ALMA MATER STUDIORUM - UNIVERSITÀ DI BOLOGNA

SCUOLA DI INGEGNERIA E ARCHITETTURA

Dipartimento di Informatica – Scienza e Ingegneria

Corso di Laurea Triennale in Ingegneria Informatica

TESI DI LAUREA
in
Fondamenti di Telecomunicazioni T

Modifiche al protocollo di trasporto LTP per migliorarne le prestazioni in presenza di perdite elevate

CANDIDATO:
Nicola Alessi

RELATORE:
Prof. Carlo Caini
CORRELATORE:
Dr. Ing Tomaso de Cola

Anno Accademico 2014/2015

Sessione III

A mio padre e a mia madre
che mi hanno sempre sostenuto
ed incoraggiato in tutte
le scelte della mia vita.
Grazie per aver creduto in me.

Indice generale

Prefazione

L'esplorazione dello spazio è divenuta possibile partendo inizialmente da osservazioni a occhio nudo delle stelle e della luna. I contributi più significativi in ambito astronomico nell'antichità sono stati dati da Ipparco (190 A.C.-120 A.C.) e Tolomeo (100 D.C.-175 D.C.). Nei secoli più recenti, le osservazioni sono state supportate da nuove tecnologie terrestri, come telescopi ottici (1608) e radiotelescopi (1931). Con il passare del tempo, e con la rapida evoluzione tecnologica, si è iniziato a sentire il bisogno di osservare i corpi celesti da una distanza più ravvicinata. L'uomo ha così inviato nello spazio e su altri pianeti diversi oggetti come satelliti artificiali (Sputnik, 1957), sonde (1958), lander (il LEM che ha portato l'uomo sulla luna nel 20 luglio 1968), rover (Lunochod 1, 1970; LVR, 1971) ecc. iniziando così ad esplorare materialmente lo spazio.
Verso la fine degli anni '90 Vint Cerf, già progettista dei protocolli TCP/IP (1978) insieme a Bob Kahn**,** ha sviluppato l'idea di estendere la rete Internet terrestre per poterla usare nello spazio: è nato così il concetto di "*Internet Interplaneraria*" (InterPlanetary Networking, IPN).
Per pensare in termini di IPN è necessario considerare uno scenario di riferimento diverso rispetto a quello per le comunicazioni terrestri [Mann].
Mentre per le comunicazioni terrestri è ragionevole considerare Round Trip Time (RTT) brevi, canali continui e basse percentuali di errore sul canale, per le comunicazioni interplanetarie occorre rivalutare queste ipotesi.
Le distanze tra i pianeti nel nostro sistema solare si presentano nell'ordine dei minuti luce (1 minuto luce è circa uguale a 18Gm) per i pianeti più vicini ,ad esempio Terra-Marte, distanti mediamente 5 minuti luce, e nell'ordine delle ore luce(1 ora luce è circa uguale a 1,08Tm) per i pianeti più lontani, ad esempio Terra-Nettuno distanti mediamente 4 ore luce [Nasa_Light].
Per questo motivo il tempo di propagazione minimo è di ordine nettamente superiore a quello terrestre, che va da qualche millisecondo a circa duecento.
Un altro problema di natura fisica è dovuto al moto di rotazione e di rivoluzione dei corpi celesti. Difatti, per poter comunicare nello spazio profondo è necessario che il percorso diretto tra il mittente e il destinatario sia libero. Un eventuale ostacolo, che potrebbe essere anche il pianeta stesso, interromperebbe il canale di comunicazione.
Allo stato dell'arte le tecnologie utilizzate per inviare i segnali dati sul canale appartengono a due famiglie: Radio Frequency Technologies (RF) e Laser Communications (Lasercom) [Deutsch].Si è osservato sperimentalmente che utilizzando le RF, è possibile considerare il canale con errori indipendenti con una probabilità di errore per pacchetto (Packet Error Ratio, PER) di solito inferiore al 3%, mentre per le Lasercom il canale presenta di solito errori correlati con PER che può arrivare anche al 15%. Le Lasercom tuttavia permettono delle velocità di trasmissione nettamente più elevate.
Risulta quindi chiaro che bisogna considerare l'ipotesi di Round Trip Time (RTT) estremamente lunghi e canali di trasmissione soggetti ad alte percentuali di errore, attivi solo in determinate finestre temporali.
Le architetture di rete usate in ambito terrestre (ad esempio quella TCP/IP) risultano inadeguate per affrontare queste tipologie di problemi.
Delay/Disruption Tolerant Networking (DTN) è l'architettura di rete proposta per le future missioni spaziali e più in generale per la realizzazione di una futura Internet interplanetaria.
L'idea alla base delle DTN è nata generalizzando i requisiti identificati per le IPN, in particolare, per situazioni dove i satelliti orbitanti possono funzionare come nodi intermedi per comunicare da o verso la superficie di un pianeta, quando la comunicazione diretta è impossibile a causa del pianeta stesso.

Mentre inizialmente l'unico scenario di riferimento erano le comunicazioni interplanetarie, successivamente l'idea si è rivelata utile anche per alcune applicazioni terrestri, sia in ambito civile che militare.
L'architettura DTN e i suoi protocolli sono stati principalmente sviluppati dalla *Internet Research Task Force (IRTF) DTN Research Group (DTNRG)*, gruppo di ricerca aperta che conta partecipanti da diverse parti del mondo.
Un altro gruppo di ricerca importante che lavora sull'architettura DTN è il *Consultative Committee on Space Data Systems (CCSDS) DTN Working Group*, che si interessa però di proporre nuove specifiche finalizzate all'utilizzo dell'architettura DTN in ambito spaziale.
A differenza del DTNRG, CCSDS è un organismo composto dalle maggiori agenzie spaziali, comprese NASA ed ESA.
L'idea alla base dell'architettura DTN è il concetto di “Store and Forward”, ovvero ogni nodo intermedio mantiene l'informazione da trasferire al nodo successivo in un database (Store) per poi inviarla quando si presentano le condizioni adatte (Forward) [RFC4838].
Licklider Transmission Protocol (LTP) [RFC5325] è un protocollo di trasporto dotato di stato, progettato per fornire una comunicazione affidabile basata sulle ritrasmissioni su dei canali caratterizzati da RTT estremamente lunghi e possibili interruzioni della connessione.
Seppur concepito inizialmente per funzionare in modo indipendente, il protocollo LTP è stato quindi inserito come *Convergence Layer* nell'architettura DTN Bundle Protocol (BP) [RFC5325].
Il software *Interplanetary Overlay Network* (ION) è un'implementazione open source dell'architettura DTN realizzata da NASA-JPL (National Aeronautics and Space Administration, Jet Propulsion Laboratory – California Institute of Technology) [ION_DOC].
La presente tesi si pone come obiettivo quello di analizzare il protocollo LTP (riferendosi in particolare ad ION) e proporre dei miglioramenti utili al caso in cui siano presenti perdite elevate.
Più in dettaglio, una prima parte introduttiva motiva l'inefficacia del TCP/IP in ambito interplanetario e introduce l'architettura DTN Bundle Protocol (Capitolo 1). La tesi prosegue con la descrizione delle specifiche del protocollo LTP (Capitolo 2), in particolar modo evidenziando come un bundle venga incapsulato in un blocco LTP, come questo sia successivamente diviso in tanti segmenti LTP e come questi vengano successivamente inviati con il protocollo UDP o con un protocollo analogo. Viene quindi presentata un'approfondita analisi delle penalizzazioni dovute alle perdite dei segmenti LTP, sia di tipo dati che di segnalazione (Capitolo 3). Quest'analisi permette di dimostrare la criticità degli effetti delle perdite, in particolare per quello che riguarda i segmenti LTP di segnalazione. Mentre in presenza di perdite basse tali effetti hanno in media un impatto minimo sul tempo di consegna di un blocco LTP (quindi del bundle in esso contenuto), in quanto avvengono raramente, in presenza di perdite elevate rappresentano un collo di bottiglia per il tempo di consegna di un blocco LTP. A partire da questo risultato sono state proposte alcune modifiche che permettono di migliorare le prestazioni di LTP (Capitolo 4) compatibilmente con le specifiche RFC in modo da garantire l'interoperabilità con le diverse implementazioni del protocollo. Successivamente nel Capitolo 5 viene mostrato come sono state implementate le modifiche proposte in ION (all'attuale versione 3.4.1). Nel capitolo finale (Capitolo 6) sono presenti i risultati numerici relativi ad alcuni test preliminari eseguiti confrontando la versione originale del protocollo con le versioni modificate contenenti i miglioramenti proposti. I test sono risultati molto positivi per elevate perdite, confermando così la validità dell'analisi e dei miglioramenti introdotti.

Capitolo 1: L'architettura Delay-/Disruption Tolerant Networking

1. Introduzione

Come già accennato nella prefazione, l'ambiente di riferimento IPN presenta le seguenti caratteristiche:

- Tempo di propagazione elevato;
- Canale di trasmissione intermittente e asimmetrico;
- Probabilità di errore sul canale a volte elevata;

Queste caratteristiche rendono impossibile l'uso dell'architettura e dei protocolli TCP/IP, come mostrato in particolare nel prossimo paragrafo per quanto riguarda il protocollo TCP [Durst]. Le reti interplanetarie sono tuttavia solo un esempio di “challenged networks”, ovvero delle reti nei quali le caratteristiche del canale impediscono l'utilizzo dei protocolli TCP/IP. E' proprio riconoscendo che reti apparentemente diverse, quali le reti spaziali, le reti sottomarine, le reti di sensori radio, le reti di emergenza, le reti militari tattiche, ecc. avevano in realtà problemi simili, che Kevin Fall ebbe l'idea di estendere l'ambito di studio dell'IPN a tutte le reti challenged, coniando il termine DTN (Delay-/Disruption Tolerant Networking). L'obiettivo è quello di promuovere una soluzione unica a tutte le reti challenged, anziché una serie di soluzioni di ambito limitato.
L'architettura DTN descritta in [RFC4838] è basata sull'introduzione di un livello al di sopra del livello di trasporto (o altri livelli ad esso inferiori), chiamato “Bundle Layer”. Le caratteristiche essenziali di questo nuovo livello sono due: la suddivisione del percorso end-to-end in più tratte, dette DTN hop, aventi agli estremi un nodo DTN, e la memorizzazione dell'informazione nei nodi DTN intermedi secondo un meccanismo detto “Store and Forward”. In questo modo le problematiche relative a ritardi e interruzioni del canale possono essere gestite *hop by hop* nel percorso tra mittente e destinatario.
In particolare, risulta ridefinita la semantica end-to-end dei protocolli di trasporto, ora limitata ad una sola tratta DTN. Conviene introdurre un esempio: in una comunicazione Terra-Marte si possono distinguere 3 tratte DTN. Una fra il mittente ed un possibile gateway DTN terrestre; una fra questo gateway ed un suo corrispondente su Marte; una fra quest'ultimo e l'utilizzatore finale, ad esempio un rover. La prima e l'ultima tratta, quella terrestre e quella marziana, non hanno particolari problemi sul canale, quindi in esse può tranquillamente essere ancora usato il protocollo TCP (limitato però all'interno della tratta). Fra I due gateway, al contrario, è del tutto impossibile, come vederemo sotto, l'utilizzo del TCP. Esso dovrà essere rimpiazzato da un protocollo progettato ad hoc per le reti spaziali, il Licklider Transmission Protocol. L'architettura DTN, suddividendo il percorso in più tratte DTN, ha il merito di:

1. isolare i problemi di una tratta dalle altre;
2. poter utilizzare su ognuna di esse il protocollo più appropriato alla risoluzione dei problemi di quella tratta;

2. Limiti e problemi dell'architettura TCP/IP per l'IPN

i. Apertura e chiusura di una connessione: three way handshake

Molte scelte progettuali del TCP[RFC793, RFC5681, RFC7414], sono giustificate dall'ipotesi

che i tempi di propagazione *one-way* e di conseguenza anche i Round Trip Time sono molto piccoli.
Partendo dall'apertura di una connessione, il TCP utilizza il ben noto protocollo di apertura “three way handshake”, riassunto in figura [Tanenbaum].

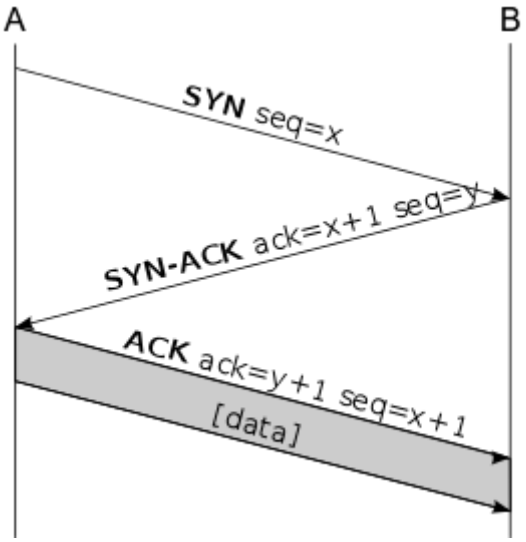

Figura 1: Three way handshake

La procedura consiste nello stabilimento di una connessione, in quanto il protocollo TCP è “Connection Oriented” fra A e B. In questa fase viene fra l'altro stimato il Round Trip Time su entrambi i nodi ed entrambi dichiarano all'altro la loro finestra di ricezione (Advertised Window, o anche Receiver Window, o rwnd), cioè la quantità di dati che sono momentaneamente disposti a ricevere, legata ai loro buffer di ricezione. Essa verrà poi aggiornata di continuo durante la connessione (Controllo di Flusso). Una volta stabilita, la connessione TCP è bidirezionale, cioè sia A che B possono mandare dati all'altro nodo.
In particolare si noti che Il nodo A inizia quindi il trasferimento dei dati verso B dopo un RTT, cioè di norma dopo pochi millisecondi. Nel caso A e B siano molto distanti si può arrivare a 200ms (fra Europa e cosa Ovest USA), o a 600ms nel caso di comunicazioni tramite un satellite GEO, mai oltre in ambito terrestre.
In ambito interplanetario i lunghi RTT rendono tutto molto più problematico. Ad esempio, con le impostazioni di default, lo stesso three way handshake sarebbe impossibile nello scenario Terra-Marte, poiché il timer legato alla ritrasmissione (RTO) del segmento SYN andrebbe a scattare prima della ricezione del corrispondente segmento SYN-ACK in quanto il suo valore iniziale è fissato ad 1s [RFC 6298]. Dopo questo primo fallimento, l'RTO iniziale viene raddoppiato (algoritmo di Exponential Backoff) e al suo scadere un altro SYN viene ritrasmesso. Nel caso in cui il SYN-ACK del primo SYN arrivi prima che sia esaurito il numero massimo di tentativi (in linux sono dati da tcp_syn_retries, che di default è settato a 5), sarebbe comunque possibile stimare il RTT e chiudere il three way handshake se la Timestamp Option è attiva. Questo perché il SYN-ACK riporterebbe il timestamp scritto nel SYN che l'ha generato, permettendo così al mittente di stabilire a quale dei SYN inviati il SYN-ACK si riferisce. Se la Timestamp Option non è attiva, viene applicato l'algoritmo di Karn. Esso prevede che gli acknowledgment provenienti da segmenti ritrasmessi vengano ignorati per l'aggiornamento del RTT, in quanto non è possibile stabilire a quale dei segmenti ritrasmessi essi facciano riferimento. Successivamente viene usata una strategia di backoff, andando ad aumentare progressivamente il valore dei timeout di ritrasmissione. Rispetto al

caso precedente inizialmente ci saranno delle ritrasmissioni a causa dell'impossibilità di stimare il RTT nella fase di three way handshake.
Tuttavia, nel caso di comunicazioni Terra-Marte, il numero massimo di tentativi verrebbe con ogni probabilità esaurito prima (dopo circa un minuto nel caso delle impostazioni di default di Linux,ovvero con tcp_syn_retries uguale a 5).
In questo scenario, inoltre, anche se fosse possibile concludere il three way handshake aumentando il numero massimo delle ritrasmissioni, il tempo di propagazione one-way sarebbe comunque elevatissimo, essendo compreso nell'intervallo fra 3 minuti e 22 minuti. Anche assumendo un ritardo di soli 5 minuti, il trasferimento dei dati potrebbe iniziare solo dopo 10 minuti al termine del three way handshake. Nel caso Terra-Nettuno, in cui il tempo di propagazione one-way (medio) è di circa 4 ore, il trasferimento dei dati inizierebbe dopo 8 ore.
Un caso limite potrebbe essere rappresentato dallo scenario Terra-Luna, in cui il tempo di propagazione one-way è di circa 1 secondo. In questo scenario, se la Timestamp Option del TCP è attiva e se si ipotizza di non avere perdite sui segmenti relativi al three way handshake, il SYN-ACK relativo al primo SYN arriva prima di aver esaurito i tentativi a disposizione, dando la possibilità al mittente di stimare correttamente il RTT.
Considerazioni analoghe possono essere fatte per la chiusura della connessione con three way handshake o four way handshake.

ii. Controllo di congestione

Dopo l'apertura della connessione, per garantire controllo di congestione [RFC5681], si entra in fase di “*Slow Start*” con valore di cwnd solitamente a 2.
In questa fase A invia cwnd segmenti a B e ad ogni ACK ricevuto incrementa di uno il valore della cwnd. Ad ogni RTT, trascurando la possibile presenza dei Delayed Ack, la cwnd raddoppia fino al raggiungimento del valore di “Slow Start Threshold” (ssthresh).

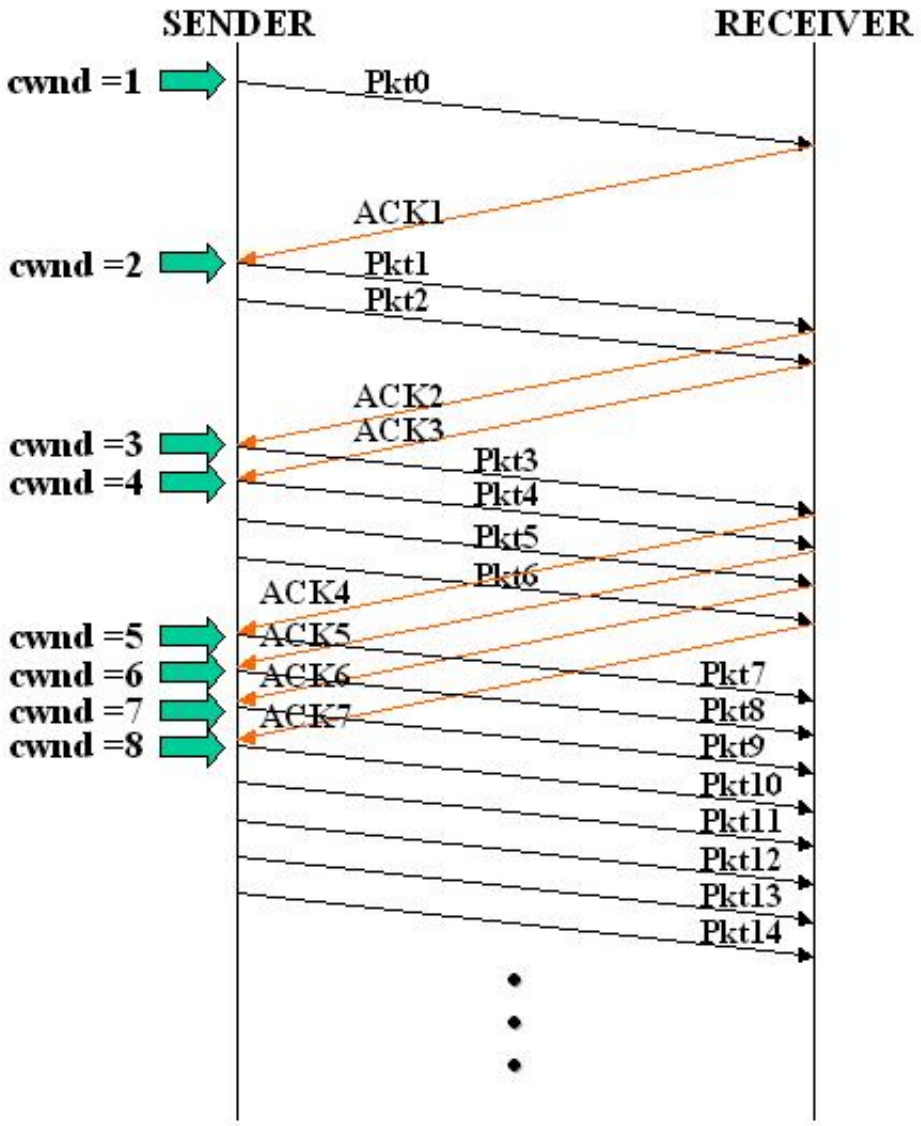

Figura 2: Slow Start

Anche in questo caso, è chiaro che considerando i lunghi RTT dello scenario di riferimento, il tempo necessario a raggiungere la situazione di regime è inaccettabile.
Superata la soglia ssthresh, si entra nella fase di *"Congestion Avoidance"* (in cui è più probabile una congestione) dove la cwnd cresce linearmente con il RTT, ovvero cwnd viene incrementata di uno ogni cwnd ACK ricevuti (quindi sostanzialmente di uno per ogni RTT).
Per individuare i pacchetti persi, sono previsti 2 meccanismi validi sia per la fase di Slow Start che per la fase di Congestion Avoidance

- Scadenza di un timer (RTO)
- Ricezione di 3 ACK duplicati (Dup-Ack)

Nel primo caso viene settata la cwnd=1 e si ricomincia in fase di Slow Start.
Nel secondo caso il mittente invia immediatamente il segmento mancante (*Fast Retransmit*) , viene settata la sstresh alla metà del valore assunto dalla cwnd all'inizio della procedura di Fast Retransmit e la finestra cwnd viene quindi posta al valore sstresh al termine della

procedura, per poi crescere linearmente (*Fast Recovery*).
TCP interpreta la perdita di pacchetti come indicatore di congestione.

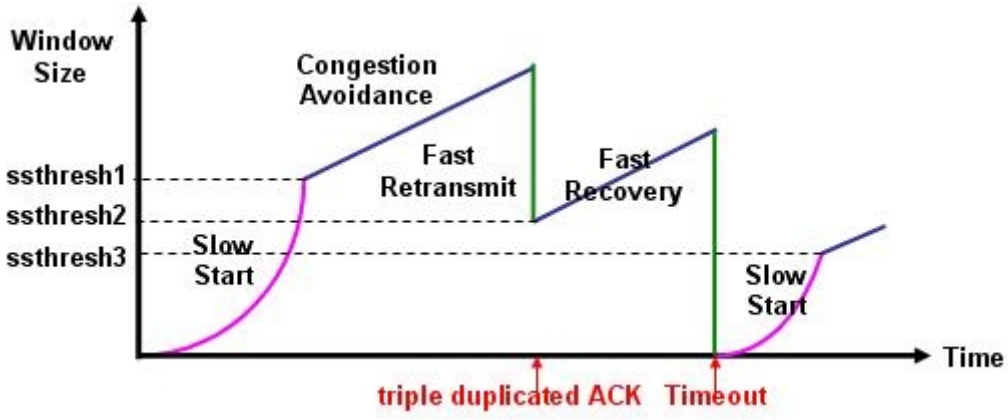

Figura 3: Fasi del controllo di congestione

I limiti imposti dallo scenario IPN di riferimento, anche in questo caso rendono i meccanismi di Congestion Avoidance inadeguati.

iii. Altre considerazioni usando l'equazione del throughput

Usando la formula del throughput [Padhye] si può calcolare la Loss Event Rate (p) richiesta per raggiungere idealmente lo steady-state.
Consideriamo uno scenario in cui il tempo minimo di RTT tra Terra e Marte sia di 8 minuti, la dimensione del pacchetto sia 1500 byte e che il ricevitore invia un ACK per ogni pacchetto ricevuto.
In questo caso si ottengono i seguenti valori:

Throughput	Loss Event Rate (p)
10 Mbps	$4{,}68 * 10^{-12}$
1 Mbps	$4{,}68 * 10^{-10}$
100 kbps	$4{,}68 * 10^{-8}$
10 kbps	$4{,}68 * 10^{-6}$

Come ci si aspettava dalle precedenti considerazioni, tali probabilità di errore richieste sono di ordine molto inferiore rispetto alle percentuali d'errore reali [RFC5325].
In questo modo è stato quindi possibile giustificare anche matematicamente l'inefficacia del TCP in ambito interplanetario (ed anche qualche difficoltà del TCP standard sulle reti ad alta velocità).
L'impossibilità di utilizzare il TCP in ambito interplanetario, ed altre considerazioni analoghe hanno inevitabilmente portato alla necessità di definire, come detto prima, una nuova architettura di rete.

3. L'architettura DTN ed il Bundle Protocol (BP)

L'architettura DTN è descritta in [RFC4838]. Essa si basa sull'introduzione del Bundle Layer, il cui protocollo, il Bundle Protocol (BP) è definito in [RFC5050] e per le applicazioni

spaziali è definito anche nello standard promulgato dal Consultative Committee for Space Data Systems [CCSDS 734.2-B-1].
Un applicazione DTN invia messaggi di lunghezza arbitraria, chiamati anche *Application Data Unit* (ADU). Gli ADU sono trasformati dal Bundle Layer in uno o più Protocol Data Unit (PDU) chiamati "bundle", i quali vengono inoltrati dai nodi DTN.
Un bundle contiene anche altre informazioni, come il nome del mittente, nome del destinatario, altri due indirizzi (report to: e custodian:) e altre informazioni aggiuntive per la consegna end-to-end.
I bundle possono essere divisi in "bundle fragment" da ogni nodo coinvolto durante la trasmissione. I frammenti sono a loro volta bundle che potrebbero essere ulteriormente frammentati. Due o più frammenti possono essere riassemblati in qualsiasi momento durante il trasferimento, formando un nuovo bundle.
Bundle Protocol [RFC5050] è l'implementazione dell'architettura DTN più usata.
Il Bundle Protocol si interfaccia con diversi layer di livello inferiore (solitamente con il livello di trasporto) attraverso il “Convergence Layer Adapter” (CLA), e prevede che ogni nodo DTN nel percorso può usare il CLA più appropriato per l'invio del bundle al prossimo nodo DTN. Si noti che fra due nodi DTN possono esserci molti nodi intermedi, non DTN (ad esempio router IP). Essi sono in pratica trasparenti al BP (sono contenuti nelle nuvolette fra un nodo DTN e l'altro).

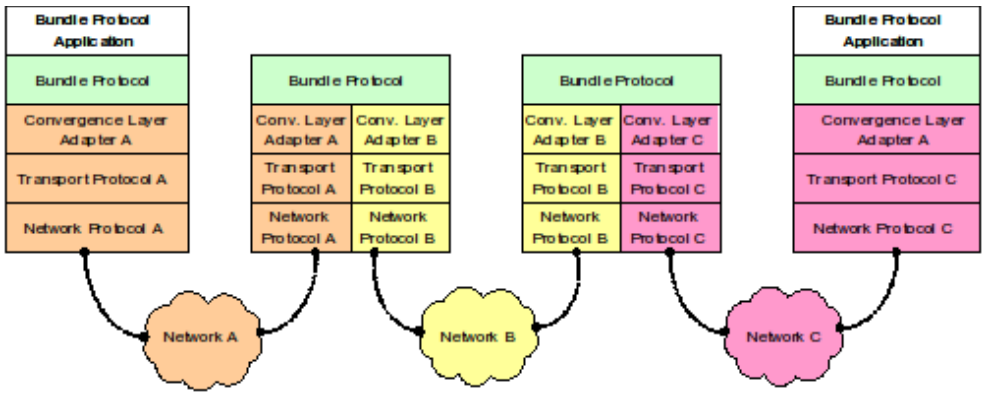

Figura 4: Architettura DTN BP

Il *Bundle Protocol Agent* (BPA) è il componente che offre i servizi del Bundle Protocol eseguendone le sue procedure.
Installando il *Bundle Protocol Agent* BPA sui nodi end-to-end e sui nodi iniziali e finali di una sottorete omogenea, il percorso end-to-end può essere diviso in diversi DTN hop.
Il requisito fondamentale per interconnettere reti eterogenee è che su ogni hop possa essere usato uno stack di protocolli differente. Nel caso in cui lo stack sia uguale, come spesso accade, l'unica differenza risiede nei diversi protocolli usati(esempio TCP,UDP,..), oppure nelle diverse versioni dello stesso protocollo. Ogni nodo DTN può usare il Convergence Layer più appropriato per l'operazione di consegna al nodo successivo, soddisfacendo quindi il requisito fondamentale per interconnettere reti eterogenee.
Poiché nelle reti Internet standard si assume una connettività continua e tempi di propagazione ridotti, i router non conservano le informazioni a lungo termine e l'informazione è persistente solo nei nodi iniziali e finali (perché recuperare un informazione dalla sorgente è conveniente piuttosto che mantenerla per tutto il percorso). Nello scenario tipico DTN, a causa dei lunghi RTT e interruzioni di canale, bisogna considerare l'assenza contemporanea dei nodi end-to-end.

Per far fronte a questo problema, DTN introduce il concetto di “custodia”. Il nodo custode si occupa di gestire l'affidabilità del messaggio, ovvero gestisce le ritrasmissioni senza dover raggiungere il mittente.
L'header di un bundle possiede un flag di richiesta per la custodia, chiamato Custody Option.
Il nodo intermedio che processa un bundle con Custody Option asserita può decidere di accettare o rifiutare la custodia.
Se la custodia è stata accettata, il nodo salva il bundle in memoria persistente, come ad esempio un hard disk locale, e informa il custode precedente di aver preso in custodia il bundle. A questo punto, il custode precedente può eliminare il bundle dalla sua memoria persistente.
Questo rende DTN molto più robusto contro le interruzioni, disconnessioni e fallimenti perché, in alcuni casi, il mittente originale non ha mai una seconda opportunità di ritrasmettere i dati; migliora inoltre le prestazioni, perché le ritrasmissioni avvengono su un percorso più breve rispetto al percorso end-to-end.
Un'altra caratteristica distintiva del Bundle Protocol sono gli “Status Report”, inviati dai nodi DTN per comunicare lo stato di trasmissione di un bundle.
Ogni Status Report si riferisce ad uno specifico bundle, ed è inviato al nodo DTN specificato nel campo “report_to” presente nell'header del bundle, il quale potrebbe non coincidere con il mittente.
Gli Status Report definiti sono:

- Bundle Reception – quando un bundle arriva in un nodo DTN
- Custody Acceptance – quando un nodo ha accettato la custodia di un bundle
- Bundle Forwarded – quando un bundle è stato inoltrato da un nodo DTN
- Bundle Deletion – quando un bundle è stato scartato o eliminato
- Bundle Delivery – quando un bundle è consegnato all'applicazione del nodo destinatario
- Acknowledged by Application – quando un bundle è stato processato da un applicazione sul nodo di destinazione. Questo generalmente implica un azione specifica da parte dell'applicazione ricevente

Gli Status Report devono essere richiesti esplicitamente dal nodo mittente settando dei bit nell'header del bundle, un nodo DTN non è obbligato ad accettare la richiesta (ad esempio per risparmiare banda).

4. Implementazioni dell'architettura DTN

Esistono diverse implementazioni open source del Bundle Protocol, le principali sono: DTN2 [DTN2_SW] e ION [ION_SF], ma ne esistono anche altre fra le quali IBR-DTN [IBR_DTN].
Dato che queste implementazioni puntano ad obiettivi differenti, presentano notevoli differenze in termini di decisioni di routing, memorizzazione dei bundle e API fornite.
La seguente sezione ha lo scopo di introdurre brevemente le caratteristiche principali di queste implementazioni. L'implementazione di riferimento della seguente tesi sarà ION, alla quale in seguito verrà dedicato una sezione più approfondita.

i. DTN2

DTN2 è l'implementazione di riferimento del Bundle Protocol introdotta dall'IRTF Delay Tolerant Networking Research Group DTNRG) [DTNRG]. Essa fornisce un framework flessibile per esperimenti relativi al DTN e può essere configurato e gestito da un'apposita

console insieme a dei file di configurazione.
È possibile estendere, tramite dei tag XML, l'implementazione di default, aggiungendo dinamicamente estensioni per il routing, memorizzazione dei bundle e Convergence Layer.
DTN2 ha due diversi moduli per la memorizzazione dei bundle: una memorizzazione basata sulla memoria non persistente (ad esempio la RAM) e una memorizzazione basata su memoria persistente (ad esempio disco fisso) che si appoggia alla libreria Berkeley DB (ovvero un DBMS open source multipiattaforma).
DTN2 contiene anche alcune applicazioni (dtnping,dtnsend,ecc) e il tool DTNperf_2[Caini] che consente di valutarne le prestazioni.

ii. ION

Interplanetary Overlay Network (ION) è un'implementazione del Bundle Protocol realizzata dalla NASA, Jet Propulsion Laboratory (JPL), finalizzata all'uso spaziale. In particolare è progettata per essere eseguita anche su hardware ridotto e sistemi operativi real-time.
ION contiene al suo interno una specie di database in cui la memorizzazione dei bundle è basata sul Simple Data Recorder (SDR), un componente già presente negli attuali mezzi spaziali.
Può essere configurato per conservare informazioni sul disco, in memoria o in entrambi.
SDR supporta un meccanismo di transazioni che garantisce integrità del database in caso di fallimento delle operazioni sul database.
È ottimizzato per funzionare considerando canali con una banda disponibile ridotta e supporta la compressione dell'header tramite la codifica “Compressed Bundle Header Encoding” (CBHE) [RFC6260].
ION include il Contact Graph Routing, un algoritmo di routing progettato appositamente per gli ambienti spaziali, dove le connessioni sono intermittenti ma la disponibilità del canale è nota in anticipo, conoscendo il moto dei corpi celesti e degli assetti spaziali coinvolti.
ION implementa il Bundle Security Protocol [RFC6257] ed anche la versione “streamlined” candidata a succedere alla prima.
Tra i Convergence Layer Adapter supportati abbiamo TCPCL, UDPCL e LTPCL.
I primi due risultano essere interoperabili con DTN2, il terzo con minime modifiche.
ION offre certe caratteristiche interessanti come connettività programmata (scheduled), non ancora implementata in DTN2. Tale funzionalità richiede tuttavia l'LTP come Convergence Layer.

Capitolo 2: Licklider Transmission Protocol (LTP)

1. Introduzione a LTP

il Licklider Transmision Protocol (LTP) è un protocollo dotato di stati progettato per garantire affidabilità basata sulle ritrasmissioni (ARQ) su canali caratterizzati da RTT estremamente elevati e/o frequenti interruzioni della connessione.
LTP è descritto in 3 RFC: le motivazioni in [RFC5325] , le specifiche del protocollo in [RFC5326] e le estensioni di sicurezza in [RFC5327]. Le ultime due sono alla base dello standard LTP per le missioni spaziali promulgato dal CCSDS [CCSDS 734.1-B-1].
Per questo motivo LTP viene usato come Convergence Layer nel Bundle Protocol sulle tratte spaziali. Esso si interfaccia sul protocollo UDP o su protocolli analoghi di uso spaziale.
Per fare fronte ai problemi evidenziati prima, riguardo al TCP, I criteri ispiratori dell'LTP sono:

1. servizio di consegna di tipo sia affidabile che non affidabile;
2. riduzione al minimo della “conversazionalità” (chattiness) del protocollo;
3. mancanza di una sessione di stabilimento connessione;
4. i dati sono organizzati in blocchi anche di grandi dimensioni; viene richiesta la conferma dell'intero blocco alla fine dell'invio (o di parti grandi di esso) anziché di singoli segmenti nel caso di servizio affidabile;
5. mancanza dei classici controlli di flusso e di congestione a retroazione; l'invio dei dati è “rate based”;
6. unidirezionalità logica;

2. Descrizione del funzionamento di LTP in assenza di perdite sui dati

Un'operazione di trasferimento inizia quando un servizio del cliente, nell'architettura DTN il BP convergence layer LTPCLA, chiede al protocollo LTP il trasferimento di un blocco di informazioni (LTP block) ad un nodo remoto. La trasmissione di un blocco viene detta sessione LTP e il protocollo può avere più sessioni in parallelo
LTP considera ogni blocco costituito da:

- una "red part", ovvero una porzione del blocco la cui trasmissione deve avvenire con affidabilità, realizzata con ritrasmissioni e acknowledgment (analogamente al TCP).
- una "green part" con semantica “best effort”, ovvero consegna senza affidabilità e quindi senza meccanismi di ritrasmissione (analogamente all'UDP).

La lunghezza di entrambe le parti può essere zero, ogni blocco può essere costituito completamente da red part ,green part o entrambe. La red part è posta all'inizio del blocco.
Il blocco, contenente uno o più bundle, quindi di dimensione variabile ma teoricamente anche molto grande (ad esempio alcuni MB), viene segmentato in diversi LTP Segment, i quali non possono essere più grandi della Maximum Segment Size (MSS) imposta dal layer sottostante.
Nel seguente paragrafo, se non specificato diversamente, si utilizzerà il termine “invio“ per indicare la trasmissione logica tra due entità LTP poste allo stello livello logico (il trasporto), sottintendendo che l'invio fisico avviene come per tutti i protocolli attraverso i vari livelli della pila dei protocolli del mittente, dall'alto (Trasporto) verso il basso (strato Fisico) e quindi viceversa nel nodo terminale. Con il termine “segmento red” verrà inteso un segmento

appartenente alla red part e con il termine “segmento green” verrà inteso un segmento appartenente alla green part.
In una trasmissione mista (formata sia da segmenti red che da segmenti green), i segmenti red non hanno una priorità maggiore rispetto ai segmenti green.
La red part indica solo che per quei segmenti è richiesta affidabilità tramite le ritrasmissioni.
I servizi che usano le API di LTP devono specificare l'identità del destinatario, la locazione dei dati da trasmettere, il numero totale di dati da trasmettere e il numero di byte che verranno trasmessi come red. Una volta iniziata la trasmissione, il servizio del client riceve una notifica con indicazione di inizio sessione. Da notare che i parametri della sessione LTP non sono negoziati, ma sono invece calcolati unilateralmente.
L'interazione tra mittente e destinatario avviene senza negoziazione, quindi alla ricezione del primo segmento del blocco, il destinatario inizia una sessione di ricezione.
Ecco i passaggi principali per il trasferimento di un blocco:

- Il mittente suddivide il blocco in segmenti e li invia logicamente al destinatario LTP, passando fisicamente per gli strati inferiori (e viceversa una volta arrivati a destinazione).
- L'ultimo segmenti red viene marcato come "End of Red Part" (EORP), indicando la fine della red-part del blocco, come un “Checkpoint” (identificato da un checkpoint serial number univoco) indicando che il ricevitore deve emettere un “Reception Report” all'arrivo del segmento.
- Viene avviato un timer per la EORP, quindi può essere ritrasmessa automaticamente se la risposta non è stata ricevuta.
- L'ultimo segmento del blocco è marcato come End of Block (EOB).

I segmenti green vengono immediatamente consegnati al servizio del cliente una volta arrivati all'LTP del destinatario, mentre per i segmenti red la consegna avviene solo dopo aver ricevuto tutto il blocco. La consegna è quindi ordinata all'interno di un blocco, anche se blocchi diversi possono essere consegnati fuori ordine.
Una volta che il destinatario riceve tutti i segmenti red della trasmissione, viene inviato un Report Segment che indica la ricezione completa.
Dato che il flusso dei dati è unidirezionale, gli acknowledgment (Reception Report) di LTP non possono essere in “piggybacking” sui segmenti dati viaggianti in direzione opposta, come accade in TCP, sono quindi trasportati in segmenti dati di segnalazione “puri”.
Il Report Segment è immediatamente trasmesso al mittente, e un timer di ritrasmissione viene avviato. Se il “Report Acknowledgment” di conferma non viene ricevuto prima del timeout, il Report Segment viene ritrasmesso.
Il mittente riceve il Report Segment, disattiva i timer per la ritrasmissione dell'EORP e invia al destinatario il segmento Report Acknowledgment. In assenza di perdite, la trasmissione della red-part è quindi terminata lato mittente e la sessione viene chiusa.
Il destinatario riceve il segmento Report Acknowledgment e disattiva i timer per la ritrasmissione del Report Segment.
La sessione di ricezione della red part è terminata e la sessione è chiusa anche lato ricevitore.
La chiusura delle sessioni implica la liberazione dei buffer della sessione lato Tx e Rx, che si rendono disponibili ad una eventuale nuova sessione. Come detto prima, possono esserci più sessioni in parallelo, a discrezione dell'utente, in relazione anche alla memoria fisica a disposizione per i buffer (spesso limitata in ambito spaziale).

3. *Calcolo dei timer*

Non essendoci negoziazioni, LTP deve calcolare accuratamente i timer di ritrasmissione. Se il tempo calcolato è troppo breve, questo costerà una ritrasmissione non necessaria.
Se il tempo calcolato è al contrario eccessivo, il tempo totale di consegna (Delivery Time) aumenta, e quindi è ritardato anche il momento in cui vengono rilasciate le risorse, cosa che può avere pesanti ripercussioni sul goodput nelle comunicazioni spaziali, a causa della mancanza di buffer liberi per accogliere nuovi blocchi.
LTP assume che il RTT sia deterministico, e che quindi possa essere stimato accuratamente in tempo reale dalle informazioni disponibili. La cosa è logica, in quanto nelle comunicazioni spaziali la componente di gran lunga principale del RTT è il tempo di propagazione, che può variare da circa 2s per la luna alle decine di minuti per Marte.
Il RTT viene calcolato, in prima approssimazione, come due volte il tempo di one-way dalla sorgente alla destinazione, aggiungendo un margine arbitrario per tenere conto delle latenze, come quella introdotta dal processore.
Il margine è un valore tipicamente di qualche secondo, un valore enorme in confronto agli standard di Internet, ma nella pratica è trascurabile paragonato ai RTT presenti nello scenario di riferimento, dove il tempo di propagazione è ,come già accennato, il fattore preponderante.
L'LTP è stato progettato per fare fronte all'intermittenza dei collegamenti spaziali, o, con terminologia DTN, è in grado di gestire i contatti “scheduled”. Questi non sono altro che le finestre temporali, note a priori, in cui è possibile un collegamento fra due nodi. Sono in altre parole delle “opportunità di trasmissione” di cui l'istante di inizio e di fine sono noti a priori e resi noti ai nodi DTN attraverso il cosiddetto “contact plan”. Se una sessione è in corso al mento della chiusura del canale (fine della finestra di trasmissione) non ha senso reinviare i segmenti di segnalazione non confermati al loro scadere previsto, in quanto il canale radio non sarebbe disponibile. Per questo motivo tutti i timer dell'LTP vengono “congelati” al momento della chiusura di una finestra e riabilitati all'apertura della finestra successiva.
Nel dettaglio,il RTT totale viene calcolato tenendo conto dei seguenti fattori:

- **Protocol Processing Time**: tempo necessario al protocollo LTP per emettere il segmento originale, ricevere il segmento originale, generare ed emettere il segmento di acknowledgment e ricevere il segmento di acknowledgment;
- **Outbound Queuing Delay**: ritardo nel mittente del segmento originale, mentre tale segmento è in coda di invio, più l'analogo ritardo del mittente dell' acknowledgment mentre tale segmento è in coda di invio;
- **Radiation Time**: tempo necessario ad inviare i bit (tramite il mezzo fisico) che compongono il segmento originale, e il tempo necessario affinché tutti i bit del segmento di acknowledgment sono stati inviati (tramite il mezzo fisico). Questa componente è tuttavia trascurabile se la bit rate è elevata;
- **Round-trip Light Time**: tempo di propagazione del segnale alla velocità della luce, in entrambe le direzioni;
- **Inbound Queuing Delay**: ritardo del ricevitore del segmento originale mentre tale segmento è nella coda di ricezione,e ritardo del ricevitore del segmento di acknowledgment mentre tale segmento è nella coda di ricezione.

4. *Ritrasmissione dei dati*

Alla ricezione di un checkpoint, il ricevitore invia al mittente un *Reception Report* indicando tutti i range contigui di segmenti red ricevuti correttamente prima del checkpoint stesso.

La perdita di uno o più segmenti red, dato che deve essere garantita la loro consegna, innesca il meccanismo di ritrasmissione.
Il Reception Report è normalmente inviato in un singolo report segment identificato da un Report Serial Number e contenente anche il Serial Number del checkpoint che lo ha generato, per confermare al mittente la sua corretta ricezione. All'interno del Reception Report Segment vi sono i Reception Claim, ovvero gli intervalli contigui di byte red ricevuti correttamente.
Per *scope* si intende l'intervallo di segmenti contigui a cui si riferisce il Report Segment che costituiscono il blocco LTP.
La dimensione massima di un Report Segment, come tutti i segmenti LTP, è vincolata dal data-link MTU (Maximum Transmission Unit); se vengono persi molti segmenti oppure il data-link MTU è molto piccolo, occorre generare Report Segment multipli suddividendo lo scope della red part.
Alla ricezione di ciascun Report Segment, il mittente ha questo comportamento

- disattiva tutti i timer del checkpoint riferito dal Report Segment ricevuto;
- invia un segmento di Report Acknowledgment;
- se i Reception Claim del Report Segment non coprono tutto lo scope, inizia la ritrasmissione dei segmenti non ancora ricevuti.
  L'ultimo segmento di un ciclo di ritrasmissione viene marcato come checkpoint e contiene il Report Serial Number del Report Segment a cui la ritrasmissione ha risposto.
  Viene avviato un timer per il checkpoint, che verrà ritrasmesso se non è arrivata alcuna risposta;
- se invece l'unione (considerata in senso insiemistico) di tutti i Reception Claim ricevuti copre tutto lo scope, viene considerata terminata la ritrasmissione dei segmenti, e di conseguenza anche l'invio della red-part. Il servizio del cliente mittente viene notificato con un'indicazione di trasmissione termina correttamente.

Alla ricezione di un segmento di Report Acknowledgment, il ricevitore disattiva i timer per il Report Segment in questione.
Alla ricezione di un checkpoint, con Report Serial Number diverso da zero, il ricevitore:

- restituisce un Report Segment che comprende i Reception Claim necessari ad una eventuale ritrasmissione ed attiva un timer (riferito al Report Segment);
- se a questo punto tutta la red-part è stata ricevuta, essa viene consegnata al servizio del client, altrimenti il ciclo di ritrasmissione continua.

La perdita di un checkpoint (o di un report segment) causa la scadenza di un timer. Quando questo avviene, il mittente ritrasmette il checkpoint (o il report segment).
I Report Segment vengono processati una sola volta, perciò quelli ridondanti (ricevuti ad esempio perché viene perso il Report Ack di conferma) vengono ignorati.

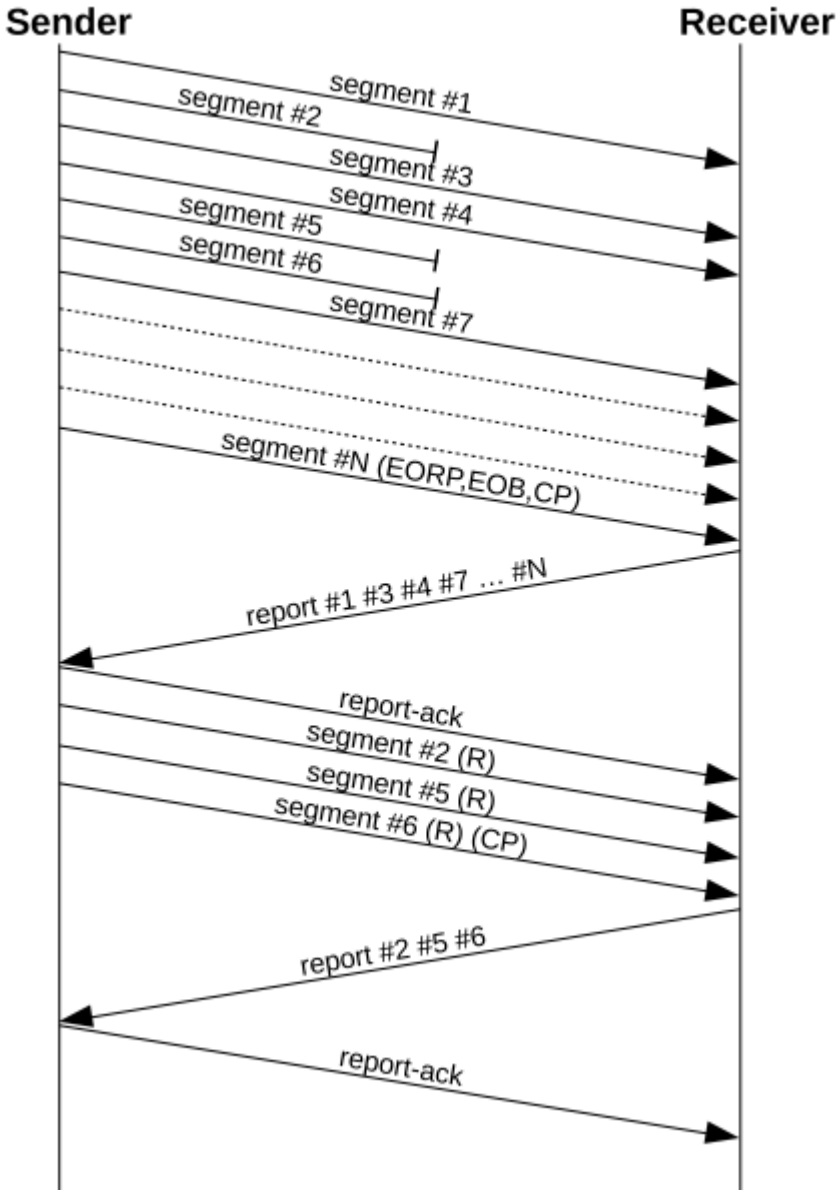

Figura 5: Esempio di sessione LTP (red) in presenza di perdite; i segmenti 2, 5 e 6 vengono persi; la loro perdita è dedotta dalle conferme mancanti all'interno del Report Segment; il Report Segment viene confermato dal Report Ack seguito dai 3 segmenti dati ritrasmessi e da un nuovo checkpoint; dato che non ci sono ulteriori perdite la sessione si chiude con il consueto scambio Report Segment Report Ack.

5. Ritrasmissioni accelerate

Le ritrasmissioni dei segmenti dati avvengono solo quando viene ricevuto un Report Segment che ne indica indirettamente la mancanza; i Report Segment sono normalmente trasmessi solo alla fine della red-part del blocco (EORP) oppure alla fine di un ciclo di ritrasmissione.
Per alcune applicazioni, potrebbe essere necessario inviare dei checkpoint intermedi.
Tali checkpoint vengono chiamati *Discretionary Checkpoint*.
Questi checkpoint sono processati esattamente allo stesso modo in cui sono processati i checkpoint standard.

6. Cancellazione di una sessione

Una sessione potrebbe essere cancellata sia dal mittente che dal destinatario per diversi motivi.
La cancellazione della sessione implica la cancellazione di tutti i segmenti in coda della sessione (cancellazione totale dei buffer), e notifica il servizio del client che la sessione è stata cancellata.
Se complessivamente non sono stati inviati o ricevuti dei segmenti, la cancellazione è immediata,altrimenti viene inviato al corrispondente un Cancel Segment.
A questo punto viene settato un timer e quindi sarà ritrasmesso automaticamente se non viene ricevuta una risposta.
Il nodo corrispondente riceve il Cancel Segment e, dopo aver eliminato tutti i segmenti in coda di output (relativi alla sessione cancellata) e notificato al client la cancellazione, invia un Cancel Acknowledgment e chiude la sessione.
Il corrispondente che riceve un Cancel Acknowledgment disattiva i timer e termina la sessione.
La perdita di un Cancel Segment o di un Cancel Acknowledgement causa la scadenza del timer. Quando questo accade il segmento viene ritrasmesso.

7. LTP in ION

L'implementazione di LTP all'interno di ION è perfettamente conforme alla [RFC5326], inoltre prevede delle aggiunte al Convergence Layer Adapter:

- I bundle passati all'LTP per la trasmissione potrebbero essere aggregati in blocchi più grandi prima della segmentazione. Controllando la dimensione dei blocchi si può controllare la quantità di traffico generato dagli acknowledgment dei blocchi ricevuti, e più in generale l'overdrive del protocollo, per uno sfruttamento più efficace della banda del canale.
- Il numero massimo di sessioni parallele costituisce una finestra di trasmissione; essa può essere utilizzata per implementare una sorta di controllo di flusso non convenzionale per i canali IPN, in quanto viene limitata la quantità di dati che devono essere salvati nei buffer di ricezione (numero max sessioni moltiplicato per la dimensione media di un blocco).

Capitolo 3: Analisi dei problemi e delle prestazioni di LTP in presenza di perdite elevate

Questo capitolo ha lo scopo di valutare gli effetti negativi prodotti dalla perdita di segmenti dati e segmenti di segnalazione (report e checkpoint) appartenenti alla red part del blocco. Lo scopo finale sarà quello di introdurre alcuni possibili miglioramenti in un caso di alte percentuali di errore sul canale, mantenendo la compatibilità con gli standard [RFC5326] e l'implementazione di ION. Poiché non sono soggetti a ritrasmissioni, i segmenti green non sono di interesse per quest'analisi. Si considerino sottintese le fasi per un invio in cui nel nodo mittente i bundle vengono aggregati in blocchi LTP , i quali vengono successivamente segmentati e i segmenti ottenuti vengono messi in coda per l'invio.
Con lo scopo di semplificare la lettura, conviene definire le seguenti metriche:

- **Delivery Time**: tempo totale di consegna di un bundle, da intendersi come tempo impiegato dal momento in cui un'applicazione del mittente (posta al livello Applicazione al di sopra del Bundle layer) decide di trasferire un'informazione (ad esempio un file o stringhe di testo) e il momento in cui tale informazione viene consegnata alla corrispondente applicazione del destinatario.
- **Penalty Time**: tempo aggiuntivo al Delivery Time, causato dalla perdita di uno o più segmenti.
- **Import Session Life Time**: tempo totale in cui la Import Session resta aperta, ovvero il tempo trascorso tra la ricezione del primo segmento e la ricezione dell'ultimo segmento della sessione.
- **Export Session Life Time**: tempo totale in cui la Export Session resta aperta, ovvero il tempo trascorso tra l'invio del primo segmento e l'invio dell'ultimo segmento della sessione.
- **Trasmissione Ideale**: trasmissione in cui tutti i segmenti vengono ricevuti senza perdite. In questa situazione,non essendoci perdite e ritrasmissioni, il Penalty Time è nullo.
  La trasmissione ideale, che ci serve come riferimento, è rappresentata dalla seguente interazione:

  1. il mittente invia tutti i segmenti (l'ultimo con l'indicazione di End of Red Part ,End of Block e Checkpoint)
  2. il destinatario riceve correttamente tutti i segmenti e invia un Report Segment che conferma i segmenti ricevuti
  3. il mittente riceve il Report Segment , invia un Report Ack e Termina la Export Session
  4. il destinatario riceve il Report Ack e termina la Import Session

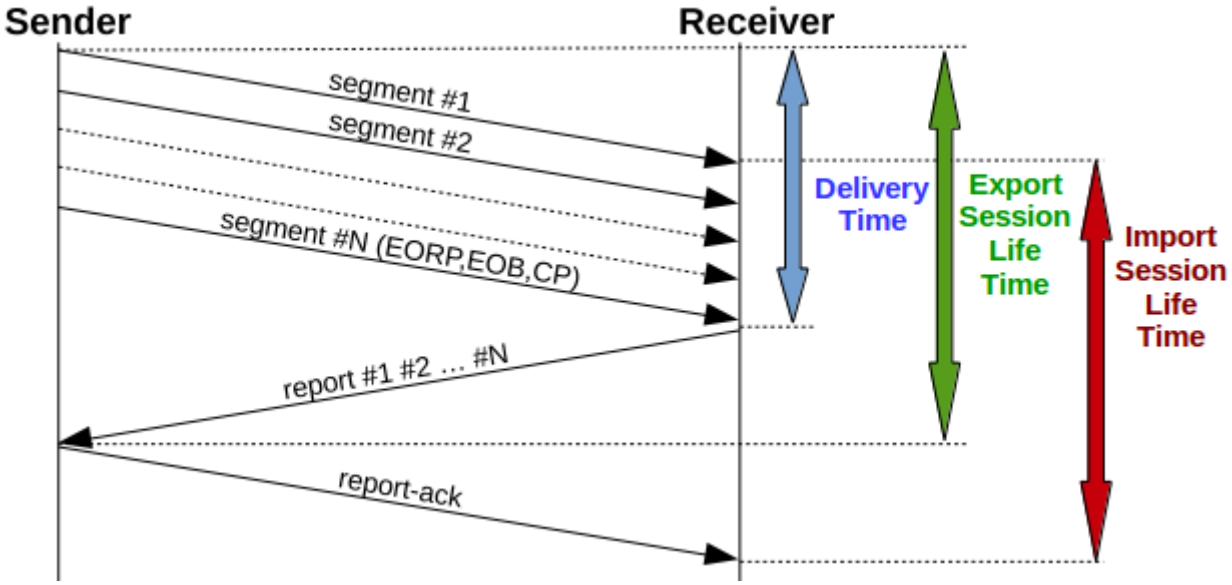

Figura 6:Trasmissione ideale: Delivery Time,Export Session Life Time(Sender) e Import Session Life Time (Receiver)

La Trasmissione Ideale serve come scenario da paragonare alle situazioni reali con perdite che esamineremo dopo.
Essendo nelle applicazioni spaziali di norma il RTT molto maggiore della somma dei tempi di invio dei segmenti e del ritardo introdotto dal processore, il Delivery Time viene a coincidere nel caso ideale con un mezzo RTT, cioè in pratica con il tempo di propagazione, che ne rappresenta ovviamente un minimo teorico assoluto, non potendo essere superata la velocità della luce. Lato mittente nel caso ideale si viene a conoscenza della corretta ricezione dopo un RTT, che è ancora una volta il minimo teorico nel caso in cui si voglia l'affidabilità. La durata della sessione è di un RTT, ma lato mittente si apre subito e si chiude all'arrivo del Report Segment. Lato destinatario si apre all'arrivo del primo segmento e si chiude all'arrivo del Report Ack. In caso di perdite si hanno ovviamente delle penalità. Nelle prossime considerazioni, il Penalty Time verrà considerato semplicemente come multiplo del RTT, trascurando la somma dei tempi aggiuntivi menzionati in precedenza. Seppur LTP permette di mitigare il problema dell'inutilizzo del canale tramite l'impiego di diverse sessioni, il Delivery Time dei singoli bundle potrebbe essere troppo elevato rispetto alla Trasmissione Ideale.
Il Life Time delle sessioni apparentemente non influenza il Delivery Time. In realtà questo non è sempre vero: per implementare il controllo di congestione non basato sulle negoziazioni, in ION il numero di Import Session ed Export Session parallele è limitato a dei valori noti, settati nei file di configurazione.
Se il mittente ha esaurito il numero di Export Session parallele e vuole inviare un bundle, la sessione verrà messa in attesa e verrà inviata riattivata non appena termina una Export Session parallela.
Se il destinatario ha esaurito il numero di Import Session parallele, il mittente non può conoscere la sua situazione e quindi procede ad inviare regolarmente. In questo caso il destinatario scarterà tutti i segmenti ricevuti. Questo rappresenta il caso peggiore sia in termini di risorse e sia in termini di Delivery Time. I problemi legati al numero massimo di sessioni non possono essere superati, in quanto il numero massimo di sessioni parallele dipende dalla memoria a disposizione sul nodo mittente e su quello ricevente (non possono essere in volo più blocchi, o meglio più parti red, di quelli memorizzabili da ambo le parti)

L'analisi punterà a comprendere se esistono situazioni in cui le sessioni durano un tempo maggiore rispetto a quanto dovrebbero, per poi avere delle indicazioni su possibili miglioramenti del protocollo.
I prossimi paragrafi evidenzieranno che la Export Session Life Time, anche in situazioni di perdite di segmenti, è circa uguale alla Import Session Life Time, ad eccezione del caso in cui viene perso il Report Ack finale.
Lo scopo principale da raggiungere per ottimizzare il protocollo, sarà quello di minimizzare il Delivery Time, Penalty Time e Import/Export Session Life Time .

1. Perdita di segmenti dati

Si consideri la seguente trasmissione:

1. il mittente invia tutti i segmenti (l'ultimo con l'indicazione di End of Red Part ,End of Block e Checkpoint)
2. vengono persi uno o più segmenti dati (diverso dall'ultimo)
3. il destinatario invia un Report Segment contenente il Reception Claim dei segmenti persi
4. il mittente riceve il Report Segment e invia un Report Ack seguito dai segmenti mancanti (l'ultimo con l'indicazione di checkpoint)
5. il destinatario riceve correttamente i segmenti ritrasmessi e invia un Report Segment che conferma la ricezione dei segmenti ritrasmessi
6. il mittente riceve il Report Segment, invia un Report Ack e termina la Export Session
7. il destinatario riceve il Report Ack e termina la Import Session

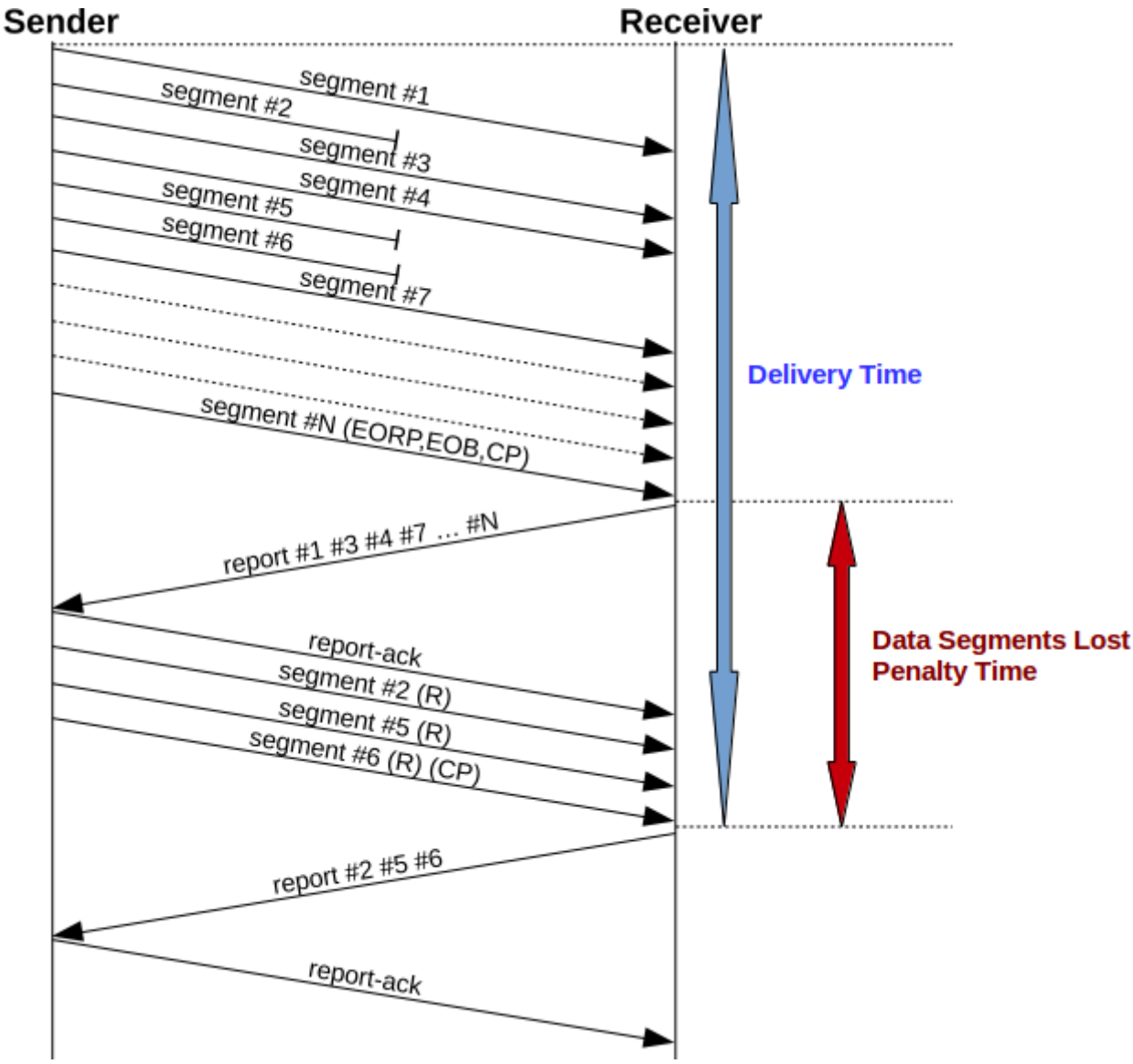

Figura 7 Trasmissione con perdita di segmenti dati

Rispetto alla Trasmissione Ideale si può notare facilmente che il Delivery Time è in ritardo di circa un RTT, ovvero il Penalty Time è pari a circa un RTT. Si noti che nella figura sopra come nelle precedenti, per motivi di leggibilità, è stato molto dilatato il tempo occorrente all'invio dei segmenti, di norma molto più piccolo del RTT nelle comunicazioni interplanetarie.
Se si considera una variante di questo scenario, in cui vengono persi uno o più segmenti anche nel ciclo di ritrasmissione (4), le ritrasmissioni causeranno un altro Penalty Time aggiuntivo pari ad un RTT.
Il seguente risultato può essere riassunto come:

> "sotto l'ipotesi di RTT>>del tempo di invio del blocco, per ogni ciclo di invio o ritrasmissione, la perdita di uno o più segmenti causa un Penalty Time pari ad un RTT"

2. *Perdita del solo segmento dati contenente il checkpoint*

Si consideri la seguente trasmissione:

1. il mittente invia tutti i segmenti al destinatario (l'ultimo con l'indicazione di End of Red Part ,End of Block e checkpoint)
2. viene perso il checkpoint e il destinatario riceve correttamente tutti gli altri segmenti
3. scade il timer del mittente che ritrasmette il checkpoint
4. il destinatario riceve correttamente il checkpoint e invia un Report Segment che conferma tutto lo “scope” del checkpoint, ovvero l'intero blocco
5. il mittente riceve il Report Segment, invia un Report Ack e termina la sessione
6. il destinatario riceve il Report Ack e termina la sessione

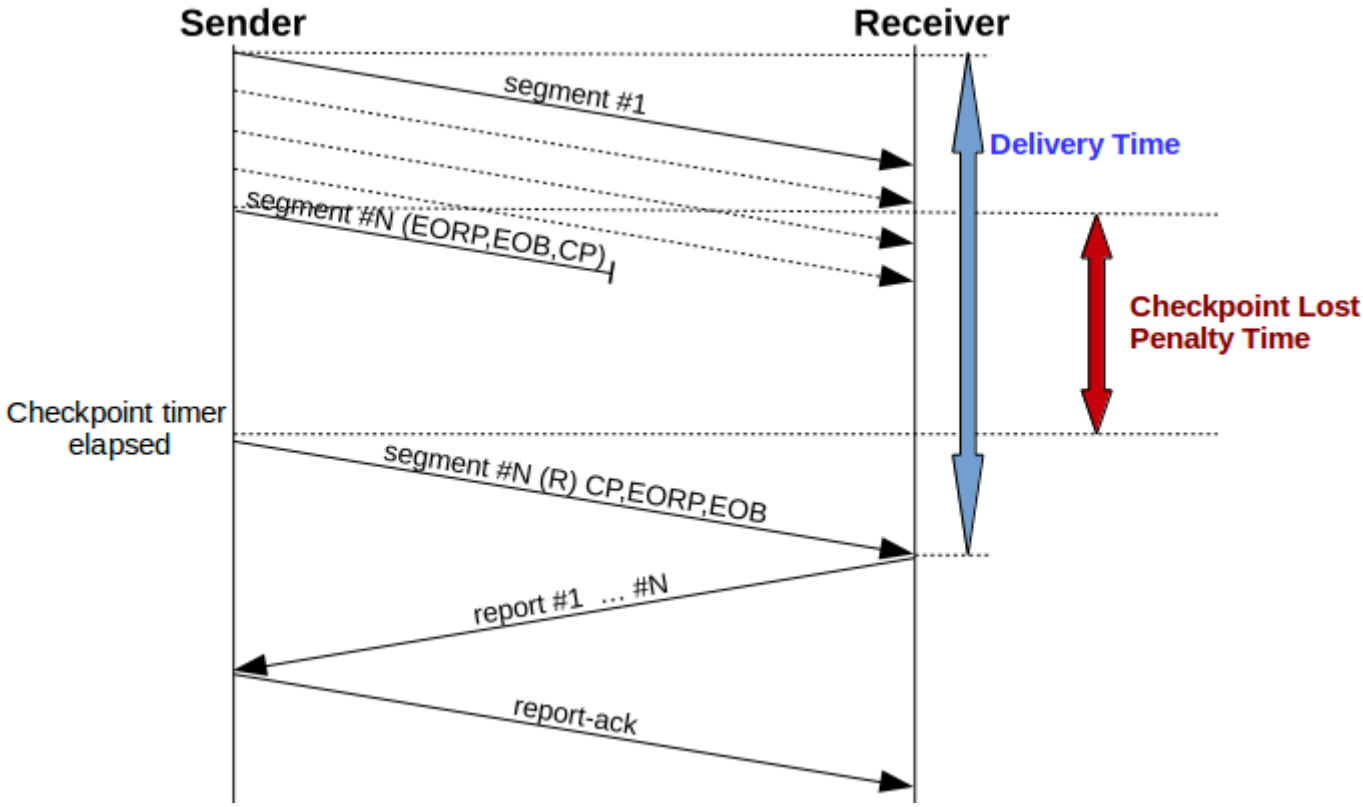

Figura 8: Trasmissione con perdita del segmento contenente il Mandatory Checkpoint

Rispetto alla Trasmissione Ideale si può notare facilmente che il Delivery Time è in ritardo di un RTT, ovvero il Penalty Time è pari al RTT.
Da notare che in questo scenario si sta analizzando la situazione in cui viene perso il Mandatory Checkpoint (checkpoint posto nell'ultimo segmento del ciclo di trasmissione).
Per i Discretionary Checkpoint (checkpoint intermedi) la situazione varia in base all'implementazione considerata. La [RFC5326] infatti prevede due tipi di comportamenti in caso di checkpoint intermedi:

1. vengono ritrasmessi solo i segmenti persi precedenti al checkpoint considerato
2. vengono ritrasmessi tutti i segmenti persi fino al momento in cui il checkpoint viene ricevuto.

In entrambi i casi il Penalty Time è sicuramente inferiore ad un RTT e teoricamente, se il ciclo

di invio presentasse moltissimi segmenti (una situazione in verità pressoché impossibile in ambito spaziale come vedremo fra poco), potrebbe essere anche nullo nel caso in cui il Report Segment causato dal checkpoint intermedio arrivasse prima che il ciclo di invio sia terminato.

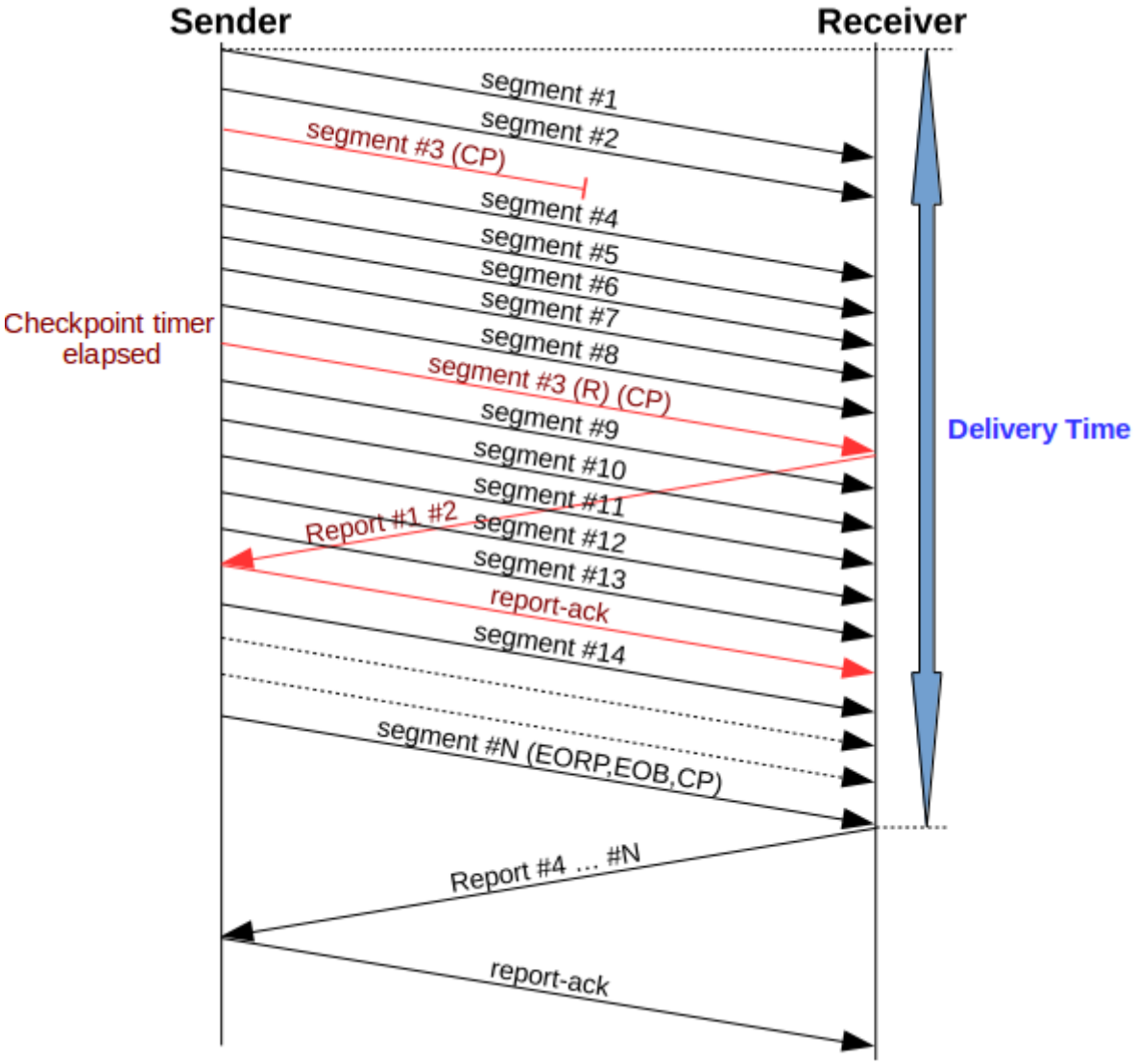

Figura 9: Trasmissione in cui viene perso il segmento dati #3 contenente, oltre ai dati, un Discretionary Checkpoint. Scaduto il timer associato al segmento perso, tale segmento viene ritrasmesso. Se il report generato arriva prima della fine del ciclo di invio, la ritrasmissione avviene senza Penalty Time

La differenza tra i due possibili comportamenti in caso di perdita di un Discretionary Checkpoint,è che nel primo caso si ha una situazione analoga alla figura 9 anche perdendo i segmenti dati #1 e #2. Nel secondo caso si avrebbe una situazione analoga anche perdendo anche uno o più segmenti compresi tra il #9 e il #13.
Ad ogni modo,la situazione in cui le ritrasmissioni avvengono entro il ciclo di invio è molto improbabile. Un esempio utilizzando dati reali è il seguente: Mars Reconnaissance Orbiter comunica con un Earth Control Center a 4Mbit/s , ovvero circa 488KB/s. Il RTT è assunto pari al minimo cioè a circa 360s (il RTT fra Terra e Marte, al contrario di quello fra Terra e Luna, è variabile nell'intervallo [6m-44m], dipendendo dalla posizione reciproca dei due

pianeti). Per avere una situazione analoga a quella mostrata in figura 9, la dimensione del blocco LTP dovrebbe essere maggiore di 488KB/s *2* 360s, ovvero circa 343,32MB. Viceversa, assumendo un più ragionevole valore di 1 MB per un bundle di grandi dimensioni (NASA configura la dimensione dei bundle in modo che la trasmissione non superi la granularità DTN, ossia 1 s), esso richiederebbe esattamente 2 s, ovvero circa lo 0.5% (cinque per mille) del RTT minimo.
Nella maggior parte dei casi di interesse,il ciclo di invio ha una durata trascurabile rispetto al RTT. In questi casi,la perdita di un Discretionary Checkpoint è assimilabile al caso di perdita di uno o più segmenti dati, analizzato nel precedente paragrafo.
I seguenti risultati possono essere riassunti come:

> "la perdita di un Mandatory Checkpoint causa un Penalty Time pari al RTT. La perdita di un Discretionary Checkpoint è analoga alla perdita di uno o più segmenti dati ordinari, che causa un Penalty Time di un RTT indipendentemente da quanti segmenti dati sono stati persi"

Per semplicità di lettura,le successive considerazioni verranno fatte analizzando il caso di perdita del Mandatory Checkpoint e il termine "Mandatory" verrà omesso. I Discretionary Checkpoint faranno quindi parte della categoria più generale dei segmenti dati ordinari.

3. Perdita del checkpoint e di altri segmenti dati

Si consideri la seguente trasmissione:

1. il mittente invia tutti i segmenti (l'ultimo con l'indicazione di End of Red Part ,End of Block e Checkpoint)
2. vengono persi uno o più segmenti dati
3. viene perso il checkpoint
4. scade il timer del mittente che ritrasmette il checkpoint
5. il destinatario riceve correttamente il checkpoint e invia un Report Segment che conferma i segmenti ricevuti correttamente
6. il mittente riceve il Report Segment e invia un Report Ack seguito dai segmenti mancanti (l'ultimo con l'indicazione di checkpoint)
7. il destinatario riceve correttamente i segmenti ritrasmessi e invia un Report Segment che conferma la ricezione dei segmenti ritrasmessi
8. il mittente riceve il Report Segment, invia un Report Ack e termina la Export Session
   il destinatario riceve il Report Ack e termina la Import Session

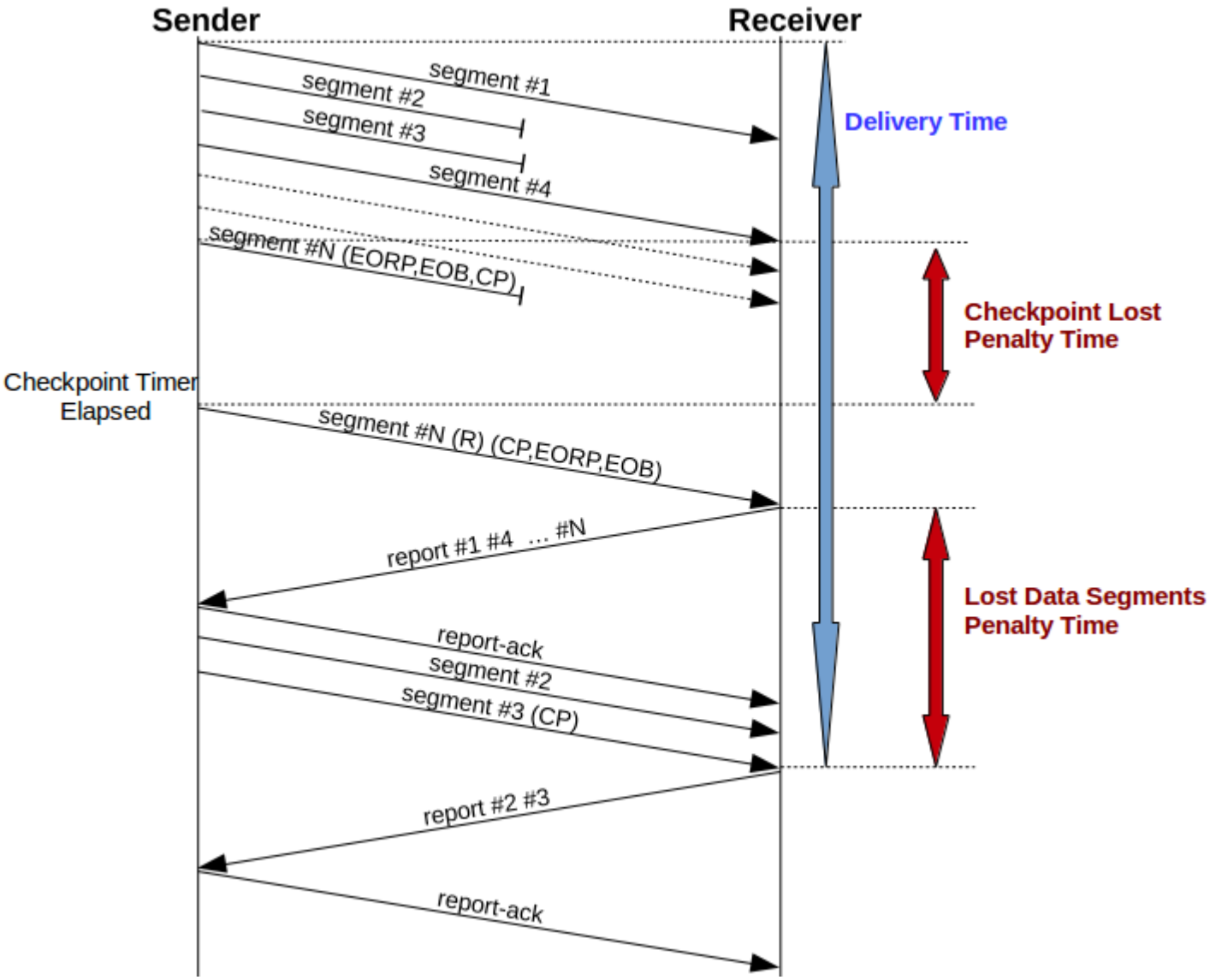

Figura 10: Trasmissione con perdite miste

Questo scenario evidenzia i problemi della combinazione dei due scenari precedenti.
Come è visibile graficamente, il Penalty Time è sempre di 1 RTT indipendentemente dal numero di segmenti dati persi; tuttavia, se viene perso anche il checkpoint, si ha una penalizzazione aggiuntiva di un altro RTT. Le due penalizzazioni si sommano, impattando negativamente sul Delivery Time.
Risulta chiaro che perdere il primo segmento dati o il primo checkpoint è equivalente. Tuttavia, una volta perso un segmento dati non ho ulteriori penalizzazioni se ne viene perso un secondo, mentre se viene perso anche il checkpoint la penalizzazione si somma.

4. Perdita di un Report Segment

Si consideri la seguente trasmissione:

1. il mittente invia tutti i segmenti al destinatario (l'ultimo con l'indicazione di End of Red Part ,End of Block e checkpoint)
2. il destinatario riceve tutti i segmenti correttamente e invia un Report Segment che li

conferma
3. il report segment viene perso
4. scade il timer del mittente che ritrasmette il checkpoint
5. il destinatario riceve correttamente il checkpoint, ma viene scartato.
6. Scade il timer del Report Segment e il destinatario invia un Report Segment che conferma tutto lo scope
7. il mittente riceve il Report Segment, invia un Report Ack e termina la sessione
8. il destinatario riceve il Report Ack e termina la sessione

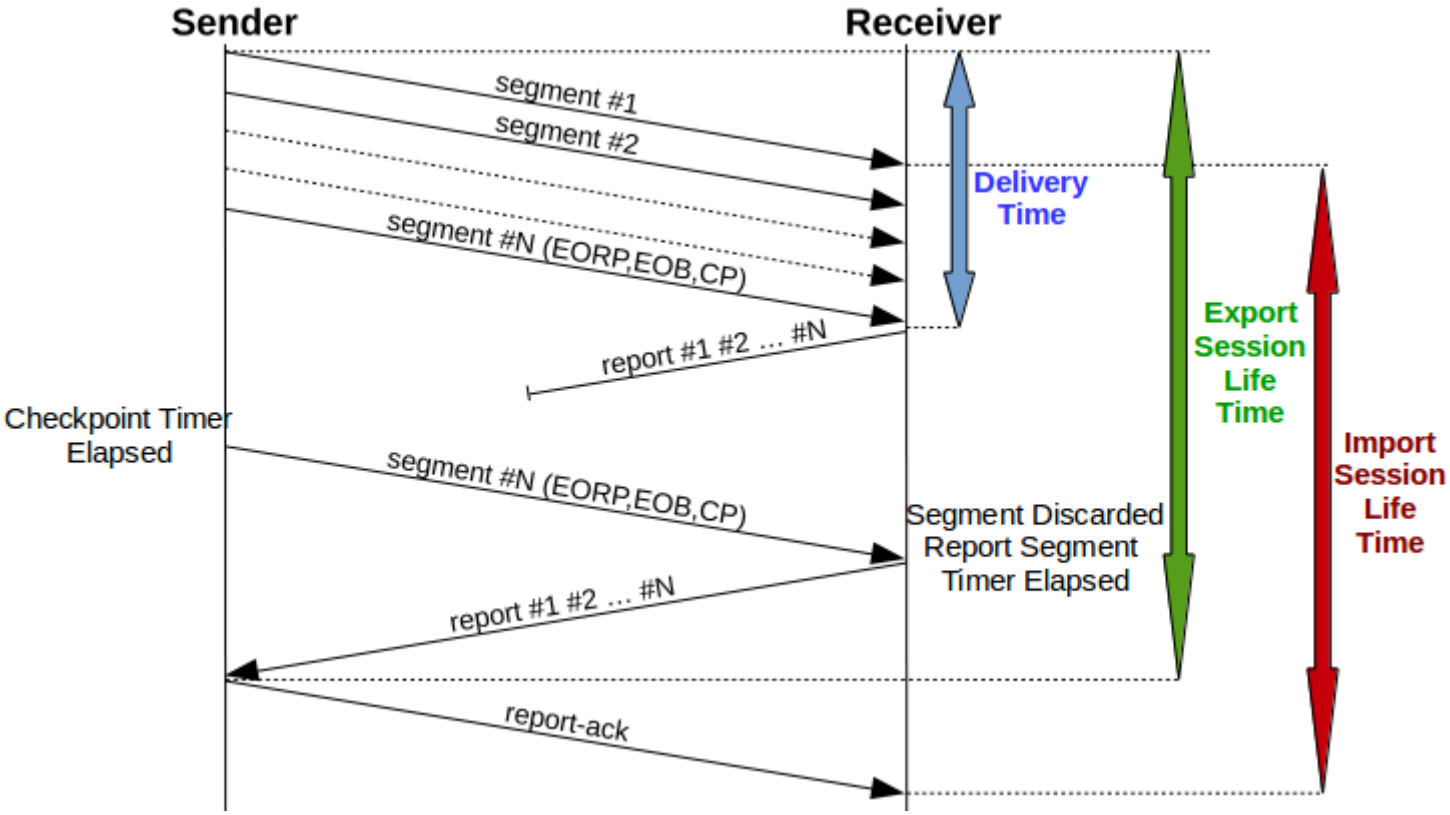

Figura 11: Trasmissione con perdita di Report Segment

Questo scenario sembra essere molto simile a quello in cui viene perso il checkpoint. In questo caso però il blocco viene consegnato al layer superiore con lo stesso Delivery Time della Trasmissione Ideale, ovvero senza Penalty Time. Gli effetti negativi sono nei Life Time delle sessioni.
Il seguente risultato può essere espresso come

> "la perdita di un Report Segment causa un ritardo di chiusura di entrambe le sessioni pari a un RTT rispetto al caso di Trasmissione Ideale"

5. *Perdita di un Report Ack finale*

Le Import Session terminano dopo aver inviato il Report Ack di conferma al Report Segment finale. Le Export Session terminano non appena ricevono tale Report Ack.
Si consideri la seguente trasmissione:

1. il mittente invia tutti i segmenti (l'ultimo con l'indicazione di End of Red Part ,End of Block e Checkpoint)

2. il destinatario riceve correttamente tutti i segmenti e invia un Report Segment che li conferma tutti
3. il mittente riceve il Report Segment e invia un Report Ack e termina la Export Session
4. il Report Ack viene perso
5. scade il timer associato alla ricezione del Report Ack
6. il destinatario ritrasmette il Report Segment
7. il mittente riceve il Report Segment ma, avendo terminato la Export Session, scarta il segmento
8. si ritorna al punto 5 finché non si esauriscono i tentativi, successivamente il destinatario termina la Import Session

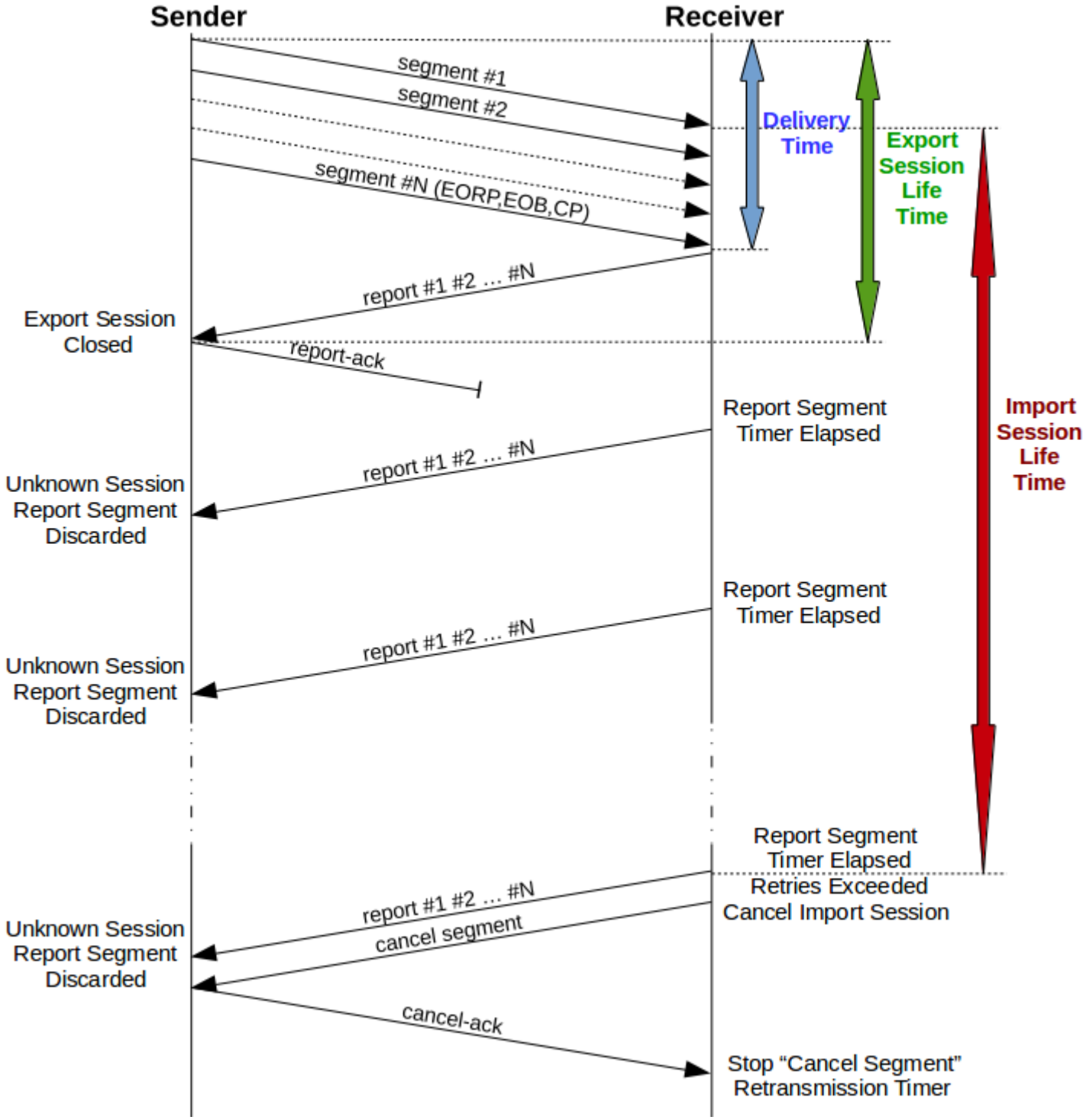

Figura 12: Trasmissione con perdita del Report Ack finale

Questo scenario rappresenta il caso peggiore. È possibile notare dalla figura 12 che il Import Session Life Time è molto maggiore sia del Delivery Time che del Export Session Life Time. Da notare che la perdita dei Cancel Segment o Cancel Ack non causa problemi in quanto la Import Session viene chiusa prima. Il seguente risultato può essere riassunto come:

> “la perdita del Report Ack finale causa un prolungamento della Import Session Life Time pari a RTT*maxTimeouts rispetto alla Trasmissione Ideale”

Capitolo 4: Soluzioni concettuali proposte

Riassumendo le considerazioni del capitolo precedente si ha che rispetto alla Trasmissione Ideale (cioè senza perdite):

1. per ogni ciclo di invio o ritrasmissione, la perdita di uno o più segmenti dati causa un Time Penalty pari ad un RTT;
2. la perdita di ogni checkpoint (Mandatory Checkpoint) causa un Penalty Time pari a un RTT;
3. la perdita di un Report Segment causa un ritardo di chiusura di entrambe le sessioni pari ad un RTT;
4. la perdita del Report Ack finale causa un prolungamento della Import Session Life Time pari a RTT*maxTimeouts;

Per limitare le penalità ed avere anche una costanza dei Delivery Time (importantissima ad esempio nelle trasmissioni video), una possibile soluzione è quella di utilizzare uno schema di codifica a livello di pacchetto (Packet Layer Coding o Erasure Coding), consistente nella trasmissione di segmenti di ridondanza, in grado di recuperare i segmenti [Apollonio]. Questa soluzione è particolarmente promettente per migliorare la robustezza di trasmissioni di green part, in quanto permette di ridurre drasticamente le perdite fino a praticamente annullarle ammesso che lo schema di codifica di pacchetto sia opportunamente configurato (es., numero di segmenti di ridondanza). Si deve tuttavia notare che tale soluzione non può garantire l'assoluta affidabilità del trasferimento dati poiché un eccessivo numero di segmenti persi (informazione e ridondanza) non può essere recuperato della codifica di pacchetto. In particolare, assumendo una codifica di pacchetto ideale (o perfetta), partendo da un blocco costituito da K segmenti LTP di informazione e generando N-K segmenti di ridondanza, il ricevitore deve almeno ricevere K segmenti (siano essi di informazione o ridondanza), per poter ricostruire correttamente il blocco iniziale; diversamente si osserveranno delle perdite di segmenti LTP. Per rendere la trasmissione completamente affidabile, sarebbe pertanto consigliabile abbinare l'uso di erasure coding con la red part. La differenza è che in caso di erasure coding il numero delle perdite dovrebbe risultare drasticamente ridotto, ma non sempre annullato. Peraltro, le minori perdite si ottengono pagando un costo in termini di maggiore complessità, riduzione della banda a disposizione a causa della trasmissione dei segmenti di ridondanza e di un aumento significativo del RTT a causa degli ulteriori ritardi introdotti dai processi di codifica e decodifica. Ad esempio per codificare un blocco (ad esempio di 4200 segmenti), occorre aspettare un certo tempo perché il blocco stesso si riempia con i segmenti di più blocchi LTP. Se essi non arrivano occorre predisporre un timeout, ma nel caso esso scatti la ridondanza introdotta sarà più alta di quella nominale in quanto il blocco è solo parzialmente pieno, ecc.
Un altro approccio, più semplice, è quello di ridondare alcuni segmenti particolarmente critici semplicemente trasmettendoli più volte, in pratica applicando un semplice codice a ripetizione. In questa tesi, si è preferito non considerare la replica di segmenti dati poiché la loro perdita introduce un Penalty Time uguale a 1 RTT, indipendentemente da quanti segmenti sono stati persi nel corso del ciclo di invio. Replicare i segmenti dati, inoltre, produce nel caso di una replicazione per N una corrispondente moltiplicazione per N dei segmenti. E' noto che con ridondanze del genere un erasure coding ben progettato può dare prestazioni decisamente superiori (se ad esempio moltiplico per due, è sufficiente avere la perdita di entrambi i segmenti per avere un ciclo di ritrasmissioni; nel caso dei FEC studiati in [Apollonio], invece

occorre la perdita di metà del totale dei segmenti di un blocco di codifica, un evento evidentemente molto più improbabile. La soluzione qui considerata invece parte dalla constatazione che la perdita di segmenti di segnalazione sia più grave della perdita di segmenti dati. Per questo motivo si ritiene che si possano migliorare le prestazioni replicando alcuni di essi, come i checkpoint, senza di fatto introdurre un grande aumento del numero totale dei segmenti trasmessi neppure nel caso ideale (dove le ritrasmissioni non occorrerebbero), dato che i segmenti di segnalazione sono pochi e solitamente molto più piccoli dei segmenti dati.

1. Cenni sui canali radio e sui canali ottici

Per comunicazioni in radio frequenza (RF), nello spazio libero (di norma nel caso delle comunicazioni spaziali), e senza considerare attenuazioni supplementari (es. attenuazioni atmosferiche dovute a pioggia se si lavora a frequenze superiori a 20 GHz) e disturbi vari (rumore termico, vento solare, etc.), si può supporre che gli errori (bit errati) sul canale siano indipendenti, per cui un canale AWGN (Additive White Gaussian Noise) è ragionevole. In questo caso, errori indipendenti a livello di bit danno luogo a perdite di segmento (erasures) indipendenti. Al contrario, per le Laser Communication (Lasercom) bisogna supporre che gli errori sul canale siano correlati, in quanto in gran parte dovuti a scintillazioni causate da turbolenze atmosferiche. In questo caso, il numero consecutivo di bit errati corrisponde ad un periodo di fading, la cui durata è solitamente superiore al tempo di trasmissione di un intero segmento LTP, causando quindi la perdita di numerosi segmenti LTP consecutivi. In questo caso, pertanto, le perdite di segmenti LTP sarebbero da considerare anch'esse correlate. Gli errori (bit errati), in realtà possono risultare correlati anche nel caso RF se si impiegano al livello fisico dei codici FEC (se fallisce la decodifica di un blocco di bit ci saranno molti errori all'interno del medesimo blocco, in caso contrario, non ci sarà nessun errore nel blocco). Viceversa, in caso di bit errati correlati “naturalmente”, essi potrebbero essere resi incorrelati dalla presenza di un interleaver di durata superiore al tempo di coerenza del canale. Ciononostante, deve essere notato che escludendo il caso di comunicazioni dati realizzate in bande di frequenza elevate (maggiori di 20 Ghz), la correlazione di bit errati è limitata a poche parole di codice di livello fisico, causando pertanto la perdita di un solo segmento LTP. Pertanto, anche in questo caso le perdite di segmenti LTP saranno indipendenti. Per completezza , va anche detto che la corruzione di segmenti è rilevata direttamente a livello fisico dove la specifica codifica di canale congiuntamente all'implementazione di codici per la rilevazione di errore (es. CRC) permettono di verificare la correttezza di un dato segmento ed eventualmente scartarlo. Per avere un pacchetto (o segmento) perso è sufficiente che vi sia almeno un bit errato dopo la codifica di canale a livello fisico. Data p la probabilità di corretta ricezione e $q=1-p$ la probabilità di errore per bit, la probabilità di avere un pacchetto di N bit senza errori, nell'ipotesi di indipendenza (cioè di canale non correlato), è data da p^N . Quella di avere almeno un errore (cioè la perdita del pacchetto), è la sua complementare ovvero $1-p^N$. Da questa formula si deduce quindi che la probabilità di perdita di un pacchetto, sempre nella ipotesi di canale non correlato, dipende dalla sua lunghezza, e che la perdita di due blocchi successivi è indipendente. Queste sono le ipotesi che vengono fatte nell'implementazione LTP di ION per calcolare in modo automatico il numero massimo di ritrasmissioni a partire dalla probabilità d'errore per bit, sotto l'ulteriore pesante ipotesi di avere segmenti di uguale dimensione. Questa ipotesi vale per i segmenti dati (tranne l'ultimo del blocco) la cui dimensione è impostata dall'utente nei file di configurazione, ma non per moltissimi segmenti di segnalazione,che generalmente sono molto più corti. Per essi, la formula sopra con N pari alla dimensione massima di un segmento porta

ad una forte sovrastima della reale probabilità di perdita. D'altro canto, la ipotesi di avere bit totalmente indipendenti è molto forte anch'essa, e porta a sottostimare o sovrastimare la reale probabilità di perdita a seconda dei casi. Più precisamente, nel caso di comunicazione laser o in presenza di eventi atmosferici in comunicazioni RF in bande di frequenza elevate, gli errori sono concentrati, pertanto il comportamento del canale a livello di pacchetto si può rappresentare come composto di due stati, “good” e “bad”. In tali condizioni, non è corretto considerare esclusivamente la probabilità d'errore media, in quanto questa metrica non cattura l'alternarsi degli stati good e bad. In particolare, lo stato good è solitamente caratterizzato da una probabilità di perdita di pacchetto molto bassa o persino nulla, mentre lo stato bad è contraddistinto da una probabilità di perdita dei pacchetti molto elevata, spesso unitaria. Qui non si vuole andare oltre per non tentare in un ambito specialistico che non compete a questa tesi. Le considerazioni sopra servono solo a mostrare le complessità del problema, spesso sottostimata dai non addetti ai lavori, e a giustificare la scelta di due diverse tecniche di ridondanza nei paragrafi successivi.

2. Primo miglioramento: ridondanza dei segmenti di segnalazione

Per comunicazioni Radio Frequencies (RF), si può supporre che gli errori sul canale siano indipendenti, mentre per le Laser Communication (Lasercom) bisogna supporre che gli errori sul canale siano correlati. Gli approcci di ridondanza proposti saranno sempre basati sulla ripetizione (N volte) di alcuni segmenti di segnalazione, in particolare dei checkpoint. Le ripetizioni considerate sono di due tipi, come mostrato sotto per il caso dei checkpoint:

- Burst di N1 checkpoint, ovvero N1 invii replicati e consecutivi dei checkpoint, tecnica adatta ai canali con errori indipendenti
- Spread di N2 checkpoint, ovvero invio di N2 checkpoint equidistanti temporalmente all'interno di un intervallo di RTT, tecnica adatta ai canali con tempi di correlazione elevati (la speranza è di avere probabilità di perdite indipendenti per invii successivi, cosa che occorre se il tempo di correlazione del canale è minore di $\frac{RTT}{N2}$)

Le due tecniche possono essere combinate senza interferire tra di loro, risultando in N2 burst di N1 segmenti di segnalazione ripetuti, equidistanti temporalmente di $\frac{RTT}{N2}$.

Le seguenti soluzioni proposte si riferiranno ai checkpoint. È semplice verificare che tali approcci risolutivi basati sulla ridondanza, hanno una validità più generale, ovvero si possono applicare a tutti i segmenti di controllo. In particolare, il caso dei burst, può essere applicato,oltre che ai Checkpoint, anche ai Report Segment, Report Ack, Cancel Segment e Cancel Ack Segment.
Il caso degli spread, può essere applicato, oltre che ai Checkpoint, anche a tutti i segmenti di segnalazione soggetti a reinvio dopo un timeout, ovvero i Report Segment e i Cancel Segment. I Report Ack e Cancel Ack sono esclusi perché non sono segmenti soggetti a reinvio.

i. Burst di checkpoint

Definisco un valore CPBURST (prima N1). Per ogni invio di checkpoint previsto normalmente dal protocollo, ne verranno inviati altri CPBURST-1. Se il canale è indipendente, la probabilità di perdere tutto il burst si ottiene elevando la probabilità di perdita di un segmento al valore di CPBURST.

Probabilità di perdita di un segmento	CPBURST	Probabilità di perdita del burst
3%	3	0.0027 %
15%	3	0.3375 %
30%	3	2.7 %
30%	4	0.81 %
50%	4	6.25 %
50%	7	0.78 %

La seguente tabella fornisce degli esempi numerici che dimostrano l'efficacia dei burst per i canali senza correlazione, pur scegliendo dei valori di CPBURST ridotti. Vale la pena ricordare che le perdite massime considerate come target dipendono dalla specifica missione spaziale presa a riferimento e delle specifiche tecnologie utilizzate a livello fisico (es. codifica di canale). La NASA ritiene ragionevole considerare perdite massime del 3% per canali RF incorrelati, corrispondenti quindi alla prima riga della tabella. In ogni caso il corretto funzionamento del protocollo LTP e le soluzioni proposte non possono dipendere dal verificarsi di un particolare scenario applicativo, per quanto importante, ma devono essere più generali da cui l'importanza anche delle altre righe della tabella, molto più pessimistiche.

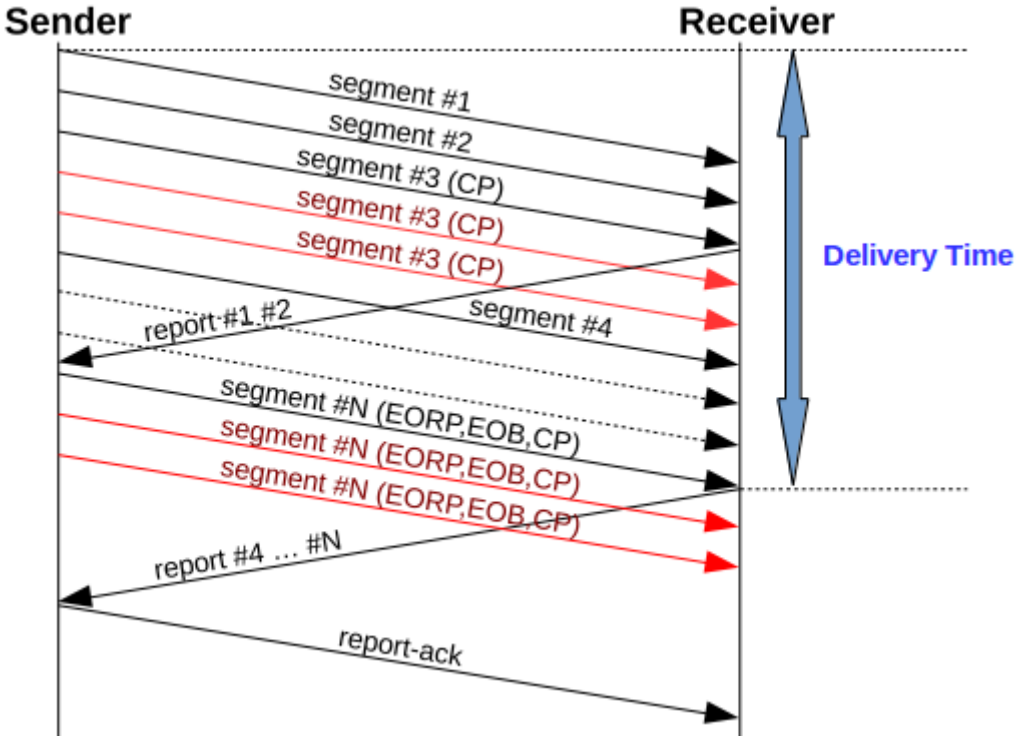

Figura 13: Trasmissione Ideale con CPBURST=3. I segmenti rossi sono i checkpoint aggiunti all'originale per formare un burst di 3 elementi; se l'originale viene ricevuto vengono scartati dal Receiver poiché già processati.

È consigliabile scegliere un valore di CPBURST che mantenga una probabilità di perdita dell'intero burst molto bassa (anche inferiore a 1%) in modo da ipotizzare ragionevolmente che almeno un segmento del burst riesca a raggiungere il destinatario. Così facendo si crea

una situazione simile a quella in cui nessun checkpoint viene perso perché i segmenti aggiuntivi vengono scartati come comportamento di default.
La seguente figura, mostra come il burst permetta di azzerare il Penalty Time dovuto alla perdita del Checkpoint rispetto al caso di perdite miste schematizzato in figura 10.

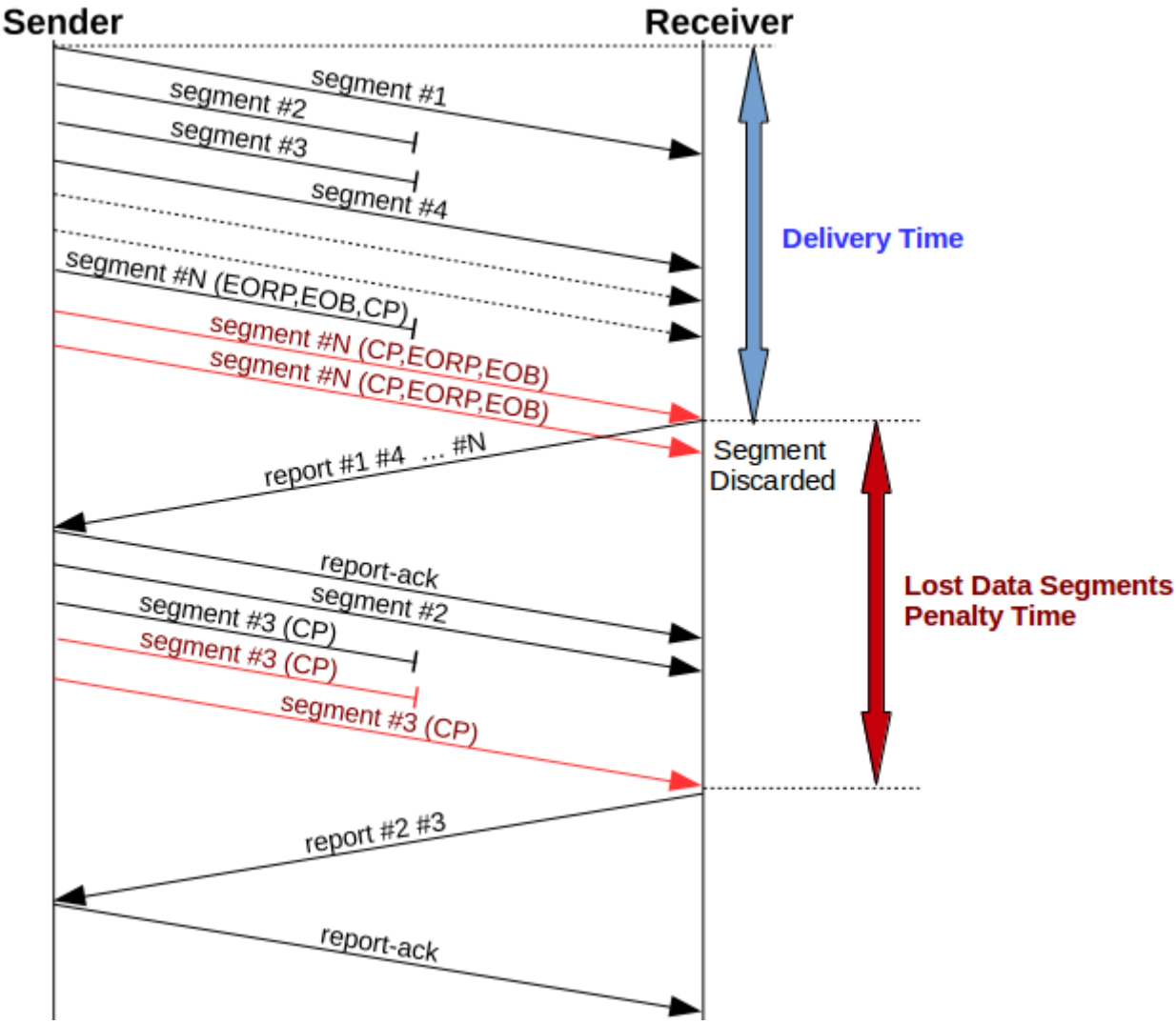

Figura 14: Trasmissione con perdite miste (dati+checkpoint), CPBURST=3. I segmenti rossi sono i checkpoint aggiunti per formare il burst.

Considerazioni analoghe possono essere fatte per tutti gli altri segmenti di segnalazione, nell'implementazione la lunghezza dei burst sarà impostabile singolarmente per ognuno di essi tramite variabili analoghe a CPBURST.

ii. Spread di checkpoint

Per i canali con probabilità di errore correlata, la soluzione mostrata precedentemente è inefficace. Questo avviene perché la probabilità di perdere un segmento immediatamente successivo ad uno perso è molto più alta della media. Questo per via del fatto che il canale potrebbe avere delle brevi interruzioni nel caso della tecnologia Lasercom, dovute a scintillazioni e turbolenze atmosferiche. Al limite, nel caso in cui la probabilità di errore nello stato bad sia molto alta, la probabilità di perdere l'intero burst può essere non molto minore di

quella di perdere un solo checkpoint. Si porti al limite la situazione per comprenderla meglio, considerando un un canale on-off. Nello stato off (il caso limite di bad) non passa nulla. Anche un buco totale nelle comunicazioni ottiche breve in termini di tempo può essere significativo in termini di bit e pacchetti persi, data l'elevata bit rate dei link ottici.
L'idea in questo caso è quella di distribuire più checkpoint all'interno di un intervallo di RTT (Spread), ma che siano opportunamente distanziati in modo da avere una correlazione il più possibile bassa con un eventuale checkpoint perso. Definendo NRIP come il numero totale di ripetizioni (prima N2) di checkpoint all'interno di un RTT.
Un modo semplice per raggiungere tale obiettivo è:

- dividere il tempo dei timer di un checkpoint (circa uguale a 1 RTT) per NRIP
- moltiplicare il numero dei tentativi di ritrasmissione permessi per NRIP

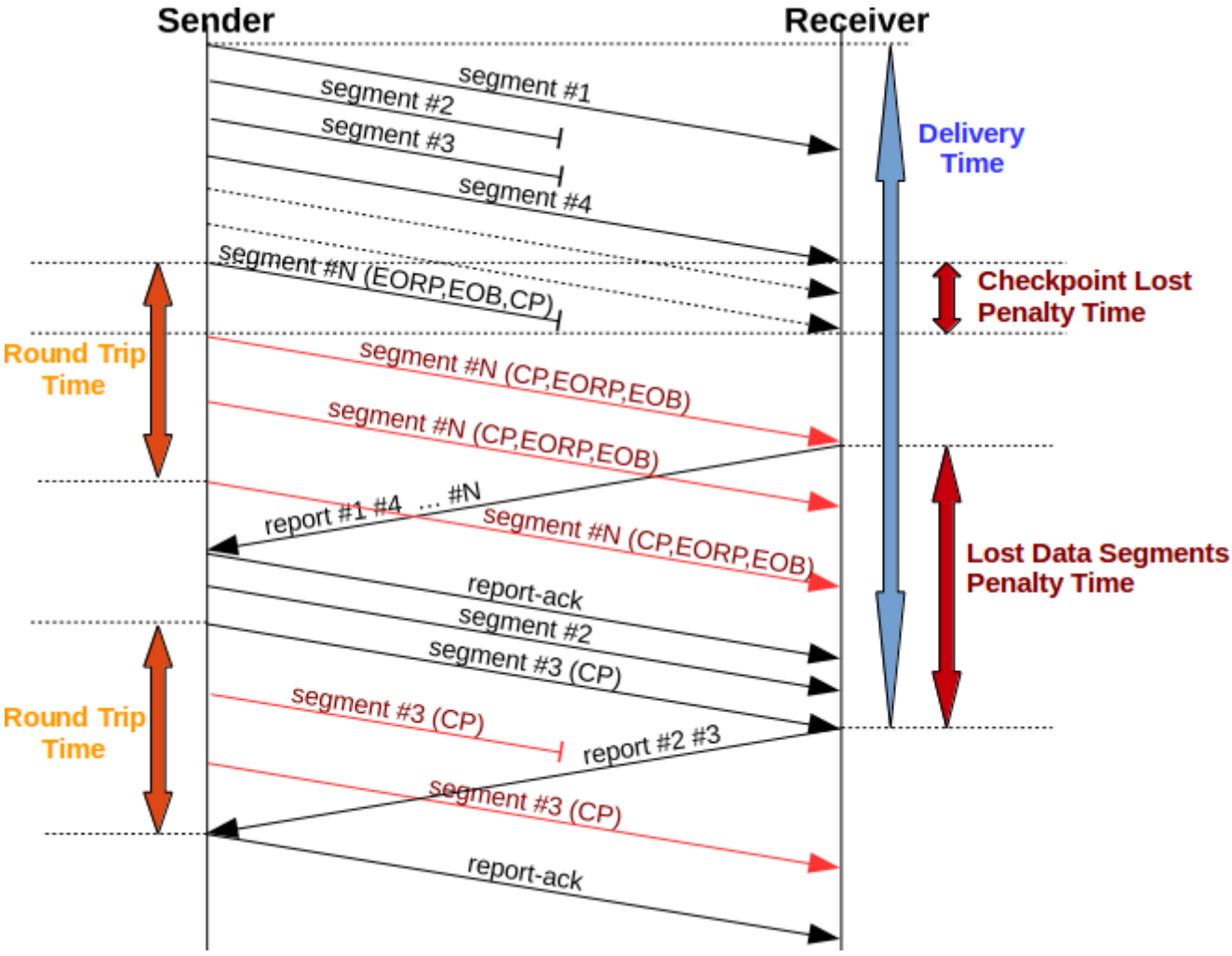

Figura 15: Spread di checkpoint con NRIP=3

In questo modo si riduce notevolmente il Checkpoint Lost Penalty Time, tempo che dipenderà da quanti checkpoint sono stati persi prima di ricevere il primo. Nel caso migliore, il primo checkpoint viene ricevuto correttamente e il Checkpoint Lost Penalty Time è nullo. In questo caso però non vi sarebbe alcun vantaggio rispetto al caso di assenza di ripetizioni. Se invece viene perso il primo, ma non il secondo,si ha una penalità di $\frac{RTT}{NRIP}$ anziché di $1\,RTT$.

Se vengono persi il primo ed il secondo ma non il terzo, la penalità sale a $\frac{2\,RTT}{NRIP}$, e così via.

3. *Secondo miglioramento: introduzione dello stato "Closing"*

Se in una trasmissione si perde il Report Ack finale, cioè quello che conferma la ricezione del Report Segment finale (completata ricezione della parte red), il valore di Import Session Life Time viene prolungato di RTT*maxTimeouts. Questo succede perché, una volta chiusa la Export Session del mittente, esso non riconosce più segmenti in arrivo da parte del destinatario e quindi i vari Report Segment ritrasmessi vengono scartati.
Si ha una situazione di disallineamento fra mittente e destinatario; il primo sa che il secondo ha ricevuto tutti i segmenti e quindi ritiene conclusa la sessione, il secondo pur sapendo ovviamente di avere ricevuto tutto, ritiene, in assenza del Report Ack finale che il mittente non lo sappia; pertanto non chiude la sessione (anche se ha già consegnalo il blocco) al livello superiore, cioè al BP, fintanto che uno dei Report Segment rinviati non viene confermato (cosa impossibile data la chiusura lato mittente) o si raggiunge il limite delle ritrasmissioni (dopodiché inizia una procedura di cancellazione).
La perdita del Report Ack finale è di gran lunga la perdita che prolunga maggiormente la Import Session Life Time. Essa avviene con probabilità pari alla probabilità di perdita di un pacchetto, cioè per un blocco ogni 33 nel caso di PER=3% (Packet Error Ratio) e uno ogni 10 nel caso di PER=10%, ecc. Magari non accade molto frequentemente, ma neanche così raramente da dover accettare la penalizzazione senza intervenire. Si tenga presente che essa potrebbe impedire l'utilizzazione di una delle possibili sessioni parallele per un lungo periodo di tempo se il numero massimo di sessioni parallele è stato raggiunto. Di conseguenza prolungherebbe anche il Delivery Time delle sessioni inutilmente in attesa. In caso di invii multipli questa variazione causa un forte ritardo di alcuni blocchi sfortunati rispetto agli altri, con conseguente grave fuori ordine.
Una prima idea potrebbe essere quella di rispondere ad ogni Report Segment ricevuto con un Report Ack anche dopo la chiusura della sessione. Questa soluzione risolverebbe il problema ma introdurrebbe seri rischi di sicurezza. Tra i problemi più critici si ha:

1. rispondendo a tutti i Report Segment provenienti da sessioni sconosciute, si da la possibilità ad un attaccante di inviare dei Report Ack ad un nodo arbitrario della rete semplicemente inviando dei Report Segment.
2. L'attaccante potrebbe inviare un burst di Report Segment con lo scopo di occupare tutta la banda disponibile, interrompendo quindi il servizio (attacco Denial of Service, DoS).

È chiaro quindi che il concetto di base è buono, ma la soluzione deve essere raffinata.
La soluzione proposta per questo problema si basa sulla introduzione di un nuovo stato della sessione, chiamato "Closing State".
Una sessione, prima di essere chiusa definitivamente, entra in questo stato in cui è in grado di rispondere solo ai Report Segment. Questo evita al destinatario di dover esaurire tutti i tentativi di ritrasmissione prima di poter chiudere la sua Import Session.
Una sessione rimarrà nello stato Closing un tempo pari a RTT*maxTimeouts, ovvero il tempo totale impiegato dal destinatario a ritrasmettere il Report Segment, considerato il caso peggiore in cui tutte le ritrasmissioni vengono perse.
In questo modo si risolve il primo rischio di sicurezza, ovvero ci si limita a rispondere soltanto a delle sessioni chiuse in passato e in un intervallo di tempo in cui ci si aspettano

delle ricezioni di Report Segment da parte dei destinatari.
Per risolvere il secondo rischio di sicurezza, legato ad attacchi di tipo DoS, bisogna anche forzare un'uscita anticipata dallo stato di Closing se viene superato il valore di maxTimeouts.

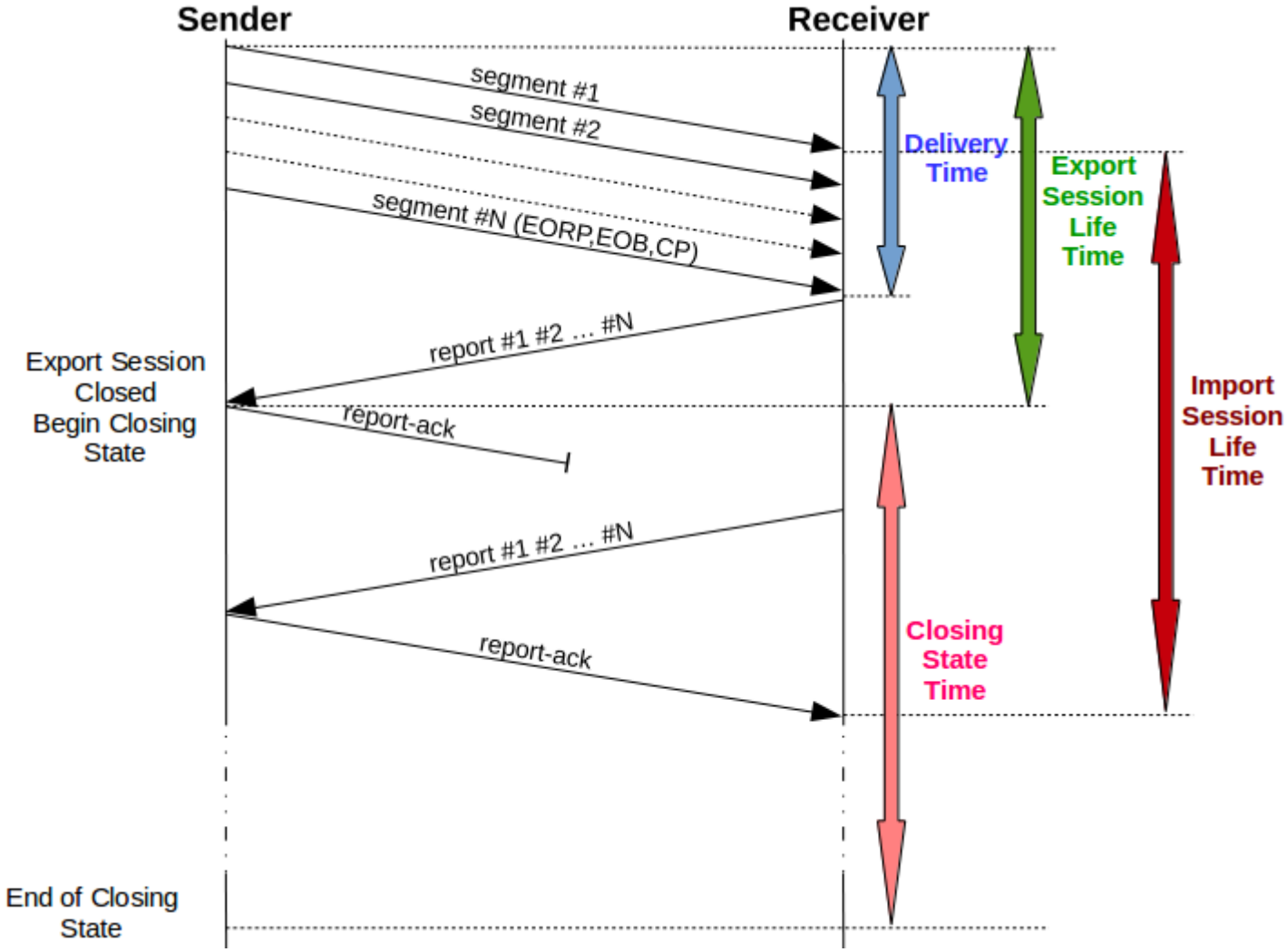

Figura 16: Trasmissione con Closing State.

Il comportamento schematizzato in figura 16 rappresenta lo scenario di utilizzo tipico, ovvero in assenza di un attacco DoS. Tale scenario è lo stesso analizzato precedentemente, si riferisce al caso in cui viene perso il Report Ack finale (situazione schematizzata in figura 12). L'introduzione del Closing State accorcia drasticamente la Import Session Life Time, permettendo eventualmente ad altri blocchi LTP in attesa di impegnare una sessione parallela.
La protezione al secondo problema di sicurezza (attacco di tipo DoS) è schematizzata nella seguente figura:

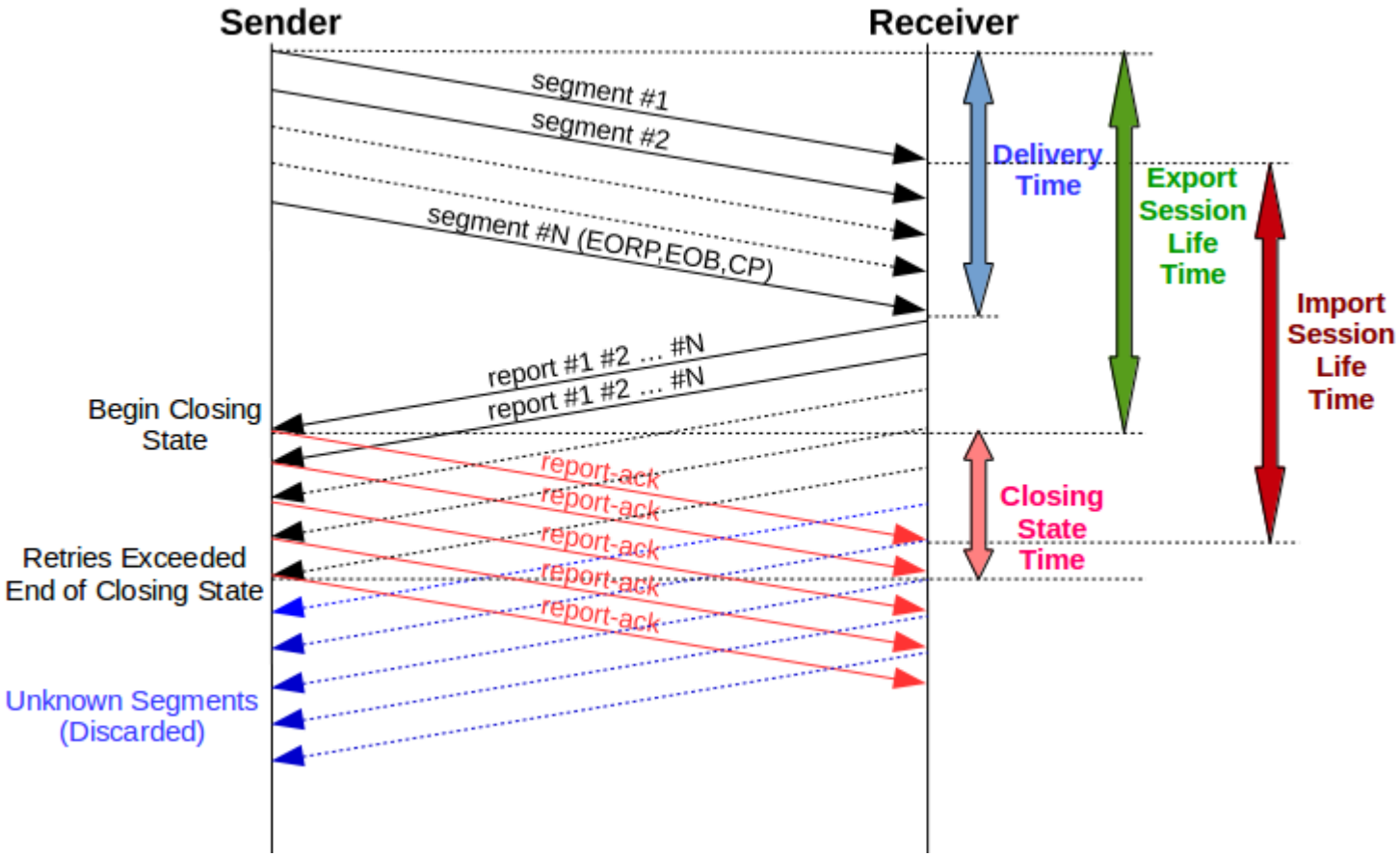

*Figura 17: Closing State DoS Protection. L'attaccante prova a inviare un burst di Report Segment. Ai primi maxTimeouts Report Segment ricevuti, viene chiuso anticipatamente il Closing State, impedendo all'attaccante di bloccare le altre trasmissioni nell'intervallo RTT*maxTimeouts*

Capitolo 5: Modifiche apportate al codice

1. Introduzione ad ION

Interplanetary Overlay Network (ION) [ION_DOC] è un implementazione dell'architettura DTN[RFC4838], progettato per lavorare sfruttando le risorse in modo efficiente, garantendo una completa funzionalità anche su hardware ridotti.
La seguente tesi fa riferimento alla versione 3.4.1, ad oggi ultima versione disponibile su sourceforge [ION_SF].
Gli endpoint sono identificati dagli Universal Record Identifiers(URI) che sono stringhe di testo ASCII con il seguente formato:

scheme_name:scheme_specific_part

Gli schemi possibili sono due, quello “dtn” e quello “ipn”. In ION possono essere usati entrambi, ma quello preferito è il secondo, la cui sintassi è:

ipn:node_number.service_number

ION comprende i seguenti pacchetti software: ICI (Interplanetary Communication Infrastructure), LTP (Licklider Transmission Protocol), BP (Bundle Protocol), DGR (Datagram Retransmission over UDP), AMS (Asynchronous Message Service), CFDP (CCSDS File Delivery Protocol), BSS (Bundle Streaming Service).
La presente tesi si pone come obiettivo di proporre dei miglioramenti all'implementazione LTP di ION, quindi verranno presentati solo gli aspetti relativi a tale implementazione, includendo i vari componenti coinvolti. Una trattazione completa è presente nella documentazione ufficiale [ION_DOC].

2. Struttura software di ION

ION è progettato per funzionare correttamente dentro un sistema operativo Real Time. Molti sistemi operativi Real Time migliorano il determinismo omettendo il supporto per la protezione della memoria. Questa caratteristica viene sfruttata per limitare lo spreco di memoria: un oggetto allocato dinamicamente può essere utilizzato da task differenti accedendo alla memoria condivisa (***Shared Memory***). L'unica complicazione risiede nel proteggere gli accessi che causano situazioni inconsistenti. Sono stati introdotti dei semafori per il controllo degli accessi.

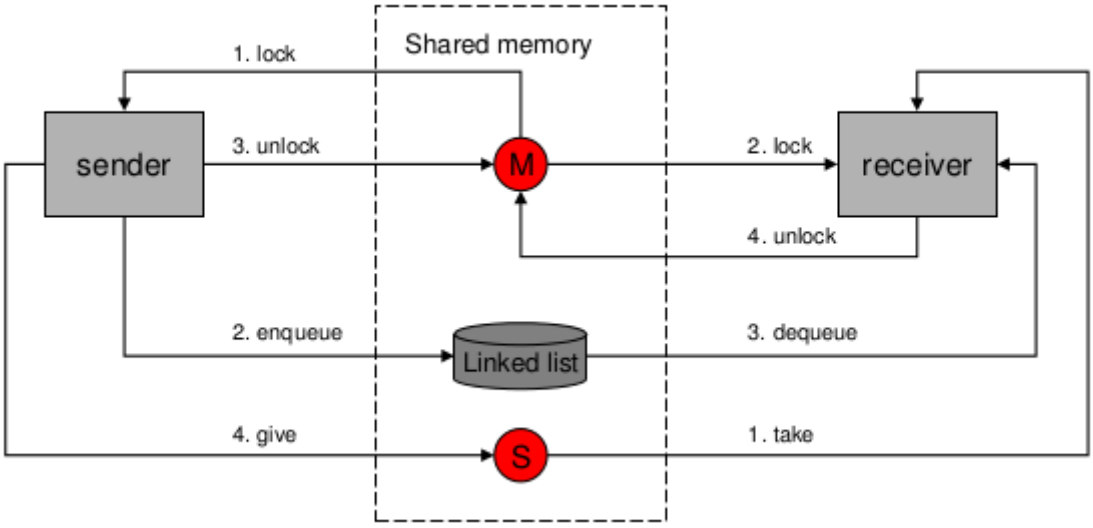

Figura 18: Comunicazione fra task differenti: il task mittente: 1)prende il lock dal semaforo di mutua esclusione (M) proteggendo gli accessi da parte di altri, 2) aggiunge oggetti alla lista, 3) rilascia il lock e 4) segnala (tramite il semaforo S) che la lista non è più vuota. Il task ricevitore: 1) viene informato che la lista non è più vuota, 2) prende il lock dal semaforo di mutua esclusione (M), 3) estrae gli oggetti dalla lista e 4) rilascia il lock.

Se l'oggetto di una lista è un puntatore a un oggetto molto pesante, un modo per abbassare l'overhead è quello di evitarne la copia in task differenti. Per questo scopo, sono stati introdotti i *Zero-copy Object* (ZCO), ovvero oggetti pesanti condivisi fra più task mediante i loro riferimenti. La deallocazione automatica è realizzata mediante la tecnica di Reference Counting, ovvero ogni oggetto possiede un contatore che indica quanti riferimenti ad esso sono presenti, quanto tale contatore è uguale a 0, l'oggetto viene deallocato.
Come è noto, il Reference Counting fallisce in situazioni di riferimenti circolari causando il problema del Memory Leak, ovvero oggetti che si referenziano circolarmente impediscono che il contatore dei riferimenti scenda a zero, e quindi non vengono deallocati. Durante le modifiche che coinvolgono degli Zero-copy Object (ZCO), il programmatore deve prestare attenzione ad evitare il Memory Leak.
Ci sono due tipologie di allocazione gestite da ION: working memory(volatile) e heap(non volatile).

- La working memory (volatile) è un pool di dimensione fissa di memoria condivisa (RAM dinamica). Viene usata dai task di ION per memorizzare informazioni temporanee di ogni tipo. Le strutture dati presenti in questa memoria possono essere condivise tra i diversi task di ION oppure create e gestite individualmente da un task di ION. L'allocazione dinamica della memoria volatile è realizzata mediante il Personal Space Management (PSM).
- Heap (non volatile) è un pool di dimensione fissa. Questo potrebbe occupare un pool di dimensione fissa di memoria condivisa, oppure potrebbe occupare un solo file di dimensioni fisse nel file system, o entrambi. Nel secondo caso, tutti i dati heap sono scritti sia nella memoria che sul file ma sono letti solo dalla memoria. L'allocazione dinamica di spazio nell'heap da parte dei task è realizzata mediante il SDR (Simple Data Recorder o anche Spacecraft Data Recorder).

3. Descrizione di alcuni servizi inclusi in ICI (Interplanetary Communication Infrastructure)

- Platform: funziona per adattare il codice ai diversi sistemi operativi supportati; esso fornisce un'interfaccia software indipendente dal sistema operativo a tutti i task di ION. Questo servizio si occupa anche di implementare l'astrazione di memoria condivisa per i sistemi operativi con memoria protetta (es Linux, Windows,..)
- Personal Space Management (PSM): consente l'allocazione dinamica di oggetti all'interno di un blocco di memoria di dimensione fissa. Un blocco gestito da PSM può far parte sia della memoria privata di un task che della memoria condivisa.
- Simple Data Recorder (SDR): sistema per gestire la memoria heap (non volatile), costruito sullo stesso modello del PSM. Si ispira al modello di database e quindi presenta delle strutture analoghe. Sono presenti funzioni per la gestione di strutture dati (come linked list) nella memoria non volatile, che vengono salvate dentro un singolo file di dimensioni fisse. SDR include un meccanismo di transazioni per proteggere l'integrità del database, assicurando che il fallimento di un'operazione sul database causa il rollback, ovvero tutte le operazioni fatte nel corso della stessa transazione vengono ignorate. Lo scopo del sistema è garantire la conservazione di uno stato del protocollo coerente anche in presenza di un riavvio improvviso del sistema.

Le modifiche al codice originale ION presentate sotto riguardano i file ltpP.h , libltpP.c e ltpclock.c. Come suggerisce il nome, i primi due sono i file sorgenti che contengono l'implementazione del protocollo LTP e il terzo riguarda la gestione degli eventi associati a scadenze temporali (ad esempio i timer).
Da notare che le variabili di tipo Object sono definite come long e rappresentano l'indirizzo di un oggetto allocato nella memoria in SDR. Allo stesso modo, sono definite le variabili di tipo Address. Entrambe funzionano in modo analogo ai puntatori del linguaggio C, la differenza è che Object deve puntare necessariamente ad un oggetto, allocato ad esempio con una sdr_malloc(), mentre quelle di tipo Address possono puntare ad un'area di memoria qualsiasi.
Il linguaggio C non è un linguaggio orientato agli oggetti ma, dato che in ION si usa il termine “Object” per identificare delle strutture allocate dinamicamente in memoria, farò anche io uso di questo termine sottintendendo questo significato.
La seguente figura autoesplicativa mostra il flusso dei dati, utile per la comprensione delle modifiche introdotte

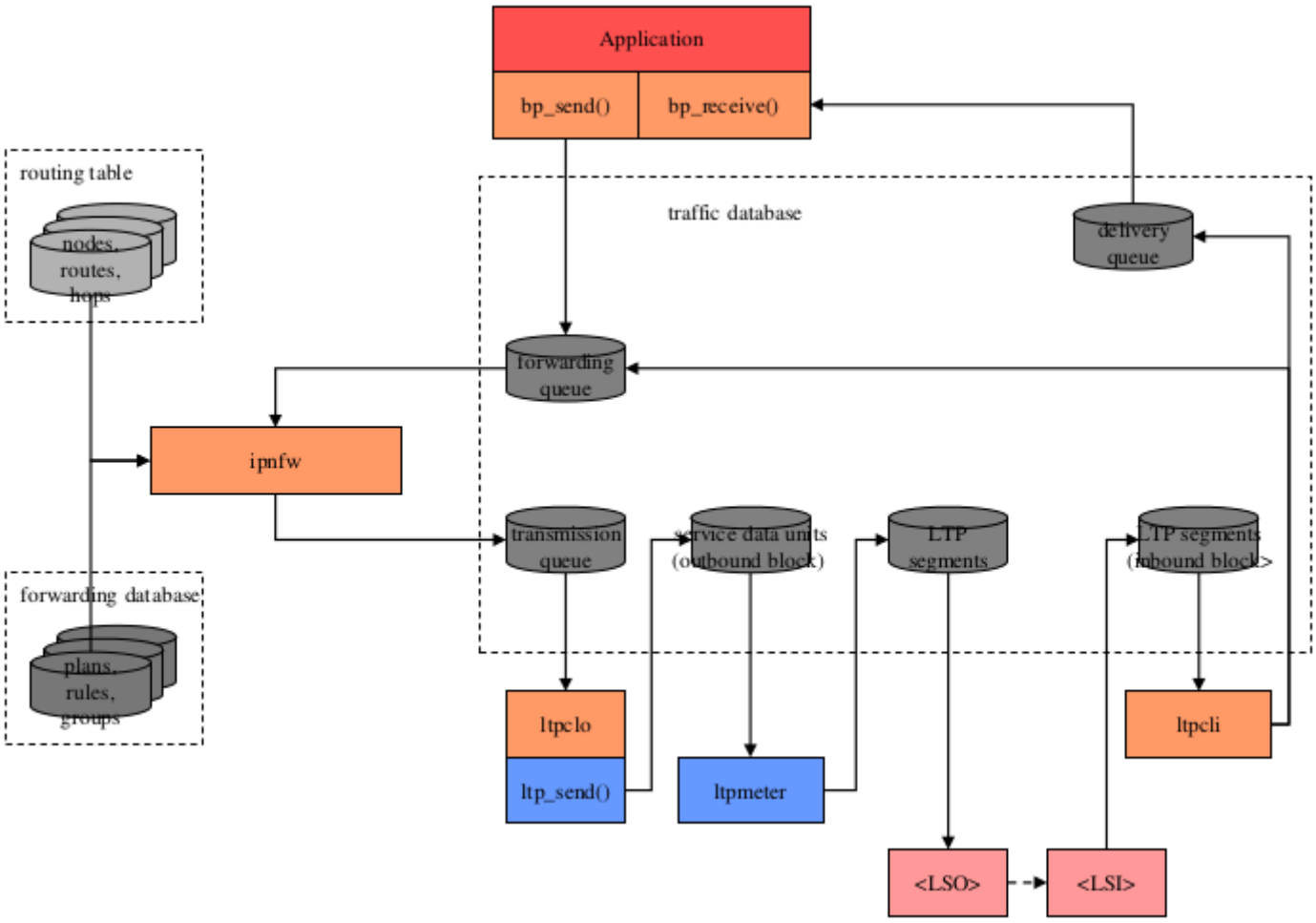

Figura 19: Flusso dei dati in ION. Le collezioni ordinate di oggetti sono mostrate come dei cilindri. Le entità in grigio scuro indicano che i dati sono gestiti dall'SDR, mentre quelle in grigio chiaro indicano che i dati sono gestiti dalla memoria volatile DRAM per migliorare le prestazioni. I rettangoli indicano elementi di elaborazione (task, processi, thread,..). Per maggiori dettagli consultare la documentazione ufficiale di ION [ION_DOC].

4. Implementazione dei Burst

Come accennato precedentemente, i burst risultano essere efficaci non solo per i checkpoint, ma per tutti i segmenti di controllo (Report Segment, Cancel Segment, Report Ack, Cancel Ack).

i. Costanti

Per implementare i burst, sono state definite 6 costanti:
Per il file ltpP.h:

```
#ifndef BURST_ENABLED
#define BURST_ENABLED   1
#endif
```

Questa costante serve per includere (settandola a un valore diverso da 0) o escludere (settandola a 0) dalla compilazione tutto il codice relativo ai burst.

Per il file libltpP.c:

```
#ifndef CHECKPOINT_BURST
#define CHECKPOINT_BURST 3
#endif

#ifndef REPORTSEGMENT_BURST
#define REPORTSEGMENT_BURST 3
#endif

#ifndef CANCELSEGMENT_BURST
#define CANCELSEGMENT_BURST 3
#endif

#ifndef REPORTACK_BURST
#define REPORTACK_BURST 3
#endif

#ifndef CANCELACK_BURST
#define CANCELACK_BURST 3
#endif
```

Queste costanti indicano la durata complessiva dei burst (3 nell'esempio), contando anche l'elemento originale. In altre parole si ha l'invio del segmento ordinario più altre X_BURST-1 repliche (2 nell'esempio). Settandoli tutti ad 1, si ottiene il comportamento normale di LTP (ovvero senza burst). Nell'esempio sono state settate tutte a 3 ma, ovviamente, è solo per facilitare il lettore: ogni costante può avere un certo valore indipendentemente dalle altre.

ii. Introduzione di funzioni in libltpP.c

Sono state predisposte due funzioni per inviare i burst:

- enqueueBurst()
- enqueueAckBurst()

Entrambe le funzioni servono per mettere in coda di invio le X_BURST-1 repliche. Le due funzioni hanno un comportamento molto simile ma differiscono, oltre che per i parametri di ingresso, per il tipo di accodamento. Questo perché l'accodamento dei segmenti di Acknowledgement è prioritario rispetto ai segmenti dati.
La funzione enqueueBurst() quindi mette le repliche in fondo alla coda, la funzione enqueueAckBurst() mette le repliche in coda con priorità, ovvero prima di ogni segmento dati ma dopo di tutti gli altri Ack eventualmente già presenti in coda.
Entrambe le funzioni richiedono, come parametro di ingesso, il segmento da replicare. La condizione principale per la chiamata di queste funzioni è quindi che tale segmento sia già costruito. Per questo motivo,queste funzioni devono essere chiamate tipicamente dopo la sdr_write() del segmento per il quale si vuole costruire il burst in modo da avere subito a disposizione un segmento quasi pronto per i successivi accodamenti.

```
int enqueueBurst(LtpXmitSeg *segment,LtpSpan *span, Object where, int
burstType)
{
	Sdr		sdr = getIonsdr();
	int i;
	Object segmentObj;
```

```
	for (i=1;i<burstType;i++)
	{
	segmentObj = sdr_malloc(sdr, sizeof(LtpXmitSeg));
	segment->pdu.timer.expirationCount=-1; //BURST CONDITION
	segment->queueListElt = sdr_list_insert_last(sdr, span-
>segments,segmentObj);
		if(where)
	segment->sessionListElt = sdr_list_insert_last(sdr,where,
segmentObj);
	sdr_write(sdr, segmentObj, (char *) segment, sizeof(LtpXmitSeg));
		}
	return 0;
}
int enqueueAckBurst(LtpXmitSeg *segment,Object spanObj, int burstType)
{
	Sdr	sdr = getIonsdr();
	int i;
	Object segmentObj;
	for (i=1;i<burstType;i++)
	{
	segmentObj = sdr_malloc(sdr, sizeof(LtpXmitSeg));
	segment->pdu.timer.expirationCount=-1; //BURST CONDITION
	segment->queueListElt =enqueueAckSegment(spanObj, segmentObj);
	sdr_write(sdr, segmentObj, (char *) segment, sizeof(LtpXmitSeg));
	}
	return 0;
}
```

Si è scelto di usare `pdu.timer.expirationCount=-1` come condizione per identificare i burst. Questo è motivato dal fatto che il codice è strutturato in modo da considerare i segmenti con `expirationCount=0` come segmenti di prima trasmissione, mentre i segmenti con valore diverso da 0 come ritrasmissioni. Per i valori maggiori o uguali a 0, `expirationCount` viene incrementato fino al valore `maxTimeouts`, a quel punto il timer non viene più rinnovato per un prossimo invio. Per i burst non si vuole questo tipo di comportamento, nessun segmento del burst deve essere soggetto a ritrasmissioni con dei timer. Per questo è stato scelto un valore negativo (-1) tale da far considerare il segmento come una ritrasmissione, perché diverso da zero, ma allo stesso tempo distinguibile dai valori “normali”, in quanto negativo.
Per gestire le varie priorità di accodamento,in `enqueueBurst()`, i segmenti vengono inseriti in fondo alla coda con `sdr_list_insert_last()`, in `enqueueAckBurst()` invece viene richiamata la funzione `enqueueAckSegment()`, già presente nel codice LTP ufficiale, che accoda il segmento passato come parametro prima di ogni segmento dati, ma dopo gli altri Ack eventualmente già presenti in coda.
Un'altra differenza tra le due funzioni è che nella prima viene eventualmente assegnato il valore di `sessionListElt`, che consente ad alcuni segmenti di essere gestiti per gruppi.
Ad esempio, nel caso di burst di un checkpoint, bisogna inserire il checkpoint nella lista dei segmenti red della Export Session ed assegnare a `sessionListElt` il puntatore del checkpoint (all'interno della lista dei segmenti red). In questo modo, ad esempio durante la chiusura o la cancellazione della sessione, quando i segmenti red non ancora spediti vengono eliminati dalla coda, vengono eliminati anche i segmenti del burst. Non tutti i segmenti hanno questa necessità, quindi la variabile `Object where`, che indica in quale lista aggiungere il segmento per la corretta gestione di gruppo, è opzionale.

iii. Elenco delle chiamate a enqueueBurst() e enqueueAckBurst()

Funzioni che usano enqueueBurst():

- **constructDataSegment()**: questa funzione si occupa di costruire e accodare un segmento dati generico a partire da una porzione del blocco LTP. Questa funzione viene chiamata ciclicamente (per costruire e accodare tutti i segmenti) dalla funzione issueSegments(), a sua volta chiamata dal main di ltpmeter (per inviare tutti i segmenti del blocco LTP) oppure dalla funzione per gestire l'arrivo di un Report Segment handleRS() (per inviare tutte le eventuali ritrasmissioni). Se il segmento costruito da constructDataSegment() è un checkpoint, deve essere accodato altre CHECKPOINT_BURST -1 volte.
- **constructRs()**: questa funzione si occupa di costruire un Report Segment in seguito all'arrivo di un checkpoint e metterlo in coda per l'invio. Tale segmento deve essere accodato altre REPORTSEGMENT_BURST -1 volte
- **enqueueCancelReqSegment()**: questa funzione si occupa di costruire un Cancel Segment (che può essere sia un “cancel by sender” che un “cancel by receiver”) e metterlo in coda per l'invio. Tale segmento deve essere accodato altre CANCELSEGMENT_BURST -1 volte.
- **ltpResendCheckpoint()**: questa funzione si occupa di ritrasmettere un Checkpoint in caso scada il timer ad esso associato. Tale segmento deve essere accodato altre CHECKPOINT_BURST -1 volte
- **ltpResendReport()**: questa funzione si occupa di ritrasmettere un Report Segment in caso scada il timer ad esso associato. Tale segmento deve essere accodato altre REPORTSEGMENT_BURST -1 volte

Funzioni che usano enqueueAckBurst():

- **constructReportAckSegment()**: questa funzione si occupa di costruire un Report Ack Segment (in seguito alla ricezione di un Report Segment) e metterlo in coda per l'invio. Tale segmento deve essere accodato (in modo prioritario rispetto ai segmenti dati) altre REPORTACK_BURST -1 volte.
- **constructCancelAckSegment()** questa funzione si occupa di costruire un Cancel Ack Segment (in seguito alla ricezione di un Cancel Segment) e metterlo in coda per l'invio. Tale segmento deve essere accodato (in modo prioritario rispetto ai segmenti dati) altre CANCELACK_BURST -1 volte.

iv. Altre modifiche per il supporto dei burst

I burst,non essendo previsti dall'attuale implementazione, hanno richiesto piccole modifiche di alcune parti del codice per evitare comportamenti imprevisti e violazioni di accesso alla memoria.

- *Funzione setTimer()*

La funzione setTimer() viene invocata in tutte le varie situazioni in cui un segmento deve essere soggetto a ritrasmissioni. Si occupa di calcolare la scadenza del timer e di inserire un evento in una lista di eventi. I segmenti del burst però non devono essere soggetti a ritrasmissioni, in quanto fanno affidamento al timer del primo segmento.
Ad esempio, all'invio di un checkpoint, è associato un timer. Se si ha CHECKPOINT_BURST

settato a tre, l'invio deve essere ripetuto altre due volte. Se ci fosse il timer per le due ripetizioni, in questo caso si avrebbero 9 ritrasmissioni (6 in più del dovuto) al primo timeout, 27 ritrasmissioni (24 in più del dovuto) e così via.
Per evitare che un segmento di un burst imposti un timer, come prologo della funzione setTimer() serve un controllo per ritornare alla funzione chiamante senza impostarlo, ovvero semplicemente:

```
if(timer->expirationCount==-1) //burst condition
            return 0;
```

- *Funzione destroyDataXmitSeg()*

Questa funzione viene invocata ciclicamente da stopExportSession() per eliminare ogni segmento dati presente in coda di invio. Ogni segmento dati con checkpoint contiene un riferimento ad un oggetto di tipo LtpCkpt, comune a tutti i checkpoint del burst. Il puntatore a tale oggetto è condiviso da tutti i segmenti ripetuti del burst di checkpoint. L'oggetto puntato però deve essere distrutto una sola volta (altrimenti ci si ritrova in una situazione di violazione SDR), quindi solo in corrispondenza della distruzione del primo checkpoint del burst.
Per evitare questo problema occorre aggiungere un semplice controllo che impedisca alla funzione di tentare di distruggere nuovamente l'oggetto di tipo LtpCkpt riferito, in corrispondenza dei segmenti con checkpoint del burst, ovvero aggiungendo un semplice controllo (modifica evidenziata):

```
if (ds→ckptListElt && ds->pdu.timer.expirationCount!=-1)    /*A checkpoint
segment and not a burst */
      {
            /*    Destroy the LtpCkpt object and its ListElt.     */
            sdr_free(sdr, sdr_list_data(sdr, ds->ckptListElt));
            sdr_list_delete(sdr, ds->ckptListElt, NULL, NULL);
      }
```

- *Funzione destroyRsXmitSeg()*

Questa funzione viene invocata ciclicamente da stopImportSession() per eliminare ogni Report Segment presente in coda di invio. Ogni Report Segment contiene una lista di Reception Claim, il cui riferimento è comune a tutti i Report Segment del burst.
Questa lista deve essere distrutta una sola volta (altrimenti ci si ritrova in una situazione di violazione SDR), ovvero in corrispondenza della distruzione del primo Report Segment del burst. Per evitare questo problema occorre aggiungere un semplice controllo che impedisca alla funzione di provare a distruggere nuovamente la lista dei Reception Claim, in corrispondenza segmenti del burst (modifica evidenziata):

```
if(rs->pdu.timer.expirationCount != -1) //burst condition
      sdr_list_destroy(sdr, rs->pdu.receptionClaims, NULL, NULL);
```

5. *Implementazione degli Spread*

Come già accennato, questa modifica riguarda tutti i segmenti di controllo soggetti a ritrasmissione, ovvero Checkpoint, Report Segment e Cancel Segment.
Rispetto alla modifica precedente, questa risulta essere molto più semplice. Per questo motivo è stata implementata prima dell'altra nel corso della tesi.

i. Costanti

È stata definita una costante nel file libltpP.c

```
#ifndef RETRANSMISSION_REDUNDANCY
#define RETRANSMISSION_REDUNDANCY 1
#endif
```

questo è il valore che in fase di analisi è stato chiamato NRIP. Il comportamento di default si ottiene settando il valore 1. Non sono permessi valori minori o uguali a zero.

ii. Macro

È stata definita una macro per arrotondare un numero float all'intero superiore

```
#define CEIL(x) (int)(x) + (1 - (int)((int)((x) + 1) - (x)))
```

iii. Modifiche di funzioni in libltpP.c

- *Funzione computeRetransmissionLimits()*

La funzione computeRetransmissionLimits(), come suggerisce il nome, calcola i limiti di ritrasmissione. Uno di questi valori calcolati è il valore di maxTimeouts, ovvero il numero di tentativi di ritrasmissione dei segmenti. Occorre aggiungere semplicemente dopo il calcolo di maxTimeouts la seguente istruzione:

```
vspan->maxTimeouts=vspan->maxTimeouts*RETRANSMISSION_REDUNDANCY;
```

- *Funzione setTimer()*

La funzione setTimer(), imposta i timer dopo aver calcolato il tempo previsto di arrivo (segArrivalTime) e il tempo previsto di attesa (ackDeadLine). I timer,nella funzione setTimer() del codice originale, sono calcolati nel seguente modo:

```
timer->segArrivalTime = currentSec + radTime + vspan->owltOutbound
                  + ((ltpdb.ownQtime >> 1) & 0x7fffffff);
GET_OBJ_POINTER(sdr, LtpSpan, span, sdr_list_data(sdr,vspan->spanElt));

timer->ackDeadline = timer->segArrivalTime
                  + span->remoteQtime + vspan->owltInbound
                  + ((ltpdb.ownQtime >> 1) & 0x7fffffff);
```

Per supportare la compatibilità tra questa modifica e l'introduzione del Closing State (descritta nei paragrafi successivi), prima di dividere per RETRANSMISSION_REDUNDANCY, è stato rimosso currentSec dal calcolo e assegno i risultati a due variabili che conterranno i valori di offset rispetto a currentSec e nelle istruzioni successive viene applicata la modifica proposta in fase di analisi. L'arrotondamento all'intero successivo serve ad evitare che la ritrasmissione corrispondente a quella che si avrebbe in situazioni normali non avvenga in un istante di tempo precedente. Questo potrebbe costare una ritrasmissione extra. Meglio ritardare una trasmissione leggermente arrotondando sempre per eccesso che anticiparla.

```
segArrivalTimeOffset = radTime + vspan->owltOutbound
                  + ((ltpdb.ownQtime >> 1) & 0x7fffffff);
GET_OBJ_POINTER(sdr, LtpSpan, span, sdr_list_data(sdr,vspan->spanElt));
```

```
ackDeadlineOffset = segArrivalTimeOffset
                  + span->remoteQtime + vspan->owltInbound
                  + ((ltpdb.ownQtime >> 1) & 0x7fffffff);
timer->segArrivalTime= currentSec+ CEIL( segArrivalTimeOffset /
(float)RETRANSMISSION_REDUNDANCY);
      timer->ackDeadline= currentSec+ CEIL(ackDeadlineOffset /
(float)RETRANSMISSION_REDUNDANCY);
```

La funzione noteClosedImport, calcola indipendentemente da setTimer il tempo per rimuovere dalla lista delle Import Session chiuse.
Occorre adattare il calcolo all'introduzione di RETRANSMISSION_REDUNDANCY (modifica evidenziata):

```
event.scheduledTime = currentTime + 10 +
            (2 * (vspan->maxTimeouts / (float)RETRANSMISSION_REDUNDANCY) *
            (vspan->owltOutbound + vspan->owltInbound));
```

6. *Implementazione dello stato "Closing"*

i. *Costanti*

È stata definita una costante nel file ltpP.h

```
#ifndef     RS_CLOSING_STATE_ENABLED
#define     RS_CLOSING_STATE_ENABLED      0
#endif
```

questa costante permette di abilitare (settandola a un valore diverso da 0) e disabilitare (settandola a 0) l'implementazione del Closing State

ii. *Strutture*

È stata introdotta la seguente struttura:

```
typedef struct
{
      Object span;
      unsigned int sessionNbr;
      LtpTimer timer;
} LtpClosing;
```

dove Object span è il puntatore all'oggetto di tipo LtpSpan che servirà per poter mettere in coda i segmenti Report Ack, sessionNbr è il numero della sessione in Closing State e serve per poter costruire il Report Segment, timer serve a far terminare il Closing State, dopo un certo tempo o dopo un certo numero di tentativi.
È stato definito quindi un nuovo tipo di evento, tra gli eventi già presenti, chiamato LtpForgetClosingSession (modifica evidenziata) :

```
typedef enum
{
      LtpResendCheckpoint = 1,
      LtpResendXmitCancel,
      LtpResendReport,
      LtpResendRecvCancel,
```

```
	LtpForgetSession
	,LtpForgetClosingSession
} LtpEventType;
```

Nella struttura definizione della struttura LtpDB, è stato aggiunto Object closingSessions (modifica evidenziata), che sarà il puntatore ad una lista di oggetti di tipo LtpClosing:

```
typedef struct
{
	uvast		ownEngineId;
	Sdnv		ownEngineIdSdnv;
	int		estMaxExportSessions;
	unsigned int	ownQtime;
	unsigned int	enforceSchedule;/*	Boolean.		*/
	double		maxBER;		/*	Max. bit error rate.	*/
	LtpClient	clients[LTP_MAX_NBR_OF_CLIENTS];
	unsigned int	sessionCount;
	Object		exportSessionsHash;
	Object		deadExports;	/*	SDR list: ExportSession */
	Object		spans;		/*	SDR list: LtpSpan */
	Object		timeline;	/*	SDR list: LtpEvent	*/
	Object		closingSessions;  /*	SDR list: LtpClosing */
	unsigned int	maxAcqInHeap;
	unsigned long	heapBytesReserved;
	unsigned long	heapBytesOccupied;
	unsigned long	heapSpaceBytesReserved;
	unsigned long	heapSpaceBytesOccupied;
} LtpDB;
```

La lista di sessioni in Closing State deve essere creata nella fase di inizializzazione del protocollo in cui anche le altre liste di LtpDB vengono create.

iii. Modifiche di funzioni in libltpP.c

Nella funzione ltpInit() è stato necessario aggiungere,all'interno della transazione SDR che inizializza tutti gli oggetti e le strutture, la seguente istruzione:

```
ltpdbBuf.closingSessions= sdr_list_create(sdr);
```

Nella funzione setTimer() occorre aggiungere una condizione per riadattare la formula per il calcolo del timer delle sessioni Closing State, che differisce dagli altri timer. Dopo il calcolo classico, occorre correggere la scadenza del timer nel caso in cui esso si stia riferendo al timer di una sessione in Closing State (modifiche evidenziate):

```
/* codice già presente dalla modifica precedente */
segArrivalTimeOffset = radTime + vspan->owltOutbound
		+ ((ltpdb.ownQtime >> 1) & 0x7fffffff);
ackDeadlineOffset = segArrivalTimeOffset
		+ span->remoteQtime + vspan->owltInbound
		+ ((ltpdb.ownQtime >> 1) & 0x7fffffff);
timer->segArrivalTime= currentSec+ CEIL( segArrivalTimeOffset /
(float)RETRANSMISSION_REDUNDANCY);
	timer->ackDeadline= currentSec+ CEIL(ackDeadlineOffset /
(float)RETRANSMISSION_REDUNDANCY);
/* correzione di timer→ackDeadLine da introdurre*/
if(event->type==LtpForgetClosingSession)
```

```
	{
	timer->ackDeadline=currentSec+
			(ackDeadlineOffset *
			vspan->maxTimeouts/RETRANSMISSION_REDUNDANCY);
	}
```

Notare che maxTimeouts viene diviso per RETRANSMISSION_REDUNDANCY. Questa cosa garantisce compatibilità con le modifiche introdotte precedentemente.

iv. Introduzione di funzioni in libltpP.c

- *Funzione addSessionToClosingStateList()*

Questa nuova funzione si occupa di aggiungere una sessione alla lista delle sessioni in Closing State, settando anche il timer.

```
int	addSessionToClosingStateList(LtpDB *ltpdb,LtpVspan *vspan,  Object
spanObj, unsigned int sessionNbr , unsigned int segmentLength)
{
	Sdr		sdr = getIonsdr();
	Object		elt;
	LtpClosing	ltpClosingBuf;
	Object		ltpClosingObj;
	LtpTimer	*timer;
	LtpEvent	ltpClosingEvent;
	time_t		currentTime=getUTCTime();

	/*preparing ltpClosing object*/
	memset((char *) &ltpClosingBuf, 0, sizeof(LtpClosing));
	memset((char *) &ltpClosingEvent, 0, sizeof(LtpEvent));
	ltpClosingObj = sdr_malloc(sdr, sizeof(LtpClosing));
	if (ltpClosingObj == 0)
		{
		putErrmsg("No space for database.", NULL);
		return -1;
		}
	ltpClosingBuf.sessionNbr=sessionNbr;
	ltpClosingBuf.span= spanObj;
	ltpClosingBuf.timer.expirationCount=0;
	sdr_write(sdr, ltpClosingObj, (char *) &ltpClosingBuf,
	sizeof(LtpClosing));
	sdr_list_insert_first(sdr, ltpdb->closingSessions, ltpClosingObj);

	/*preparing ltpTimer object*/
	ltpClosingEvent.refNbr2=sessionNbr;
	ltpClosingEvent.type=LtpForgetClosingSession;
	timer = &ltpClosingBuf.timer;
	if (setTimer(timer, ltpClosingObj + FLD_OFFSET(timer,
&ltpClosingBuf),  currentTime, vspan, segmentLength, &ltpClosingEvent) < 0)
	{
		putErrmsg("Can't set timer.", NULL);
		return -1;
	}
	/*debugInfo*/
#if LTPDEBUG
	char  rsbuf[256];
```

```
	putErrmsg("List of Session in Closing state :", NULL );
	for (elt = sdr_list_first(sdr, (_ltpConstants())->closingSessions);
elt; elt = sdr_list_next(sdr, elt))
	{
		ltpClosingObj = sdr_list_data(sdr, elt);
		sdr_read(sdr, (char *) &ltpClosingBuf,
ltpClosingObj,sizeof(LtpClosing));
		sprintf(rsbuf, "Span:%lu SessionNumber:
%u",ltpClosingBuf.span,ltpClosingBuf.sessionNbr);
		putErrmsg(rsbuf, NULL);
	}
#endif

	return 0;
}
```

Da notare che questa funzione non avvia alcuna transazione SDR. Per non incorrere in problemi di database inconsistente, la funzione chiamante deve aver avviato precedentemente una transazione SDR.
Non è stata avviata alcuna transazione perché, come vedremo, questa funzione viene chiamata soltanto da handleRS() all'interno di una transazione già avviata.

- *Funzione acknowledgeReportFromClosingState()*

Questa nuova funzione si occupa di mettere in coda un Report Ack per conto di una sessione nel Closing State. Se viene superato maxTimeouts, forza l'eliminazione della sessione dalla lista delle sessioni in Closing State.

```
int acknowledgeReportFromClosingState(LtpVspan *vspan,unsigned int
sessionNbr,unsigned int rptSerialNbr)
{
	Sdr		sdr = getIonsdr();
	Object		elt;
	Object		ltpClosingObj;
	LtpClosing	ltpClosingBuf;
	LtpSpan		spanBuf;

for (elt = sdr_list_first(sdr, (_ltpConstants())->closingSessions); elt;
elt = sdr_list_next(sdr, elt))
{
ltpClosingObj = sdr_list_data(sdr, elt);
sdr_stage(sdr, (char *) &ltpClosingBuf, ltpClosingObj,sizeof(LtpClosing));

if (ltpClosingBuf.sessionNbr == sessionNbr)
	{
	if(ltpClosingBuf.timer.expirationCount == vspan->maxTimeouts )
		{
		putErrmsg("Forget Session, Reason: Retries Exceeded",
utoa(ltpClosingBuf.sessionNbr));
		sdr_list_delete(sdr,elt,NULL,NULL);
		}
	else
		{
		putErrmsg("Sending RA to session in \"closing\" state.",
utoa(sessionNbr));
```

```
            if (constructReportAckSegment(&spanBuf, ltpClosingBuf.span ,
sessionNbr,rptSerialNbr))
                  {
                  putErrmsg("Can't send another RA segment.", NULL);
                  return 0; //fail
                  }

                  ltpClosingBuf.timer.expirationCount++;
                  sdr_write(sdr, ltpClosingObj, (char *) (&ltpClosingBuf),
sizeof(LtpClosing)); //update
            }
      return 1; //success
      }
}
return 0; // fail
}
```

Da notare che questa funzione non inizia alcuna transazione SDR. Per non incorrere in problemi di database inconsistente, la funzione chiamante deve aver avviato precedentemente una transazione SDR.
Non è stata avviata alcuna transazione perché, come vedremo, questa funzione viene chiamata soltanto da handleRS() all'interno di una transazione già avviata.

- *Funzione ltpForgetClosingSession()*

Questa nuova funzione viene chiamata da ltpclock.c in seguito della scadenza del timer. Si occupa di rimuovere una certa sessione dalla lista delle sessioni in Closing State.

```
int ltpForgetClosingSession(unsigned int sessionNbr)
{
Sdr               sdr = getIonsdr();
Object            elt;
Object            ltpClosingObj;
LtpClosing        ltpClosingBuf;

CHKERR(sdr_begin_xn(sdr));

for (elt = sdr_list_first(sdr, (_ltpConstants())->closingSessions); elt;
elt = sdr_list_next(sdr, elt))
      {
      ltpClosingObj = sdr_list_data(sdr, elt);
      sdr_read(sdr, (char *) &ltpClosingBuf,
ltpClosingObj,sizeof(LtpClosing));

      if (ltpClosingBuf.sessionNbr == sessionNbr)
            {
            sdr_list_delete(sdr,elt,NULL,NULL);

      #if LTPDEBUG
      putErrmsg("Forget Session, Reason: Timer Elapsed", utoa(sessionNbr));
      #endif

            break;
            }
      }
      if (sdr_end_xn(sdr) < 0)
```

```
            {
            putErrmsg("Error occurred removing Session from Closing State",
utoa(sessionNbr));
            return -1; //failure
            }
      return 0; //success
}
```

Da notare che questa funzione, a differenza delle precedenti, si occupa anche di iniziare e finire la transazione SDR associata all'operazione. Questo perché, come vedremo a breve, questa funzione è chiamata dalla funzione dispatchEvents() di ltpclock.c che non avvia alcuna transazione.

v. Sospensione dei timer

L'implementazione LTP di ION prevede che i timer vengano sospesi nei momenti in cui è prevista l'interruzione del canale, intesa come interruzione prevista dai file di configurazione e non come interruzione occasionale dovuta a guasti, malfunzionamenti o cause non previste.
Quando si verifica l'interruzione del canale, viene chiamata la funzione ltpSuspendTimers() per sospendere i timer legati alle ritrasmissioni, quando invece il canale è nuovamente attivo viene chiamata la funzione ltpResumeTimers() per ripristinarli.
Il Closing State è realizzato mediante una lista, nella quale ogni elemento viene rimosso dopo la scadenza di un timer ad esso associato. Il comportamento che si vuole ottenere è quello che il Closing State abbia una durata che tenga conto delle varie interruzioni del canale. Per questo motivo i timer associati al Closing State devono essere sospesi e ripristinati in modo opportuno. È stato necessario quindi aggiungere questo comportamento ad entrambe le funzioni.

- *Codice aggiunto a ltpSuspendTimers()*

```
Object      ltpClosingObj;
LtpClosing  ltpClosingBuf;

for (elt = sdr_list_first(sdr, (_ltpConstants())->closingSessions); elt;
elt = sdr_list_next(sdr, elt))
      {
      ltpClosingObj = sdr_list_data(sdr, elt);
      sdr_stage(sdr, (char *) &ltpClosingBuf, ltpClosingObj,
sizeof(LtpClosing));
      /*    Suspend Closing State forget timer.       */
      timer = &ltpClosingBuf.timer;
      suspendTimer(suspendTime, timer, ltpClosingObj + FLD_OFFSET(timer,
&ltpClosingBuf), qTime, priorXmitRate,
LtpForgetClosingSession,0,ltpClosingBuf.sessionNbr, 0);
      }
```

- *Codice aggiunto a ltpResumeTimers()*

```
Object      ltpClosingObj;
LtpClosing  ltpClosingBuf;
for (elt = sdr_list_first(sdr, (_ltpConstants())->closingSessions); elt;
elt = sdr_list_next(sdr, elt))
      {
      ltpClosingObj = sdr_list_data(sdr, elt);
      sdr_stage(sdr, (char *) &ltpClosingBuf, ltpClosingObj,
```

```
sizeof(LtpClosing));
	if (ltpClosingBuf.timer.state != LtpTimerSuspended)
		{
		continue;	/*	Not suspended.	*/
		}
	timer = &ltpClosingBuf.timer;
	if (resumeTimer(resumeTime, timer,ltpClosingObj + FLD_OFFSET(timer,
&ltpClosingBuf),qTime, remoteXmitRate, LtpForgetClosingSession,0,
ltpClosingBuf.sessionNbr, 0) < 0)
		{
		putErrmsg("Can't resume timers for span.",itoa(span-
>engineId));
		sdr_cancel_xn(sdr);
		return -1;
		}
	}
```

vi. Chiamate alle nuove funzioni

- *Funzione addSessionToClosingStateList()*

Questa funzione viene chiamata da handleRS(), ovvero la funzione lanciata alla ricezione di un Report Segment. La seguente porzione di codice indica la parte in cui il Report Segment ricevuto è quello finale, quindi viene chiusa la ExportSession distruggendo tutti i segmenti in uscita. Successivamente a questa fase, la sessione deve essere messa nella lista delle sessioni in Closing State (modifiche evidenziate):

```
if (claim->offset == 0 && claim->length == sessionBuf.redPartLength)
{
	ltpSpanTally(vspan, POS_RPT_RECV, 0);
	MRELEASE(claim);  /*	(Sole claim in list.)	*/
	lyst_destroy(claims);
	if (sessionBuf.redPartLength == sessionBuf.totalLength
	|| sessionBuf.stateFlags & LTP_EOB_SENT)
	{
	stopExportSession(&sessionBuf); //distrugge tutti i segmenti
	closeExportSession(sessionObj);
	ltpSpanTally(vspan, EXPORT_COMPLETE, 0);

	int segmentLength=segment->pdu.headerLength	+ segment-
>pdu.contentLength + segment->pdu.trailerLength;
	addSessionToClosingStateList(ltpdb,vspan,spanObj,sessionNbr,
segmentLength);

}
```

- *Funzione acknowledgeReportFromClosingState()*

Anche questa funzione viene chiamata da handleRS(), ovvero la funzione lanciata alla ricezione di un Report Segment. La seguente porzione di codice indica il punto in cui viene ricevuto un Report Segment di provenienza sconosciuta. Tale segmento, potrebbe provenire da una sessione in Closing State, quindi bisogna provare a rispondere (modifiche evidenziate)

```
CHKERR(sdr_begin_xn(sdr));
getSessionContext(ltpdb, sessionNbr, &sessionObj,&sessionBuf, &spanObj,
```

```
&spanBuf, &vspan, &vspanElt);
if (spanObj == 0) //unknown provenance
	{
	if(acknowledgeReportFromClosingState(vspan,sessionNbr,rptSerialNbr))
		{
		if (sdr_end_xn(sdr) < 0)
			{
			putErrmsg("Can't send another RA segment.", NULL);
			MRELEASE(newClaims);
			return -1;
			}
		return 0;
		}
#if LTPDEBUG
	putErrmsg("Unknown Session:Discarding report.", NULL);
#endif
	MRELEASE(newClaims);
	sdr_exit_xn(sdr);
	return 0;
	}
```

- *Funzione ltpForgetClosingSession()*

In ltpclock.c, viene invocata la funzione dispatchEvents() per ogni secondo trascorso. Questa funzione controlla ogni evento presente ed eventualmente ne invoca la funzione corrispondente. Avendo definito il tipo di evento LtpForgetClosingSession, occorre dire a ltpclock.c quale funzione di libltpP.c invocare (modifica evidenziata) nel caso in cui l'evento si verificasse:

```
switch (event.type)
	{
	case LtpResendCheckpoint:
	result = ltpResendCheckpoint(event.refNbr2,event.refNbr3);
	break;			/*	Out of switch.		*/

	case LtpResendXmitCancel:
	result = ltpResendXmitCancel(event.refNbr2);
	break;			/*	Out of switch.		*/

	case LtpResendReport:
	result = ltpResendReport(event.refNbr1,event.refNbr2, event.refNbr3);
	break;			/*	Out of switch.		*/

	case LtpResendRecvCancel:
	result = ltpResendRecvCancel(event.refNbr1,event.refNbr2);
	break;			/*	Out of switch.		*/

	case LtpForgetSession:
	sdr_list_delete(sdr, event.parm, NULL, NULL);
	result = 0;
	break;			/*	Out of switch.		*/

	case LtpForgetClosingSession:
	result= ltpForgetClosingSession(event.refNbr2);
	break;
```

```
default:          /*    Spurious event.         */
result = 0; /*    Event is ignored. */
}
```

Capitolo 6: Analisi delle prestazioni

1. Introduzione

Per valutare le modifiche introdotte è stato necessario effettuare dei test. I tool usati sono [DTNperf_3], [Virtualbricks] e macchine virtuali con sistema operativo Debian 7 GNU/Linux in cui è stato installato ION 3.4.1.
DTNperf_3 è un tool di valutazione client server progettato per stimare il goodput e per fornire dei log degli Status Report in un'architettura DTN Bundle Protocol (BP).
La terza versione di DTNperf include il pieno supporto per DTN2 e ION.
Virtualbricks è un front end che combina le funzionalità di QEMU (software Linux per l'emulazione delle macchine virtuali) e VDE (software che emula dei dispositivi di rete, come ad esempio gli switch).
Il testbed è costituito da due macchine virtuali, connesse attraverso un emulatore di canale (Netemu) con nel quale è possibile stabilire i tempi di propagazione one-way e la PER (Packet Error Ratio), in entrambe le direzioni sia in modo simmetrico che in modo asimmetrico.
Per interpretare al meglio i risultati, conviene cercare di evitare interferenze da sessioni LTP parallele. Per questo motivo è preferibile inviare un bundle per volta.
Questo può essere ottenuto facilmente utilizzando la Window Option di DTNperf_3: settando tale parametro a 1 nel client solo un bundle per volta può essere in trasmissione.
Il principio di funzionamento di DTNperf_3 può essere riassunto nelle seguenti 4 fasi:

1. Il client, in esecuzione sul nodo mittente, genera un bundle e lo trasmette al server;
2. il bundle viene consegnato al server, in esecuzione sul nodo di destinazione;
3. viene inviato un ACK al client;
4. il client riceve l'ACK e ricomincia dal punto 1

Il ciclo si interromperà per uno dei seguenti motivi:

- è stata inviata la quantità di dati richiesta inizialmente
- si è esaurito il tempo massimo dato come parametro di input

Il Delivery Time può essere calcolato dagli Status Report “delivered”, che contengono il timestamp di generazione del bundle e il timestamp di consegna del bundle. Questa operazione è molto semplice ma richiede che le macchine in cui sono in esecuzione il client e il server di DTNperf_3 abbiano gli orologi di sistema perfettamente allineati.
Seppur questo possa essere un grosso problema relativo in alcuni scenari reali, non costituisce un problema con i test su macchine virtuali, poiché in entrambe l'orologio di sistema è sincronizzato perfettamente con l'orologio della macchina host.

2. Parametri di configurazione per i test

Per valutare l'impatto del Closing State si è deciso di impostare nello scope di entrambi con una sola Import Session e una sola Export Session.
La dimensione del bundle è settata a 100kB, il segmento LTP a 1024B, quindi si hanno circa 100 segmenti per bundle. Il parametro di Aggregation Size dell'LTP è impostato a un valore minore di 100kB, in modo che quando il bundle viene passato all'LTPCLA, esso venga direttamente incapsulato in un blocco LTP ed immediatamente segmentato. Ogni segmento LTP viene a sua volta incapsulato in un datagramma UDP.
La velocità di trasmissione del canale utile al livello LTP è impostata a 10 Mbit/s, la quale

garantisce un tempo di trasmissione del bundle (corrispondente a un blocco LTP) di circa 0.08s.
Le impostazioni dei tempi di propagazione sono tipicamente tra le più critiche e il RTT dipenderà da esse. Ad ogni modo, l'analisi introdotta nel terzo capitolo di questa tesi permette di realizzare dei test indipendenti dal valore effettivo del RTT, pensando i vari intervalli di tempo come multipli del RTT. Infatti, sotto l'ipotesi di avere un RTT molto maggiore del tempo di trasmissione del blocco e assumendo che il timer di ritrasmissione sia settato ad un valore poco più grande del RTT (ovvero RTT + margine), il Delivery Time può essere normalizzato al RTT, diventando indipendente dal valore di RTT attuale.
Queste considerazioni nella pratica permettono di svolgere i test usando valori di RTT estremamente ridotti, minimizzando la durata complessiva del test, ottenendo dei risultati validi per ogni RTT possibile.
Nell'attuale implementazione di ION il minimo consentito è 2 secondi a causa della granularità del tempo che impedisce di settare dei tempi minori di 1 secondo, pertanto i test sono stati svolti utilizzando un RTT pari a 2 secondi.
Tutti i test sono stati effettuati con questi parametri, riportando i risultati al variare della PER (Packet Error Ratio), con perdite indipendenti e considerando il canale simmetrico (ipotesi non problematica visto che i risultati sono espressi in funzione del RTT).
Come evidenziato dal quarto capitolo nei cenni sui canali radio e sui canali ottici, se si considera uno scenario più realistico potrebbe essere troppo riduttivo ragionare in termini di Packet Error Ratio perché questo implicherebbe che ogni pacchetto abbia la stessa probabilità di essere perso. Potrebbe essere quindi più accurato ragionare in termini di BER (Bit Error Ratio) in cui si valuta la probabilità di perdere un bit piuttosto che l'intero pacchetto. Questo comporterebbe ad avere PER variabile per ogni pacchetto, che cresce all'aumentare della dimensione del pacchetto.
Lo scopo di questi test è, tuttavia, quello di verificare la correttezza e l'efficacia delle tecniche adottate, senza pretesa di fornire risultati numerici eccessivamente accurati.

3. Risultati Numerici

Nei test sono stati considerati 4 scenari LTP:

1. Originale, così come fornito da ION 3.4.1, etichettato come “Original” nelle figure
2. Closing State, etichettato come “ClosingState” nelle figure
3. Burst (con lunghezza dei burst=3 per tutti i segmenti di segnalazione) più Closing State, etichettato semplicemente come “Burst” nelle figure
4. Spread (con NRIP=3) più Closing State, etichettato semplicemente come “Spread” nelle figure

Non sono stati considerati i singoli casi di Burst o Spread perché il miglioramento della Closing State non interferisce con le ridondanze.
La seguente tabella mostra i risultati numerici dei test effettuati, in termini di valore medio espresso in RTT/2, al variare della PER.
Si noti che poiché il Delivery Time minimo in assenza di perdite è pari a RTT/2, i valori mostrati in tabella rappresentano anche il fattore moltiplicativo del Delivery Time dovuto alle perdite. Ad esempio nel caso dell'algoritmo originale il Delivery Time con PER=3% sarà pari a 3,96 volte il valore che avrebbe avuto in assenza di perdite (PER=0%)

PER	Original (RTT/2)	ClosingState (RTT/2)	Burst (RTT/2)	Spread (RTT/2)
1%	2.3	2.81	2.66	2.35
3%	3.96	3.84	3.1	3.3
10%	10.07	6.21	4.5	5.63
20%	20.01	12.49	6.54	8.47
30%	79.7	24.58	10.14	12.46

Riportando i seguenti risultati in un grafico si ha:

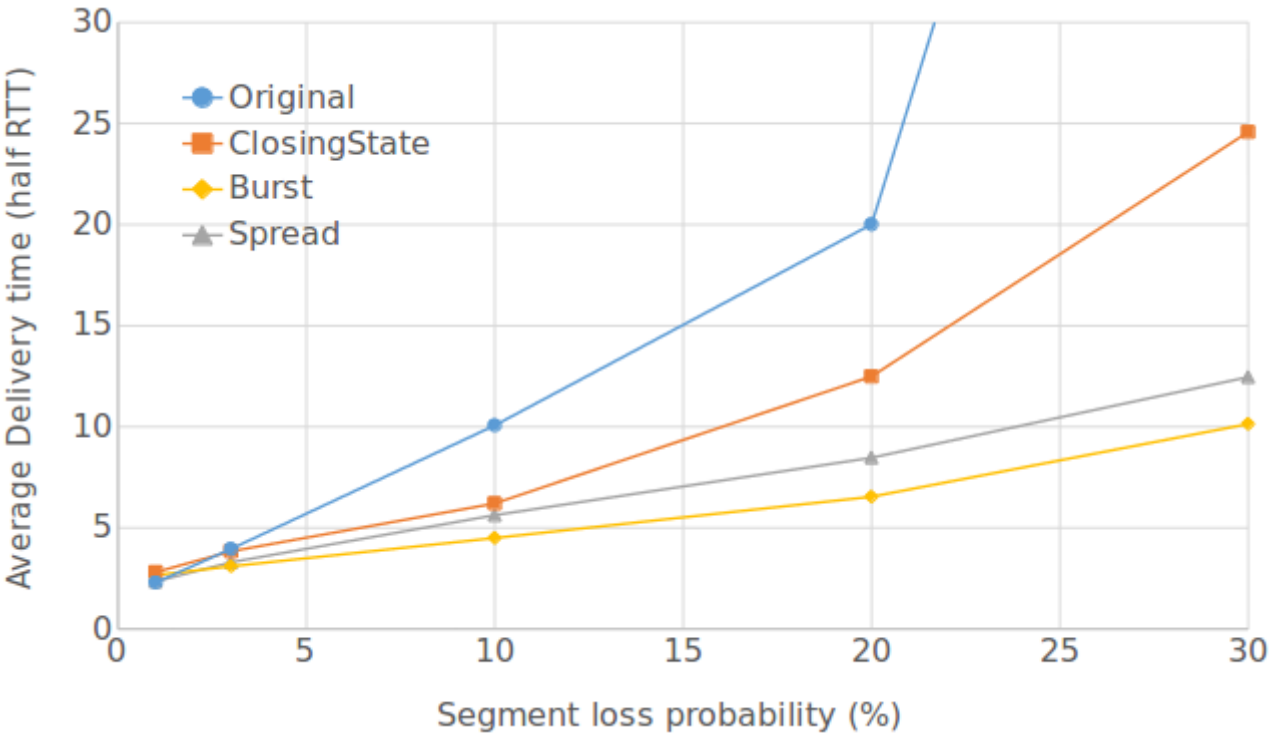

Figura 20: Risultati dei test in termini di Delivery Time medio in funzione della PER (Packet Error Ratio). Paragonando le curve si può osservare che per valori di PER bassi (<1%) il valore medio è pressoché uguale in tutti i casi. Si ha un lieve miglioramento al 3% di perdita. Per valori più alti, dal 10% in poi, si ha un visibile miglioramento.

Questi risultati confermano quanto si era già detto nel capitolo 9: se il canale presenta delle perdite indipendenti, il burst risulta la più efficace delle tecniche implementate.
Il vantaggio introdotto dalle nuove tecniche non è significativo (in termini di media) per valori di PER inferiori al 10%, perché le problematiche da risolvere riguardano solo le situazioni sfortunate in cui si hanno delle perdite, come la perdita del Report Ack finale (nel caso del Closing State) o anche la perdita di altri segmenti di segnalazione (nel caso di Burst e Spread).
Ad ogni modo, se si va a confrontare la varianza del Delivery Time, come mostrato nella figura seguente, si può notare che i miglioramenti introdotti hanno un impatto significativo a partire da valori di PER minori (già a partire dal 3%). Come per la media, per valori di PER elevati si presenta una grossa differenza in termini di varianza.
La seguente tabella mostra la varianza del Delivery Time espressa in RTT/2, calcolata in

funzione della PER:

PER	Original (RTT/2)	ClosingState (RTT/2)	Burst (RTT/2)	Spread (RTT/2)
1%	1.12913	2.0900802	1.31307	1.021543
3%	7.96	3.9943434	0.59596	0.818182
10%	85.03	10.066566	2.030303	3.79101
20%	292.05	32.151414	3.624646	7.645556
30%	1187.34	114.28646	9.495354	13.46303

Riportando i seguenti risultati in un grafico si ha:

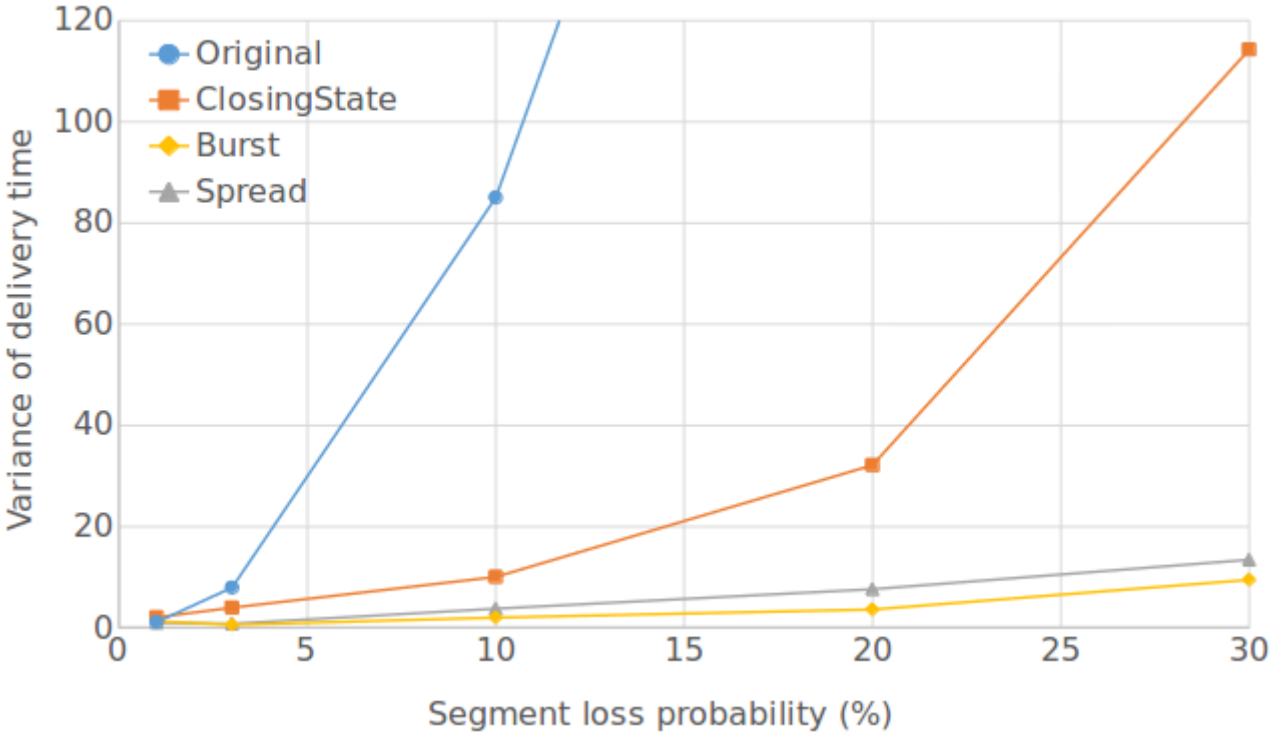

Figura 21:Risultati dei test in termini di varianza del Delivery Time in funzione della PER (Packet Error Ratio). Si può notare che anche al 3% di perdite la varianza del Delivery Time assume valori minimi sia in caso di Burst che Spread. Notevole anche l'impatto del Closing State preso singolarmente.

L'analisi della varianza ci suggerisce l'ipotesi che ci sia una grande differenza in termini di Delivery Time tra i casi più frequenti e il caso più “sfortunato”. Per verificare questa osservazione, si può confrontare la mediana (il 50° percentile della funzione di ripartizione) del Delivery Time con il 90° percentile della funzione di ripartizione del Delivery Time del bundle.
La seguente tabella mostra la mediana del Delivery Time espressa in RTT/2, calcolata in funzione della PER:

PER	Original (RTT/2)	ClosingState (RTT/2)	Burst (RTT/2)	Spread (RTT/2)
1%	3	3	3	3
3%	3	3	3	3
10%	6	5	5	5
20%	12	11	7	8
30%	32	23	9	12

Riportando i seguenti risultati in un grafico si ha:

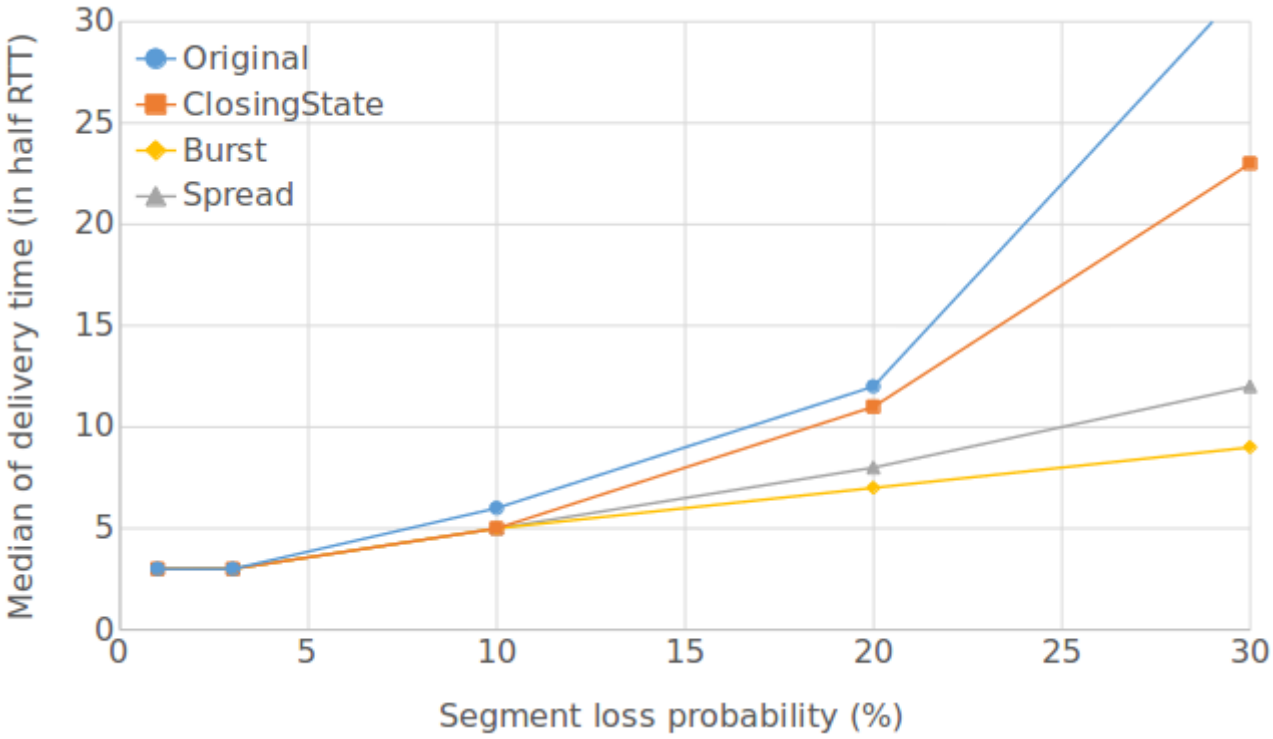

Figura 22: Risultati dei test in termini di mediana del Delivery Time in funzione della PER (Packet Error Ratio). Si può notare che per percentuali di perdite molto basse l'effetto dei miglioramenti sulla mediana sono molto piccoli, in particolare se PER<3% non si hanno miglioramenti. Per valori superiori al 20% invece si possono osservare notevoli differenze.

La seguente tabella mostra il 90° percentile del Delivery Time espresso in RTT/2, calcolato in funzione della PER:

PER	Original (RTT/2)	ClosingState (RTT/2)	Burst (RTT/2)	Spread (RTT/2)
1%	3	4	4	3
3%	6	6	3	4.1
10%	24	10	7	8
20%	44.2	20	9	12
30%	79	37	14	18

Riportando i seguenti risultati in un grafico si ha:

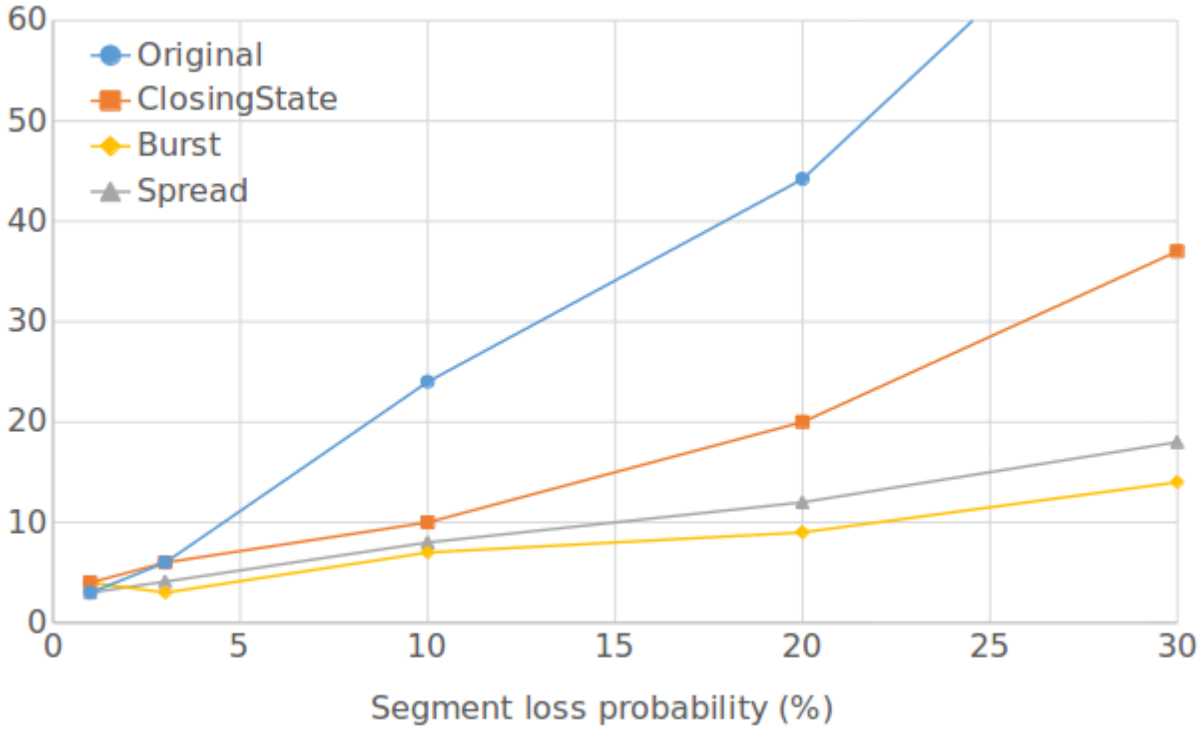

Figura 23: Risultati dei test in termini di 90° percentile del Delivery Time in funzione della PER (Packet Error Ratio).

Conclusioni

Nel corso della tesi è stato esaminato in dettaglio il comportamento del protocollo LTP in presenza di elevate perdite. Dopo avere individuato due possibili cause di debolezza nella mancata protezione dei segmenti di segnalazione, e nella mancanza di uno stato di closing nella procedura di chiusura di sessione, ne è stata quantificata la gravità in termini di RTT di penalizzazione, grazie all'introduzione di un modello originale, semplificato ma realistico in ambito spaziale. Si è quindi iniziato il progetto delle modifiche da introdurre per risolvere i problemi evidenziati, passando quindi alla loro implementazione all'interno della versione LTP presente in ION, il pacchetto DTN sviluppato e mantenuto dalla NASA-JPL.

Da ultimo, è stata valutata l'efficienza dei miglioramenti introdotti in termini di Delivery Time. I risultati preliminari mostrati nella tesi sono incoraggianti in quanto mostrano un notevole miglioramento dei valori medi per perdite molto elevate ed una significativa diminuzione della varianza, anche per PER relativamente basse, dovuta alla forte riduzione del Delivery Time nel caso peggiore.

Le modifiche introdotte, presentate in dettaglio nella tesi, sono pienamente compatibili con lo standard IETF del protocollo [RFC5326], anche se necessariamente non del tutto conformi.

Anche se le modifiche presentate riferiscono l'implementazione ION, l'analisi concettuale del quarto capitolo ha permesso di trarre delle conclusioni valide per tutte le implementazioni di LTP.

La mia speranza è che queste modifiche vengano prese in considerazione e possibilmente accettate in modo da poter essere incluse, se non altro come sperimentali, nella prossima versione di ION.

Bibliografia

[Apollonio] P. Apollonio, C. Caini ,T. de Cola, G. Liva *;* “Erasure Error Correcting Codes Applied to DTN Communications”, http://amslaurea.unibo.it/6852/4/apollonio_erasure_error_correcting_codes_applied_to_dtn_communications.pdf, Università di Bologna ,2014 , Last Access: 28/02/2016

[Caini] C. Caini;R. Firrincieli;T. de Cola;I. Bisio;M. Cello;G. Acar,”Mars to Earth communications through orbiters: Delay-Tolerant/Disruption-Tolerant Networking performance analysis”, «INTERNATIONAL JOURNAL OF SATELLITE COMMUNICATIONS AND NETWORKING», 2014, 32, pp. 127 - 140

[CCSDS 734.1-B-1] CCSDS 734.1-B-1 “Licklider Transmission Protocol (LTP) for CCSDS” , http://public.ccsds.org/publications/archive/734x1b1.pdf , Blue Book. Issue 1. May 2015 , Last Access: 28/02/2016

[CCSDS 734.2-B-1] CCSDS 734.2-B-1 “CCSDS Bundle Protocol Specification” , http://public.ccsds.org/publications/archive/734x2b1.pdf, Blue Book. Issue 1. September 2015 , Last Access: 28/02/2016

[Deutsch] L. Deutsch ;J. Yuen; S. Townes; “Science and Technology: Research Topic Details” , http://scienceandtechnology.jpl.nasa.gov/research/ResearchTopics/topicdetails/?ID=67 , NASA Jet Propulsion Laboratory – Last access: 21/12/2015

[Durst] Robert C. Durst; Patrick D. Feighery; Keith L. Scott ; “Why not use the Standard Internet Suite for the Interplanetary Internet?” http://www.ipnsig.org/reports/TCP_IP.pdf, 2002 , Last access: 28/02/20156

[DTNRG] “RTF Delay-Tolerant Networking Research Group (DTNRG)” , https://irtf.org/dtnrg,- Last Access: 28/02/2016

[DTN2_SW] “DTN2” https://sourceforge.net/projects/dtn/files/DTN2/ - Last Access: 28/02/2016

[DTNperf_3] C. Caini, A. D'Amico, M. Rodolfi, “A further enhanced tool for Delay-/Disruption- Tolerant Networking Performance evaluation”, Dec 2013

[IBR_DTN] “IBR-DTN” https://www.ibr.cs.tu-bs.de/projects/ibr-dtn/ Last Access: 28/02/2016

[ION_DOC] S. Burleigh; “Interplanetary Overlay Network (ION) - Design and Operation” ,https://sourceforge.net/projects/ion-dtn/files/ion-3.4.1.tar.gz/download, Jet Propulsion Laboratory , 2012, Last Access: 28/02/2015

[ION_SF] S. Burleigh; “Delay-Tolerant Networking suitable for use in spacecraft”

, http://sourceforge.net/projects/ion-dtn/ , Last Access: 28/02/2015

[Mann] A. Mann, “Google’s Chief Internet Evangelist on Creating the Interplanetary Internet" , http://www.wired.com/2013/05/vint-cerf-interplanetary-Internet/, Wired, 2013 – Last Access: 28/02/2015

[Nasa_Light] Unknown Author ,“How fast does light travel from the Sun to each of the planets?" , http://image.gsfc.nasa.gov/poetry/venus/q89.html, NASA's Goddard Space Flight Center , Last Access: 28/02/2015

[Padhye] Padhye, J., Firoiu, V., Towsley, D. and J. Kurose, "Modeling TCP Throughput: A Simple Model and its Empirical Validation", Proc. ACM SIGCOMM 1998

[RFC 793] Postel, J., “Transmission Control Protocol – Protocol Specification” , Internet RFC 793, Sep. 1981, https://tools.ietf.org/html/rfc793 Last Access: 28/02/2016

[RFC 4838] V. Cerf , S. Burleigh, A. Hooke, L. Torgerson, R. Durst ,K. Scott, K. Fall,H. Weiss, “Delay-Tolerant Networking Architecture” , Internet RFC 4838, April. 2007, https://tools.ietf.org/html/rfc4838 Last Access: 28/02/2016

[RFC 5050] K. Scott, S. Burleigh, “Bundle Protocol Specification”, Internet RFC 5050, Nov. 2007, https://tools.ietf.org/html/rfc5050 Last Access: 28/02/2016

[RFC 5325] S. Burleigh , M. Ramadas, S. Farrell, “Licklider Transmission Protocol – Motivation” , Internet RFC 5325, Sep. 2008, https://tools.ietf.org/html/rfc5325 Last Access: 28/02/2016

[RFC 5326] M. Ramadas, S. Burleigh , S. Farrell, “Licklider Transmission Protocol – Specification” , Internet RFC 5326, Sep. 2008, https://tools.ietf.org/html/rfc5326 Last Access: 28/02/2016

[RFC 5327] M. Ramadas, S. Burleigh ,S. Farrell, “Licklider Transmission Protocol – Security Extensions” , Internet RFC 5327, Sep. 2008, https://tools.ietf.org/html/rfc5327 Last Access: 28/02/2016

[RFC 5681] M. Allman,V. Paxson, E. Blanton, “TCP Congestion Control” , Internet RFC 5681, Sep. 2009, https://tools.ietf.org/html/rfc5681 Last Access: 28/02/2016

[RFC 6298] V. Paxson,M. Allman, J. Chu, M. Sargent, “Computing TCP's Retransmission Timer” , Internet RFC, June 2011, https://tools.ietf.org/html/rfc6298 Last Access: 28/02/2016

[RFC 7484] M. Duke , R. Braden,W. Eddy, E. Blanton,A. Zimmermann, “A Roadmap for Transmission Control Protocol (TCP) Specification Documents” , Internet RFC 7484, Feb. 2015, https://tools.ietf.org/html/rfc7484 Last Access: 28/02/2016

[Tanenbaum] Andrew S. Tanenbaum and David J. Wetherall, *Computer Networks*, ed. 5th, Prentice Hall, 2011.

[Virtualbricks] P. Apollonio, C. Caini, M. Giusti, D. Lacamera, “Virtualbricks for DTN satellite communications research and education”