5.6: Attivazione processi Air e Ground

un generatore potente di numeri pseudo-casuali (uniformi) con periodo molto lungo 10^{6001}.

Le distribuzioni di probabilità qui impiegate sono l'esponenziale e la normale, la prima è specificata dal solo parametro λ che rappresenta l'inverso del valore medio μ della distribuzione, mentre la seconda è specificata dal valore medio μ e dalla deviazione standard σ. L'esponenziale è scelta per determinare il tempo tra la generazione di un evento e quello successivo, spesso riferito come *Interarrival Time*, questo è il caso dell'entrate dei velivoli nello spazio aereo nel CoordinatorSupplier e delle richieste di transazione ACL nei livelli AirSupplier e GroundSupplier. Mentre la distribuzione Normale è usata per il tempo di risposta del pilota e del con-

trollore nei livelli AirApplication e GroundApplication, tale distribuzione è abbastanza tipica per i tempi di risposta che dipendono da molteplici fattori (percettivi, cognitivi e di decisione). Comunque grazie alla struttura modulare dell'applicazione realizzata, è semplice sostituire queste distribuzioni con altre ottenute ad esempio da valori empirici.

5.1.5 Implementazione StatHandler per misure

Il pacchetto Nekostat fornito in Neko consente di catturare gli eventi che occorrono durante l'esecuzione, per poi trasformarli nelle quantità di interesse. Affinché questo sia possibile (si veda il paragrafo relativo a NekoStat nel capitolo 4), è necessario implementare una classe *StatHandler* specifica per l'applicazione (*UserApplStatHandler*), ed inserire le chiamate ad un *StatLogger* nei punti opportuni del codice applicativo per la registrazione degli eventi. Ricordiamo che le quantità di interesse sono i parametri TRN e $RCTP$, il primo è il tempo riferito ad una transazione CPDLC, mentre il secondo è dato dalla differenza tra TRN e il tempo del risponditore TTR. Con l'inserimento delle chiamate allo StatLogger per la registrazione degli eventi (Event Nekostat) relativi all'invio di una richiesta ($reqSnt$) ed alla ricezione della risposta ($resRcv$) è possibile ottenere la misura di TRN come differenza tra i tempi associati ai due eventi: $t_{resRcv} - t_{reqSnt}$. Quando una transazione include una risposta intermedia (es. *STANDBY*) sono associate due misure di TRN: è registrato un ulteriore evento $resParz$ e calcolato il primo TRN come $t_{resRcv} - t_{reqParzSnt}$ ed il secondo come $t_{resRcv} - t_{reqSnt}$. Il calcolo della misura $RCTP$ è leggermente più complicato, poiché coinvolge eventi su processi diversi. Prima, dal lato del risponditore, è necessario registrare gli eventi di ricezione

di una richiesta ($reqRcv$) e l'invio della risposta ($resSnt$), i quali permettono di calcolare il parametro TTR come differenza dei tempi associati agli eventi ($t_{resSnt} - t_{reqRcv}$). Poi, è possibile calcolare la misura $RCTP$ di una transazione sottraendo TTR dalla misura TRN (corrispondenti alla stessa transazione). Analogamente al calcolo del TRN per una transazione con risposta intermedia (es. *STANDBY*), anche il calcolo dell'$RCTP$ richiede di determinare due valori.
La raccolta e il trattamento degli eventi per il calcolo delle misure ha luogo durante la fase di chiusura dell'applicazione distribuita. Nel rispetto del funzionamento di Nekostat, questa operazione è svolta dal processo Coordinator (il processo Master) ed è attivata con il messaggio *NS_STOP* dopo lo scambio dei messaggi di terminazione (*END*) tra il Coordinator ed i processi Air e Ground.

5.2 Il livello di rete

Il livello di rete è concettualmente differente rispetto a quello previsto in Neko, è prevista infatti una rete di controllo parallela alla rete di comunicazione che non si limita solo alla procedura di inizializzazione e terminazione dei processi (Master e Slave), ma rimane attiva per tutta la durata dell'applicazione distribuita. In questo paragrafo sono presentate entrambe le reti di comunicazione e di controllo.

5.2.1 Rete di comunicazione

La rete di comunicazione è una rete ATN (Aeronautical Telecommunications Network) di test, configurabile grazie al contributo di Airtel (azienda

partner OTE fornitrice di router ATN). Airtel ha reso disponibile anche una versione limitata dell'interfaccia XTI (X/Open Transport Interface) [30] per poter utilizzare il servizio di trasporto orientato alla connessione (TP4). Questa versione di XTI è sviluppata per il sistema operativo Linux ed usa un meccanismo basato sui socket per la comunicazione tra processi (IPC), per questa ragione non è richiesto alcun supporto per gli STREAMS (alternativa ai sockets nelle versioni commerciali di UNIX) previsti invece nella versione standard di XTI [31].

Interfaccia XTI (X/Open Transport Interface)

XTI (X/Open Transport Interface) definisce un'interfaccia al servizio di trasporto indipendente dallo specifico protocollo impiegato. Tale interfaccia è formata da una libreria di funzioni in linguaggio C, attraverso le quali è possibile aprire e chiudere connessioni, trasmettere e ricevere dati.
La connessione tra due utenti di trasporto segue il paradigma Client/Server, il Client è la parte attiva che effettua la richiesta di connessione, mentre il Server rimane in attesa di ricevere tale richiesta. Prima della connessione, entrambi gli utenti abilitano un canale con il loro fornitore di trasporto e gli associano un indirizzo ATN (TSAP) attraverso le primitive *t_open()* e *t_bind()*. Il Client effettua una richiesta di connessione usando la primitiva *t_connect()* con argomenti il descrittore di file associato al canale (ottenuto da *t_open()*) e l'indirizzo del Server. Tale richiesta è notificata al Server attraverso la funzione *t_listen()*. Il Server può accettare la connessione con *t_accept()* su un canale alternativo a quello di ascolto, così può gestire la connessione corrente ed allo stesso tempo rimanere in attesa di altre richieste di connessione. Una volta effettuata la connessione è possibile il trasferimento di dati (bytes) con *t_snd()* e *t_rcv()*; quest'ultima ha semanti-

ca bloccante ovvero si blocca finché non ci sono dati disponibili. Il rilascio della connessione può avvenire in qualsiasi momento dal Server oppure dal Client con la primitiva *t_snddis()*. La notifica della terminazione della connessione avviene attraverso la primitiva *t_rcvdis()*. Terminata la connessione è opportuno chiudere il canale aperto con le primitive *t_close()* e *t_unbind()*.

XtiNetwork

L'applicazione CPDLC realizzata in Neko è scritta nel linguaggio Java e non può utilizzare direttamente l'interfaccia XTI per la rete ATN. E' necessario allora ricorrere alla tecnologia JNI (Java Native Interface) che consente ad una applicazione Java di invocare metodi scritti in linguaggio C. Questo ha permesso l'implementazione di un livello di rete Neko chiamato XtiNetwork per utilizzo dell'interfaccia XTI.
Per il nostro scopo è implementata un'apposita libreria C di supporto a XtiNetwork (una volta compilata si presenta come libXtiNetwork.so) che consente di richiamare le primitive XTI per stabilire connessioni di trasporto, trasferire dati e chiudere le connessioni. Tale libreria è utilizzabile dal livello di rete XtiNetwork attraverso i seguenti metodi nativi:

```
private native void XtiServer(String device);
private native int  XtiClient(byte[] address, String device);
private native void XtiSendConnect(int fd);
private native void XtiSendDisconnect(int fd);
private native void XtiReceiveDisconnect(int fd);
private native void XtiSend(int fd, byte[] content);
private native byte[] XtiReceive(int fd);
```

Il livello XtiNetwork è usato dai processi Air e Ground come rete di comunicazione per i messaggi CPDLC, non è usato invece dal processo Coordinator in quanto invia e riceve solo messaggi di controllo. XtiNetwork unisce in singolo livello le funzionalità di comunicazione previste per i processi Air e Ground, anche se queste sono tra loro differenti. Infatti, i processi Air sono entità passive che attendono di ricevere una richiesta di connessione, mentre i processi Ground sono la controparte attiva abilitata ad effettuare tale richiesta.
Ogni processo Air ha un proprio Server XTI attivato attraverso il metodo nativo *XtiServer()*, mentre i processi Ground hanno la necessità di comunicare con più processi Air contemporaneamente ed hanno quindi un Client XTI attivo per ognuno di questi processi.

Apertura connessione

Quando XtiNetwork del processo Ground riceve il messaggio CPDLC *START_REQ* (relativo alla procedura CPDLC Start), viene attivato un Client XTI attraverso il metodo nativo *XtiClient(byte[] address, String device)* con due parametri: *address* specifica l'indirizzo ATN (array di bytes) associato al destinatario e *device* identifica il fornitore di trasporto del Client. *XtiClient()* effettua la chiamata alle primitive XTI *t_open()* e *t_bind()* con i parametri passati, e restituisce il descrittore di file (fd) associato al canale aperto con il fornitore di trasporto. Successivamente, il Client invia una richiesta di connessione di trasporto *XtiSendConnect(int fd*) utilizzando l'fd ottenuto come parametro. Il Server XTI del destinatario, precedentemente attivato, può accettare la richiesta richiamando *t_accept()* ed inviare una callback all'XtiNetwork del processo Air corrispondente che contiene il descrittore di file associato alla connessione. Come già sottolineato, questo

descrittore di file è diverso da quello con cui il Server rimane in ascolto con la primitiva *t_listen()*.

Invio e ricezione messaggi

Dopo aver stabilito la connessione, XtiNetwork permette l'invio e la ricezione di messaggi attraverso due thread distinti *SenderThread* e *DeliverThread* (fig. 5.7). *SenderThread* attende di ricevere messaggi CPDLC dai livelli superiori, dopodiché utilizzando come parametri l'fd di connessione e l'array di bytes ottenuto dalla serializzazione del messaggio, invoca il metodo nativo *XtiSend(int fd, byte[] content)* per inviare l'array di bytes con la primitiva *t_snd()*. Il processo Air ha un singolo fd relativo all'unico destinario possibile (il processo Ground controllore), mentre il processo Ground ne ha molteplici e seleziona ogni volta quello opportuno. La serializzazione di un messaggio CPDLC (un oggetto Neko Message) restituisce il seguente array di bytes: i primi 4 byte sono relativi al tipo di messaggio (valore intero), i successivi 4 byte al numero di transazione CPDLC (valore intero) associato al messaggio, e i restanti sono bytes generati casualmente (il numero dei bytes è preso da distribuzione Normale) che rappresentano il contenuto informativo di un messaggio CPDLC.

DeliverThread invoca il metodo nativo *XtiReceive(int fd)* per richiamare la primitiva *t_rcv()*, la quale si blocca fino alla ricezione di dati sulla connessione specificata dal parametro fd. Poiché un processo Ground può comunicare con più processi Air in parallelo, XtiNetwork deve disporre di un *DeliverThread* per ciascuna connessione aperta con i processi Air. Una volta ricevuti i dati (array di bytes), questi vengono deserializzati ottenendo un messaggio CPDLC da recapitare ai livelli superiori.

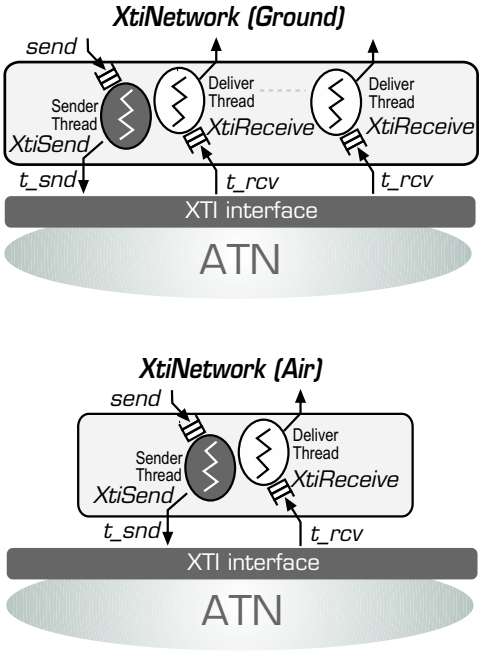

Figura 5.7: Struttura di XtiNetwork

Nella tabella seguente sono riassunti i messaggi Neko CPDLC trasmessi attraverso la rete comunicazione:

Chiusura connessione

La procedura di *CPDLC End* implementata al livello applicativo ASE consente di avviare la fase di disconnessione tra due processi. Alla ricezione del messaggio *END_RES* dal processo Air (in risposta ad *END_REQ* inviato dal processo Ground), l'XtiNetwork del processo Ground invoca il metodo *XtiSendDisconnect(int fd)* per richiamare la primitiva XTI di disconnessione *t_snddis()* con il descrittore di file corrispondente al processo Air, e termina il *DeliverThread* corrispondente. Dall'altra parte, l'XtiNet-

CLEARANCE_REQ	*CLEARANCE*
LACK	*WILCO*
UNABLE	*STANDBY*
ERROR	*TIMEOUT*
START_REQ	*START_RES*
END_REQ	*END_RES*

Figura 5.8: Messaggi CPDLC su rete di comunicazione

work del processo Air riceve l'indicazione di disconnessione attraverso la primitiva *t_rcvdis()* ed anch'esso termina il proprio *DeliverThread*.

5.2.2 Rete di controllo

Come già accennato questa applicazione CPDLC impiega un'altra rete, basata sulla Neko TCPNetwork, per lo scambio di messaggi di controllo, l'inizializzazione e la terminazione dell'intera applicazione distribuita. Tale rete è utilizzata, a differenza di XtiNetwork, da tutti i processi Air, Ground e Coordinator per tutta la durata dell'esecuzione dell'applicazione. L'introduzione di una rete parallela a quella di comunicazione, è stata motivata dall'esigenza di coordinare l'applicazione cercando di minimizzare l'impatto sulla sua esecuzione. In questo modo, il flusso di controllo non rappresenta alcun ostacolo al flusso di comunicazione.
Di seguito sono riportati i messaggi trasmessi su rete di controllo.

ENTRANCE	*END*
MAPPING	*UNMAPPING*

Figura 5.9: Messaggi su rete di controllo

5.3 Configurazione ed esecuzione

L'applicazione CPDLC presentata è realizzata seguendo la filosofia di Neko ovvero permettendo sia la simulazione che l'esecuzione su rete reale, senza la necessità di modificare il codice applicativo. Poiché l'interesse è rivolto soprattutto alla valutazione del servizio CPDLC su un'infrastruttura di rete ATN, la simulazione ha qui un ruolo marginale e non è usata per ricavare le quantità di interesse. La simulazione è comunque importante perché serve ad esercitare l'applicazione (in fase di debugging) per verificarne la correttezza.

5.3.1 La rete ATN di test

L'azienda Airtel (partner OTE) ha supportato la sviluppo di questa tesi fornendo i principali componenti dell'architettura ATN: *End Systems* ed *Intermediate Systems* (si veda cap. 2). In particolare, Airtel ha rilasciato un software *ATN Access Router* per il sistema operativo Linux, che consente di configurare End Systems ed Intermediate Systems attraverso opportuni scripts (sempre forniti da Airtel). Per le esecuzioni reali dell'applicazione CPDLC è utilizzata una configurazione di rete ATN (fig. 5.10) formata dai seguenti cinque PC industriali (Pentium III 500 Mhz.) con sistema operativo Linux (kernel 2.4.29):

- *PC1 (Ground End System)*: *End System* di terra utilizzato dai controllori (Processi Ground) dotato dei protocolli **ES-IS** (per lo scambio di messaggi di Hello tra End Systems ed Intermediate Systems), **CLNP** (il protocollo di rete) e **TP4** (il protocollo di trasporto orientato alla connessione).

- *PC2 (Ground/Ground Router 1)*: *Intermediate System* di terra dotato dei protocolli **ES-IS**, **IS-IS** (per lo scambio di messaggi di Hello tra gli Intermediate Systems) e **CLNP**.

- *PC3 (Ground/Ground Router 2)*: *Intermediate System* identico al precedente, è impiegato per garantire il funzionamento della rete ATN in caso di guasto del primo *Intermediate System*.

- *PC4 (Air/Ground Router)*: *Boundary Intermediate System* (BIS) di terra per la comunicazione con i velivoli (Air End Systems), oltre ai protocolli **ES-IS**, **IS-IS** e **CLNP**, impiega il protocollo **IDRP** (per il routing tra domini differenti).

- *PC5 (Air End Systems)*: Molteplici End System di bordo utilizzati dai piloti (Processi Air), dotati del protocollo di routing **IDRP**, di rete **CLNP** ed infine di trasporto **TP4**.

I primi quattro sistemi formano un dominio di terra adatto a rappresentare un'infrastruttura di rete reale di un centro di controllo del traffico aereo, mentre l'ultimo sistema forma un dominio per ciascun velivolo (Air End Systems) presente. Ciò giustifica l'impiego del protocollo *IDRP* per lo scambio delle informazioni di routing tra domini differenti. Per ragioni ovvie i sistemi di bordo (Air End Systems) sono fissi, perciò il protocollo

IDRP utilizza solo percorsi (routes) statici e non offre alcun supporto alla mobilità.
L'End System di terra può comunicare con un solo router (Ground/Ground) alla volta, scelto attraverso i protocolli *IS-IS* e *ES-IS*. Oltre a disporre di due router (Ground/Ground) per garantire il funzionamento di ATN anche quando uno di loro è guasto, è presente un'altra forma di ridondanza a livello di sottorete. Infatti, i sistemi appartenenti al dominio di terra (PC1, PC2, PC3 e PC4) sono collegati tra loro attraverso due sottoreti *Ethernet* distinte, mentre un'altra sottorete *Ethernet* collega il router (PC4) ai sistemi di bordo (PC5). Nel primo caso, l'impiego delle sottoreti *Ethernet* è giustificabile perché si vuole rappresentare una rete locale (Lan) di un centro di controllo, nel secondo caso invece, la sottorete *Ethernet* è inadeguata perché si vuole rappresentare la sottorete wireless VDL2 che collega la stazione di terra ai velivoli. Si ricorre perciò ad uno strumento software chiamato **NetEm** per emulare il comportamento della sottorete VDL2 [33].

Emulazione della sottorete VDL2 con NetEm

NetEm [32] è uno strumento integrato nelle ultime versioni del kernel Linux (a partire dalla 2.4.28) utile per riprodurre il comportamento di reti di varia tipologia. NetEm agisce sulla coda di uscita (FIFO) dei pacchetti *Ethernet* mantenuta nel kernel, basandosi su parametri impostati dall'utente come il ritardo, la perdita, la duplicazione e il riordino dei pacchetti. Senza entrarne nei dettagli, l'utente di NetEm può associare al ritardo una certa distribuzione attraverso la media (μ), la deviazione standard (σ) e la correlazione (ρ). Di default, NetEm usa una distribuzione normale specificata dalla media e dalla deviazione standard. Nel nostro caso, la sottorete VDL2 è emulata semplicemente associando una distribuzione nor-

male al ritardo con i valori ottenuti dalle simulazioni ACTS (Aeronautical Communications Technologies Simulator) di Eurocontrol per il programma Link2000+ [34, 35]. In particolare, come valore medio si usa il ritardo di trasmissione (Avg. TX_RC Delay) di un pacchetto, il quale rappresenta il tempo medio osservato tra l'inserimento nella coda del trasmettitore e la ricezione completa e corretta del pacchetto da parte del destinatario. Questo valore tiene conto anche dei tempi di ritrasmissione necessari a compensare la perdita di pacchetti a livello fisico, dovuta principalmente al fenomeno delle collisioni.

5.3.2 Configurazione dell'applicazione CPDLC

L'applicazione CPDLC è configurata attraverso un singolo file che permette di impostare oltre ai parametri tipici per l'utilizzo di Neko e NekoStat, anche altri parametri specifici per questa applicazione. Di seguito sono riportati i principali parametri:

- *simulationTime*: il tempo di simulazione/esecuzione.
- *network*: la lista delle reti Neko utilizzate, nel nostro caso *XtiNetwork* e *TCPNetwork* rispettivamente come reti di comunicazione e di controllo.
- *process.num*: il numero totale dei processi Neko.
- *process.*i*.inizializer*: la classe *inizializer* che consente di creare l'opportuno stack dei livelli per il processo i.
- *slave.*i: l'indirizzo Ip e la porta Tcp del processo slave i ($i > 0$).

- *end_system.*i: l'indirizzo ATN (TSAP) e il fornitore di trasporto (l'indirizzo Ip e la porta Tcp previsti dall'interfaccia XTI) del processo slave i ($i > 0$).
- *n_controllers*: il numero dei processi Ground (i controllori di volo).
- *n_aircrafts*: il numero dei processi Air (i velivoli).
- *groundACLinterarrival*: media della distribuzione per la generazione delle transazioni ACL (Ground).
- *airACLinterarrival*: media della distribuzione per la generazione delle transazioni ACL (Air).
- *aircraftInterarrival*: media della distribuzione per il tempo tra un'entrata di un velivolo e l'altra.
- *groundResponseParams*: media e deviazione della distribuzione per il tempo di risposta (Ground).
- *airResponseParams*: media e deviazione della distribuzione per il tempo di risposta (Air).
- *groundMissionParams*: media e deviazione della distribuzione per la durata del controllo dei velivoli.
- *messageLength*: media e deviazione della distribuzione per la dimensione dei messaggi CPDLC.

I processi Neko sono numerati da 0 a $n - 1$, dove 0 è il *Master* ed i restanti sono gli *Slaves*. Tali processi sono inizializzati attraverso il parametro *process.*i*.inizializer* nel seguente modo: il Master 0 come processo Coordinator, gli Slaves da 1 a k come processi Ground (supposto k il valore del

parametro *n_controllers*) e gli $n - k - 1$ Slaves come processi Air. Nelle esecuzioni reali sulla rete ATN di test presentata, affinché lo scambio dei messaggi di comunicazione tra i processi Ground ed Air sia possibile, è necessario avviarli sugli End Systems opportuni. Perciò, i processi Ground sono avviati sulla macchina dell'End System di terra (PC1), mentre tutti i processi Air su quella degli End Systems dei velivoli (PC5). L'esecuzione su una stessa macchina (PC5) di più istanze di End System ed altrettanti processi Air, comporta un notevole consumo di memoria; per questa ragione si è deciso di avviare il processo Coordinator sul PC1.
Come già più volte sottolineato, i processi Coordinator, Ground ed Air scambiano tra loro messaggi di controllo attraverso la rete Neko TCPNetwork. Quest'ultima sfrutta un collegamento Ethernet tra il PC1 e il PC5 indipendente alla rete di comunicazione ATN. Sempre su questo collegamento, PC1 e PC5 condividono l'accesso alla rete Internet per la sincronizzazione con un Server NTP primario (IEN Galileo Ferraris ntp1.ien.it [27]).

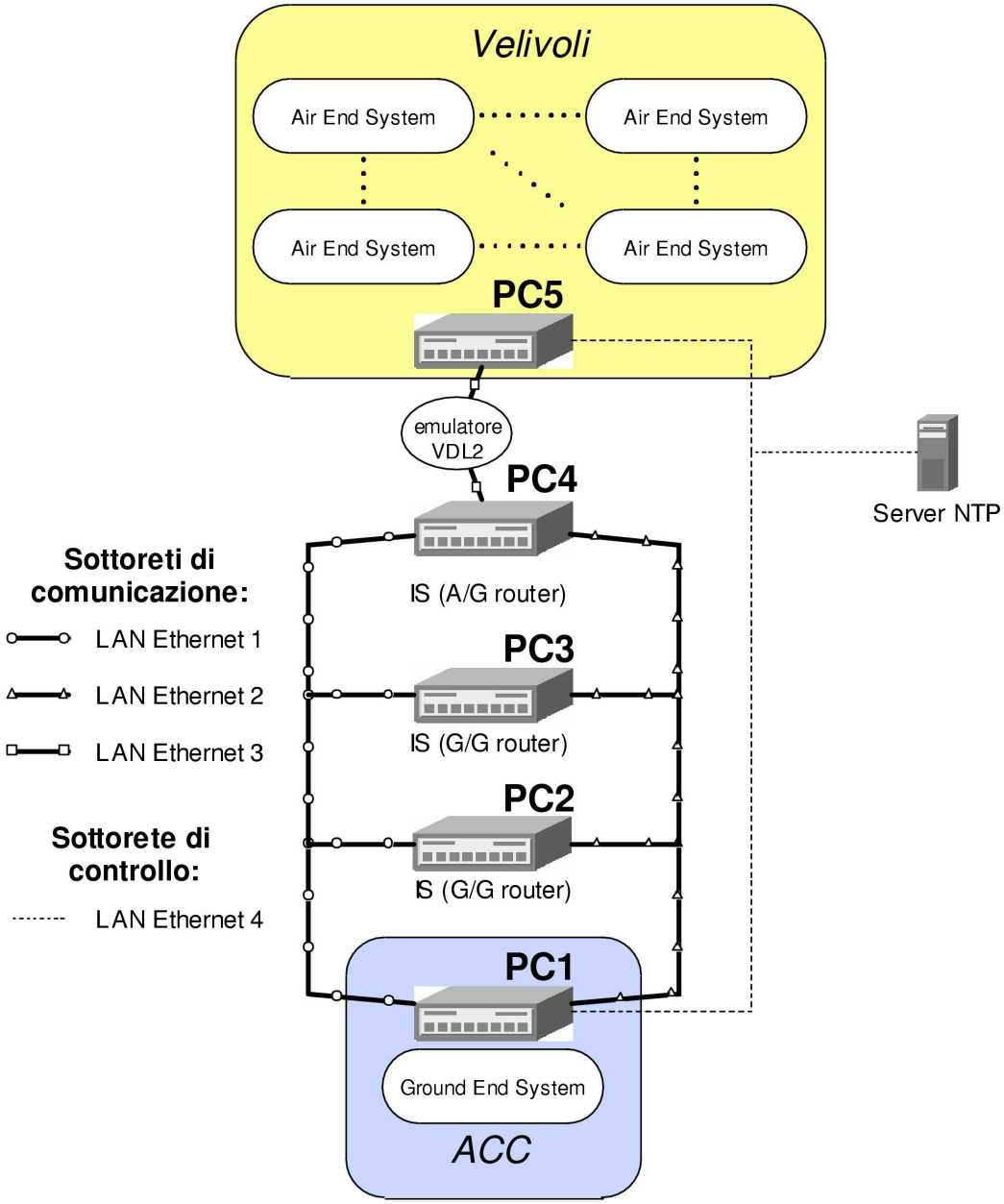

Figura 5.10: Configurazione rete ATN di test

Capitolo 6

Esperimenti ed analisi dei risultati

In questo capitolo vengono descritti gli esperimenti per la valutazione delle quantità di interesse relative al servizio offerto dall'applicazione CPDLC sulla rete ATN. In generale, la scelta dei parametri di ingresso e delle assunzioni strutturali ha effetto sui risultati del sistema. Poiché è impossibile effettuare esperimenti per tutte le possibili combinazioni di valori e condizioni di ingresso, è opportuno nella definizione degli esperimenti, stabilire quali sono i fattori che hanno maggiore rilevanza rispetto all'obiettivo di studio ed alle misure di interesse.

6.1 Definizione esperimenti

Gli esperimenti sono definiti in base a due fattori principali: il numero dei Ground/Ground routers attivi e il traffico CPDLC.

6.1.1 Il primo fattore: numero dei Ground/Ground routers attivi

Il primo fattore interessa aspetti strutturali della rete ATN, prendendo come riferimento il tempo di esecuzione del sistema vengono identificate quattro possibili configurazioni:

Configurazione A: Un solo Ground/Ground router attivo per tutto il tempo di esecuzione.

Configurazione B: Due Ground/Ground routers attivi per tutto il tempo di esecuzione.

Configurazione C: Due Ground/Ground routers inizialmente attivi ed uno è disattivato a metà esecuzione.

Configurazione D: Due Ground/Ground routers inizialmente attivi, uno è disattivato ad un terzo dell'esecuzione e ripristinato raggiunti i due terzi dell'esecuzione.

Le prime due configurazioni di rete sono *statiche* ovvero l'intero sistema funziona con lo stesso numero di routers attivi per tutta la sua esecuzione, al contrario le altre due configurazioni sono *dinamiche*. Nelle ultime due configurazioni di rete, l'evento di disattivazione di un router può essere inteso come l'inserimento di un guasto nel sistema (Fault Injection), allo scopo di valutare quanto la presenza di guasti sulla rete si riflette sulle performance del servizio CPDLC.

6.1.2 Il secondo fattore: traffico CPDLC

L'altro fattore, il traffico CPDLC, è caratterizzato dall'ammontare delle transazioni ACL (iniziate da terra e dai velivoli) e dipende principalmente dai seguenti parametri: il numero dei velivoli controllati, la durata del controllo dei velivoli e la frequenza delle transazioni ACL. Per rappresentare diverse situazioni di traffico CPDLC relative ad un centro di controllo (ACC) durante il suo funzionamento, si possono fissare i primi due parametri e variare solo il numero dei velivoli controllati. Occorre notare che nel contesto di En-Route, cioé durante la fase più lunga del controllo di un velivolo, un ACC può controllare un numero rilevante di velivoli in uno stesso momento, ma avere comunque delle transazioni CPDLC con frequenza piuttosto bassa. Purtroppo nell'applicazione CPDLC realizzata, i sistemi di bordo (Air End Systems) dei velivoli sono tutti eseguiti su una stessa macchina (PC5) ed è necessario limitare il numero massimo dei processi Air contemporaneamente attivi (si usa 20 come valore per il parametro *n_aircrafts* dell'applicazione), evitando così di consumare tutta la memoria disponibile (512 MB). Tuttavia è possibile aumentare la frequenza delle transazioni CPDLC per ottenere il traffico CPDLC desiderato. Si possono considerare due profili di traffico reale CPDLC che corrispondono rispettivamente al controllo di 75 e 125 velivoli durante un'ora di servizio di un ACC. Dove per ciascun velivolo, si prevede una durata di controllo di 35 minuti (uguale al tempo medio registrato dall'ACC di Roma[48]) ed all'incirca 12 transazioni ACL sia iniziate da terra che dai velivoli (Eurocontrol ne prevede 32 per un volo di 90 minuti[35]). Nel primo profilo sono generate all'incirca 1800 transazioni ACL totali sia iniziate da terra che dai velivoli (75 velivoli $\times 12$ transazioni $\times 2$ modalità $= 1800$), mentre nel secondo profilo se ne hanno circa 3000. Nell'applicazione CPDLC realizzata, tutte

queste transazioni ACL sono ottenibili con tempi di controllo più brevi (in media 800 secondi per 75 velivoli e 600 per 125 velivoli) ma aumentando, come abbiamo detto, la frequenza delle transazioni ACL. In particolare, sia dal lato terra che aria, sono ridotti i tempi di generazione tra una richiesta ACL e l'altra. L'incremento delle transazioni ACL fa aumentare anche gli eventi registrati, causando dei problemi a NekoStat durante il calcolo delle misure di interesse, probabilmente dovuti al fatto che gli eventi sono mantenuti in memoria e non scritti su disco. Per questa ragione si è deciso allora di ridurre il tempo di esecuzione dell'applicazione a 30 minuti nel secondo profilo.

Le tabelle 6.1 e 6.2 mostrano i parametri applicativi utilizzati rispettivamente nel primo e nel secondo profilo di traffico. I parametri applicativi comuni ad entrambi i profili riguardano i processi totali (25), i processi Ground relativi ai Controllori (4), i processi Air ovvero il massimo numero di velivoli contemporaneamente attivi (20), i tempi di risposta dei controllori e dei piloti (media 4 sec, deviazione 1 sec) scelti in modo da non superare i 5 sec. come dai risultati delle simulazioni Link2000+ con l'impiego di interfacce HMI avanzate[42], ed infine la lunghezza in bytes dei messaggi (media 20, deviazione 5) scelta in modo da avvicinarsi alla dimensione reale dei messaggi CPDLC. Mentre variano i parametri i tempi di esecuzione rispettivamente 3600 sec e 1800 sec, i tempi di entrata dei velivoli (in media ogni 48 sec nel primo caso e 28 sec nel secondo), la durata del controllo ed i tempi di generazione tra una richiesta ACL e l'altra.

Inoltre, per rappresentare in modo appropriato i due profili di traffico nell'esecuzione dell'applicazione CPDLC, è opportuno considerare differ-

enti valori di ritardo per l'emulatore NetEm della sottorete VDL2, perché come evidenziato dalle simulazioni ACTS di Eurocontrol, il ritardo VDL2 aumenta al crescere del volume dei dati da trasmettere. Per impostare i valori di ritardo opportuni per i due profili di traffico è usato il tempo medio di trasmissione TX_RC Delay ottenuto dalle simulazioni ACTS. Nel primo profilo, NetEm inserisce un ritardo medio di 399 ms (con deviazione 12 ms) sia ai pacchetti in uscita da PC4 (Uplink) sia quelli in uscita da PC5 (Downlink). Mentre nel secondo profilo, NetEm inserisce un ritardo di 399 ms (con deviazione 12 ms) ai pacchetti di Uplink e di 500 ms (con deviazione 15 ms) a quelli di Downlink.

Profilo 1 con 75 velivoli controllati **Parametri:**
simulationTime = 3600 sec
process.num = 25
n_controllers = 4
n_aircrafts = 20
groundACLinterarrival = 12 (media)
airACLinterarrival = 30 (media)
aircraftInterarrival = 48 sec (media)
groundResponseParams = 4 sec, 1 sec (media, deviazione)
airResponseParams = 4 sec, 1 sec (media, deviazione)
groundMissionParams = 800 sec , 60 sec (media, deviazione)
messageLength = 20,5 (media, deviazione)
ritardo VDL2 Uplink = 399 ms , 12 ms (media, deviazione)
ritardo VDL2 Downlink = 399 ms , 12 ms (media, deviazione)

Figura 6.1: Profilo 1: i parametri

Profilo 2 con 125 velivoli controllati **Parametri:**
simulationTime = 1800 sec
process.num = 25
n_controllers = 4
n_aircrafts = 20
groundACLinterarrival = 7 sec (media)
airACLinterarrival = 10 sec (media)
aircraftInterarrival = 28 sec (media)
groundResponseParams = 4 sec, 1 sec (media, deviazione)
airResponseParams = 4 sec, 1 sec (media, deviazione)
groundMissionParams = 600 sec , 45 sec (media, deviazione)
messageLength = 20,5 (media, deviazione)
ritardo VDL2 Uplink = 399 ms , 12 ms (media, deviazione)
ritardo VDL2 Downlink = 500 ms , 15 ms (media, deviazione)

Figura 6.2: Profilo 2: i parametri

6.2 Gli esperimenti

Dati quattro livelli per il primo fattore (il numero di Ground/Ground routers attivi) e due per il secondo (il traffico CPDLC) diventano così otto gli esperimenti possibili (figura 6.3).

6.2.1 Stima delle quantità

Ciascuno esperimento consiste nell'effettuare n esecuzioni (batch) indipendenti (l'indipendenza è data dalla generazione di numeri casuali) che consentono di stimare con un intervallo di confidenza del 95% le seguenti quantità: il tempo medio di transazione (TRN) sia lato terra TRN_{ground}

	Profilo 1: 75 velivoli	**Profilo 2: 125 velivoli**
Configurazione A	ExpA_1	ExpA_2
Configurazione B	ExpB_1	ExpB_2
Configurazione C	ExpC_1	ExpC_2
Configurazione D	ExpD_1	ExpD_2

Figura 6.3: Esperimenti

che lato aria TRN_{air}, ed il tempo tecnico medio di transazione ($RCTP$) sia lato terra $RCTP_{ground}$ che lato aria $RCTP_{air}$.

Si considerano delle variabili casuali X_j che hanno come valore la media delle N misure effettuate per una quantità di interesse nel j-esimo batch ($j = 1..n$). Il valor medio μ di quella quantità può essere stimato con la media degli X_j, definita come $\bar{X}(n) = \dfrac{\sum_{j=1}^{n} X_j}{n}$, in un intervallo di confidenza del $100(1-\alpha)$% ($\alpha = 0.05$ per il 95%) dato da:

$$\bar{X}(n) \pm \delta = t_{n-1,1-\alpha/2}\sqrt{\frac{S^2(n)}{n}}$$

Dove $t_{n-1,1-\alpha/2}$ rappresenta il valore della distribuzione t Student per $1-\alpha/2$ con $n-1$ gradi di libertà e $S^2(n)$ è la varianza degli X_j. Per un intervallo di confidenza del 95% è usato $t_{n-1,0.975}$. Il numero dei batch è scelto in maniera tale che l'errore commesso sulla media (la metà dell'ampiezza dell'intervallo di confidenza) sia al massimo il 10% della media μ della quantità. In altre parole, si suppone di aver determinato un intervallo di confidenza per la media μ partendo da un numero fissato n di batch, allora per raggiungere un errore relativo di γ sono richiesti $n^*(\gamma)$ batch.

$$n^*(\gamma) = min\left\{ i \geq n : \frac{t_{i-1,1-\alpha/2}\sqrt{S^2(n)/i}}{|\bar{X}(n)|} \leq \gamma' \right\}$$

dove $\gamma' = \gamma/(1+\gamma)$ rappresenta la correzione di γ affinché si possa ottenere l'errore relativo desiderato. Nel nostro caso, per un errore relativo del 10% è usato $\gamma = 0.10$ e $\gamma' = 0.09$.

6.2.2 Problema del transiente iniziale

Un sistema CPDLC reale rientra tra quei sistemi che iniziano la propria esecuzione una sola volta e continuano per un tempo indefinito. Poiché si vogliono stimare le medie delle performance di tale sistema, non è significativo considerare il periodo di tempo detto transiente iniziale necessario al sistema per stabilizzarsi. E' opportuno allora individuare questo transiente iniziale per scartare poi tutte le misure ad esso associate. Date m misure $Y_1, Y_2, ..., Y_m$ per un certa quantità si utilizza la media $\bar{Y}(m,l) = \frac{\sum_{i=l+1}^{m} Y_i}{m-l}$ dove l sono le misure relative al transiente iniziale. In particolare per determinare il numero l, è usata una procedura grafica (Welch) [10] che consiste di quattro passi principali:

1. Si effettuano n esecuzioni (5 o 10 batch) ottenendo per ciascuna m misure. Sia Y_{ji} la i-esima misura del batch j-esimo ($j = 1, 2, ..., n; i = 1, 2, ..., m$).

2. Si calcola $\bar{Y}_i = \sum_{j=1}^{n} Y_{ji}/n$ per $i = 1, 2, ..., m$

3. Si definisce la media mobile $\bar{Y}_i(w)$ nel modo seguente:

$$\bar{Y}_i(w) = \begin{cases} \dfrac{\sum_{s=-w}^{w} \bar{Y}_{i+s}}{2w+1}, & se\ i = w+1, ..., m-w \\ \\ \dfrac{\sum_{s=-(i-1)}^{i-1} \bar{Y}_{i+s}}{2i-1}, & se\ i = 1, ..., w \end{cases}$$

dove w è un intero positivo tale che $w \leq \lfloor m/4 \rfloor$

4. Si rappresenta graficamente la media mobile $\bar{Y}_i(w)$ per $i = 1, 2, ..., m-w$ e si sceglie l come il valore i per cui $\bar{Y}_i(w)$ sembra convergere.

La media mobile $\bar{Y}_i(w)$ permette di evidenziare l'andamento (trend) delle misure in modo molto più comprensibile rispetto ad usare direttamente $\bar{Y}_i$.

6.2.3 Profilo 1: 75 velivoli controllati

Risultati ExpA_1

Per l'esperimento ExpA_1 (un solo router attivo per tutta la durata dell'esecuzione e primo profilo di traffico) sono effettuate 8 esecuzioni (batch) che consentono il raggiungimento di un errore relativo inferiore alla soglia massima consentita (del 10%) per tutte le quantità. Quest'ultime eccetto TRN_{air} presentavano già dopo i primi 5 batch un errore relativo basso, mentre TRN_{air} ha raggiunto un errore accettabile, ma comunque lontano rispetto alle altre quantità, solo al termine delle 8 esecuzioni.
La tabella in figura 6.4 mostra per ciascuna quantità di interesse: la media, l'intervallo di confidenza del 95% e la deviazione standard ottenute con il procedimento descritto nel paragrafo 6.2.1 dalle misure relative alle

transazioni (1800 circa) effettuate nelle varie esecuzioni. Da queste misure sono escluse le prime misure che si riferiscono al transiente iniziale, determinato come già detto, attraverso la procedura grafica vista precedentemente (media mobile con $w = 30$). In particolare, per le quantità $RCTP_{ground}$, $RTCP_{air}$ e TRN_{ground} si osserva un andamento stabile dopo le prime 50 misure, mentre per la quantità TRN_{air} il comportamento è più vicino alla media dopo 100 misure, ma non si può certo affermare che sia convergente. Questo evidenzia maggiore incertezza nella stima della media di TRN_{air}. E' da sottolineare che questa quantità, il tempo di richiesta e risposta delle transazioni iniziate dall'aereo, si differenzia dalla sua corrispondente lato terra TRN_{ground}, perché introduce un ulteriore overhead legato al tempo di attesa nella coda di richieste mantenuta dal controllore. Le medie ottenute per le quantità $RCTP_{ground}$ ed $RCTP_{air}$ sono simili, ma soprattutto è importante notare l'incidenza del ritardo VDL2 (in media 399ms sia in Uplink che Downlink) in una transazione CPDLC ovvero circa il 90% della quantità $RCTP$, mentre solo il 10% è il contributo dei livelli applicativi e della rete ATN.
Infine, nei batch effettuati non sono registrate perdite di messaggi applicativi.

	Media (ms) $\bar{X}(n)$	Intervallo di confidenza 95% $[\bar{X}(n) - \delta, \bar{X}(n) + \delta]$	Deviazione (ms) S
$RCTP_{ground}$	888.91	$[884.62, 893.19]$	5.13
$RTCP_{air}$	887.24	$[881.15, 893.34]$	7.29
TRN_{ground}	6022.25	$[5957.26, 6087.24]$	77.74
TRN_{air}	7140.32	$[6604.98, 7675.65]$	640.34

Figura 6.4: risultati ExpA_1

Risultati ExpB_1

Nell'esperimento ExpB_1 sono effettuati lo stesso numero di batch e calcolati i risultati (fig. 6.5) seguendo il procedimento utilizzato precedentemente. Anche se in questo esperimento sono utilizzati due ruoter contemporaneamente attivi anziché un solo, non si notano differenze apprezzabili sui risultati ottenuti rispetto all'esperimento precedente e non si registrano neanche in questo caso perdite di messaggi applicativi. Quindi poter disporre di due router durante l'esecuzione, non riduce i tempi delle transazioni e non incrementa le performance del servizio CPDLC. Probabilmente, l'introduzione di un meccanismo di bilanciamento del carico (load balancing) avrebbe permesso di ottenere prestazioni migliori rispetto al caso precedente.

	Media (ms) $\bar{X}(n)$	Intervallo di confidenza 95% $[\bar{X}(n) - \delta, \bar{X}(n) + \delta]$	Deviazione (ms) S
$RCTP_{ground}$	883.51	$[878.70, 888.32]$	5.75
$RTCP_{air}$	892.23	$[886.75, 897.71]$	6.56
TRN_{ground}	5983.76	$[5943.26, 6024.26]$	48.45
TRN_{air}	7122.08	$[6544.40, 7689.76]$	679.03

Figura 6.5: risultati ExpB_1

Risultati ExpC_1

Per questo esperimento sono sufficienti 5 esecuzioni per rimanere al di sotto della soglia di errore consentita. La tabella (fig. 6.6) mostra i risultati ottenuti sempre con lo stesso procedimento, scartando le prime 50 misure per $RCTP_{ground}$, $RTCP_{air}$ e TRN_{ground}, e le prime 100 per TRN_{air}. Anche

qui si nota la maggiore variabilità di TRN_{air} rispetto alle altre quantità. L'introduzione di un guasto attraverso la disattivazione del primo router G/G dopo 30 minuti di esecuzione, provoca lo scambio di nuove informazioni di routing (protocolli IS-IS ed ES-IS) che fa aumentare leggeremente i tempi $RCTP$; sono infatti osservate alcune misure alte (fino a 12 sec.) durante il periodo centrale dell'esecuzione ovvero proprio quando il primo router è disattivato.
Oltre a questi ritardi, che rientrano comunque nei 16 secondi del requisito per il parametro $RCTP$ (fig. 3.7), si osservano anche delle perdite di messaggi applicativi. Queste causano la terminazione prematura della transazione associata, ciò accade per ciascuna esecuzione al massimo due volte sulle 1800 circa transazioni effettuate. Questo comportamento del sistema è ovviamente indesiderato perché è attesa dal protocollo di trasporto TP4 una consegna affidabile dei messaggi attraverso il meccanismo di ritrasmissione.

	Media (ms) $\bar{X}(n)$	Intervallo di confidenza 95% $[\bar{X}(n) - \delta, \bar{X}(n) + \delta]$	Deviazione (ms) S
$RCTP_{ground}$	935.99	$[876.80, 995, 18]$	47.67
$RTCP_{air}$	900.94	$[884.59, 917.28]$	13.17
TRN_{ground}	6082.24	$[5981.77, 6182.72]$	80.92
TRN_{air}	7086.69	$[6711.29, 7462.08]$	302.33

Figura 6.6: risultati ExpC_1

Risultati ExpD_1

Come nell'esperimento precedente durante il periodo di disattivazione del primo router (in questo caso dopo 20 minuti) si evidenziano alcune misure $RCTP$ alte, in particolare, in una esecuzione si raggiunge una punta di 16 sec. dal lato air, mentre le altre si aggirano intorno ai 10-12 sec. Sempre nel periodo critico del guasto del router, sono localizzate inoltre delle perdite di messaggi applicativi. Per questa ragione valgono le stesse considerazioni del caso precedente.
Il ripristino del primo router non comporta alcun beneficio in termini di prestazioni CPDLC, in quanto una volta che il guasto è rilevato dai componenti ATN, sono diffuse nuove informazioni di routing e il sistema di terra (Ground End System) seleziona il secondo router come predefinito fino al termine dell'esecuzione. Al momento del ripristino del primo router sono comunque diffuse delle informazioni di routing, che però non influenzano i tempi delle transazioni.

	Media (ms) $\bar{X}(n)$	Intervallo di confidenza 95% $[\bar{X}(n) - \delta, \bar{X}(n) + \delta]$	Deviazione (ms) S
$RCTP_{ground}$	920.47	$[888.25, 952.69]$	38.54
$RTCP_{air}$	907.81	$[879.31, 936.30]$	34.08
TRN_{ground}	6035.28	$[5983.98, 6086.59]$	61.37
TRN_{air}	6868.06	$[6542.64, 7193.49]$	389.25

Figura 6.7: risultati ExpD_1

Riepilogo risultati (profilo 1)

Per il primo profilo di traffico le medie stimate delle quantità rientrano abbondantemente nei requisiti specificati per il 95% delle transazioni ACL CPDLC (si veda la figura 3.7 del capitolo 3). In particolare, in accordo alle simulazioni ATCS [34] il requisito di 16 secondi per i tempi $RCTP$ appare giustificato in una infrastruttura di rete completa. Anche le quantità TRN sono molto più basse rispetto ai requisiti, ma queste a differenza dei tempi $RCTP$, dipendono da molti fattori legati ai tempi risposta degli operatori come il carico di lavoro, la loro esperienza ed il livello di usabilità delle interfacce HMI, risultando quindi più difficili da prevedere.
Nel grafico (fig. 6.8) che rappresenta le medie delle quantità per gli esperimenti effettuati, non si notano differenze apprezzabili tra i vari esperimenti; perciò le performance del servizio CPDLC non migliorano con l'impiego di due router anziché uno solo, ma neanché degradano in presenza di un router guasto nel sistema.

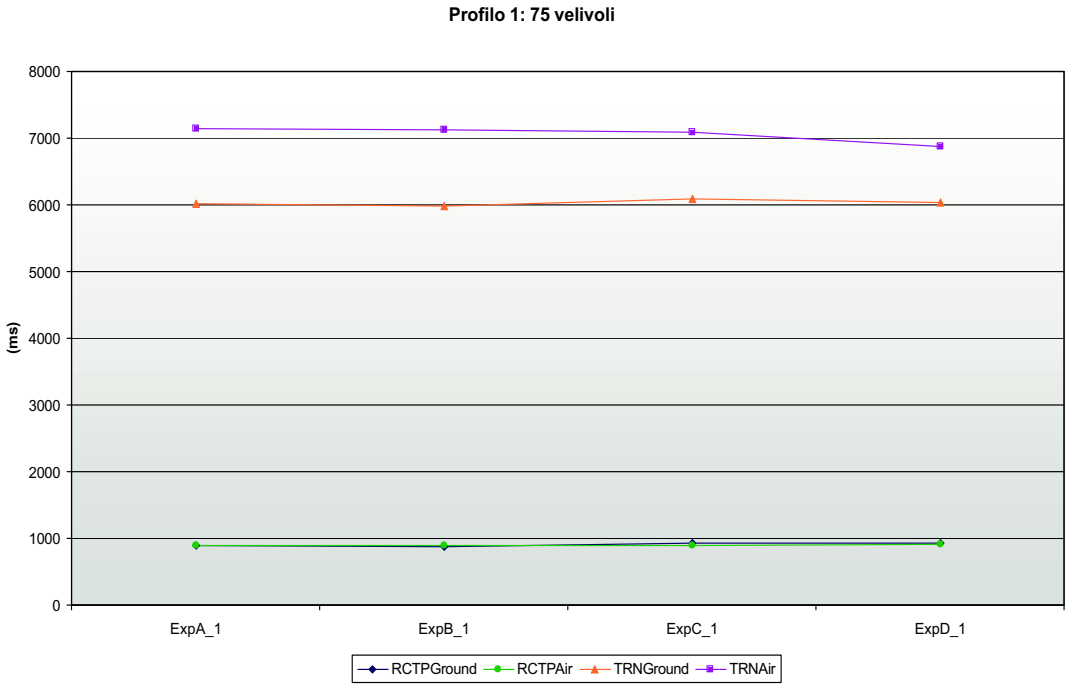

Figura 6.8: Profilo 1: medie delle quantità

6.2.4 Profilo 2: 125 velivoli controllati

Risultati ExpA_2

L'esperimento ExpA_2 considera un carico applicativo maggiore rispetto agli esperimenti precedenti, corrispondente alle transazioni effettuate (1500 circa) da 125 aerei durante 30 minuti di esecuzione del sistema CPDLC. Sono effettuate 8 esecuzioni (batch) distinte utilizzando la configurazione di rete ATN con un solo router attivo e il secondo profilo applicativo (fig. 6.2). In modo analogo ai primi esperimenti, questi batch sono sufficienti per raggiungere un errore relativo inferiore a quello massimo tollerato del 10%. La tabella (fig. 6.9) riporta i valori calcolati secondo il proced-

imento descritto nel paragrafo 6.2.1, anche qui per risolvere il problema del transiente iniziale sono scartate le prime misure delle quantità: 50 per $RCTP_{ground}$, $RTCP_{air}$, TRN_{ground} e 100 per TRN_{air}.
Dalla tabella risulta che i tempi $RCTP$ sono molto vicini tra loro, ma è opportuno sottolineare che in questo caso il ritardo VDL2 rappresenta l'85% degli $RTCP$ contro il 90% osservato nell'esperimento ExpA_1. D'altra parte è interessante osservare la variazione del contributo applicativo ovvero la differenza tra TRN ed $RCTP$ rispetto all'esperimento ExpA_1. Mentre il contributo applicativo lato terra ($TRN_{ground} - RCTP_{ground}$) si riduce dell' 1.3%, quello lato aria ($TRN_{air} - RTCP_{air}$) invece aumenta di circa il 4%. Questo indica che l'aumento della domanda del sistema CPDLC ovvero l'incremento dei velivoli da controllare si riflette maggiormente, anche se in maniera tollerabile, sul carico di lavoro degli controllori e conseguentemente sui tempi di risposta percepiti dai piloti. Infine, durante questo esperimento non sono osservate perdite di messaggi applicativi.

	Media (ms) $\bar{X}(n)$	Intervallo di confidenza 95% $[\bar{X}(n) - \delta, \bar{X}(n) + \delta]$	Deviazione (ms) S
$RCTP_{ground}$	1049.23	$[1045.21, 1053.26]$	4.94
$RTCP_{air}$	1056.24	$[1048.80, 1063.68]$	9.12
TRN_{ground}	6116.60	$[6052.93, 6180.28]$	78.10
TRN_{air}	7555.61	$[6854.43, 8256.79]$	860.03

Figura 6.9: risultati ExpA_2

Risultati ExpB_2

Non si notano particolari differenze con l'esperimento precedente, ciò rafforza quanto detto a proposito dell'esperimento ExpB_1 ovvero che le performance CPDLC non sono sensibili al numero di router attivi. E' inoltre possibile osservare l'incremento della deviazione standard soprattutto per la quantità TRN_{air}, evidenziando una maggiore variabilità ed incertezza sulla stima di questa quantità rispetto alle altre.

	Media (ms) $\bar{X}(n)$	Intervallo di confidenza 95% $[\bar{X}(n) - \delta, \bar{X}(n) + \delta]$	Deviazione (ms) S
$RCTP_{ground}$	1048.62	$[1040.15, 1057.09]$	11.02
$RTCP_{air}$	1060.25	$[1050.84, 1069.67]$	12.25
TRN_{ground}	6163.48	$[6084.50, 6242.46]$	102.75
TRN_{air}	7549.91	$[6771.64, 8328.18]$	1012.49

Figura 6.10: risultati ExpB_2

Risultati ExpC_2

Per quanto riguarda la disattivazione del primo router a metà esecuzione, in questo caso dopo 15 minuti, si registrano ancora una volta alcuni ritardi per le quantità $RCTP$ sia lato terra che aria. I ritardi che oscillano dai 10 ai 13 secondi (solo una volta si è osservato 17 secondi) sono pochi e tutti localizzati nel periodo di disattivazione. A differenza dell'esperimento ExpC_1, è stata osservata una minore frequenza di perdite di messaggi applicativi; perciò possiamo affermare che in questo caso il guasto inserito ha avuto meno impatto sul sistema.

	Media (ms) $\bar{X}(n)$	Intervallo di confidenza 95% $[\bar{X}(n) - \delta, \bar{X}(n) + \delta]$	Deviazione (ms) S
$RCTP_{ground}$	1081.87	$[1066.39, 1097.34]$	20.13
$RTCP_{air}$	1105.93	$[1077.85, 1134.01]$	36.53
TRN_{ground}	6200.39	$[6163.40, 6237.28]$	48.12
TRN_{air}	7677.11	$[6993.11, 8361.11]$	889.85

Figura 6.11: risultati ExpC_2

Risultati ExpD_2

L'esperimento ExpD_2, analogo ad ExpD_1, consiste nel disattivare il primo router dopo 10 minuti di esecuzione per riattivarlo successivamente raggiunti i 20 minuti. I risultati ottenuti (fig. 6.12) evidenziano dei tempi di richiesta/risposta lato aereo leggermente inferiori rispetto all'esperimento precedente, soprattutto per $RTCP_{air}$ si ottiene all'incirca la stessa media di quella calcolata negli esperimenti ExpA_2 ed ExpB_2. Per quanto riguarda il verificarsi dei ritardi e delle perdite di messaggi applicativi, non si notano nel complesso sostanziali differenze rispetto all'esperimento precedente. L'unica eccezione è una singola misura di 36 secondi registrata per la quantità $RTCP_{ground}$.

	Media (ms) $\bar{X}(n)$	Intervallo di confidenza 95% $[\bar{X}(n) - \delta, \bar{X}(n) + \delta]$	Deviazione (ms) S
$RCTP_{ground}$	1094.76	[1048.66, 1140.86]	59.98
$RTCP_{air}$	1065.43	[1054.83, 1076.03]	13.79
TRN_{ground}	6264.61	[6164.63, 6364.60]	130.08
TRN_{air}	7295.41	[6638.02, 7952.79]	855.22

Figura 6.12: risultati ExpD_2

Riepilogo risultati (profilo 2)

Anche negli esperimenti per il secondo profilo di traffico, le medie stimate sono tutte inferiori a quanto specificato nei requisiti (fig. 3.7). Inoltre, come è evidenziato dal grafico (fig. 6.13) non si notano particolari differenze in termini di prestazioni CPDLC al variare della configurazione di rete ATN. A tale proposito valgono le stesse considerazioni fatte nel riepilogo degli esperimenti del primo profilo. Infine, occorre osservare che questo aumento di traffico dati è ben tollerato dal sistema perché non ha conseguenze così significative sulle performance del servizio CPDLC.

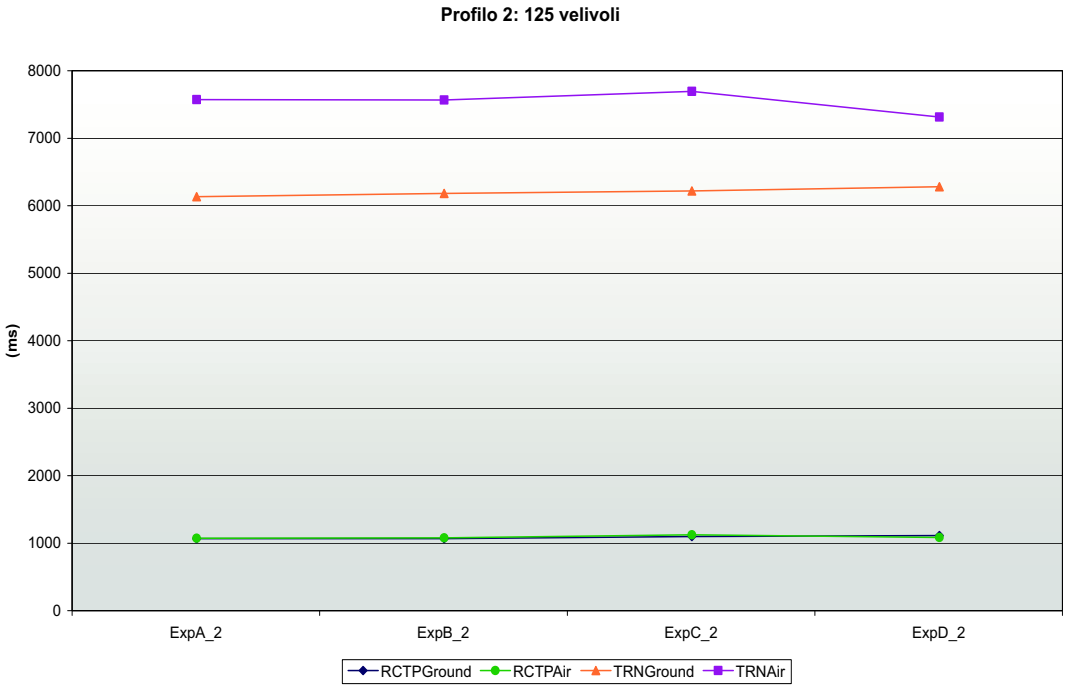

Figura 6.13: Profilo 2: medie delle quantità

Capitolo 7

Conclusioni

In questa tesi abbiamo dapprima introdotto i concetti teorici, le metodologie e gli strumenti più comuni per l'analisi e la valutazione quantitativa dei sistemi. Da una descrizione ad alto livello applicabile a tutti i sistemi, ma in modo particolare a quelli distribuiti, ci siamo addentrati nel contesto specifico presentando la rete ATN, ovvero l'infrastruttura di comunicazione per lo sviluppo dei futuri servizi applicativi per la gestione e il controllo del traffico aereo. Tra questi servizi abbiamo descritto il CPDLC per lo scambio di dati tra pilota e controllore, evidenziandone finalità, modalità operative e requisiti.
L'analisi quantitativa del servizio CPDLC considerando la reale infrastruttura di comunicazione alla base, è stata possibile attraverso una tecnica simulativa/sperimentale per mezzo dello strumento Neko corredato di un'apposita libreria (XtiNetwork) per il supporto di ATN. Tale strumento ha permesso lo sviluppo di un prototipo di sistema CPDLC su rete ATN che può essere facilmente adattabile, grazie alla struttura modulare dei livelli applicativi, per rispondere a future esigenze. Tra queste il testing

di altre procedure operative, lo sviluppo di specifici meccanismi (Fault-tolerant) di tolleranza ai guasti, la verifica di copertura dei guasti inseriti, ad esempio, attraverso dei livelli applicativi intermedi con una tecnica di Fault-Injection via software. In definitiva, questo prototipo di sistema può rappresentare un importante punto di partenza per l'analisi e lo sviluppo di nuove applicazioni distribuite.

Ringraziamenti

A Pierluigi Fantappié e Paolo Maltese per aver ispirato e sostenuto questo bel progetto, al Prof. Andrea Bondavalli ed al suo assistente Lorenzo Falai perché grazie alla loro competenza e passione lo hanno reso ancora più interessante. Ad Andrea Berti per avermi aiutato nei rapporti con Airtel, a Santi Ibarz e Frank O'Connor perché senza di loro non sarebbe stata possibile l'installazione di una vera rete ATN.
A Peter Urban che in poche email è riuscito a darmi preziose indicazioni su Neko. A tutti i dipendenti OTE del reparto ARC System Test con i quali ho condiviso gran parte delle giornate, in particolare, Claudio Borgioli e Stefano Calamandrei per le loro utili osservazioni.

Sono grato alla mia famiglia, soprattutto ai miei genitori e mia sorella Beatrice per il loro costante sostegno in questi anni di studio, a Maria Giulia per essermi davvero vicina, ed alla sua famiglia. Infine, desidero ringraziare i miei amici, alcuni dei quali anche compagni di studio.

Bibliografia

[1] Tanenbaum, van Steen. Distributed Systems, Principles and Paradigms, Book, Prentice Hall (2002).

[2] N.Lynch. Distributed Algorithms, Book, Morgan Kaufmann (1996).

[3] Coulouris, Dollimore, Kindberg. Distributed Systems, Concepts and Design, book, Addison-Wesley (2001).

[4] J.C. Laprie. Dependable Computing and Fault Tolerance: Concepts and Terminology , Article, IEEE (1985).

[5] A. Avizienis, J.C. Laprie, B. Randell. Fundamental Concepts of Dependability, Technical report, LAAS-CNRS (2000).

[6] D. P. Siewiorek, R. S. Swarz. Reliable Computer Systems - Design and Evaluation - Third Edition, Book, A K Peters Ltd. (1998).

[7] L. Hu, I. Gorton. Performance Evaluation for Parallel Systems: A Survey, University of NSW, Australia (1997).

[8] R. Sahner, A. Puliafito, K. S. Trivedi. Performance and Reliability Analysis of Computer Systems, Book, Springer (1995).

[9] K. S. Trivedi, M. Malhotra. Reliability and Performability Techniques and Tools: A Survey, Survey, MMB (1993).

[10] A. M. Law, W. D. Kelton. Simulation Modelling Analysis - Third Edition, Book, McGraw-Hill (2000).

[11] A. M. Law, M.G. McComas. How to build valid and credible simulation models, Proceedings of the 2001 Winter Simulation Conference.

[12] M. Kaâniche, J. C. Laprie, J. P. Blanquart. Dependability Engineering of Complex Computing Systems, Article, IEEE (2000).

[13] M. Lane. Predicting the reliability and satefy of commercial software in advanced avionic systems, Article, BCC (2000).

[14] N.C. Audsley, M. Burke. Distributed Fault-Tolerant Avionic Systems - A Real-Time Perspective, Article, IEEE (1998).

[15] J. Banks. Introduction to Simulation, Proceedings of the 1999 Winter Simulation Conference.

[16] J. Carreira, H. Madeira, J. G. Silva. Xception: A Techinque for the Experimental Evaluation of Dependability in Modern Computers, Article, IEEE (1998).

[17] J. A. Clark, D. K. Pradhan. Fault Injection: A method for validating computer-system dependability, Article, IEEE (1995).

[18] K. S. Trivedi. Probability and Statistics with Reliability, Queuing and Computer Science Applications, Book, Prentice-Hall (1982).

[19] P. Urban, X. Defago, A. Schiper. Neko: A single Environment to Simulate and Prototype Distributed Algorithms, Journal of Information Science and Engineering 18 (2002).

[20] L. Falai. Metodologie e strumenti per la valutazione quantitativa sperimentale e simulativa di algoritmi distribuiti, Tesi di laurea, Università degli studi di Firenze (2004).

[21] S. Bernardi, C. Bertoncello, S. Donatelli, G. Franceschinis, R. Gaeta, M. Gribaudo, A. Horváth. GreatSPN in the new Millenium, Università di Torino ed Università del Piemonte Orientale.

[22] A. Bondavalli, S. Chiaradonna, F. Di Giandomenico, I. Mura. Dependability Modeling and Evaluation of Multiple Phased Systems using DEEM, article, IEEE (2000).

[23] G. Clark, T. Courtney, D. Daly, D. Deavours, S. Derisavi, J. M. Doyle, W. H. Sanders, P. Webster. The Mobius Modeling Tool, 9th international Workshop on Petri Nets and Performance Models (2001).

[24] NS-2, website, www.isi.ed/nsnam/ns

[25] Scalable Simulation Framework website, website, www.ssfnet.org

[26] The Colt Distribution - Open Source Libraries for High Performance Scientific and Technical Computing in Java, website, hoschek.home.cern.ch/hoschek/colt

[27] IEN Galileo Ferraris website, website, www.ien.it

[28] S. Zerkowitz (IATA). Is the Need for a New ATM Operational Concept a Stategic Necessity?, article, IEEE (2003).

[29] O. V. Prinzo. Data-Linked Pilot Reply Time on Controller Workload and Communication in a Simulated Terminal Option, FAA (2001).

[30] Linux XTI Programmers's Reference 0.3, Airtel (2004).

[31] Specification for the X/Open Transport Interface (XTI), X/Open Company Ltd.

[32] S. Hemminger. Network Emulation with NetEm, Open Source Development Lab (2005).

[33] Signal-in-Space Minimum Aviation System Performance Standards (MASPS) for Advanced VHF Digital Data Communications Including Compatibility With Digital Voice Techniques, RTCA/DO-224A (2000).

[34] ACTS Aeronautical Communications Technologies Simulator - Results VDL2 1.4, Eurocontrol (2005).

[35] ACTS Aeronautical Communications Technologies Simulator - Validation Report, Eurocontrol (2006).

[36] K. J. Viets, C. G. Ball. Validating a Future Operational Concept for En Route Air Traffic Control, IEEE (2001).

[37] J. C. Knight. Safety Critical Systems: Challenges and Directions, ACM (2002).

[38] ED-110A/DO-280A Interoperability Requirements Standard For ATN Baseline 1, RCTA and EUROCAE (2004).

[39] Link2000+ Network Planning Document, Eurocontrol (2002).

[40] ATC Data Link Manual for Link2000+ Services, Eurocontrol (2004).

[41] PETAL-II Transition and Final Report, Eurocontrol (2002).

[42] ANNEX C Operational Performance Assessment (OPA), RCTA and EUROCAE (2006).

[43] C. Dhas, T. Mulkerin, C. Wargo, R. Nielsen, T. Gaughan. Aeronautical Related Applications Using ATN and TCP/IP Research Report, NASA (2000).

[44] Comprehensive ATN Manual (CAMAL), FANS (1999).

[45] Manual of Technical Provisions for the Aeronautical Telecommunication Network (Doc. 9705), ICAO (2002).

[46] Standards and Recommended Practices (SARPs) for the Aeronautical Telecommunication Network (ATN) 3rd Edition, Helios (2001).

[47] ED-120 Safety and Performance Requirements Standard For Initial Air Traffic Data Link Services In Continental Airspace (SPR IC), RCTA and EUROCAE (2004).

[48] ATFM and Capacity Report 2004, Eurocontrol (2005).

[49] H. Tekura, A. Hori, Y. Ishikawa, M. Sato. PM: An operating system coordinated high performance communication library, Proceedings of High-Performance Computing and Networking Conference (1997).

[50] M. Hayden. The Ensemble system, Technical Report, Department of Computer Science, Cornell University (1998).

[51] J.F. Hermant, G. L. Lann. A protocol and correctness proofs for real-time high-performance broadcast networks, Proceedings of 18th International Conference on Distributed Computing Systems (1998).

[52] SimJava, website, www.dcs.ed.ac.uk/home/hase/simjava/

[53] NTP: The Network Time Protocol, website, www.ntp.org