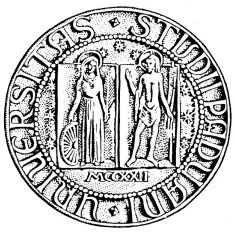


UNIVERSITÀ DEGLI STUDI DI PADOVA
LAUREA MAGISTRALE IN
INGEGNERIA DELLE TELECOMUNICAZIONI

STUDIO E SIMULAZIONE DI ALGORITMI DI ARQ IBRIDO PER SISTEMI DI TELECOMUNICAZIONE LTE

Autore: Giovanni Tomasi
Relatore: prof. Michele Zorzi
Correlatori: Marco Miozzo (CTTC), Marco Mezzavilla

Anno Accademico: 2010/2011
Data: 14 Marzo 2011

INDICE

INTRODUZIONE

Il *3GPP -3rd Generation Partnership Project-* si sta impegnando nella definizione delle nuove tecnologie le quali porteranno significativi miglioramenti alle precendenti quali UMTS/HSPA. *Bit rate* maggiori sia in *downlink* che in *uplink* e minori latenze di rete sono alcuni esempi di obiettivi da raggiungere.

Una delle principali caratteristiche che fornisce robustezza e permette l'aumento delle *performance* è l'*HARQ*. La tecnica *ARQ Automatic Retransmission reQuest* viene comunemente utilizzata da anni in sistemi di comunicazione per aumentare l'affidabilità delle trasmissioni. Questa semplice tecnica permette al ricevitore di richiedere una eventuale ritrasmissione di un pacchetto errato, tramite la serie di protocolli *stop and wait - go back N - selective repeat*.

La *HARQ -Hybrid Automatic Retransmission reQuest-* funziona allo stesso modo dell'*ARQ* con le non piccole differenze che l'informazione contenuta nelle ritrasmissioni può essere diversa *IR Incremental Redundancy* e quindi la rivelazione di errori nel pacchetto sarà in funzione del modo con cui si ricombinano le varie informazioni relative alle diverse ritrasmissioni *SC Soft Combining*. Viene appunto definita *ARQ Ibrida* perchè oltre alla richiesta di ritrasmissione che svolge la parte *ARQ* si affianca la *soft combining* che utilizza la *FEC Forward Error Correction*.

Di simulatori di rete e simulatori di *physical layer* ne sono stati creati diversi, usati come strumenti da diversi ricercatori sia in ambito aziendale che in ambito privato. Pochi di questi sono però disponibili al pubblico, sia

per l'utilizzo che per lo sviluppo di nuovi moduli. Un'eccezione ai simulatori proprietari è *ns-3 network simulator 3* simulatore di rete ad eventi discreti usato principalmente per scopi didattici e di ricerca, successore del noto *ns-2*. ns-3 è un framework libero protetto da licenza *GNU GPLv2* ed è completamente scritto in *C++* e in piccola parte in *Python*.

Struttura tesi

CAPITOLO 1: si riportano le principali caratteristiche dei sistemi *LTE*.

CAPITOLO 2: si introduce il perchè serve una astrazione del livello fisico in un simulatore come *ns-3* ed il come viene fatta con una focus marcato sull'astrazione per mezzo di misure dell'informazione mutua.

CAPITOLO 3: si spiega con accuratezza come composto il sistema *HARQ* in *LTE*, le sue varianti ed come si può modellizare tale tecnica per mezzo dell'utilizzo della *link abstraction*.

CAPITOLO 4: si riportano le caratteristiche e approssimazioni dell'implementazione del *downlink scheduler*, *harq entity* e *link abstraction* fatta prima su *octave/matlab* e successivamente su *ns-3* per poi mostrare e giustificare i risultati ottenuti.

APPENDICE: si riportano i *link* utili.

Ringraziamenti

Marco Mezzavilla, Marco Miozzo e prof. Michele Rossi per avermi permesso di svolgere la tesi

Gli amici e colleghi
Diego Altolini per l'effetto Taxi
Michael Sterchele per avermi insegnato cos'è un pèche
Slash per la sua filosofia nel prendere l'ingegneria
Federico Favaro e l'intera GE senza la quale ci sarebbe stata la depressione (o c'è stata? mah...)
i coinquilini Cracco Andrea e Fabrizio Zago per le sfide a Street Fighter II
e ovviamente la mia famiglia

La tesi è stata scritta in LaTeX 2ε

CAPITOLO

1

LTE OVERVIEW

Nel Novembre del 2004 il *3GPP* ha cominciato un progetto per definire l'evoluzione di lungo termine della tecnologia cellulare *UMTS* ovvero la *Long Term Evolution* della rete *UMTS*. Questo progetto si fa conoscere attraverso le specifiche della *Evolved UMTS Terrestrial Radio Access (E-UTRA)* e della *Evolved UMTS Terrestrial Radio Access Network (E-UTRAN)*. Una prima versione e raccolta si trova nella *Release 8* delle specifiche *3GPP* mentre per applicazioni commerciali ancora oggi nel 2010 ci si trova in fase di test. Si mostra la *road map* in figura 1.1

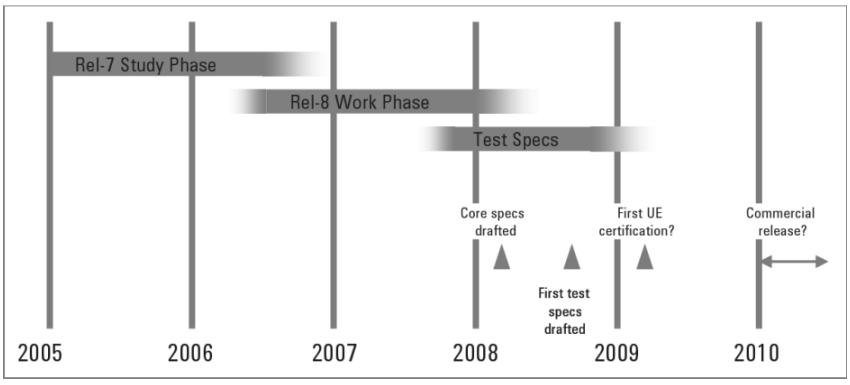


Figura 1.1: road map

Questo capitolo vuole solo riportare in breve le caratteristiche chiave, senza scendere nel dettaglio, anche perchè saggi e testi sulla descrizione dell'*LTE* si trovano duvunque. Riporto le informazioni essenziali per enumerazioni.

Gli obiettivi principali, motivo del tanto impegno, sono

- *performace* migliori
  - 100 [Mbit] di *rate* di picco in *downlink*, 50 [Mbit] in *uplink*
    - 1 [Gbit] per *LTE Advance*, l'evoluzione ulteriore dell'*LTE*
    - gestione utenti a bordo cella veloce
    - riduzione latenze per gli utenti (ordine dei 10 [ms])
    - banda scalabile fino ai 20 [Mhz]
- compatibilità
  - inter-operabilità con i sistemi *GSM/EDGE/UMTS*
  - utilizzo degli spettri 2G e 3G e nuovi spettri
  - supporto all'*handover* e *roaming* per le reti mobili esistenti
- svariate applicazioni
  - spettri *TDD* e *FDD*
  - mobilità fino a 350 [Km/h]
  - grande varietà di terminali (PC, telefonini, ...)

1.1 Architettura LTE

- *LTE* comprende l'evoluzione
  - della *radio access* attraverso *E-UTRAN*
  - degli aspetti 'non radio' sotto il termine *SAE Aystem Architecture Evolution*
- L'intero sistema composto sai da *LTE* che *SAE* viene chiamato *EPS Evolved Packet System*
- a livello più alto, la rete comprende
  - la *CN Core Network*, chiamata *EPC Evolved Packet Core* in *SAE*
  - la rete di accesso *E-UTRAN*.
- si definisce un *bearer* un flusso di pacchetti IP marchiati da una specifica *QoS Quality of Service* fra il *Gateway* ed l'*Ue User Terminal*
- la *CN* è la responsabile di tutto il controllo della *Ue* e dell'instaurazione dei *bearer*.

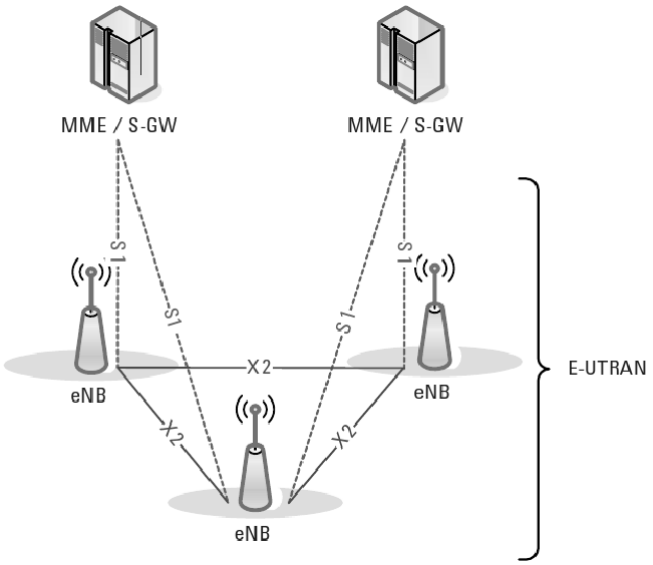


Figura 1.2: architettura LTE

- i nodi logici principali in *EPC* sono
  - *P-GW PDN Gateway*
  - *S-GW Serving Gateway*
  - *MME Mobility Management Entity*
- *EPC* include ulteriori nodi e funzioni quali
  - *HSS Home Subscriber Servin*
  - *PCRF Policy Control and Charging Rules Function*
- *EPS* fornisce un percorso al *bearer* con determinata *QoS* mentre il controllo per applicazioni multimediali viene fornito dall'*IMS IP Multimedia Subsystem*, considerato esterno all'*EPS*
- *E-UTRAN* contiene unicamente le base station 'evolute' chiamate *eNodeB* o *eNB*

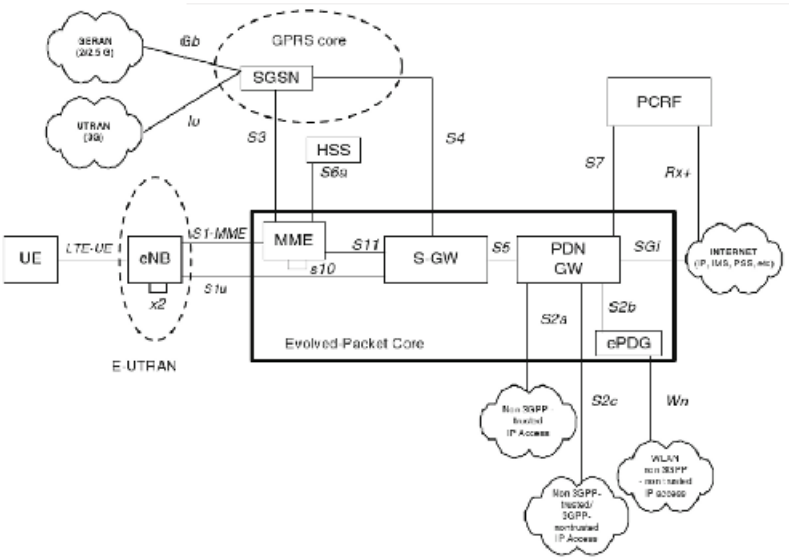


Figura 1.3: altra architettura LTE

1.2 Interfaccia Radio LTE

- le *eNodeB* ed le *Ue* sono dotate sia di protocolli di *control plane* che di *data plane*
- i dati entrano nel processo di trasmissione sotto forma di pacchetti IP provenienti dai *SAE bearer*

1.2.1 Livelli Protocollari

- un pacchetto IP passa attraverso entità multiple dei protocolli
  - **PDCP Packet Data Convergence Protocol**
    - compressione dell'*header* IP tramite *ROHC Robust Header Compression*
    - crittografia e protezione di integrità dei dati trasmessi
  - **RLC Radio Link Control**
    - segmentazione/concatenazione
    - gestione delle ritrasmissioni
    - *forward-up* in-sequeza per i *layers* superiori
  - **MAC Medium Access Control**
    - gestisce le ritrasmissioni *HARQ*
    - *scheduling uplink* e *downlink*

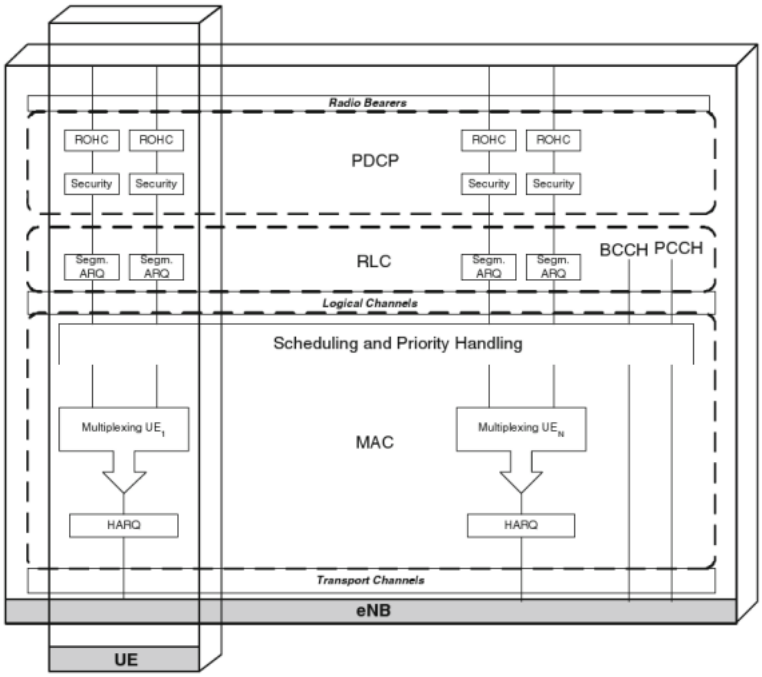


Figura 1.4: architettura radio LTE

- **PHY Physical Layer**
  - codifica/decodifica
  - modulazione/demodulazione
  - *multi-antenna mapping*
  - altre funzioni tipiche del livello fisico

1.2.2 Canali di Comunicazione

- *RLC* offre servizi al *PDCP* nella forma di *radio bearer*
- *MAC* offre servizi al *RLC* nella forma di canali logici
- *PHY* offre servizi al *MAC* nella forma di *canali di trasporto*
- un canale logico si caratterizza in funzione dell'informazione che porta, e generalmente si classifica in
  - canale di controllo, usato per trasmettere informazione di controllo e configurazione necessaria per l'operatività del sistema *LTE*
  - canale di traffico, usato per i dati utente
- un canale di trasporto si caratterizza per come e per quali caratteristiche l'informazione è trasmessa all'interfaccia radio

Si mostra ora in figura 1.5 la *channel mapping* fra i canali di diverso livello

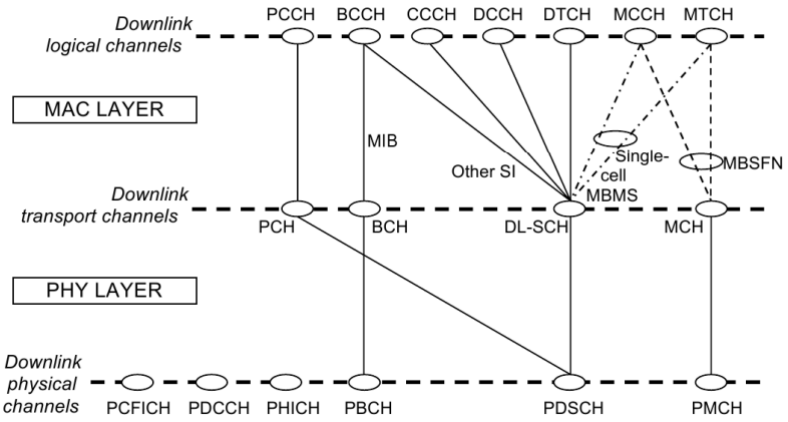


Figura 1.5: *channel mapping*

- *BCCH* broadcast
- *CCCH* common
- *PCCH* pagin
- *MCCH* multicast
- *DTCH* dedicated traffic
- *MTCH* multicast traffic
- *DL-SCH* downlink shared
- *MCH* multicast
- *BCH* broadcast
- *PCH* pagin

1.2.3 RLC Radio Link Control

- dipendentemente dalle decisioni dello *scheduler*, una certa qunatità di dati viene prelevata dal *buffer SDU Service Data Unit* dell'*RLC*, e viene segmentata/concatenata per formare la *RLC PDU Protocol Data Unit*. In *LTE* la dimensione delle *RLC PDU* varia dinamicamente

- ogni *RLC PDU* ha un *header* il quale contiene, tra le varie cose, un *sequence number* per la consegna in sequenza e per la gestione delle ritrasmissioni *ARQ*
- un meccanismo di ritrasmissione opera fra le entità *RLC* del trasmettitore e ricevitore, e serve come ultima icentificazione di errore dopo la *HARQ* al livello *MAC* la quale aumenta di molto la robustezza della trasmissione

1.2.4 MAC Medium Access Control

- i dati nel canale di trasporto sono organizzati in *TB Transport Block*
- ad ogni *TTI Transmission Time Interval* un solo *TB* di una certa dimensione viene trasmesso all'interfaccia radio (in assenza di *spatial multiplexing*)
- ogni *TB* ha associato un *TF Transport Format*, il quale specifica come il *TB* verrà trasmesso (*TB size*, *MCS Modulation and Coding Scheme*, *antenna mapping*)
- variando il *TF* si varia il *rate* di trasmissione, basti pensare a cosa la variazione della codifica, modulazione e eventuali tecniche *MIMO* comporta

Scheduler

- lo *scheduler eNodeB* controlla le risorse tempo frequenza sia per il *downlink* che per *uplink*. Dinamicamente controlla i terminali e per ognuno di essi determina quale risorse in *DL-SCH* gli verranno assegnate
- alloca dinamicamente risorse alle *Ue* ad ogni *TTI*
- la strategia di *scheduling* non è fissata dallo standard *3GPP* ma viene lasciata ai costruttori. Buone regole base per la strategia prevedono che lo *scheduler*
  - sceglie il miglior allocamento dati per le *Ue* in base alla loro condizione di canale
  - preferisca allocare risorse in trasmissione alla *Ue* con vantaggiose condizioni di canale
- una buona strategia di *scheduling* abbisogna di informazione
  - delle condizioni del canale al termiale
  - dello stato dei *buffer* e le priorità dei flussi dati
  - della situazione dell'interferenza dovuta alle celle vicine (se c'è una sorta di coordinazione fra le *eNB*)

indice CQI	modulazione	rate ·1024	bit per simbolo
0	out of range		
1	QPSK	78	0.15
2	QPSK	120	0.23
3	QPSK	193	0.38
4	QPSK	308	0.60
5	QPSK	449	0.88
6	QPSK	602	1.18
7	16QAM	378	1.48
8	16QAM	490	1.91
9	16QAM	616	2.47
10	64QAM	466	2.73
11	64QAM	567	3.32
12	64QAM	666	3.90
13	64QAM	772	4.52
14	64QAM	873	5.12
15	64QAM	948	5.55

Tabella 1.1: tabella degli *MCS* o *CQI* per *LTE* (*MCS* descritto a 4 [bit])

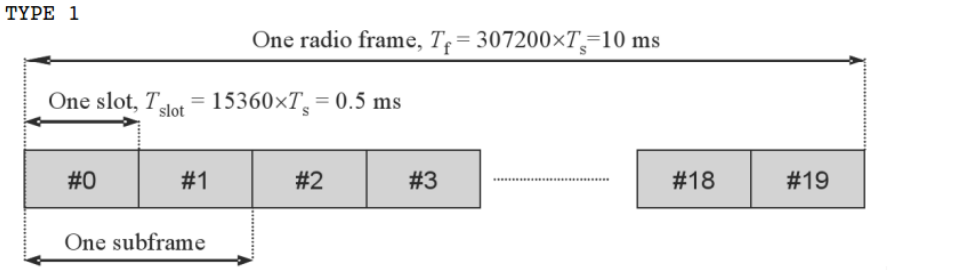
- le *Ue* trasmettono
  - lo stato del canale il quale descrive la qualità del canale sia nel tempo che in frequenza
  - informazione necessaria per il processing nel caso di *antenna mapping*
- lo *scheduler downlink LTE* considera questi algoritmi
  - *FSS Frequency Selective Scheduling*
  - *FDS Frequency Diverse Scheduling*
  - *PFS Proportional Fair Scheduling*
- il cordinamento per la gestione dell'interferenza inter-cella fa sempre parte dello *scheduler*

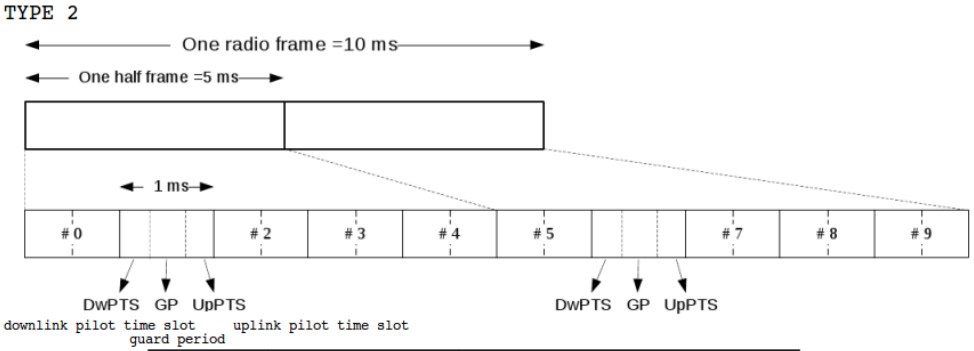
HARQ Hybrid-ARQ

- nella *H-ARQ* sono usati otto processi *stop and wait* per gestire la trasmissione e ritrasmissione dei *TB*
- il meccanismo di *HARQ* non è applicabile a tutti le tipologie di traffico (il traffico *broadcast* ne è esente mentre per *DL-SCH* e *UL-SCH* lo supporta)

1.2.5 PHY Physical Layer

- *LTE* utilizza trasmissioni *OFDMA* con prefisso ciclico in *downlink* e *SC-FDMA* con prefisso ciclico in *uplink*
- sono supportati tre modalità di *duplexing*: *full duplex FDD*, *half duplex FDD* e *TDD*
- ci sono due tipologie di *frame* di trasmissione
  - *type* 1, per trasmissioni *full* o *half duplex FDD*
  - *type* 2, per trasmissioni *TDD*
- un *frame* radio ha durata di 10 [ms] e contiene 20 *slots* (durata *slot* è 0.5 [ms])
- due *slots* adiacenti formano un *subframe* di durata 1 [ms]
- *LTE* supporta le modulazioni per sottoportante: QPSK, 16-QAM, 64-QAM
- il canale *broadcast* usa solo la modulazione QPSK
- dimensione massima di un blocco di informazione è 6144 [bit]
- per l'*error detection* si usano dei *CRC-24*





Configuration	Switch-point periodicity	Subframe number									
		0	1	2	3	4	5	6	7	8	9
0	5 ms	D	S	U	U	U	D	S	U	U	U
1	5 ms	D	S	U	U	D	D	S	U	U	D
2	5 ms	D	S	U	D	D	D	S	U	D	D
3	10 ms	D	S	U	U	U	D	D	D	D	D
4	10 ms	D	S	U	U	D	D	D	D	D	D
5	10 ms	D	S	U	D	D	D	D	D	D	D
6	10 ms	D	S	U	U	U	D	S	U	U	D

Figura 1.6: tipologie di *frame*

1.2.6 Trasmissione Downlink

- lo *scheduler eNB* alloca *RB Resource Block*, che sono la minima quantità di informazione allocabile, agli utenti nel tempo
- uno *slot* consiste in 6 (per prefisso ciclico lungo) o 7 (per prefisso ciclico corto) simboli *OFDM*
- il numero di sottoportanti disponibili varia dipendentemente dalla banda di trasmissione. Lo spazio fra le sottoportanti è fisso
- per permettere la stima di canale in trasmissioni *OFDM* vengono inseriti dei *reference symbols* nella griglia tempo frequenza *OFDM*
- tre tipi di *reference symbols* sono definiti per il *downlink LTE*
  - *reference symbols Cell-speficic*, trasmessi in ogni *downlink subframe* e coprono l'intera banda *downlink* della cella
  - *reference symbols Ue-speficic*, trasmessi all'interno di un *RB* assegnato ad una *Ue* in *DL-SCH*
  - *reference symbols MBSFN*

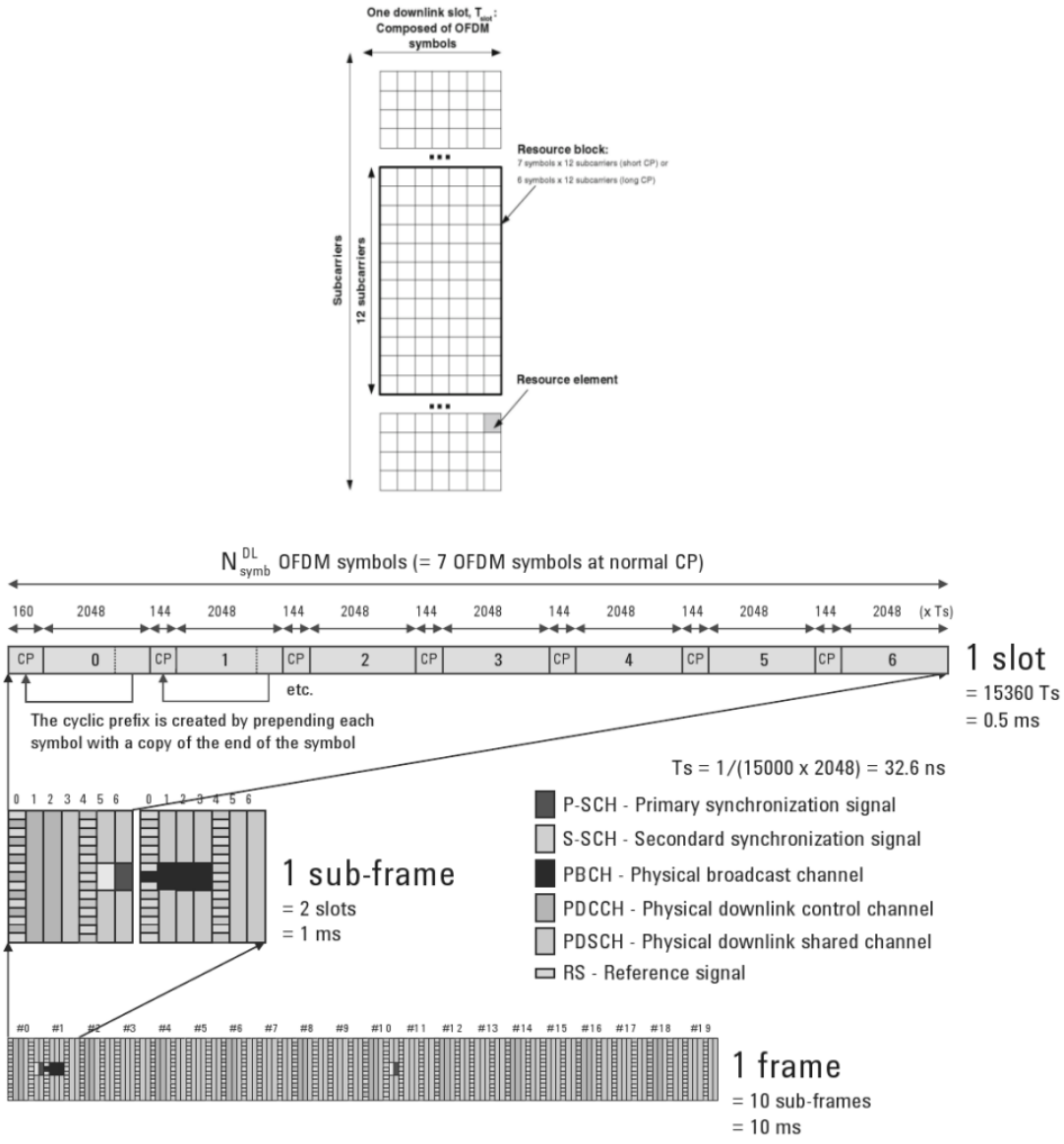


Figura 1.7: raffigurazione di un *RB* e di un *downlink frame*

CAPITOLO

2

LINK ABSTRACTION

L'obiettivo dell'astrazione del livello fisico *phycal layer abstraction* è quello di predirre in maniera accurata le performance come il *PER Packet Error Rate* con una bassa o comunque accettabile complessità computazionale. Un pacchetto può essere segmentato in più *code blocks*, e se per esempio lo si suddivide in J blocchi, la PER risulta

$$.PER = 1 - \sum_{j=1}^{J} (1 - BLER_j) \tag{2.1}$$

dove il *BLER Block Error Rate* è la probabilità di errore di ogni singolo *code block*.

Se si pensa di simulare in maniera reale il livello fisico ci si trova a dover elaborare segnali attraverso codificatori, modulatori, mappatori, convoluzioni col canale per non parlare di equalizzazioni e decodifiche ed il tutto per trasmissioni fra più stazioni radio base e terminali, e magari utilizzando anche tecniche di *MIMO Multiple Input Multiple Output*. Ci si rende subito conto, soprattutto per chi abbia frequentato esami di *trasmissione numerica* con annesse simulazioni *matlab*, della proibitiva complessità computazionale che avrebbe un tale simulatore di rete (simulazioni per qualche miglione di campioni di un singolo equalizzatore *forward-backward* prendono facilmente ore).

2.1 Link-Level simulator e System-Level simulator

Un'analisi di *link-level* permette di trovare le prestazioni fra un'*eNodeB* ed una *Ue*, le quali sono informazioni fondamentali di partenza. Le prestazioni in un ambiente reale, dove si prevede la presenza di più *eNodeB* le quali coprono un'area di servizio nella quale si muovono un grande numero *Ue* vengono valutate invece da un'analisi di *system-level*.
Una rappresentazione grafica dei principali organi di un *systen-level simulator* la si trova in figura 2.1. Interpretando il diagramma come rappresentazione

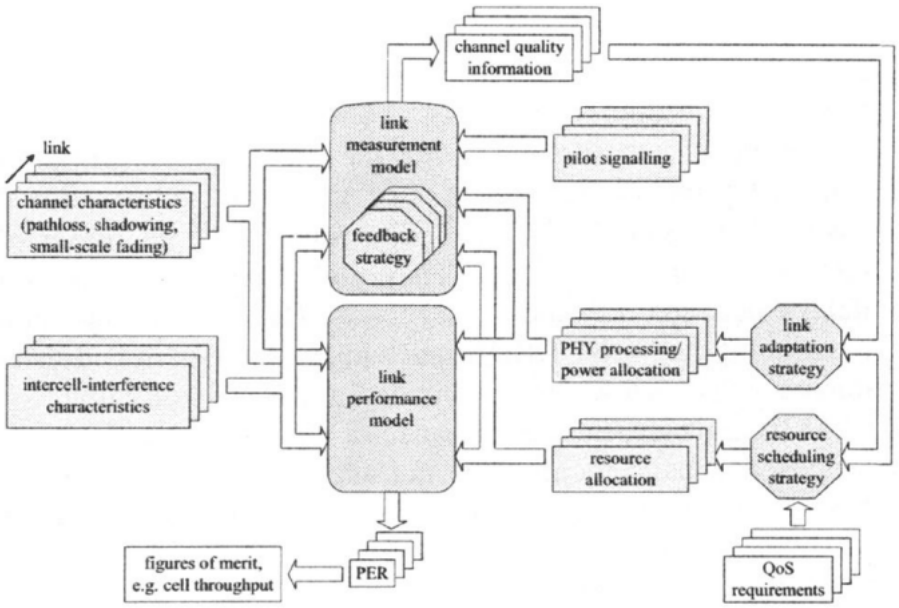


Figura 2.1: schema a blocchi di un simulatore system-level

di una singola cella si notano subito come i blocchi periferici del tipo *channel quality information* oppure *channel characteristic* sono a multipla istanza in accordo col numero di *Ue* presenti nella cella, infatti per ogni utente ci saranno le sue condizioni di canale ed la sua elaborazione del segnale, ma si nota con maggior enfasi la presenza dei due blocchi centrali *link measurement model* e *link performance model*. Essi sono il fulcro del sistema. Il *link measurement model* modella le misure che devono essere effettuate per la *link adaptation* ed la *resource allocation* mentre il *link performance model* serve per trovare il *PER* dipendentemente dalla *power allocation* e dall'elaborazione del segnale. Si sottolinea come tutti e due i modelli infieriscono nell'analisi del *PER*, però mentre il *measurement model* dipende strettamente dal sistema fisico in cui si opera, tipo di transceiver, strategie d'accesso, multiplexing

spaziale, e quindi è difficile una sua generalizzazione, il *link performance model* ha la possibilità di essere generalizzato al fine di poter comparare diversi sistemi con *measurement model* differente (*Wi-Fi* o sistemi basati su modulazione *OFDM* come *WiMax* o *LTE*). L'attenzione nei prossimi paragrafi verrà concentrata tutta sul *performance model*.

2.2 Link Performance model

Volendo procedere con l'astrazione del livello fisico ci si pone la domanda su come fare. Da ora in avanti le considerazioni verranno fatte solo per il sistema *LTE* anche se si possono tener buone per qualsiasi sistema basato su modulazione *OFDM*. Supponendo di aver già a disposizione un *measurement model* più o meno buono il quale fornisce le misurazioni dei *SINR Signal to Interference plus Noise Ratio* si vuol stimare la *BLER* relativa alla trasmissione di un *code block* su un determinato numero di sottoportanti. Dato le caratteristiche *LTE* di *fast resource scheduling* e *link adaptation* la stima della *BLER* dovrà tener conto delle condizioni istantanee di canale e di interferenza. Un modello generale adatto per questo scopo può essere quello di figura 2.2.

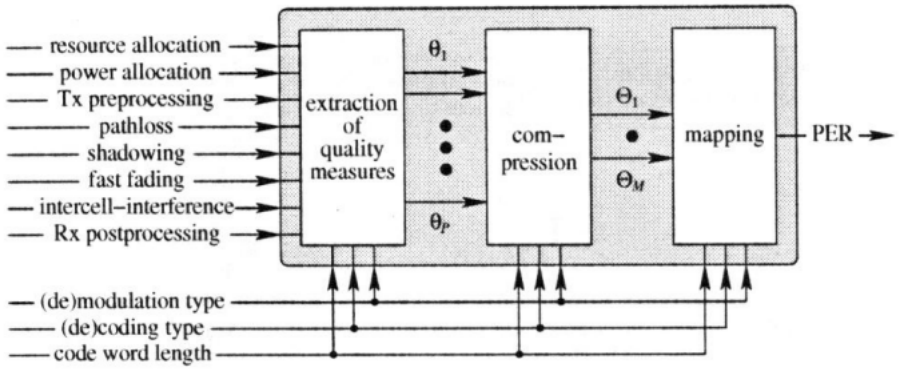


Figura 2.2: link-performance model generico

Dall'insieme di tutte le caratterisctiche del sistema vengono estratte p misure di qualità, per $p \in [1, P]$, riferite ad un *code block* (che nel nostro caso saranno direttamente i *SINR* relativi a tutte le sottoportanti assegnate ad un *code block*). Tipicamente questo insieme di paramentri viene poi compresso in uno o due paramentri θ_m questo per semplificare l'operazione finale di mappatura nel relativo *BLER* (in figura si vede scritto *PER* ma sarebbe è più corretto considerarlo come *BLER* in quanto si tratta del tasso di errore di un

code block non di un intero pacchetto, che spesso viene appunto segmentato in più *code block* allocati anche su diversi *subframe*).

Per diminuire il più possibile la complessità di un simulatore si utilizzano un solo parametro compresso θ ridefinito col nome di *eSinr effective Sinr* ed una funzione di mappatura uni-dimensionale fra il θ ed il *BLER*. Viene definita l'equazione che modella in maniera generale il blocco di compressione

$$SINR_{eff} = \alpha_1 \cdot I^{-1}\left(\frac{1}{P}\sum_{p=1}^{P} I\left(\frac{SINR_p}{\alpha_2}\right)\right) \tag{2.2}$$

considerando come misure di qualità i *system level SINR* delle diverse sottoportanti, ovvero i *SINR* derivanti da una simulazione che tiene conto dell'interferenza prodotta dalla presenza di più *Ue*. I parametri α_1 ed α_2 sono introdotti al fine di considerare gli effetti del codice e della modulazione, ovvero i diversi *MCS*, anche se per questioni di semplicità il più delle volte vengono posti uguali $\alpha_1 = \alpha_2 = \beta$, e così verrà fatto fino alla fine della tesi. Poichè la *mapping function* è una funzione fissata a priori, la qualità della stima del *BLER* dipende fortemente dalla funzione I.

In letteratura sono state proposte varie funzioni I:

La metrica *EESM Exponential Effective Sinr Metrics*

$$SINR_{eff} = -\beta \cdot ln\left(\frac{1}{P}\sum_{p=1}^{P} exp^{\left(-\frac{SINR_p}{\beta}\right)}\right) \tag{2.3}$$

con

$$I(SINR_p) = exp^{(-SINR_p)} \tag{2.4}$$

ed I derivata dal limite di *Chernoff* sulla probabilità di errore.

La metrica *CESM Capacity Effective Sinr Metrics*:

$$SINR_{eff} = -\beta \cdot \left(2^{\frac{1}{P}\sum_{p=1}^{P} log_2\left(1+\frac{SINR_p}{\beta}\right)} - 1\right) \tag{2.5}$$

con

$$I(SINR_p) = log_2(1 + SINR_p) \tag{2.6}$$

La metrica *LESM Logaritmic Effective Sinr Metrics*:

$$SINR_{eff} = 10^{-\beta \cdot \sigma_l^2 (log_{10}(SINR_p))} \cdot 10^{\frac{1}{P}\sum_{p=1}^{P} log_{10} SINR_p} \tag{2.7}$$

con

$$I(SINR_p) = log_{10}(SINR_p) \tag{2.8}$$

usando

$$\alpha_1 = 10^{-\beta \cdot \sigma_l^2},\ \alpha_2 = 1 \tag{2.9}$$

e σ_l^2 = varianza di $SINR_p$

La metrica *MIESM Mutual Information Effective Sinr Metrics* la quale verrà ampiamente trattata dopo aver mostrato le prestazioni delle diverse metriche.

2.2.1 Ottimizzazione del *Performance model*

Lo scopo del *link performance model* è quello di stimare più accuratamente possibile la *BLER* per una trasmissione. Fissata quindi una metrica da utilizzare rimane il compito di trovare i valori di β per ogni *MCS* ed fissare la *mapping function*. Nell'ipotesi di simulazine con una canale *AWGN* ci si trova ad evere $SINR_{eff} = SINR_p$,$p \in [1, P]$, per le metriche *CESM*, *EESM* ed *MIESM*. Questo significa che la curva delle prestazioni ($SINR \longleftrightarrow BLER$) con canale *AWGN* è esattamente la *mapping function* voluta. Si fissa appunto la *mapping funcion* per procedere poi a trovare il paramentro β.

Considando di avere a disposizione un simulatore completo di livello fisico, figura 2.3

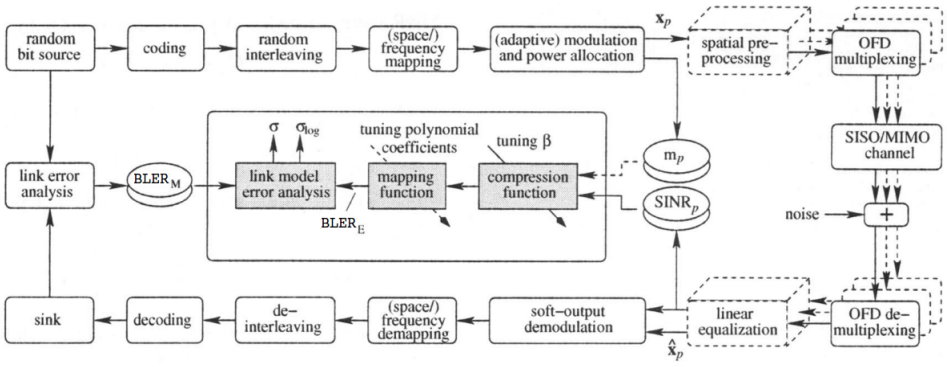


Figura 2.3: simulatore link-level e link-performance traning

si possono trovare i β ottimi tali per cui il $BLER_E$ *BLER Estimated* risultante dal *performance model* sia prossimo il più possibile al $BLER_M$ *BLER Measured* ottenuto dalla vera catena di equalizzazione e decodfica. Una possibile soluzione consiste nell'utilizzare il metodo ai minimi quadrati

$$\beta_{opt} = \arg\min_{\beta} \sum_{c=1}^{N_c} \mid \Delta e_c(\beta) \mid^2 \tag{2.10}$$

MCS	codifica e *rate*	modulazione	lunghezza parola di codice [bit]
1	CC 1/2	QPSK	408
2	CC 1/2	16-QAM	824
3	CC 3/4	16-QAM	1240
4	TC 1/3	16-QAM	544

Tabella 2.1: schemi di modulazione e codifica *(CC Convolutional Coding, TC Turbo Coding)*

con

$$\Delta e_c(\beta) = BLER_{E,c}(\beta) - BLER_{M,c} \tag{2.11}$$

dove N_c sono il numero di canali differenti usati nel processo di ottimizzazione (due canali si possono dire differenti se la matrice di canale o il rumore oppure la densità di potenza dell'interferenza differisce).

2.2.2 Risultati

I risultati sono stati presi da un articolo nel quale si spiega come sia stato utilizzato un sistema *link-level OFDM* del tipo in figura 2.3 con 416 sotto-portanti spaziate di $39,0625$ [KHz] ($Banda_{canale} \approx 16.25$ [MHz]). I prossimi risultati serviranno ad evidenziare la metrica migliore e più performante ed a giustificare la scelta di tale metrica per il simulatore *ns-3-lte*.

In tabella 2.1 si riportano i vari *MCS*, mentre in tabella 2.2 si riportano i valori di β_{opt} ottenuti con *AWGN based mapping function* dimostrando come effettivamente l'ottimizzazione sia fattibile.

Si riportano ora le prestazioni delle varie metriche in funzione della curva reale della *BLER* e degli *MCS*. In tutte e due le figure si osserva come le stime della *BLER* seguano più o meno bene l'andamento reale descritto dalla curva tratteggiata. Nella figura 2.4 si mostra come *LESM* e *CESM* danno stime più confuse e meno accurate delle stime con *EESM* ed *MIESM*, e tutto questo per dati protetti da codici convoluzionali.
Per la figura 2.5 si possono fare le stesse considerazioni dove però questa volta le stime più accurate vengono eseguite da *CESM* ed *MIESM* e per dati codificati con turbo codici.

MCS	metrics	β_{opt}
1	LESM	-0.27
1	CESM	0.09
1	EESM	1.64
1	MIESM	1.11
2	LESM	-0.18
2	CESM	0.21
2	EESM	5.25
2	MIESM	1.14
3	LESM	0.37
3	CESM	0.00
3	EESM	7.47
3	MIESM	1.07
4	LESM	-0.36
4	CESM	0.92
4	EESM	3.24
4	MIESM	1.02

Tabella 2.2: ottimizzazione di β per metriche diverse

Si può quindi affermare come la metrica basata sull'informazione mutua sia la più adatta e porti ad ottime prestazioni. Aggiungo inoltre che ai fini della tesi, sviluppata seguendo lo standard *LTE*, siano più rilevanti le prestazioni ottenute con turbo codici in quanto nella tecnica *HARQ* viene utilizzato un codificatore turbo, motivo in più per scegliere la metrica *MIESM* (si guardi bene le prestazioni per *MCS* 4).

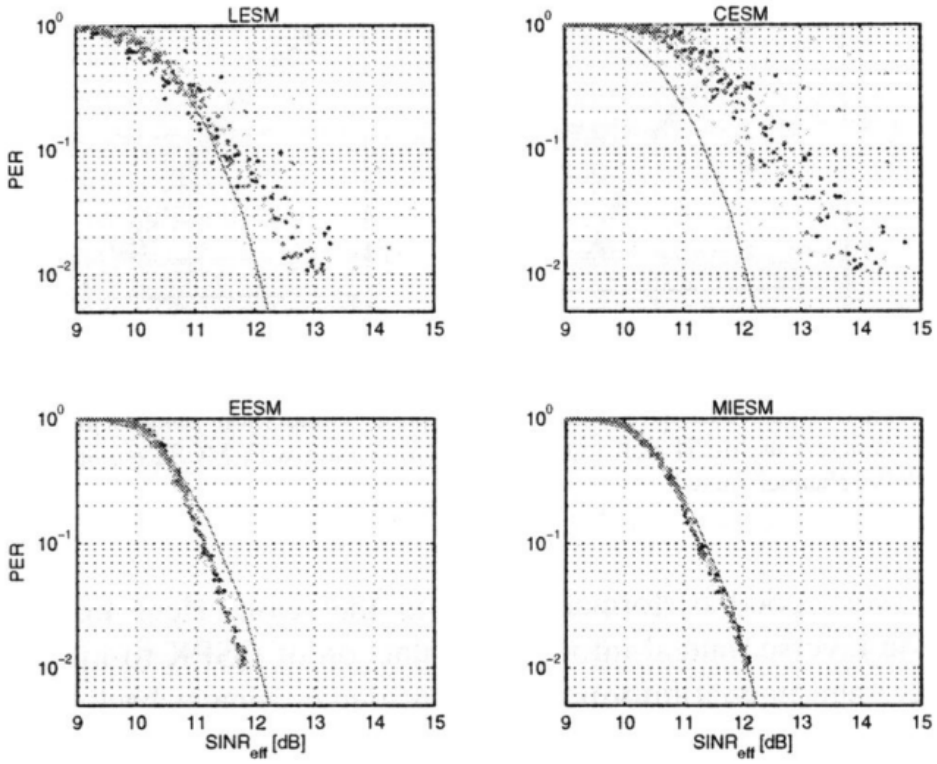


Figura 2.4: performance con *MCS* 1

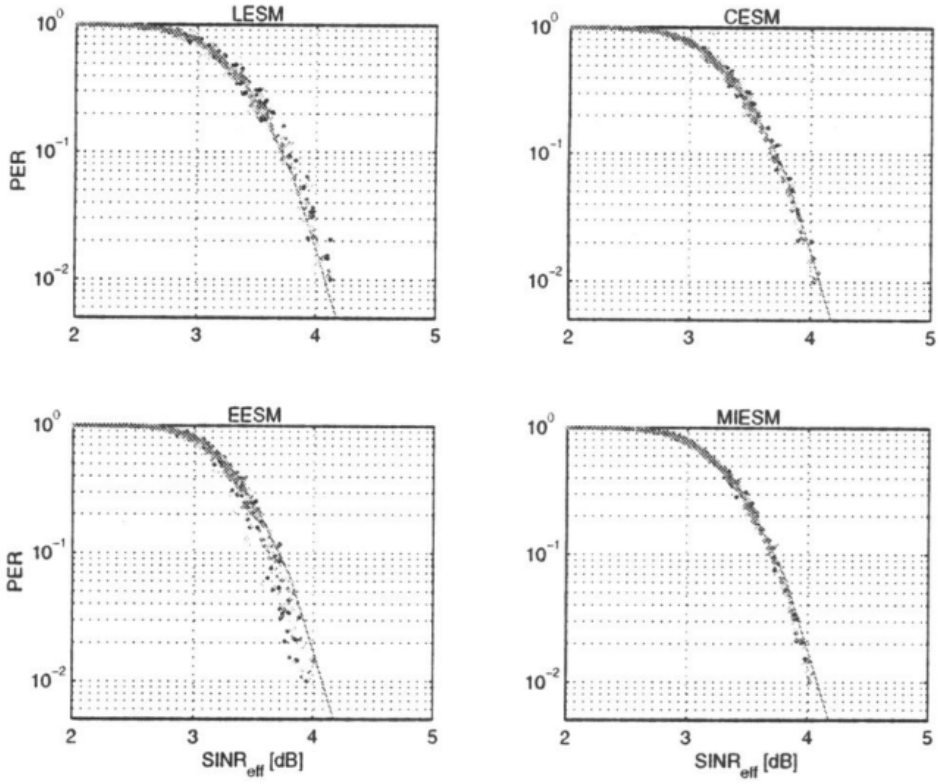


Figura 2.5: performance con *MCS* 4

2.3 Mutual Information Effective Sinr Metrics

Questa metrica si basa sul calcolo dell'informazione mutua fra i bit/simboli trasmessi e i bit/simboli ricevuti. Questo calcolo vuole quantificare il grado di somiglianza che c'è fra l'informazione trasmessa e quella ricevuta.

L'accuratezza di una metrica *MIESM* dipende dal canale equivalente sul quale la metrica viene definita. La capacità è l'informazione mutua basata su un canale gaussiano con ingresso gaussiano. La capacità limitata dalla modulazione è l'informazione mutua di un canale a livello di simbolo.

Il calcolo dell'informazione mutua per bit codificato può essere fatto attraverso l'informazione mutua a livello di simbolo ricevuto, in questo caso l'approccio si chiama *RBIR Received Bit mutual Information Rate*. Un metodo alternativo è quello di calcolare direttamente l'informazione mutua a livello di bit con l'approccio *MMIB Mean Mutual Information per Bit*.

Lo schema procedurale *MIESM* è raffigurato in figura 2.6. Per ogni sottoportante si ha il relativo *SINR*. I simboli contenuti su un *coded block* tramesso sulle sottoportanti $n \in N$ saranno affetti dal $SINR_n$ per la durata di un *subframe* (1 [ms]).

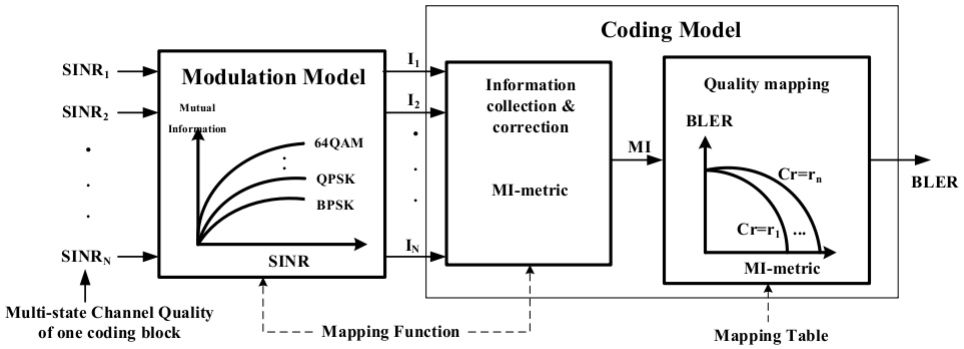


Figura 2.6: *phy abstraction* attraverso metrica *MIESM*

2.3.1 Received Bit Mutual Information Rate

Per un sistema *SISO/SIMO* l'informazione mutua per simbolo *SI Symbol Information* è data dalla formula

$$SI(SINR_n, m(n)) = log_2 M - \frac{1}{M} \cdot \sum_{m=1}^{M} E_U \left(log_2 \left(1 + \sum_{k=1, k \neq m}^{M} exp\left(\frac{- \mid X_k - X_m + U \mid^2 - \mid U \mid^2}{(1/SINR_n)} \right) \right) \right) \tag{2.12}$$

dove U è una variabile gaussiana complessa a media nulla e varianza $1/(2SINR_n)$, $m(n)$ è il numero di bit contenuti nel n_{-esimo} simbolo, M l'ordine di modulazione ed X è il segnale tramesso.

La formula mostra come l'informazione per simbolo sia $log_2 M$ al quale valore viene sottratto un fattore correttivo che per $SINR_n \to$ inf è nullo. Sostanzialmente riconduce al concetto logico che dice che se ad una sottoportante è associato un *SINR* basso allora la probabilità che il simbolo ricevuto differisca dal simbolo trasmesso è maggiore e quindi l'informazione mutua fra i simboli diminuisce.

Assumendo di usare N sottoportanti per trasmettere un *code block*, l'informazione mutua normalizzata per bit ricevuto (*RBIR*) risulta

$$RBIR = \frac{\sum_{n=1}^{N} SI(SINRNn, m(n))}{\sum_{n=1}^{N} m(n)} \tag{2.13}$$

Le curve di $SI(SINR_n, m(n))$ vengono generate con simulazioni di sistema con anessa implementazione di livello fisico, dalle quali poi si ricava il *RBIR* per mapparlo succesivamente nella *BLER*.
Queste curve non sono state trovate quindi non si è potuto proseguire per questa strada, ma si è deciso di utilizzare l'approccio *MMIB*.

2.3.2 Mean Mutual Information per Bit

La più accurata approsimazione che possiamo fare sulle prestazioni del decodificatore, la si può ottenere definendo un canale di informazione a livello del decoder stesso ovvero analizzando l'informazione mutua fra i bit di ingresso e la *LLR Log Likehood Ratio*. La *LLR* è un parametro calcolato dal decoder ed indica se il bit decodificato *'assomiglia'* più a 1 o 0

$$LLR(b_i) = ln\left(\frac{P[y \mid b_i = 1]}{P[y \mid b_i = 0]}\right) \tag{2.14}$$

dove y è il simbolo ricevuto.

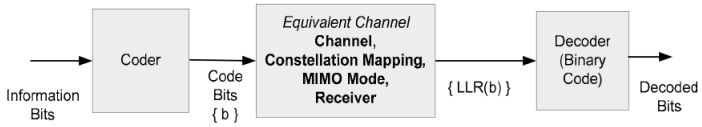


Figura 2.7: canale di informazione a *decoder-level*

L'informazione mutua *MI* di un bit codificato e bit ricevuto dipende dalla costellazione a cui appartiene e può esprimersi

$$I(b, LLR) = \frac{1}{m} \sum_{i=1}^{m} I(b_i, LLR(b_i)) \tag{2.15}$$

dove $I(b_i, LLR(b_i))$ è l'informazione mutua fra il bit di ingresso ed la *LLR* dell'i-esimo bit nella mappa di modulazione. Come si vede, la *LLR* riflette il

a_1	-0.04210661
a_2	0.00181492
b_1	0.209252
b_2	-0.142675
c_1	-0.00640081
c_2	-0.0822054
d_2	0.0549608

Tabella 2.3: parametri per la funzione J

processo di demodulazione , caratteristica non modellata nella *symbol level MI* di *RBIR* descritto sopra e questa è la differenza principale fra i due metodi *MIESM*.

Più in generale l'informazione mutua media considerando l'osservazione di N simboli di una parola di codice può esser trovata attraverso la formula

$$MI = \frac{1}{mN} \sum_{n=1}^{N} \sum_{i=1}^{m} I(b_i, LLR(b_i)) \tag{2.16}$$

La funzione $I(b_i, LLR(b_i))$ è strettamente dipendente dal *SINR* del simbolo *qam*, quindi si può esprimere la formula precedente in maniera più semplificata

$$MI = \frac{1}{mN} \sum_{n=1}^{N} \sum_{i=1}^{m} I_{m,b_i^{(n)}}(SINR_n) = \frac{1}{N} \sum_{n=1}^{N} I_m(SINR_n) \tag{2.17}$$

L'informazione mutua dipende dal *SINR* di ogni simbolo modulato (indice n) e dall'indice del bit codificato i, e varia in funzione dell'ordine di costellazione m.

Per la *BPSK*, la *LLR PDF* risulta gaussiana ed la *MIB Mutual Information per Bit* può esser espressa della funzione J

$$J(sinr) \approx \begin{cases} a_1 sinr^3 + b_1 sinr^2 + c_1 sinr, & \text{se } x \leq 1.6363 \\ 1 - exp^{(a_2 sinr^3 + b_2 sinr^2 + c_2 sinr + d_2)}, & \text{se } x > 1.6363 \end{cases}$$

E' stato dimostrato la *LLR PDF* per tutte le altre modulazioni può essere approssimata da un'insieme di distribuzioni gaussiane le quali non si sovrappongono ad alti *SINR*. Quel che ne risulta è che il corrispondente *MIB* può essere espresso come somma delle funzioni J

$$I_m(sinr) = \sum_{k=1}^{K} a_k J(c_k sinr), \text{ con } \sum_{k=1}^{K} a_k = 1 \tag{2.18}$$

funzione *MIB*		**approsimazione numerica**
$I_2(sinr)$	*QPSK*	$J(2\sqrt{sinr})$
$I_4(sinr)$	16-*QAM*	$\frac{1}{2}J(0.8\sqrt{sinr}) + \frac{1}{4}J(2.17\sqrt{sinr}) + \frac{1}{4}J(0.965\sqrt{sinr})$
$I_6(sinr)$	64-*QAM*	$\frac{1}{3}J(1.47\sqrt{sinr}) + \frac{1}{3}J(0.529\sqrt{sinr}) + \frac{1}{3}J(0.366\sqrt{sinr})$

Tabella 2.4: *MI* a seconda delle modulazioni

Le funzioni ottimizzate per le modulazioni *QPSK*, *16-QAM* e *64-QAM* sono riportate in tabella 2.4

La figuta 2.8 mostra l'andamento dell'informazione mutua per bit a seconda delle varie modulazioni

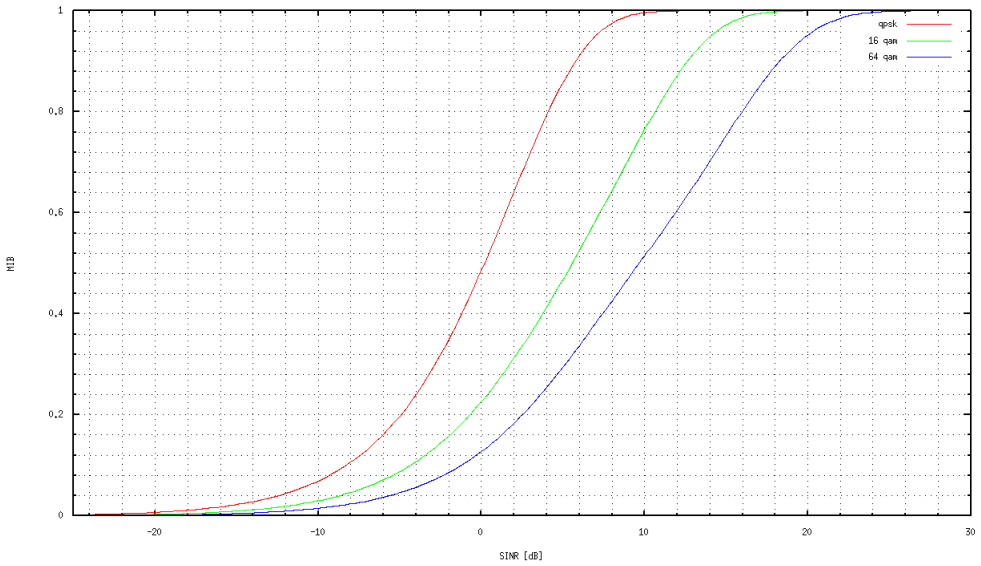


Figura 2.8: mappa *SINR-MIB*

Una volta calcolato il $MMIB = \frac{1}{N}\sum_{n=1}^{N} I_m(SINR_n)$ sulle sottoportanti su cui trasmettiamo il *code block* serve una *mapping function* che ci permetta di arrivare all'agognato *BLER*.
Si possono usare delle *AWGN reference curves* per tutti i tipi di *MCS* oppure approssimare tutte le curve con delle funzioni parametriche, come si riporta di seguito

$$BLER_{MCS} = \frac{1}{2}\left[1 - erf\left(\frac{MMIB - b_{MCS}}{\sqrt{2}\cdot c_{MCS}}\right)\right], c \neq 0 \tag{2.19}$$

In figura 2.9 si mostrano le curve *MMIB-BLER* parametrizzate usando 6 diffrenti *MCS* con rate 1/2e3/4 per lo standard 802.16e. Si nota subito che in prima approssimazione le curve possono essere assunte indipendenti dalla modulazione *QAM*, diversamente dal *code rate* e dalle lunghezze dei *code block*.

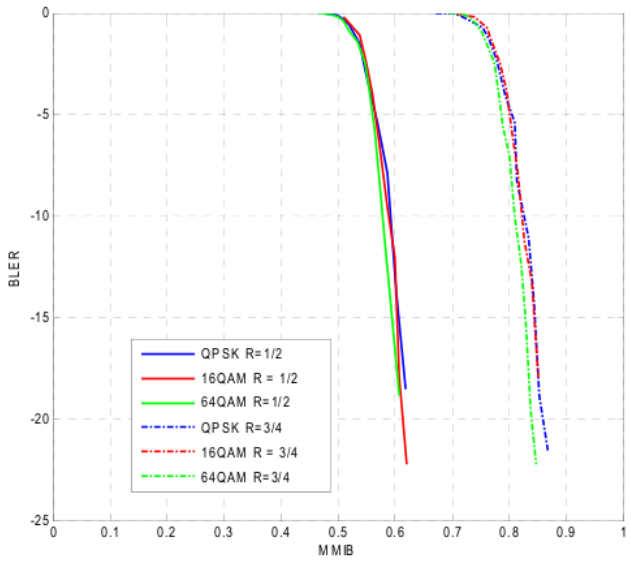


Figura 2.9: mappa *MMIB-BLER*

Si può quindi portare un'ulteriore approsimazione nel quale i parametri b e c dipendono solo dal *code rate* e dalla *code block size*, perciò

$$BLER_{BCR} = \frac{1}{2}\left[1 - erf\left(\frac{MMIB - b_{BCR}}{\sqrt{2} \cdot c_{BCR}}\right)\right], c \neq 0 \qquad (2.20)$$

Si riporta in tabella 2.5 i valori di b e c trovati con simulazioni con canale *AWGN*

BCR	*code rate*	lunghezza parola di informazione [bit]	lunghezza parola di codice [bit]	b_{BCR}	c_{BCR}
1	1/2	432	864	0.5512	0.0307
2	1/2	480	960	0.5512	0.0307
3	3/4	432	576	0.7863	0.03375
4	2/3	384	576	0.7082	0.0300
5	5/6	480	576	0.8565	0.02622

Tabella 2.5: parametri per la *mapping function* approssimata

La figura 2.10 mostra l'andamento del *BLER* in funzione del parametro *MMIB*

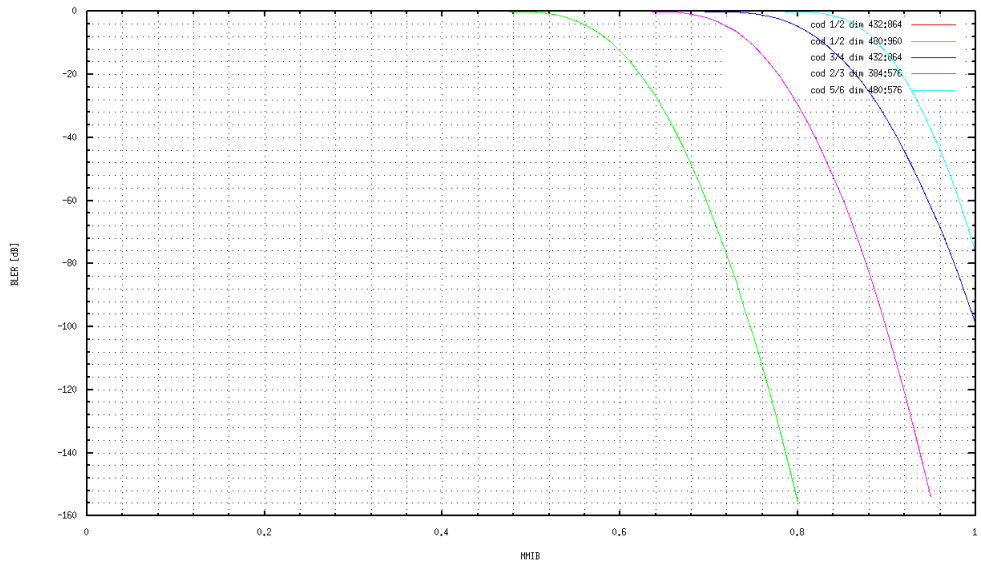


Figura 2.10: mappa *MMIB-BLER*

CAPITOLO

3

HARQ

La tecnica *HARQ Hybrid ARQ* serve per aumentare la robustezza della trasmissione.

E' un meccanismo che permette di chiedere le ritrasmissioni di pacchetti o informazione inerente ad un pacchetto ed in un certo senso lo si puó vedere come una retroazione grazie al quale si informa il trasmettitore che il pacchetto ricevuto è corrotto.

Il protocollo *HARQ* fa parte del *MAC layer* mentre il meccanismo di *soft combining* è gestito dal *physical layer*, motivo per il quale la *soft combining* subirà una procedura di astrazione. Si riporta la figura 3.1 esplicativa sulle relazioni fra i vari blocchi dei livelli *PHY-MAC*.

3.1 Harq a livello *MAC*

Lo *scheduler* comunica con l'entità *HARQ* per informarsi se ci sono eventuali ritrasmissioni e nel caso applica la sua politica di schedulazione decidendo quando e come ritrasmettere.
L'*HARQ* non viene utilizzata per tutti i tipi di traffico, per esempio il traffico *broadcast* ne è esente. La tecnica viene usata nei canali *DL-SCH* ed *UL-SCH*.

In *LTE* il protocollo *HARQ* consiste in una sequenza di otto processi *stop and wait*. Ad ogni *TTI* lo *scheduler*, nel caso non ci siano ritrasmissioni, prende vari pacchetti dai *bearer* e compone un *TB transport block*, e poi tramite l'*HARQ entity* da il compito a un processo libero di trasmetterlo.

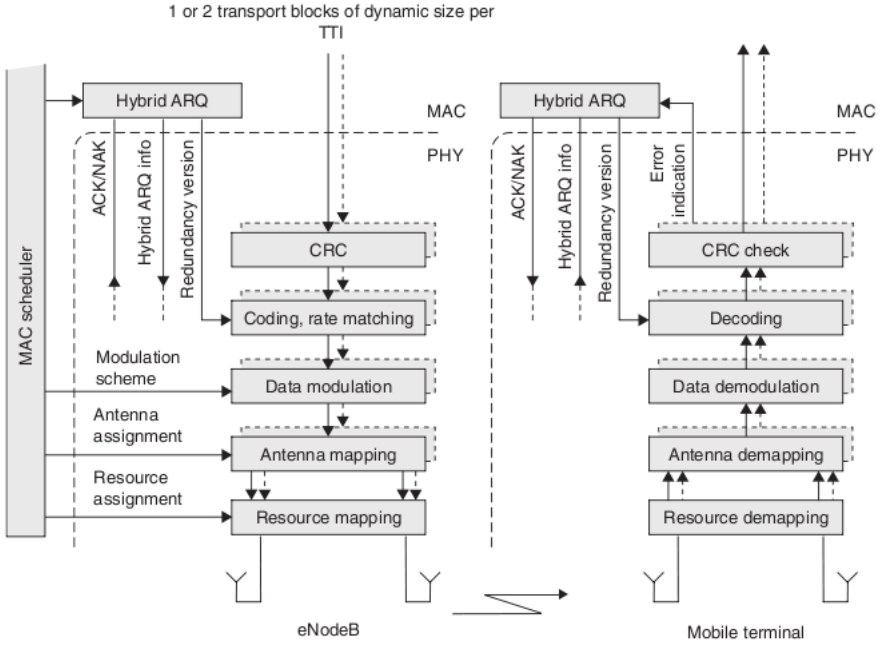


Figura 3.1: blocchi funzionali dei livelli *MAC* e *PHY*

Inoltre crea il messaggio di controllo da trasmettere nel canale fisico di controllo *PDCCH Physical Downlink Control Channel* dove viene scritta la mappa di allocazione, il *MCS* ed l'informazione *HARQ* la quale contiene il processo harq a cui è stato assegnato il *TB*, il parametro *new data indicator* che specifica quando un *TB* contiene dati nuovi oppure una ritrasmissione ed il parametro *RV Redundancy Version* che specifica il numero di ritrasmissione. Si riporta un esempio di messaggio in tabella **??**.

Il ricevitore cercherà di decodificare il *TB* ed informerà il trasmettitore sull'esito positivo o negativo della decodifica, e nel caso di esito negativo ne richiederà la ritrasmissione. Chiaramente il trasmettitore deve sapere a che processo è associato un *TB* trasmesso per poi capire a che processo direzionare le ricezioni degli *acknowledgement*. La figura 3.2 mostra gli otto processi *HARQ* al lavoro.

In questo specifico caso un *TB* è stato segmentato in *3 code blocks* ed i processi hanno la capacità di gestire le ritrasmissioni di singoli *code block*.

Il canale sul quale l'*eNodeB* trasmette gli *acknowledgement* è il *PHICH Physical Hybrid-ARQ Indicator Channel* mentre i canali sui quali un terminale *Ue* trametta gli *acknowledgement* sono il *PUSCH Physical Uplink Shared Channel*, nel caso abbia opportunità di farlo, oppure nel *PUCCH*, nel caso l'*eNodeB* non gli abbia riservato risorse (si ricorda che è la *base station* ad allocare risorse alle *Ue*). Inoltre in *LTE* il ritardo di trasmissione degli

DCI format 1A
resource block assignment
modulation and coding scheme
redundancy version
new data indicator
harq process number
pucch transmitting power control
RNTI

Tabella 3.1: PDCCH message

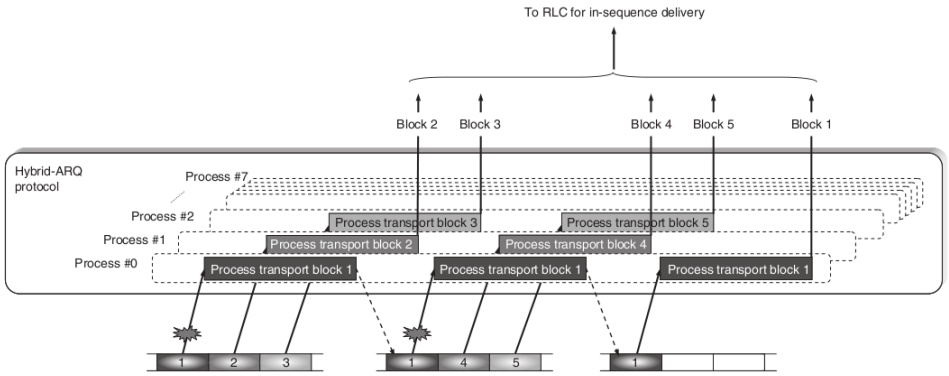


Figura 3.2: gli otto processi *HARQ stop and wait*

ACK/NAK per una *Ue* è fisso a 4 *TTI Time Transmission Interval*.

Il protocollo *HARQ* in *downlink* (da *eNodeB* a terminale) è asincrono nel senso che le ritrasmissioni possono avvenire in qualsiasi momento, a partire da una ricezione di un *NAK*, ed utilizzando un processo diverso, anche se nelle configurazioni finora viste si è usato sempre lo stesso. Le ritrasmissioni in *uplink* invece funzionano in maniera sincrona in quanto avvengono a determinati intervalli di tempo, dopo l'*acknowlegement* negativo. Nel protocollo sincrono l'intervallo di tempo dopo il quale effettuo una ritrasmissione viene fissato dallo *scheduler* della *eNodeB* ad inizio trasmissione. Questo permette all'*eNodeB* di sapere esattamente quando aspettarsi la ritrasmissione di un *TB*.

Si eccenna che oltre all'*ARQ ibrido* è previsto un altro meccanismo di ritrasmissione, a livello *RLC*. Infatti il meccanismo *HARQ* è veloce ma lascia un tasso di errore non ancora accettabile, mentre una ritrasmissione a

livello *RLC* è più lenta ma garantisce, assieme all'*HARQ* una ricezione quasi *error free*, e tutto questo è necessarioper garantire un'ottima affidabilità di trasmissione, cosa voluta dall'onnipresente procollo di trasporto *TCP*.

3.2 Harq a livello *PHY*

Se la tecnica *HARQ* a livello *MAC* ha un compito decisionale, scegliendo ad esempio su quale processo deve trasmettere o con quale *redundancy version*, il livello fisico si trova a dover mettere in pratica tali scelte.

Si mostra la catena di operazioni a cui viene sottoposto un *TB* all'interno del *PHY layer*, figura 3.3.

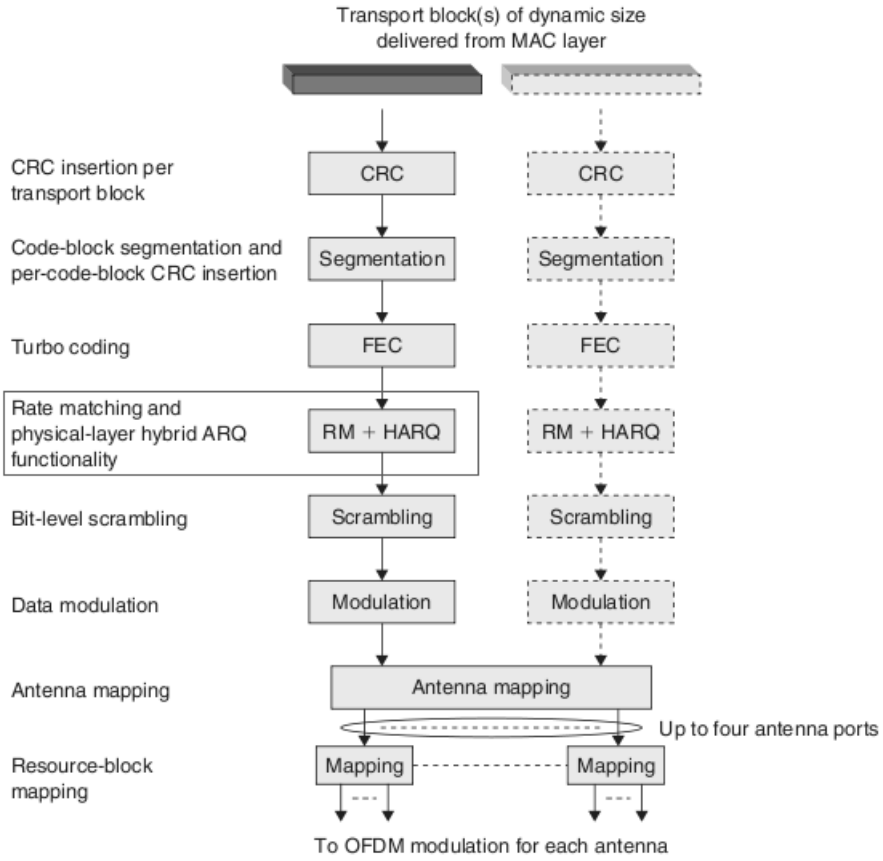


Figura 3.3: blocchi funzionali del *physical layer*

Ricevuto il *TB* la elaborare, la prima operazione consiste nella eventuale segmentazione in *code blocks*, e poi il calcolo e l'inserzione del *CRC*, figura 3.4.

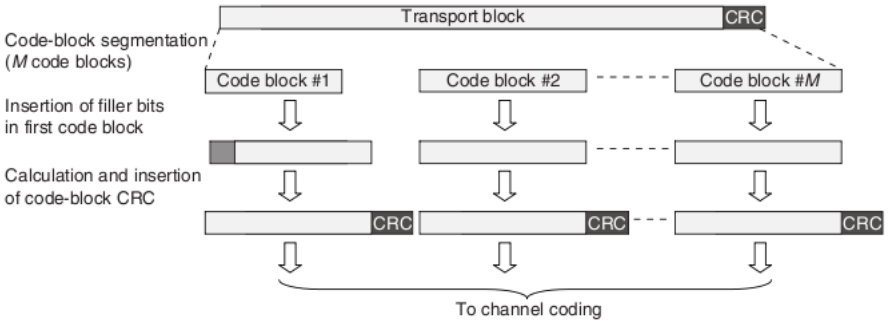


Figura 3.4: segmentazione ed *CRC insertion*

L'operazione successiva, fondamentale per l'*ARQ ibrido*, è la codifica turbo la quale triplica la dimensione di un *code block* o *TB* nel caso non ci sia segmentazione, figura 3.5.

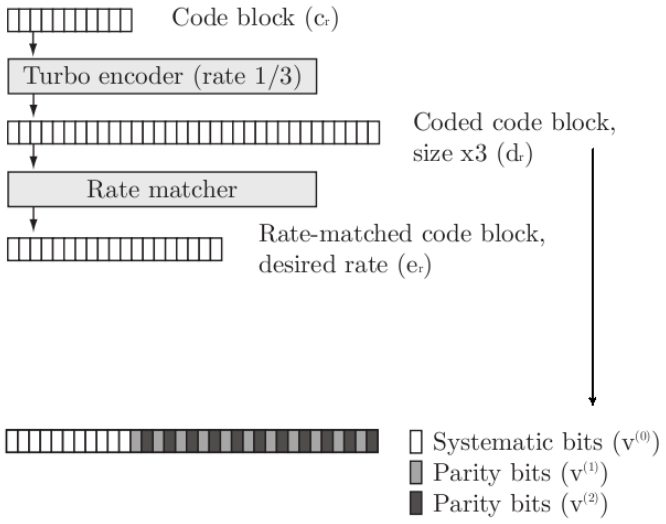


Figura 3.5: turbo codifica

La prima terza parte del *code block* codificato è identica al *code block* in ingresso al *turbo encoder* ed viene definita parte sistematica, il resto sono bit di ridondanza che verranno utilizzati nelle eventuali ritrasmissioni *HARQ*. I bit in uscita dal codificatore vengono sottoposti ad un'operazione di selezione dal blocco di *rate matching* pilotato dall'entità *HARQ* di livello *MAC*. Qualora il *code block* codificato sia alla prima trasmissione, si trasmette solo la sua parte sistematica, mentre se si tratta di ritrasmissione viene trasmessa una porzione del *code block* codificato dipendente dal tipo di *soft combining* che il sistema *HARQ* utilizza. *Rate matcher* in figura 3.6.

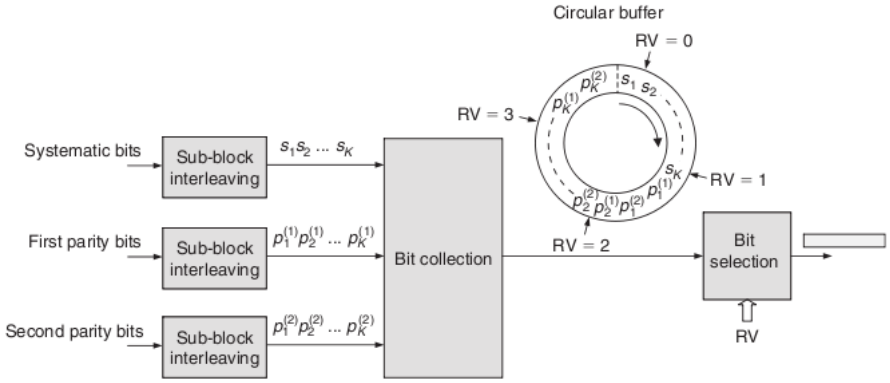


Figura 3.6: *rate matcher*

3.2.1 Soft combining

E' la tecnica che permette di dicidere quali bit trasmettere nelle ritrasmissioni, e che ha il suo corrispettivo in ricezione dove si deve avere la capacità di ricombinare i bit ricevuti con quelli delle precedenti trasmissioni, al fine di ottenere una nuova parola di codice la cui probabilità di venir decodificata corretamente sia maggiore. Gli studi fatti sui codici turbo e sul loro *rate matching* permettono quindi di avere più di una modalità di scelta sull'infomazione da tramettere nelle ritrasmissioni. Se infatti si poteva pensare di ritrasmettere la stessa informazione, ed ancora si può fare, ed è definita *soft combining - CC Chase Combining*, sono previste ritrasmissioni con informazione in parte uguale ed in parte diversa, tecnica difinita *soft combining P-IR Partial Incremental Redundancy* (o IR versione 1) ed ritrasmissioni con solamente nuova informazione, definita *soft combining H-IR Full Incremental Redundancy* (o IR versione 2). Si riportano le figure esemplificative, figura 3.7.

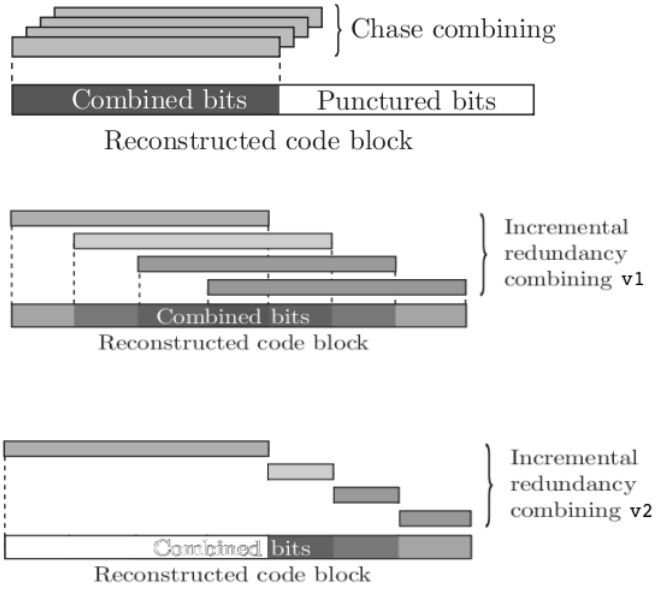


Figura 3.7: *soft combining CC, P-IR, F-IR*

3.2.2 *PHY abstraction* della *Soft combining*

L'astrazione completa del livello fisico deve quindi conglobare al suo interno tutte le operazioni che sarebbero svolte dalla *FEC* in funzione della *soft combining* usata. Come specificato nel capitolo *Link Abstraction* ci si basa su astrazione per mezzo dell'informazione mutua. Il compito sarà quello di capire che informazione mutua può avere un *code block* avendo a disposizione l'informazione mutua della sua trasmissione $MI_{sistematica}$ ed l'informazione mutua delle sue ritrasmissioni $MI^{i}_{ridondanza}$ con $i \in [1, 2, 3]$ dove i è l'indice del numero ritrasmissione.

Per la *Chase Combining* l'equazione utilizzata prevede di sommare i *SINR* della prima trasmissione con le successive ritrasmissioni per poi estrarre l'informazione mutua da questa somma:

$$M_I = \frac{1}{N} \sum_{n=1}^{N} I_m \left(\sum_{j=1}^{q} SINR_{jn} \right) \tag{3.1}$$

dove q è il numero di ritrasmissioni, I_m è l'informazione mutua per una sottoportante ed N è il numero di sottoportanti associate alla trasmissione del *code block* sotto esame.

Per la tecnica *Incremental Redundancy* oltre la possibilità di avere la stessa informazione bisogna tener conto anche di ridondanza completamente

nuova e non precedentemente trasmessa. Nella realtà il *rate matcher* ed il *decoder* del ricevitore prende la nuova informazione [bit] e la combina con l'informazione già in suo possesso e tenta di decodificare la nuova parola di codice ottenuta. Dovendo astrarre questa procedura, noi utilizzeremo le funzioni di estrazione dell'informazione mutua sia per le ridondanze che per la parte sistematica a partire dai loro *SINR*, i quali si possono assumere indipendenti per il lasso di tempo che intercorre fra loro, che è almeno 4 *TTI* in *downlink*.

L'informazione mutua dopo la ricombinazione può esser descritta dall'equazione

$$M_{Inew} = \frac{N_{pre} \cdot M_{Iold} + N_{NR} \cdot M_{Irid} + f_1(f_1^{-1}(M_{Iold}) + f_1^{-1}(M_{Irid}))}{N_{pre} + N_{NR} + N_R} \tag{3.2}$$

dove f_1 è la funzione di mappatura $SINR \rightarrow MI$, N_{NR} sono i nuovi bit di ridondanza trasmessi, N_R sono i bit ripetuti della precedente trasmissione, N_{pre} sono i bit codificati non ritrasmessi, M_{Iold} è l'informazione mutua vecchia del blocco ed M_{Irid} e l'informazione mutua della parte ora ritrasmessa.

CAPITOLO

4

IMPLEMENTAZIONE E RISULTATI

Il sistema *HARQ* e la *link abstraction* sono stati si implementati all'interno del simulatore *ns3*, attraverso nuovi moduli e modifica di classi (programmazione ad oggetti) già esistenti, ma prima è stata studiata la realizzazione del canale in *ns3* per poi riportare la stessa in *octave*, software *open source* fratello (minore) di *matlab*, ottimo stumento per l'analisi numerica e matematica. Questo al fine di ottenere risultati in un tempo ragionevolmente breve (3 mesi, dopo i primi 2 di stato dell'arte). Inoltre la stesura in classi *C++* è meno immediata e un po distoglie dagli effettivi contenuti, mentre la programmazione funzionale di *octave/matlab* è snella e diretta.

Di seguito si riportano i parametri di sistema, tabella 4.1

Banda di sistema	5 [Mhz]
Banda un sottocanale	180 [Khz]
Numero sottocanali	25
Numero sottoportanti per canale *OFDM*	300
Numero sottoportanti per sottocanale	12
TTI	$1[ms]$
Modulazione	QPSK, 16-QAM, 64-QAM (no-AMC)
Code rate	$1/2, 1/2$ (block size differente),$2/3, 3/4, 5/6$
Numero *Harq Processes*	8
Harq type	CC, P-IR, F-IR
Numero massimo di ritrasmissioni	3
Canale	Multipath e AWGN
Acknowledgement delay	4 *TTI* in *downlink* (3GPP)
Numero di *code block* per *TB*	5

Tabella 4.1: parametri per implementazione sistema *LTE*

Per un'allocazione semplice, si sono considerati pacchetti di simboli e non di bit. Ogni pacchetto consiste in un *code block*, ed ogni *code block* viene allocato su 5 sottocanali per la durata di un *subFrame*. L'allocazione viene quindi fatta per sottocanali e non sottoportanti così da semplificare notevolmente la realizzazione (sia in *octave* che per *ns-3-LTE*). Si presenta l'allocazione in figura 4.1. Con riferimento alla figura **??** invece, si riporta il

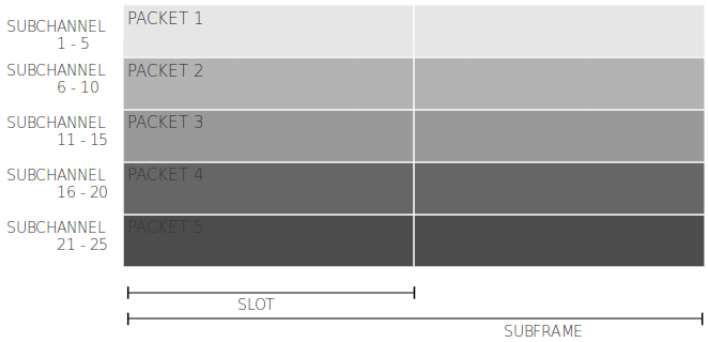


Figura 4.1: allocazione di un *subFrame*

calcolo di quanti simboli compongono un pacchetto:

- $NS_{frame} = N_{scr} \cdot N_{sch} \cdot N_{ofdm-frame} = 42000$
  - NS_{frame} numero di simboli in un *frame*
  - $N_{scr} = 12$, numero di sottoportanti per sottocanale

  - $N_{sch} = 25$, numero di sottocanali
  - $N_{ofdm-frame} = 14 \cdot 10 = 140$, numero di simboli *OFDM* su un *frame*

- $NS_{dati-frame} = NS_{frame} - NS_{control} = 28296$
  - $NS_{data-frame}$ numero di simboli dati in un *frame*
  - $NS_{control} = 13704$, numero di simboli di controllo

- $mN_{data-frame} = NS_{control}/N_{sf} \approx 2830$
  - $mN_{data-frame}$ numero medio di simboli dati in un *subframe*
  - $N_{sf} = 10$, numero di *subframe* per *frame*

- $mN_{data-pack} = mN_{data-frame}/P_{sf} = 566$
  - $mN_{data-pack}$ numero medio di simboli dati in un pacchetto
  - $P_{sf} = 5$, numero di pacchetti trasmessi in un *TB*

Nell'utilizzo della *soft combining* si deve specificare la quantità di simboli che si intende ritrasmettere. Per la *Chase Combining* si ritrasmettono i pacchetti interi di 566 simboli mentre per *Incremental Redundancy* si deve tener conto che il turbo codice triplica la dimensione di ogni pacchetto, quindi si hanno a disposizione $2 \cdot 566 = 1132$ simboli di ridondanza. In *P-IR* per ogni ritrasmissione si trasmettono $1132/3 \approx 377$ nuovi simboli di ridondanza, in modo che dopo 3 ritrasmissioni si sia trasmesso tutto il pacchetto codificato, e si ripetono $566 - 377 = 189$ simboli. In *F-IR* si trasmettono solamente i simboli di ridondanza nuovi, 377 simboli.

Ad ogni ritrasmissione, anche se c'è uno solo uno dei cinque pacchetti errato, il processo *HARQ* ritrasmette l'intero *TB*, ovvero tutti e cinque i pacchetti. Questo per semplificazione nella gestione delle ritrasmissioni, altrimenti si avrebbe dovuto implementare uno *scheduler* che tenesse conto di quanti pacchetti si debbono ritrasmettere e nel caso non sia l'intero *TB* prendesse pacchetti dal *layer* superiore per creare un nuovo *TB* con pacchetti sia da ritrasmettere che nuovi.
Questo è un lieve spreco di risorse, che però non impatta più di tanto sulle performance.

Qui ci si è soffermati a studiare ed implementare uno *scheduler downlink* quindi lo *scheduler* all'interno della *eNodeB*. Questo *scheduler* alloca risorse solo per una singola *Ue* (anche qui per limiti di tempo, e perchè altrimenti sarebbe stata una tesi sull'allocazione più che sul *HARQ*). Grazie a questo fatto si può affermare che lo *scheduler downlink* è quasi uguale allo *scheduler uplink* il quale giace all'interno della *Ue*, questo perchè comunque un terminale trasmette solo ad una *base station* quindi avviene allocazione per un singolo ricevitore. La differenza è che le latenze per gli *acknowledgement* non è fissa a 4 *TTI* come pre uno *scheduler downlink* ma è a discrezione del *eNodeB*.

I risultati ottenuti si possono interpretare derivanti da simulazioni di *link level* non considerando scenari con più *Ue* ed interferenze, i quali si studiano con simulazioni di *system level*.

4.1 Implementazione *Octave*

Il sistema-*LTE octave* lo si può dividere in tre blocchi, il canale, lo *scheduler* della *eNodeB* ed il modello di errore il quale congloba in se tutta la procedura di *link abstraction*, la gestione dei processi *HARQ* ed la *soft combining*.

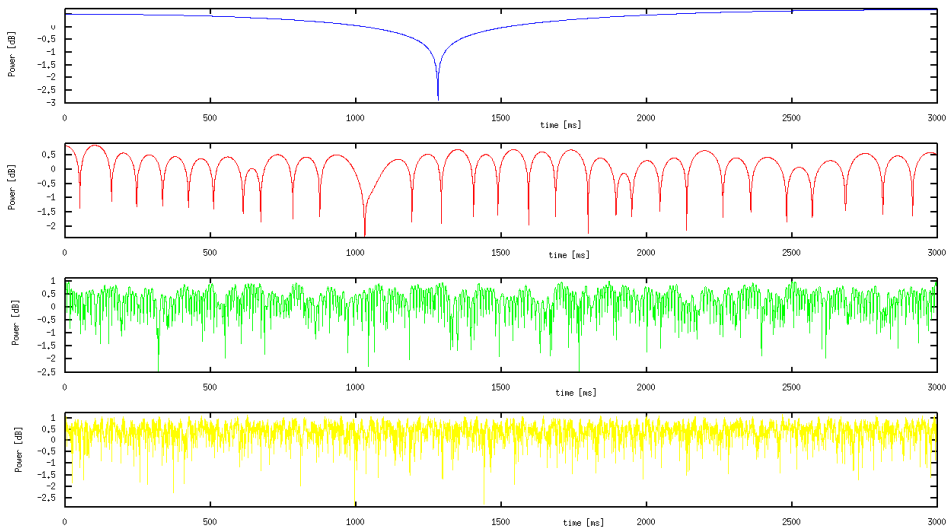
4.1.1 Canale

L'equazione che permette di trovare la potenza ricevuta per ogni sottocanale è

$$Prx_i(t) = Ptx_i(t) + multipath_i(t) - pathloss - shadowing - penetration_{loss} \tag{4.1}$$

dove il pedice i indica il sottocanale e t specifica il tempo in TTI.

Il *multipath* tiene conto delle riflessioni multiple nelle vicinanze della *Ue*, e viene trovato in funzione della sua velocità. Sono state generate 16 sequenze di *multipath* con un simulatore di Jakes il quale modella l'andamento *multipath* tramite una somma di sinusoidi. Le sequenze sono in funzione di quattro range di velocità ($v < 3[Km/h]$, $3 < v < 90[Km/h]$, $90 < v < 120[Km/h]$, $v > 120[Km/h]$) e dal numero di cammini (6, 8, 10 e 12 cammini). Queste sequenze coprono un intervallo di 3 [s]. Si riportano le sequenze, fissato un numero di cammini pari a 6, in figura **??**, tutte le 16 sequenze raggruppate per numero di cammini in figura **??** ed le sequenze in funzione dei cammini fissata la velocità della *Ue* in figura **??**.



Ad ogni intervallo di 250 [ms] e per ogni sottocanale il simulatore seleziona in maniera casuale una finestra di 250 [ms] dalla sequenza di Jakes relativa alla velocità della *Ue*, ed viene scelto casualmente il numero di cammini (si crea l'andamento $multipath_i(t)$ per 250 [ms]). Si suppone quindi che all'interno di un intervallo di 250 [ms] la velocità e posizione di un terminale sia pressochè costante.

In questo modo non si avrà della correlazione in frequenza, diversamente dalla realtà, però la semplificazione può tenere considerando che fra ogni sottoportante c'è una spaziatura di 180 [Khz].
Si mostrano delle raffigurazioni tempo-frequenza di due realizzazioni *multipath* in figura 4.3

Il *pathloss* si trova conoscendo la posizione della *Ue*, ed in accordo con lo standard *3GPP* in ambiente urbano, l'equazione matematica è

$$pathloss = I + 37.6 \cdot log_{10}(D) \tag{4.2}$$

con $I = 128.1$ per frequenza centrale del canale *OFDM* $f_0 = 2$ [Ghz] ed R la distanza fra *eNodeB* ed *Ue*.

Lo *shadowing* è modellato con una variabile aleatoria *log-normal*, il cui istogramma su 10000 realizzazioni si trova in figura 4.4. Ad ogni sottocanale è associata una realizzazione della variabile.

Il *penetration loss* è un margine costante di 10 [dB].

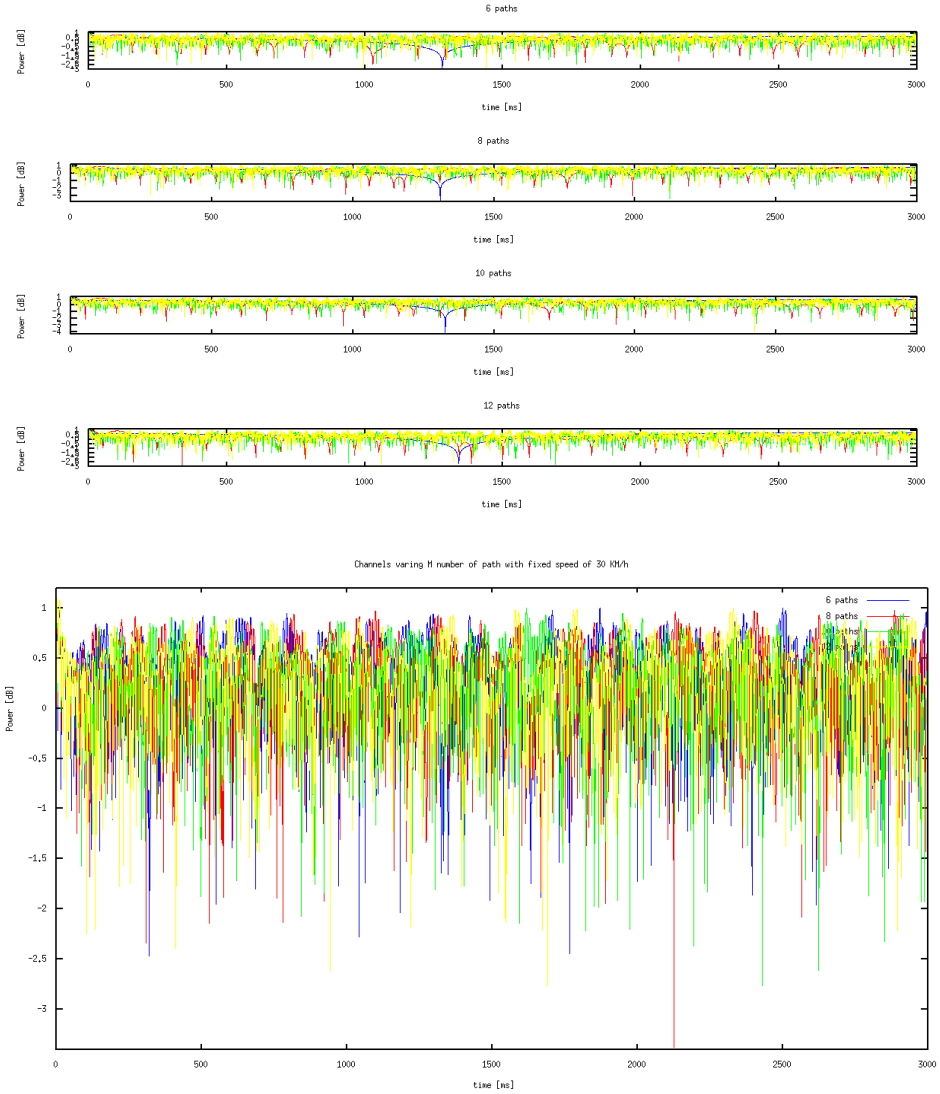


Figura 4.2: sequenze di jakes

4.1.2 Scheduler

Differentemente da *ns3* dove l'applicazione che genera pacchetti nel tempo è un'entità a se stante ed autonoma con una sua politica di generazione, in octave si è deciso di creare all'inizio simulazione tutti i pacchetti da trasmettere e cederli un po alla volta allo scheduler man mano esso abbia possibilità a trasmetterli.

Si riporta il diagramma di flusso dello *scheduler downlink*, figura 4.5 Porto una breve descrizione dei blocchi più 'misteriosi'

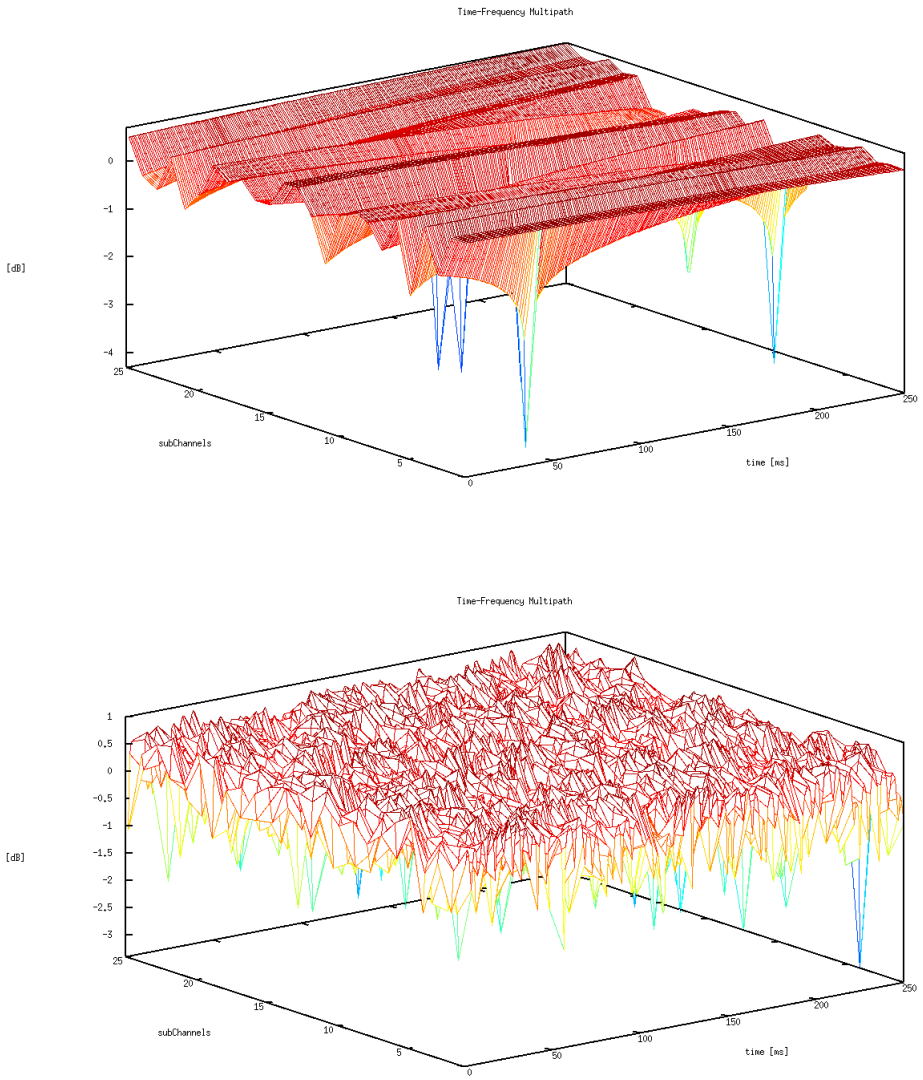


Figura 4.3: andamento tempo-frequenza del *multipath*

- $RV < 4$: si va a vedere se la massima RV fra i vari pacchetti allocati nel *TB* è minore del massimo numero di ritrasmissioni permesso. (3 ritrasmissioni oppure si può vedere come 4 trasmissioni dello stesso pacchetto)

- *check the correct packets and RTX the TB*: si va a ritrasmettere il *TB* ma si tengono corretti i pacchetti trasmessi precedentemente in

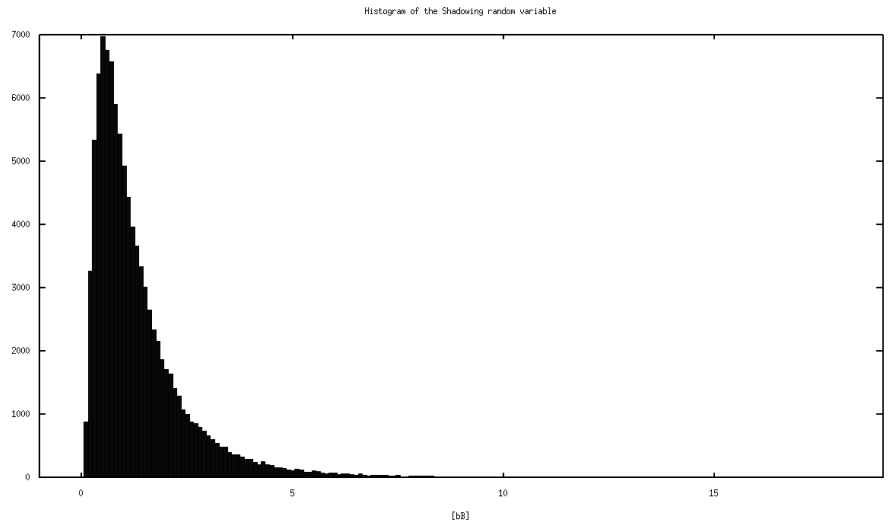


Figura 4.4: instogramma dello *shadowing*

maniera corretta e si applica la procedura di *soft combining* solo ai pacchetti errati

- *forward up the correct packets and drop the bad packets*: si salvano i pacchetti che alla fine di tutte le ritrasmissioni risultano corretti mentre si scartano quelli alla fine errati

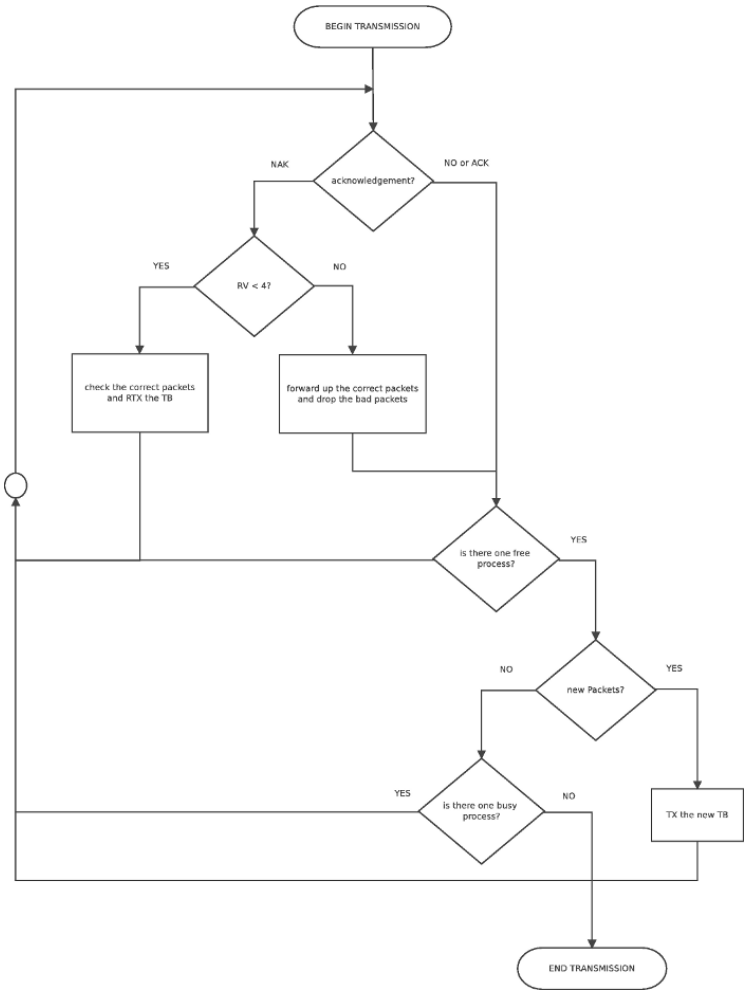


Figura 4.5: *flow chart* dello *scheduler downlink*

4.1.3 Il modello di errore e la gestione della *HARQ*

Deciso il numero di pacchetti trasmessi in un *TB* è stata creata una tabella di otto righe quanto i processi *HARQ* ed un numero di colonne pari al pacchetti allocati in un *TB* più due, nel quale

- la prima colonna indica lo stato dei processi [*busy* / *free*]
- la seconda colonna contiene la massima RV fra tutti pacchetti facenti parte dello stesso *TB*, la quale consente di capire quante volte ho già trasmesso il *TB*
- dalla terza all'ultima colonna si contengono la RV dei pacchetti associati ai vari processi

In questo modo, una volta scelto il processo che deve trasmettere o ritrasmettere, il processo conosce la RV di ogni pacchetto poichè è scritta nella sua righa di tabella, ed una volta che si esegue il modello di errore la si può aggiornare.

Il modello di errore basato sulla mutua informazione è l'implementazione pari pari delle funzioni contenute nel capitolo 2, quindi non mi dilungo. L'unica cosa rilevante è come il modello di errore debba conoscere la tabella dei processi *HARQ* in modo da potersi istruire al fine di eseguire la *soft combining*. Quindi posso affermare che sì il modello di errore astrae il *layer* fisico ma ha bisogno di uno stretto colloquio con il *MAC layer* creando una struttura *cross layer*. Inoltre bisogna tener in memoria informazioni mutue passate ed *sinr* associati alle sottoportanti per poter utilizzare la *soft combining*.

4.1.4 Risultati

Variazione numero massimo di ritrasmissioni

Fissata la modulazione $16-qam$ e figura per figura le varie tecniche di *HARQ soft combining*, si riportano i grafici prestazionali in funzione del numero massimo di ritrasmissioni, ovvero le performance per sistemi senza, con una, due o tre ritrasmissioni.

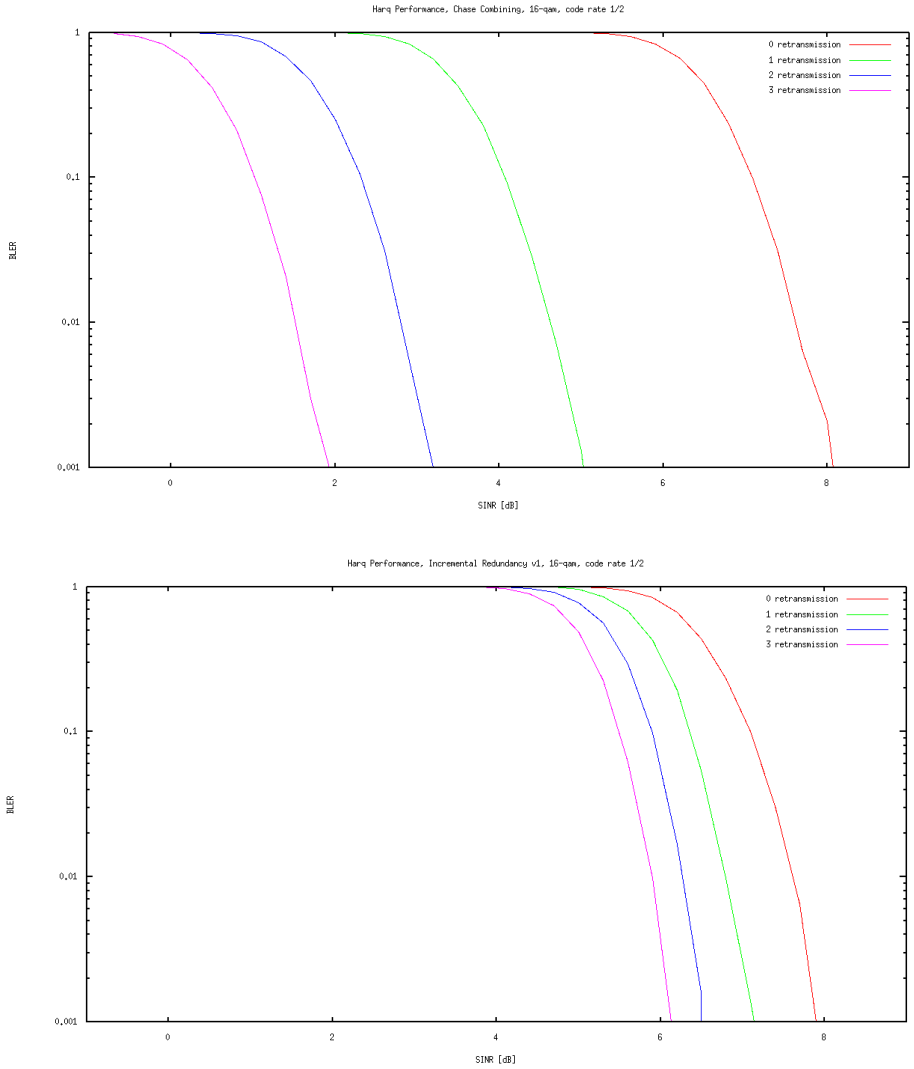


Figura 4.6: performance in funzione del numero massimo di ritrasmissioni

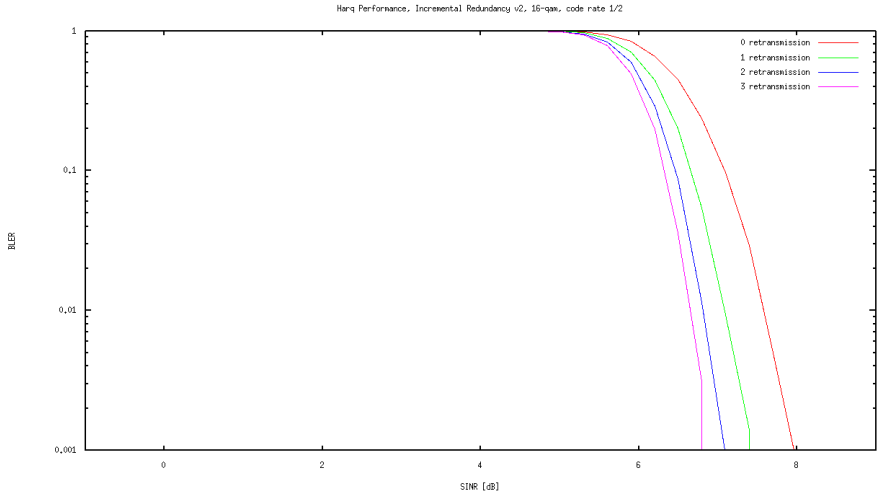
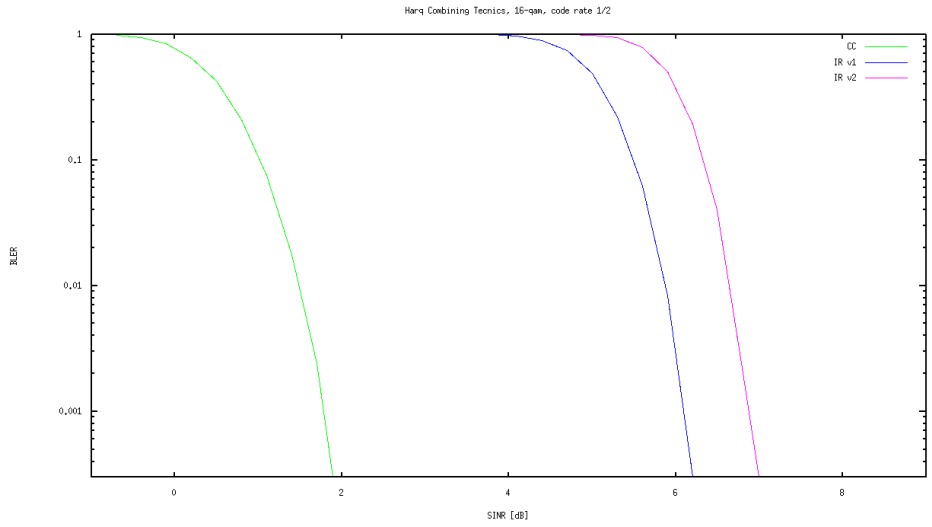


Figura 4.7: performance in funzione del numero massimo di ritrasmissioni

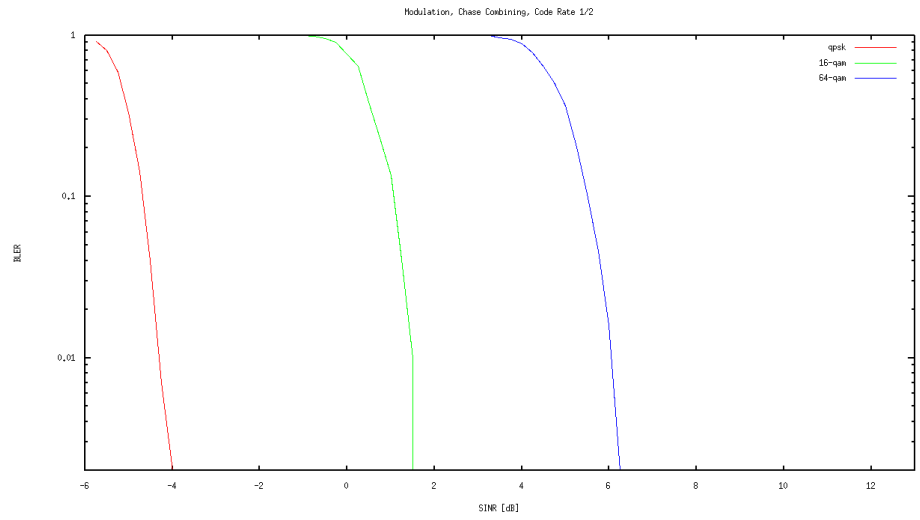
Dalle prestazioni della *soft combining* per le varie tecniche, il risultato immediato è che la *Chase Combining* sorpassa di molto le due tecniche di *Incremental Redundancy*. La motivazione principe sta nella formula che modella il suo comportamento, la quale permette di ottenere informazioni mutue elevate dalla ricombinazione di ridondanze e parte sistematica, e questo perchè si va a sommare i $SINR_{ridondanze}$ ed i $SINR_{sistematica}$ per poi rimappare la somma su una nuova informazione mutua. In *incremental redundancy* invece le *performance* diminuiscono man mano ci avviciniamo ad una *Full-IR*.

Si può vedere anche che il guadagno più netto si ha dal non ritrasmettere all'avere una massima ritrasmissione.

Raffronto fra le varie tecniche di *HARQ soft combining*



Variazione della modulazione



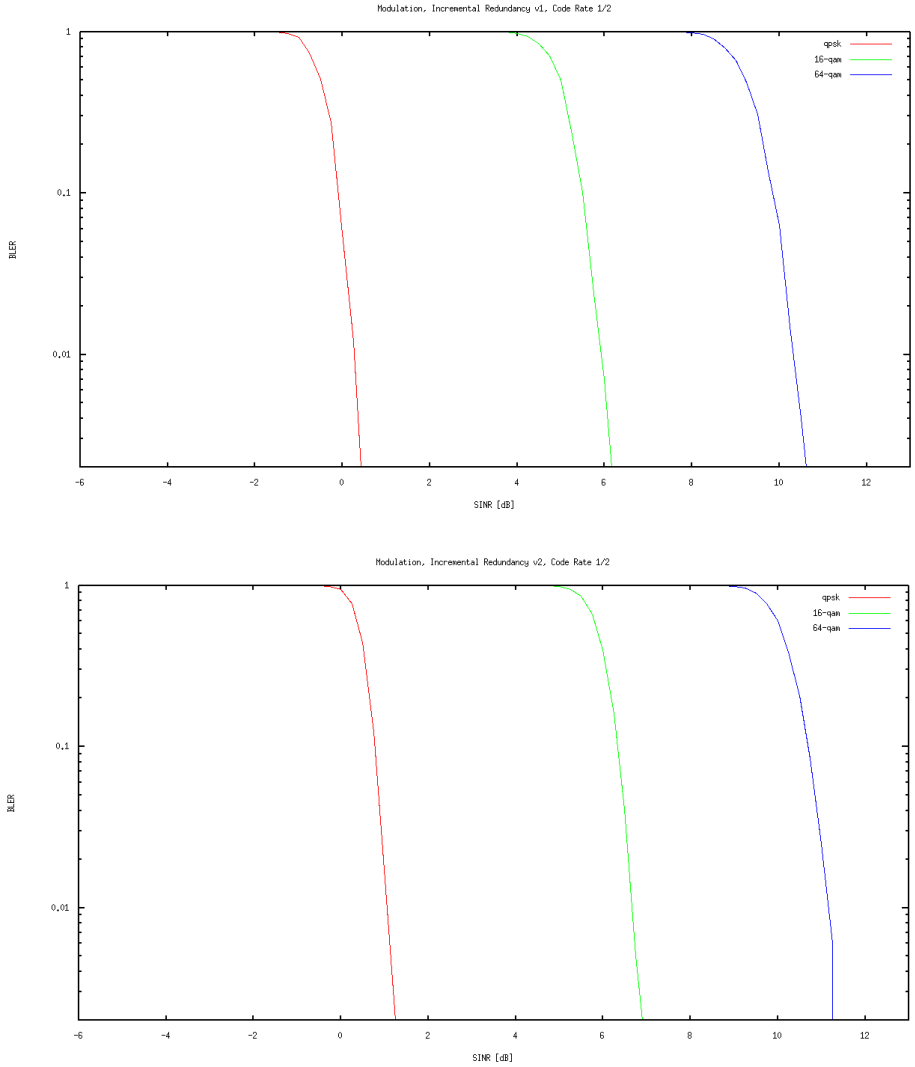


Figura 4.8: performance in funzione delle modulazioni

Throughput

Alla fine di ogni simulazione sono stati registrati alcuni dati necessari al calcolo del *throughput*. Riporto l'equazione con la quale è stato calcolato il *throughput* [bit/sec]:

$$throughput = \frac{(NP_{tx} - NP_{drop}) \cdot NS_p \cdot Mod_p \cdot Cod_p}{T_{simulation}} \tag{4.3}$$

- NP_{tx}, numero di pacchetti trasmessi

- NP_{drop}, numero pacchetti scartati
- NS_p, numero di simboli per pacchetto
- Mod_p, ordine di modulazione del pacchetto
- Cod_p, rate di codifica del pacchetto
- $T_{simulation}$, tempo totale di simulazione

Il numero di pacchetti trasmessi è la quantità di pacchetti prodotti dall'applicazione mentre i pacchetti scartati sono l'effetto di più di tre ritrasmissioni non andate a buon fine. I simboli per pacchetto moltiplicati per l'ordine di modulazione e *code rate* mi permette di dare un contenuto in [bit] dei dati effettivi ed il tempo di simulazione è l'intervallo di tempo trascorso fra l'inizio della trasmissione del primo pacchetto e l'ultimo il quale considera tutti i ritardi di ritrasmissione dovuti ad errori nei pacchetti.

Nei grafici si riportano i *throughput* per le varie coppie [modulazione / codifica] e si vuol evidenziare la loro distanza dalla massima capacità del canale DATI *LTE* (non si tengono conto di tutti i somboli utilizzati per le portanti pilota, l'accesso multiplo o i dati di controllo) calcolata dalla formula di *Shannon* modificata e adattata al caso

$$C = F \cdot B \cdot log_2(1 + SINR) \tag{4.4}$$

dove B è la banda del canale *LTE*

$$B = \frac{N_{sc} \cdot N_{sc} \cdot N_{ofdm}}{T_{subframe}} \tag{4.5}$$

con

- $N_{sc} = 12$, numero di sottoportanti per sottocanale
- $N_{sc} = 25$, numero di sottocanali
- $N_{ofdm} = 14$, numero di simboli *OFDM* su un *subFrame*
- $T_{subframe} = 0.001$ [s], TTI

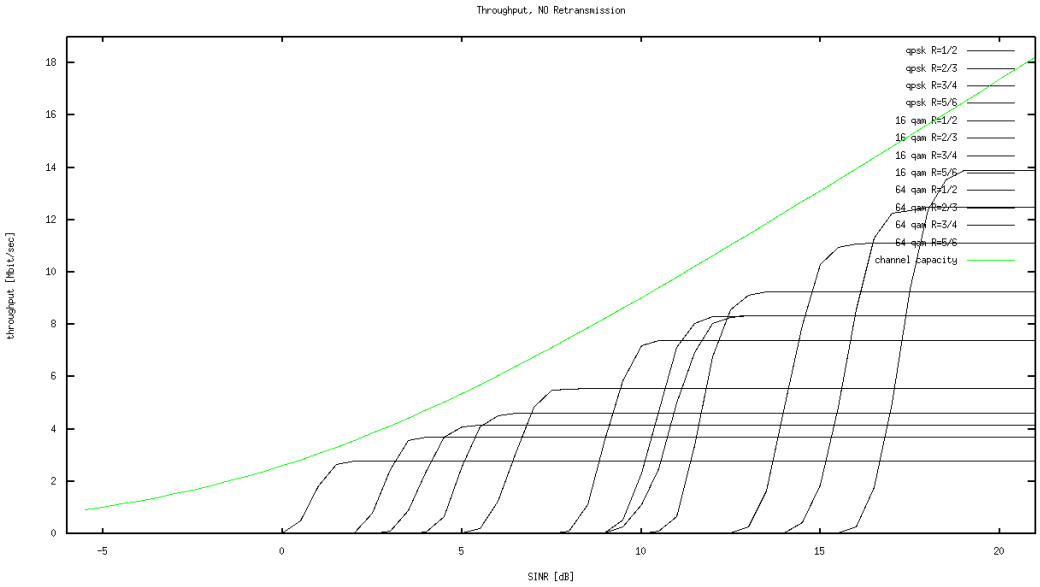
e F è un fattore di correzzione

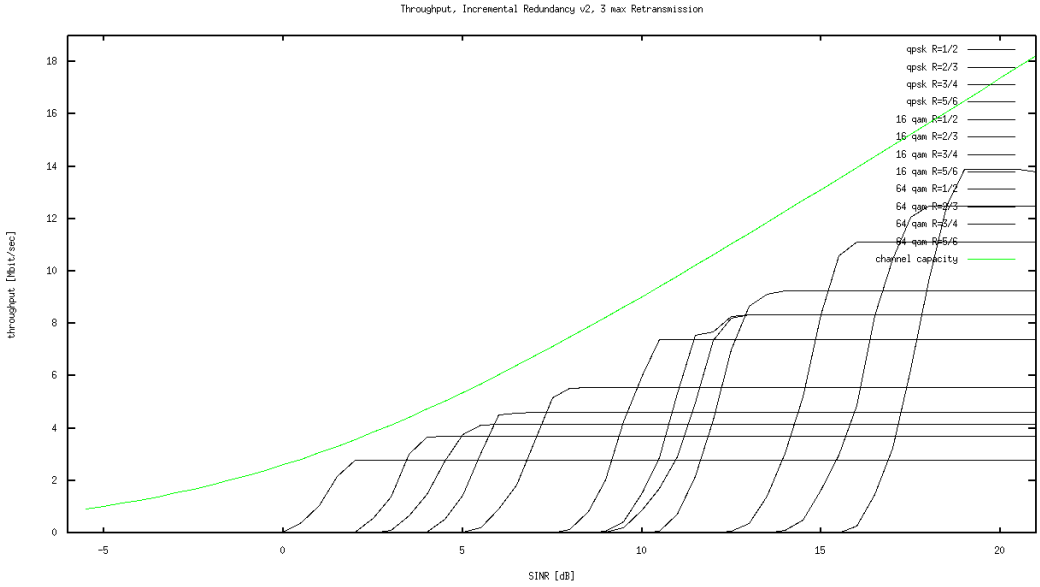
$$F = \underbrace{\frac{T_{frame} - T_{cp}}{T_{frame}}}_{CPloss} \cdot \underbrace{\frac{N_{RB} - N_{RB-control}}{N_{RB}}}_{CSloss} \tag{4.6}$$

con

- $T_{frame} = 0.010$ [s], intervallo di *frame*
- $T_{cp} = 5$, numero di sottocanali
- $N_{ofdm} = 14$, numero di simboli *OFDM* su un *subFrame*
- $N_{RB} = 25 \cdot 14 \cdot 10 = 3500$, numero totale di *RB Resours Block* in un *frame*

- $N_{RB-control} = ((25 \cdot 4) + 12) \cdot 10 + (3 \cdot 5 + 1) + 3 \cdot 2 = 1142$, numero totale di *RB Resours Block* di controllo e segnalazione in un *frame*
- *CPloss Cyclic Prefix loss*
- *CFloss Control Symbol loss*


Throughput, NO Retransmission
throughput [Mbit/sec]
SINR [dB]
qpsk R=1/2
qpsk R=2/3
qpsk R=3/4
qpsk R=5/6
16 qam R=1/2
16 qam R=2/3
16 qam R=3/4
16 qam R=5/6
64 qam R=1/2
64 qam R=2/3
64 qam R=3/4
64 qam R=5/6
channel capacity


Throughput, Incremental Redundancy v2, 3 max Retransmission
throughput [Mbit/sec]
SINR [dB]
qpsk R=1/2
qpsk R=2/3
qpsk R=3/4
qpsk R=5/6
16 qam R=1/2
16 qam R=2/3
16 qam R=3/4
16 qam R=5/6
64 qam R=1/2
64 qam R=2/3
64 qam R=3/4
64 qam R=5/6
channel capacity

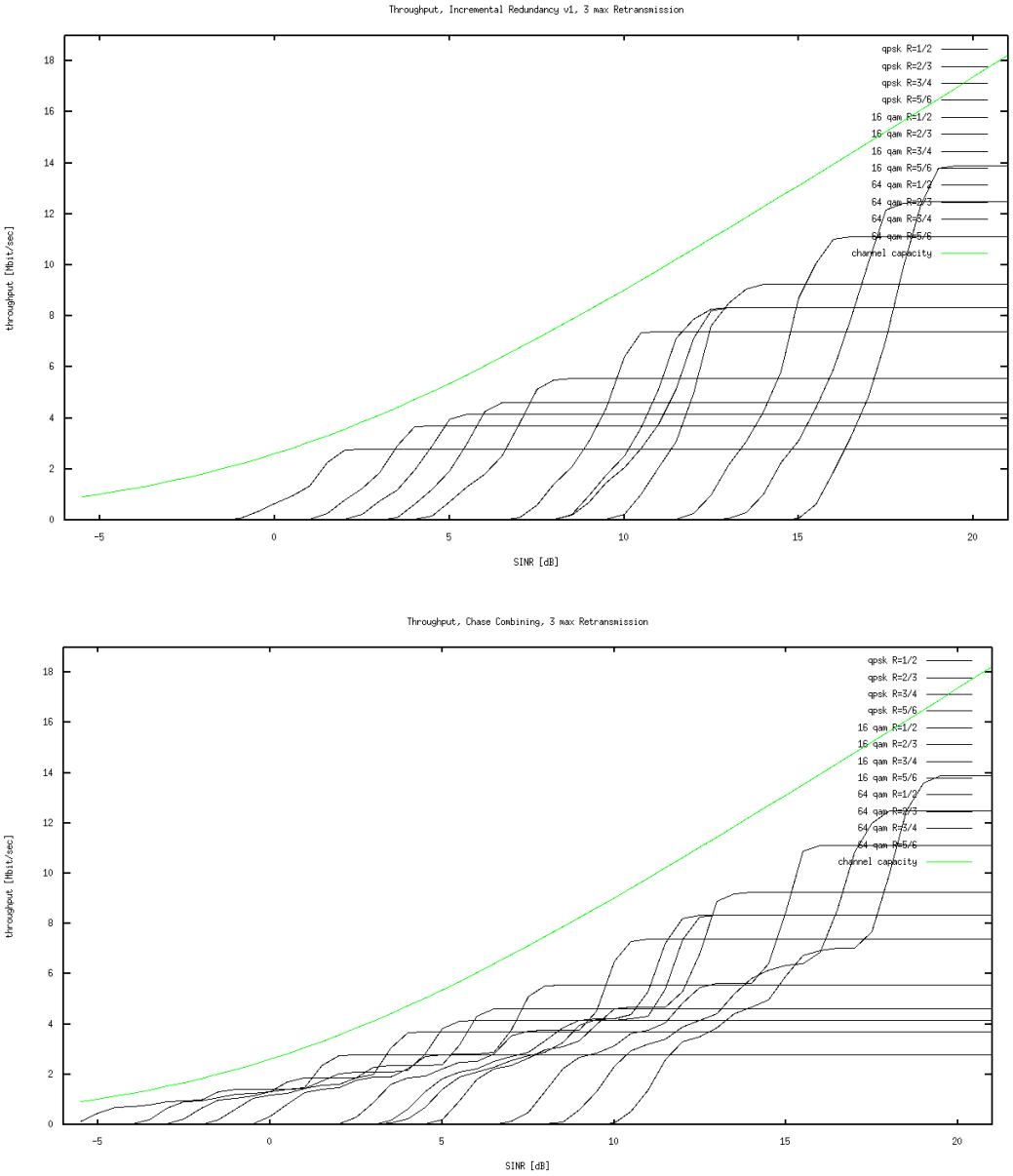
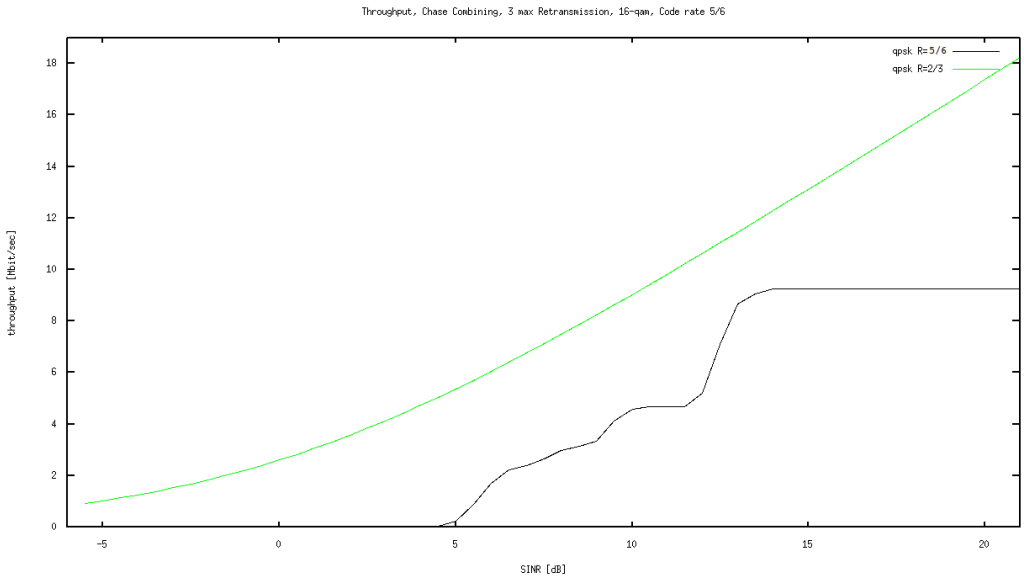


Figura 4.9: *throughputs* per le varie tecniche di *soft combining*



Interessante soprattutto il *throughput* ottenuto con la *Chase Combining* che mostra come si riesca a trasmettere a livelli di *SINR* non raggiungibili dalle altre tecniche (-5 [dB]), e mostra particolari andamenti 'ondosi' fino al raggiungimento del *SINR* dopo il quale si ha *throughput* costante.

4.2 Implementazione *ns-3-lte*

ns-3 è un simulatore di rete ad eventi discreti dove il *core* del sistema ed i vari moduli sono scritti in *C++*. Tramite *ns-3* si ha la capacità di creare topologie anche complesse di rete ed studiarne le *performance*. Per scaricare gratuitamente il simulatore guardare l'appendice.

4.2.1 ns-3

Gran parte del codice si trova nella *directory /src* ed è organizzato secondo figura 4.10

- *core*: è la base del simulatore e contiene le classi che definiscono per esempio
  - *object model*, cos'è un oggetto
  - *object names*, l'identificazione degli oggetti
  - *pointer*, puntatori
  - *callback*, chiamate a funzioni
  - *logging*, stampa a schermo
  - *tracing*, stampa su file

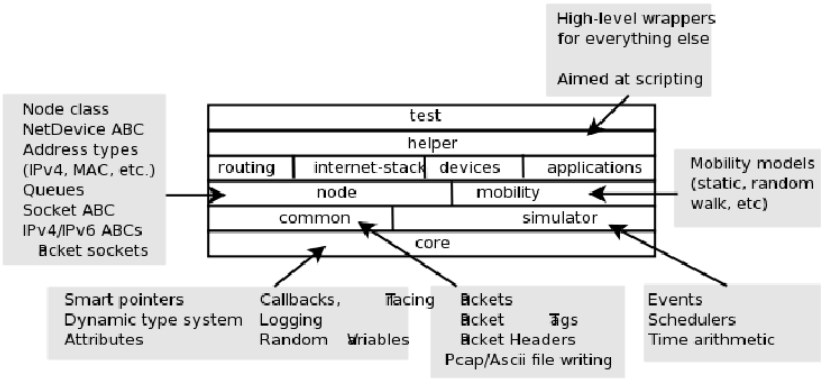


Figura 4.10: organizzazione della cartella */src*

- *simulator*: contiene classi per la realizzazione di simulazioni a tempo discreto
  - *event*, cos'è un evento
  - *scheduler*, gestisce la priorità degli eventi
  - *time*, cos'è il tempo nel sistema
- *common*: contiene classi di utilizzo generale
  - *packet*, cos'è un pacchetto
  - *header*, cos'è un *header*
  - *spectrum model*, come viene gestito lo spettro
  - *error model*, come si realizza un modello di errore generale
- *node*: contiene classi che vogliono in qualche modo definire cos'è un nodo, il quale avrà una sua mobilità, infatti nella figura il blocco *node* affianca *mobility*
  - *ipv6-header*, cos'è un indirizzo IPv6
  - *channel*, le caratteristiche base di un canale
  - *tcp-socket*, come creare una connesione *tcp*
- *routing*: contiene le classi per le varie politiche di routing
- *internet-stack*: contiene le classi per la realizzazione dei moduli dello stack protocollare
  - *arp-l3-protocol*, implementa la procedura *address resolution protocol*
  - *icmpv4*, implementa la procedura *internet control message procedure*
  - *tcp-l4-protocol*, implementa la procedura *transport control protocol*
- *application*: contiene le classi che simulano varie applicazioni

  - *ping6*
  - *udp-echo*

- *devices*: contiene le classi che descrivono i *layers* dipendenti dalle tecnologie radio
  - *csma*, implementa una trasmissione *carrier sense multiple access*
  - *wifi*, implementa lo standard 802.11.*x*
  - *lte*, implementa lo standard *LTE*, ancora in fase iniziale
- *helper*: contiene classi che aiutano una configurazione di nodi e dispositivi, utile ai *test*
- *test*: contiene le classi per la definizione dei scenari e la simulazione

Prima sono stati portati solo alcuni esempi di classi presenti nelle varie sottosezioni.

4.2.2 ns-3-lte

Per una spiegazione del sistema *LTE* implementato fino a ottobre del 2010, documentazione e contatti agli autori si invita a navigare i *links* in appendice.

4.2.3 ns-3-lte con *HARQ*

Questa sessione è utile soprattutto per chi conosce già il sistema *ns-3-lte* privo del sottosistema *HARQ*, ma anche ai nuovi adepti di *ns-3-lte*.

Aggiunta nuove classi

In figura 4.11 si riporta la relazione fra le classi per il funzionamento dell'astrazione e l'utilizzo della *HARQ*. Dalla classe *lte-spectrum-phy* si dovrebbe lanciare il metodo *RunErrorModel* fornendo al *mi-error-model* i *sinr* calcolati per sottocanale. A sua volta il *mi-error-model* esegue le sue procedure e va a scrivere le varie informazioni di errore sulla *harqProcessTable* appartenente alla *harq-entity*.

Sono state fatte delle semplificazioni per vedere se le classi funzionanano correttamente.

Sono state aggiunte le nuove classi (l'interfaccia si trova nei file.h, mentre l'implementazione si trova nei file.cc)

- *enb-downlink-packet-scheduler*
- *enb-uplink-packet-scheduler*
- *ue-downlink-packet-scheduler*

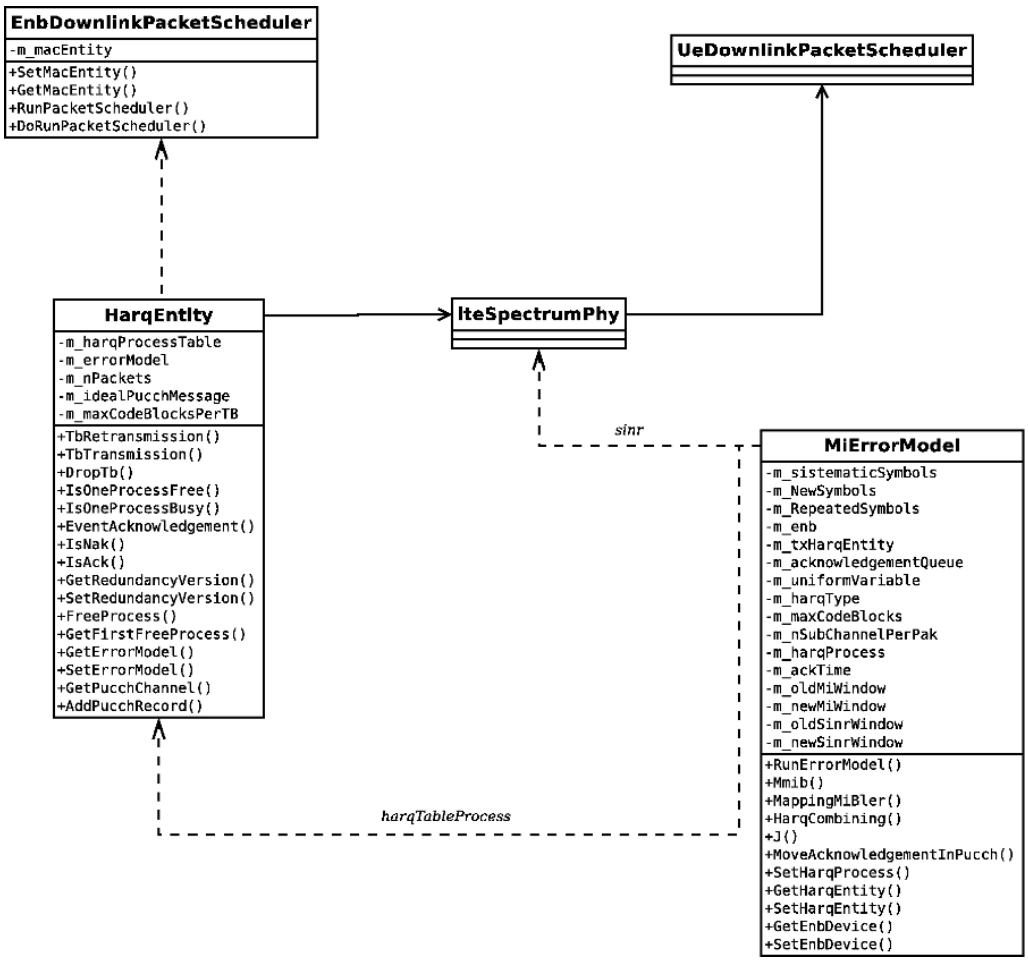


Figura 4.11: *class diagram* del sistema con l'*HARQ* innestata

- *ue-uplink-packet-scheduler*
- *harq-entity*
- *mi-error-model*

Si è voluto separare i vari *scheduler* in quanto si differenziano molto in base alla direzione di trasmissione ed in base anche all'appartenenza ad una *base station* o *Ue*. Ai fini della tesi è stato sufficente implementare lo *scheduler downlink eNodeB* nel quale è stata scritta la procedura di figura 4.5. Il modo con cui lavora lo *scheduler* e le approssimazioni usate sono spiegate nel paragrafo precedente di *octave*. Si sottolinea come la trasmissione dei messagi nel *PDCCH* all'interno della classe *enb-downlink-packet-scheduler* non è stata ultimata, e le bozze di codice commentato non è testato e molto probabilmente neanche funzionante.

La *harq-entity* ha il compito di tener traccia delle ritrasmissioni per ogni *code block*. Il numero massimo di *code block* può essere specificato nel costruttore del metodo, ed è stato posto a 5. Si può specificare anche il numero di pacchetti per *TB* per una trasmissione in modo da aiutarci nella gestione della *harqTableProcess* (per semplificazione si è deciso di gestire una *harqTable* dove il numero massimo di *code block* uguaglia il numero di pacchetti per *TB*). Nel costruttore non si specifica l'*error model* ma viene settato successivamente. La *harqTableProcess* non è altro che un vettore bidimensionale in si inseriscono le informazioni specificate nel paragrafo precedente. Inoltre si crea il canale *idealPucchMessage* gestito direttamente dalla *harq-entity* la quale maneggia gli *acknowledgement*. I vari metodi sono comprensibili direttamente dal titolo e mi soffermo sulle scorciatoie immesse nei metodi di trasmissione e ritrasmissione

- *HarqEntity::TbTransmission*: il compito del metodo è quello di informare il modello di errore su che processo si trasmette, prendere il *burst* di pacchetti e trasmetterli *enb->startTransmission*. Invece poichè non ho avuto il tempo di ricavare i *sinr* ed il puntatore alla *harq-entity* trasmittente in *lte-spectrum-phy*, non ho richiamato la vera procedura di trasmissione, ma ho specificato direttamente le informazioni di modulazione e codifica al modello di errore nel metodo senza usare l'*amc-module* e ho lanciato il metodo *MiErrorModel::RunErrorModel* che simula comunque una trasmissione. E' evidente che non ho passato quindi il *packet-burst* al canale. Comunque non è un problema, in quanto a livello finale l'unica differenza sta nel fatto che i *sinr* non saranno creati dal canale in funzione del suo andamento ma in maniera artificiosa all'interno del *mi-error-model*. Comunque il luogo giusto per far partire l'*error model* sarebbe nella funzione *LteSpectrumPhy::EndRx*, basta quindi risalire al puntatore della *harq-entity* per prendere il modello di errore associato ed eseguire un *run*. Inoltre faccio notare come il modello di errore avendo a disposizione tutte le informazioni necessarie per capire come trattare i singoli *code block*, andrebbero spostate le decisioni per passare o meno i pacchetti ai livelli superiori tramite le *callback* in *lteSpectrumModel::EndRX* alla fine del metodo *MiErrorModel::RunErrorModel*. Infatti la riga di codice -456: if (rxOk)- dell'implementazione *lteSpectrumModel* presuppone sapere se un *TB* è corretto o meno, quando però la faccenda è più complicata con *HARQ* e le infomazioni necessarie per il *forward up* sono estraibili direttamente dalla *harqTableProcess*, motivo per cui, spostare questa gestione alla fine del *RunErrorModel* dovrebbe semplificare il lavoro. Per quanto riguarda i pacchetti trasmessi basta sostanzialmente memorizarli in una coda in ricezione (aggiungendo come variabili oggetto dei *packet burst* dove memorizzare i *TB*, in *MiErrorModel*), e questo per ogni processo, e poi una volta eseguito il modello di errore, nel momento finale sulla decisione degli errori, si decide quali pacchetti mandare su, quali scartare e quelli da mantenere nella coda di memorizzazione in

attesa della ritrasmissione di ridondanza che si suppone potrà portare a una decodifica corretta (risottolineo che queste code non sono state implementate per meri motivi di tempo).

- *HarqEntity::TbRetransmission*: in questo metodo si ritrasmette un *TB*. Poichè l'idea non ancora implementata è quella di mantenere in memoria i *TB* non decodificati correttamente, non si necessita la ritrasmissione da parte della *eNodeB*. Per avere però un calcolo effettivo dei *sinr* si voleva trasmettere un *TB* virtuale che verrà successivamente scartato. Anche qui la modulazione ed la codifica vengono specificate a priori e si arrangia il modello di errore a capire di quale ritrasmissione si tratta.

La classe *mi-error-model* ha il compito di astrarre il livello fisico a partire dalla conoscenza dei vari *sinr* dei sottocanali, dalla modulazione/codice, dal tipo di *HARQ soft combining* e dalla configurazione della stessa *HARQ*. Nel costruttore si specificano le dimensioni dei pacchetti, e delle ridondanze (spiegato bene nei capitoli e paragrafi precedenti). Si specifica inoltre anche anche il tipo di *soft combining usata*, l'entità *HARQ* associata al modello di errore, il numero di sottoportanti associate alla trasmissione di un *code block* (considerando l'allocazione precedentemente presentata), un contenitore di *acknowledgement* i quali vengono presi a spostati nel *PUCCH* gestito dall'entità *HARQ* dopo l'intervallo di 4 *TTI* e poi tutte le strutture per la memorizzazione dei *sinr* e delle informazioni mutue, necessarie per la ricombinazione dei dati comuni. Commento brevemente i vari metodi

- *MiErrorModel::J*: serve per estrarre l'informazione mutua fra i simboli di un sottocanale a partire dal suo *sinr*. Dipende strettamente dalla modulazione
- *MiErrorModel::Mmib* serve per estrarre l'informazione mutua da un blocco di sottoportanti dove si trasmette con specifica modulazione per poi farne la media e ritornare il *MMIB*
- *MiErrorModel::MappingMiBler*: serve per mappare il *MMIB* al *BLER*, con una esplicita dipendanza dalla codifica.
- *MiErrorModel::HarqCombining*: serve ad implemenatare il meccanismo di *harq recombining* e mette a disposizione le tecniche *Chase Combining* e *Redundancy Version* con tutte le sue varianti (basta cambiare il numero di simboli ritrasmessi e di ridondanza nuova).
- *MiErrorModel::RunErrorModel*: esegue l'algoritmo per il modello di errore. Qui viene creato un vettore costante di *sinr* e trovato il *BLER* associato ad un *resours block* tenendo conto anche delle ritrasmissioni e delle ricombinazioni. Si decide l'evento di errore e poi si crea l'*acknowledgement* da memorizzare nella *acknowledgement queue*. Per completare il metodo, sarebbe opportuno come detto prima creare dei contenitori di *TB* che tengano in memoria quelli errati all'interno di una *Ue*. In questo modo quando una ritrasmissione va a buon fine si può passare all'*Ue MAC* i pacchetti corretti del *TB* o scartarli se dopo 4 ritrasmissioni non sono ancora corretti. Alla fine quindi dell'esecuzione

del modello di errore non si è passato il controllo all'*ue-downlink-scheduler* e questo per mancanza di tempo, però è la prosecuzione naturale della trasmissione.

Le approssimazioni fatte sono le stesse di quelle descritte nella sezione precedente.

Modifica classi preesistenti

Sono state modificate le classi già esistenti

- *enb-mac-entity*
  - aggiunte come variabili oggetto i puntatori a *enb-uplink-packet-scheduler*, *enb-downlink-packet-scheduler* ed *harq-entity* in quanto i tre componenti fanno effettivamente effettivamente parte del *MAC layer*, ed essendo il *MAC* di una *eNodeB* si va e specificare lo stesso per gli *scheduler*
  - aggiunta dei metodi per l'accesso ed il *setting* degli *scheduler* e della *harq-entity*
  - nel costruttore si istanziano gli oggetti da legare ai puntatori, e si va ad istanziare il modello di errore *mi-error-model* per poi inserire un riferimento reciproco fra l'*harq-entity* ed il *mi-error-model*. In questo modo l'entità *HARQ* conosce il relativo modello di errore usato, ed il modello di errore conosce a quale entità *HARQ* chiedere le informazioni necessarie per eseguire la *soft combining*
- *ue-mac-entity*
  - considerazioni analoghe a quelle precedenti, con la differenza che ora lavoriamo all'interno di una *Ue*
- *ideal-control-message*
  - *PdcchMapIdealControlMessage : ideal-control-message*: all'interno di un *IdealPdcchRecord* dove precedentemente veniva scritta l'*MCS* e il ricevente relativo al *code block* si è aggiunta l'*HARQ info* che specifica a quale processo è assegnata la trasmissione e che *redundancy version* ha la trasmissione
  - *PucchMapIdealControlMessage : ideal-control-message*: è stato creato il canale ideale di controllo per l'*uplink PUCCH* All'interno di questo canale (è un contenitore di *record*) si registrano i singoli messaggi di controllo trasmessi dalle *Ue* in risposta all'*eNodeB* e contengono il puntatore della *Ue* trasmittente, l'*acknowledgement* relativo e il processo *HARQ* al quale va direzionato l'*ACK/NAK*

Esempio eseguibile

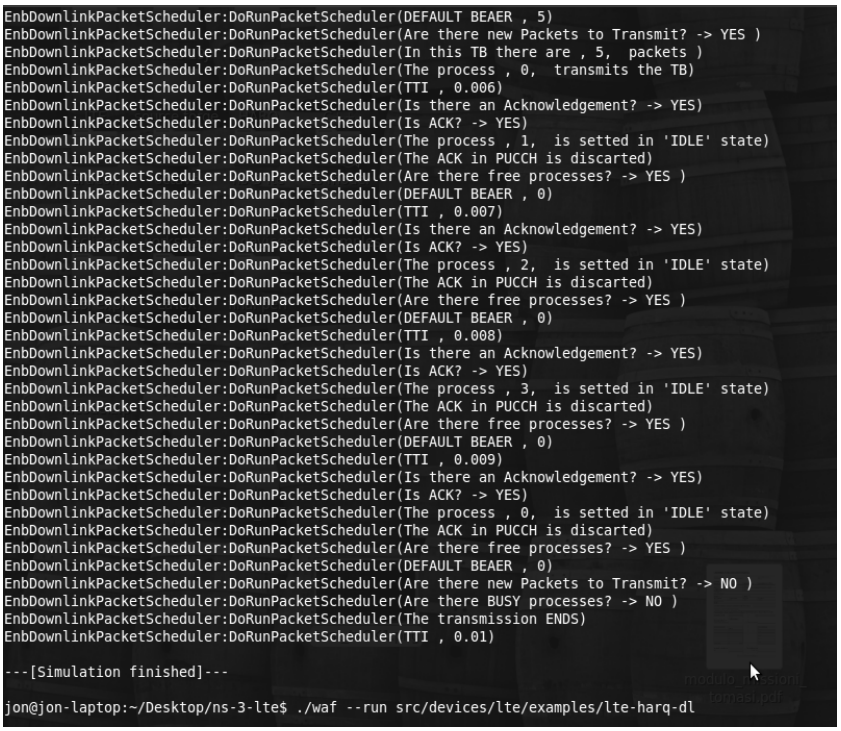
Lo scenario creato è il più semplice possibile. Si crea un *eNodeB* ed un'unica*Ue*. Si installano i dispositivi di rete per una comunicazione *LTE*. Per l'*eNodeB* si configura il dispositivo di rete inserendo l'indirizzo della *Ue* nel registro *Ue-manager*. Si configurano i canali *uplink* e *downlink*, quest'ultimo

con 25 sottocanali. Si posiziona la *eNodeB* fissa a terra mentre si configura la mobilità dell'*Ue* con il modello *ConstantVelocityMobilityModel*. Si installa la pila procollare internet nei due nodi in modo da assegnare gli indirizzi IP statici. Si creano 25 pacchetti, i quali dovrebbero simulare l'applicazione sorgente, si inseriscono i pacchetti all'interno del *default bearer* del *RRC*. Si da al via alla simulazione la quale attiverà lo *scheduler*.

La prima simulazione prevede la trasmissione su canale con *SINR* molto alti i quali non causano errori. Si riporta il *loggin* della trasmissione

```
---[Starting simulation]---

EnbDownlinkPacketScheduler:DoRunPacketScheduler(Number of packets per Transport block , 5)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Start Transmission)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there an Acknowledgement? -> NO)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Are there free processes? -> YES )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(DEFAULT BEAER , 25)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Are there new Packets to Transmit? -> YES )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(In this TB there are , 5,  packets )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 0,  transmits the TB)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(TTI , 0.002)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there an Acknowledgement? -> NO)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Are there free processes? -> YES )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(DEFAULT BEAER , 20)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Are there new Packets to Transmit? -> YES )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(In this TB there are , 5,  packets )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 1,  transmits the TB)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(TTI , 0.003)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there an Acknowledgement? -> NO)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Are there free processes? -> YES )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(DEFAULT BEAER , 15)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Are there new Packets to Transmit? -> YES )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(In this TB there are , 5,  packets )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 2,  transmits the TB)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(TTI , 0.004)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there an Acknowledgement? -> NO)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Are there free processes? -> YES )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(DEFAULT BEAER , 10)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Are there new Packets to Transmit? -> YES )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(In this TB there are , 5,  packets )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 3,  transmits the TB)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(TTI , 0.005)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there an Acknowledgement? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is ACK? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 0,  is setted in 'IDLE' state)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The ACK in PUCCH is discarted)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Are there free processes? -> YES )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(DEFAULT BEAER , 5)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Are there new Packets to Transmit? -> YES )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(In this TB there are , 5,  packets )
```



La seconda simulazione prevede la trasmissione su canale con *SINR* molto bassi i quali questa volta causano errori continui. Si riporta il *loggin* della trasmissione di seguito. Qui si possona fare alcune considerazioni; la prima è che come detto precedentemente i processi *HARQ* usati su un simulatore con *acknowledgement* ideali con ritardo 4 *TTI* saranno non più di quattro; la seconda è che se si fa il calcolo del numero di *TTI* necessari a trasmettere 25 pacchetti, ovvero cinque *TB*, su un canale che causa deterministicamente errori si trova

$$nTTI = 1 + nMaxTx \cdot ackDelay + nMaxTx \cdot ackDelay = 1 + 16 + 16 = 33 \tag{4.7}$$

dove $nMaxTx$ sono il numero massimo di trasmissioni di un *TB*. Nel tempo di 17 *TTI* si ritrasmettono quattro volte l'informazione relativa al primo *TB*, quindi è il tempo necessario perchè il processo *HARQ* 0 trasmetta, ritrasmetta e scarta il *TB*. Altri 16 *TTI* sono necessari per la trasmissione dell'ultimo *TB*, infatti se i primi quattro *TB* vengono trasmessi dai primi quattro processi *HARQ*, l'ultimo viene trasmesso dal processo 0 una volta scartato il *TB* primo. Nella simulazione riportata nelle figure si riscontra il tutto questo.

```
Hybrid-ARQ and MI-Error-Model example

---[Starting simulation]---

EnbDownlinkPacketScheduler:DoRunPacketScheduler(Number of packets per Transport block , 5)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Start Transmission)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there an Acknowledgement? -> NO)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Are there free processes? -> YES )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(DEFAULT BEAER , 25)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Are there new Packets to Transmit? -> YES )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(In this TB there are , 5,  packets )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 0,  transmits the TB)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(TTI , 0.002)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there an Acknowledgement? -> NO)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Are there free processes? -> YES )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(DEFAULT BEAER , 20)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Are there new Packets to Transmit? -> YES )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(In this TB there are , 5,  packets )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 1,  transmits the TB)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(TTI , 0.003)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there an Acknowledgement? -> NO)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Are there free processes? -> YES )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(DEFAULT BEAER , 15)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Are there new Packets to Transmit? -> YES )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(In this TB there are , 5,  packets )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 2,  transmits the TB)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(TTI , 0.004)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there an Acknowledgement? -> NO)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Are there free processes? -> YES )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(DEFAULT BEAER , 10)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Are there new Packets to Transmit? -> YES )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(In this TB there are , 5,  packets )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 3,  transmits the TB)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(TTI , 0.005)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there an Acknowledgement? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is Nak? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 0,  is setted in 'RTX' state)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The NAK in PUCCH is discarted)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(How many msg in PUCCH? , 0)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there NAK and RV < 4? -> YES)
```

```
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 0,  retransmits)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(TTI , 0.006)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there an Acknowledgement? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is Nak? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 1,  is setted in 'RTX' state)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The NAK in PUCCH is discarted)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(How many msg in PUCCH? , 0)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there NAK and RV < 4? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 1,  retransmits)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(TTI , 0.007)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there an Acknowledgement? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is Nak? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 2,  is setted in 'RTX' state)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The NAK in PUCCH is discarted)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(How many msg in PUCCH? , 0)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there NAK and RV < 4? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 2,  retransmits)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(TTI , 0.008)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there an Acknowledgement? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is Nak? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 3,  is setted in 'RTX' state)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The NAK in PUCCH is discarted)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(How many msg in PUCCH? , 0)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there NAK and RV < 4? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 3,  retransmits)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(TTI , 0.009)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there an Acknowledgement? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is Nak? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 0,  is setted in 'RTX' state)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The NAK in PUCCH is discarted)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(How many msg in PUCCH? , 0)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there NAK and RV < 4? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 0,  retransmits)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(TTI , 0.01)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there an Acknowledgement? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is Nak? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 1,  is setted in 'RTX' state)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The NAK in PUCCH is discarted)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(How many msg in PUCCH? , 0)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there NAK and RV < 4? -> YES)
```

```
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 1,  retransmits)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(TTI , 0.011)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there an Acknowledgement? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is Nak? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 2,  is setted in 'RTX' state)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The NAK in PUCCH is discarted)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(How many msg in PUCCH? , 0)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there NAK and RV < 4? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 2,  retransmits)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(TTI , 0.012)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there an Acknowledgement? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is Nak? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 3,  is setted in 'RTX' state)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The NAK in PUCCH is discarted)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(How many msg in PUCCH? , 0)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there NAK and RV < 4? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 3,  retransmits)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(TTI , 0.013)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there an Acknowledgement? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is Nak? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 0,  is setted in 'RTX' state)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The NAK in PUCCH is discarted)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(How many msg in PUCCH? , 0)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there NAK and RV < 4? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 0,  retransmits)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(TTI , 0.014)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there an Acknowledgement? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is Nak? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 1,  is setted in 'RTX' state)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The NAK in PUCCH is discarted)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(How many msg in PUCCH? , 0)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there NAK and RV < 4? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 1,  retransmits)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(TTI , 0.015)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there an Acknowledgement? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is Nak? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 2,  is setted in 'RTX' state)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The NAK in PUCCH is discarted)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(How many msg in PUCCH? , 0)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there NAK and RV < 4? -> YES)
```

```
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 2,  retransmits)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(TTI , 0.016)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there an Acknowledgement? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is Nak? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 3,  is setted in 'RTX' state)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The NAK in PUCCH is discarted)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(How many msg in PUCCH? , 0)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there NAK and RV < 4? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 3,  retransmits)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(TTI , 0.017)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there an Acknowledgement? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is Nak? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 0,  is setted in 'RTX' state)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The NAK in PUCCH is discarted)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(How many msg in PUCCH? , 0)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there NAK with RV--MAX ? -> YES )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 0,  drop the TB and is setted in 'IDLE' mode)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Are there free processes? -> YES )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(DEFAULT BEAER , 5)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Are there new Packets to Transmit? -> YES )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(In this TB there are , 5,  packets )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 0,  transmits the TB)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(TTI , 0.018)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there an Acknowledgement? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is Nak? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 1,  is setted in 'RTX' state)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The NAK in PUCCH is discarted)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(How many msg in PUCCH? , 0)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there NAK with RV--MAX ? -> YES )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 1,  drop the TB and is setted in 'IDLE' mode)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Are there free processes? -> YES )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(DEFAULT BEAER , 0)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(TTI , 0.019)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there an Acknowledgement? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is Nak? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 2,  is setted in 'RTX' state)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The NAK in PUCCH is discarted)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(How many msg in PUCCH? , 0)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there NAK with RV--MAX ? -> YES )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 2,  drop the TB and is setted in 'IDLE' mode)
```

```
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 2,  drop the TB and is setted in 'IDLE' mode)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Are there free processes? -> YES )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(DEFAULT BEAER , 0)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(TTI , 0.02)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there an Acknowledgement? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is Nak? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 3,  is setted in 'RTX' state)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The NAK in PUCCH is discarted)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(How many msg in PUCCH? , 0)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there NAK with RV--MAX ? -> YES )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 3,  drop the TB and is setted in 'IDLE' mode)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Are there free processes? -> YES )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(DEFAULT BEAER , 0)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(TTI , 0.021)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there an Acknowledgement? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is Nak? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 0,  is setted in 'RTX' state)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The NAK in PUCCH is discarted)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(How many msg in PUCCH? , 0)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there NAK and RV < 4? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 0,  retransmits)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(TTI , 0.022)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there an Acknowledgement? -> NO)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Are there free processes? -> YES )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(DEFAULT BEAER , 0)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(TTI , 0.023)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there an Acknowledgement? -> NO)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Are there free processes? -> YES )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(DEFAULT BEAER , 0)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(TTI , 0.024)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there an Acknowledgement? -> NO)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Are there free processes? -> YES )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(DEFAULT BEAER , 0)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(TTI , 0.025)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there an Acknowledgement? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is Nak? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 0,  is setted in 'RTX' state)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The NAK in PUCCH is discarted)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(How many msg in PUCCH? , 0)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there NAK and RV < 4? -> YES)
```

```
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Are there free processes? -> YES )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(DEFAULT BEAER , 0)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(TTI , 0.029)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there an Acknowledgement? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is Nak? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 0,  is setted in 'RTX' state)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The NAK in PUCCH is discarted)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(How many msg in PUCCH? , 0)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there NAK and RV < 4? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 0,  retransmits)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(TTI , 0.03)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there an Acknowledgement? -> NO)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Are there free processes? -> YES )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(DEFAULT BEAER , 0)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(TTI , 0.031)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there an Acknowledgement? -> NO)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Are there free processes? -> YES )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(DEFAULT BEAER , 0)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(TTI , 0.032)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there an Acknowledgement? -> NO)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Are there free processes? -> YES )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(DEFAULT BEAER , 0)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(TTI , 0.033)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there an Acknowledgement? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is Nak? -> YES)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 0,  is setted in 'RTX' state)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The NAK in PUCCH is discarted)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(How many msg in PUCCH? , 0)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Is there NAK with RV--MAX ? -> YES )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The process , 0,  drop the TB and is setted in 'IDLE' mode)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Are there free processes? -> YES )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(DEFAULT BEAER , 0)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Are there new Packets to Transmit? -> NO )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(Are there BUSY processes? -> NO )
EnbDownlinkPacketScheduler:DoRunPacketScheduler(The transmission ENDS)
EnbDownlinkPacketScheduler:DoRunPacketScheduler(TTI , 0.034)

---[Simulation finished]---

jon@jon-laptop:~/Desktop/ns-3-lte$
```

CAPITOLO

5 APPENDICE

5.1 Links

Simulatore ns-3 e documentazione

`http://www.nsnam.com`

Codice e documentazione sui progetti ns-3 dell'Università di Padova, Dipartimento Ingegneria dell'Informazione

`http://sourceforge.net/projects/ns3-lte/`

Ultimi aggiornamenti dei ns-3 e ultima versione di ns-3-lte

`http://code.nsnam.org/?sort=lastchange`

BIBLIOGRAFIA

[1] K. Brueninghaus, D. Astdlyt, T. Silzert, S. Visuri, A. Alexiou, S. Karger, G. Seraji, *Link Performance Models for System Level Simulations of Broadband Radio Access Systems*, IEEE 16th International Oymposium on Personal, Indoor and Mobile Radio Communications, 2005

[2] F.L. Aguilar, G.R. Cidre, J.R. París, *Effective SNR Mapping Algorithms for Link Prediction Model in 802.16e*, IEEE, 2009

[3] C.M. Mehelfuhrer, M. Wrulich, J.C. Ikuno, D. Bosanska, M. Rupp, *SIMULATING THE LONG TERM EVOLUTION PHYSICAL LAYER*, 17th European Signal Processing Conference EUSIPCO, 2009

[4] J.C. Ikuno, M. Wrulich, M. Rupp, *PERFORMANCE AND MODELING OF LTE H-ARQ*, EURASIP, 2009

[5] IEEE 802.16 Broadband Wireless Access Working Group, *IEEE 802.16m Evaluation Methodology Document (EMD) - Evaluation Methodology for P802.16m-Advanced Air Interface*, IEEE, 2008

[6] N. Baldo, M. Miozzo, *Spectrum-aware Channel and PHY layer modeling for ns3*, 2009

[7] X. Chen, Z. Fei, J. Kuang, W. Sun, L. Wan, *The Application of the MI-based Link Quality Model for Link Adaptation of Rate Compatible LDPC Codes*, IEEE, 2008

[8] E. Dahlman, S. Parkvall, J. Skold, P. Beming, *3G Evolution HSPA and LTE for Mobile Broadband*, 2nd Edition, 2008

[9] Ahmed Hamza, *Long Term Evolution (LTE) - A Tutorial*, 2009

[10] 3GPP, *3GPP TS 36.300 V10.1.0*, 2010