Presentazione

L'algoritmo di scheduling Eligibility Based Round Robin: analisi delle prestazioni e confronto con il Deficit Round Robin

Svolto da:

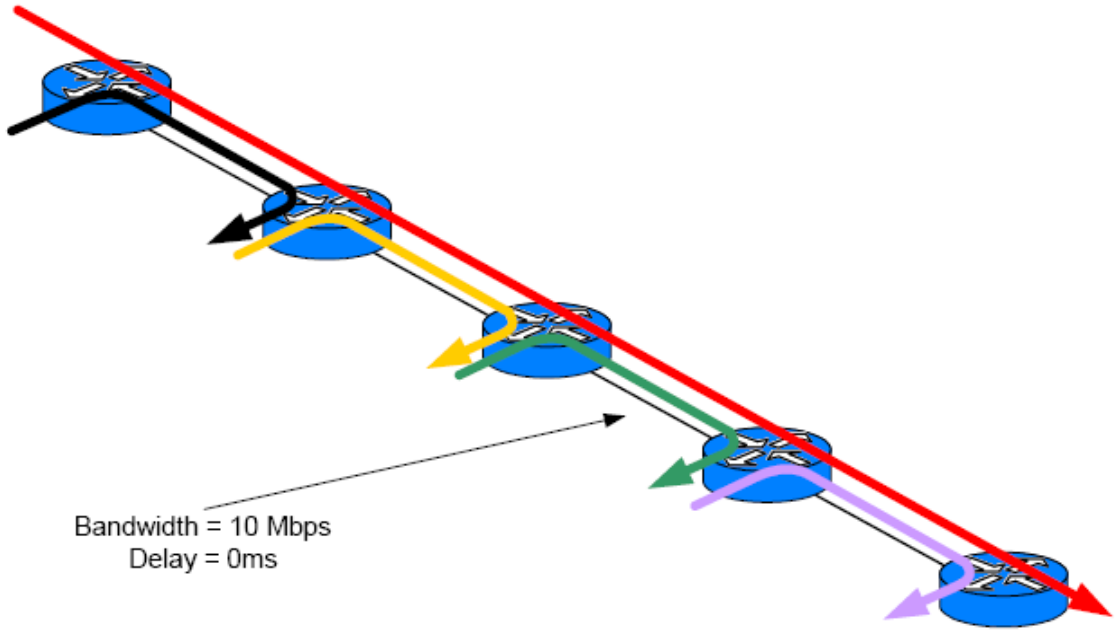
Massimiliano Atzori
Fabio Cappellini
Francesca Guerrieri
Nicola Salvo

Algoritmo EBRR

Scenari di simulazione e metriche utilizzate

Scenario di simulazione

- ✓ Tandem di N nodi collegati da un link fisico a 10Mbps
- ✓ Un particolare flusso, Tagged Flow, invia il traffico fino al nodo più a valle
- ✓ Su ogni nodo, K flussi inviano il proprio traffico verso il nodo immediatamente successivo
- ✓ Tutti i flussi usano UDP come protocollo di trasporto
- ✓ L'algoritmo di scheduling sui nodi:
  - deve garantire un rate minimo a ciascun flusso
  - deve fare in modo che il Tagged Flow non perda mai pacchetti



Scenario di simulazione e metriche

- ✓ Algoritmi di scheduling presi in esame: EBRR – DRR
- ✓ Tre scenari:

	Nodi	Dimensione pacchetti (byte)		Rate di invio (Kbps)			
				Scenario Standard		Code piene	
		Tagged Flow	Flussi Untagged	Tagged Flow	Flussi Untagged	Tagged Flow	Flussi Untagged
1 scenario	[2, 10]	1000	[100, 1500]	1000	1000	1000	1500
2 scenario	5	[200, 1500]	[100, 1500]	1000	1000	1000	1500
3 scenario	5	1000	[100, 1500]	[200, 1000]	1000	[200, 1000]	1500

- ✓ Parametri di simulazione:
  - Tempo di simulazione 150 s
  - Tempo di warm-up 5 s
  - Numero minimo/massimo di run 2/12
- ✓ Metriche utilizzate:
  - Ritardo end-to-end dei pacchetti del tagged flow: valor medio (OWD) e distribuzione cumulativa (OWD CDF)
  - Jitter end-to-end dei pacchetti del Tagged Flow: media del valore assoluto (IPDV) e distribuzione cumulativa (IPDV CDF)

Algoritmo EBRR

Descrizione dell'algoritmo di scheduling

Eligibility-Based Round Robin

Algoritmo EBRR

- ✓ L'Eligibility-Based Round Robin è una disciplina di scheduling di tipo Round Robin: ogni volta che viene servito, un flusso può trasmettere un determinato quantitativo di traffico all'interno di un round

- ✓ Ipotesi:

  - N flussi trasmettono condividendo un canale di capacità C

  - pacchetti appartenenti a flussi diversi vengono memorizzati in code FIFO distinte

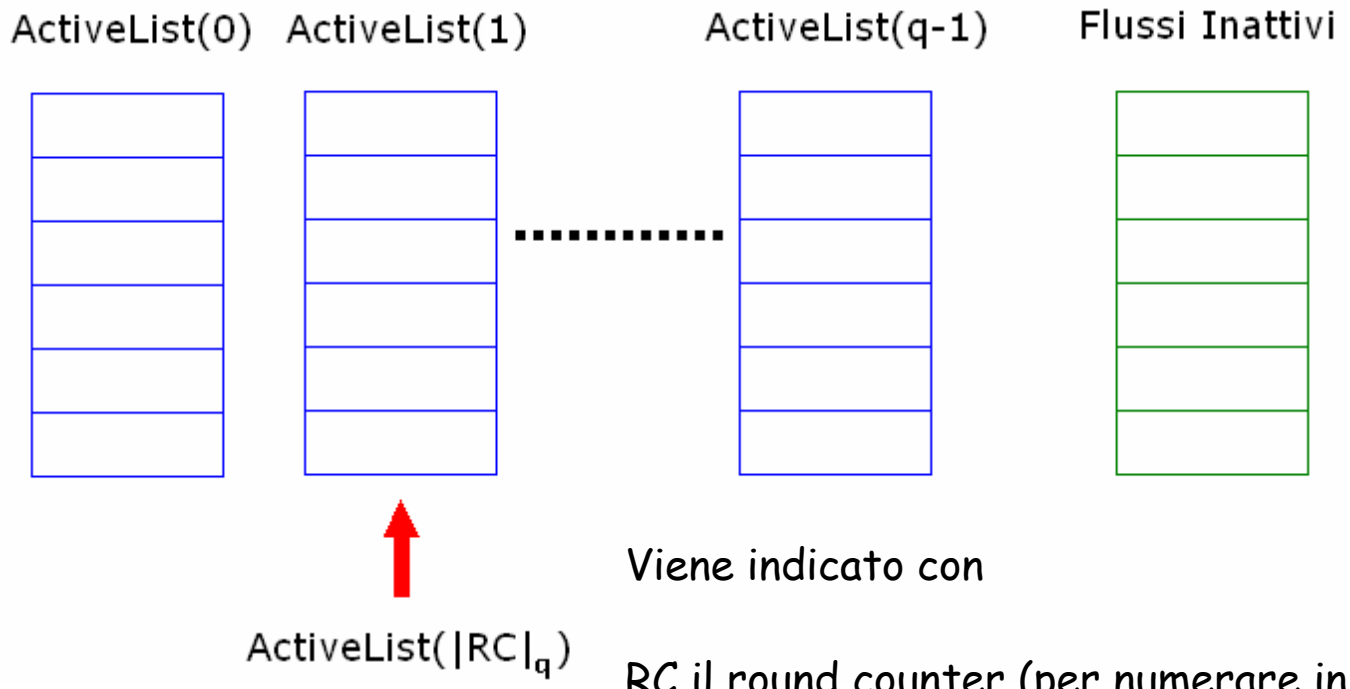
  - ogni flusso richiede un rate minimo ρ_i tale che valga la relazione $\sum_{i=1}^{N} \rho_i = C$

  - ad ogni flusso viene assegnato un quanto ϕ_i in modo tale che $\frac{\phi_i}{\phi_j} = \frac{\rho_i}{\rho_j} \quad \forall i, j$

  - il quanto rappresenta le unità di traffico che il flusso può idealmente trasmettere all'interno di un round

  - ad ogni flusso è associato un credit counter C_i che tiene traccia del servizio ricevuto in un round: viene perciò incrementato di ϕ_i unità all'inizio di un round e decrementato ogni qual volta il flusso viene servito

Active List



Viene indicato con

RC il round counter (per numerare in ordine crescente i round)

$ActiveList(|RC|_q)$ la lista dei flussi eleggibili nel round corrente

$ActiveList(|RC+k|_q)$ la lista dei flussi eleggibili nel round RC + k

✓ Ad ogni round, lo scheduler servirà solo i flussi nella lista corrente: una volta servito, un generico flusso *i* vedrà il suo credit counter C_i decrementato di ϕ_i

Active List

- ✓ Se $C_i \leq 0$, il flusso i, nell'ipotesi che rimanga backlogged, verrà servito nuovamente nel round EC_i

$$EC_i = RC + 1 + \left\lfloor \frac{-C_i}{\phi_i} \right\rfloor$$

- ✓ Di conseguenza

  - viene immediatamente inserito nella lista $ActiveList\ (|EC_i|_q)$

  - il suo credit counter è aggiornato $C_i = C_i + \left(1 + \left\lfloor \frac{-C_i}{\phi_i} \right\rfloor\right) \cdot \phi_i$

- ✓ Qualora il flusso diventi inattivo, pur aggiornando i contatori EC e C, verrà inserito nella lista dei flussi inattivi. All'arrivo di un nuovo pacchetto:

  - se $RC \geq EC_i$ il flusso verrà inserito nella lista corrente e il credit counter posto uguale al quanto

  - se $RC < EC_i$ il flusso verrà inserito nella lista $ActiveList\ (|EC_i|_q)$

Algoritmo EBRR

- ✓ Rispetto al DRR (Deficit Round Robin)

- il credit counter di un flusso può assumere valori negativi; tuttavia, non appena diventa negativo o nullo, il flusso non può più ricevere ulteriore servizio nel round corrente

- quando un flusso è eleggibile per la trasmissione, non può trasmettere più pacchetti consecutivi. Una volta trasmesso il primo, se il credit counter lo consente, viene inserito in fondo alla lista dei flussi eleggibili per la trasmissione nel round corrente

- l'utilizzo dell'eligible counter e del debito fa sì che nella lista corrente si trovino esclusivamente flussi che possono trasmettere: la dimensione del quanto non è perciò vincolata alla dimensione massima del pacchetto. Ciononostante, considerando che

$$-(L_{i_{\max}} - 1) \le C_i \le \phi_i \qquad i = 1,2,\ldots,N$$

affinché l'algoritmo funzioni correttamente, è necessario che si verifichi

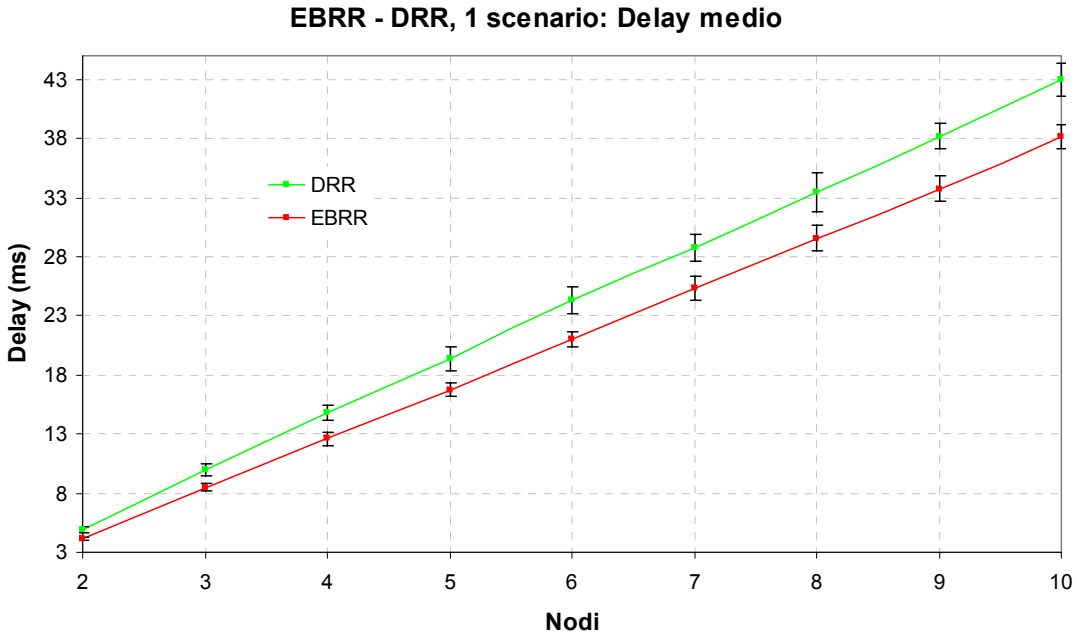
$$q \ge 2 + \max_{i=1,2,\ldots,N} \left\lfloor \frac{L_{i_{\max}} - 1}{\phi_i} \right\rfloor$$

Primo scenario EBRR

Primo scenario:
Prestazioni al variare del numero dei nodi

- ✓ Nodi: 2 ÷ 10
- ✓ Active List: 2 ÷ 9
- ✓ Dimensione pacchetti tagged: 1000 Bytes
- ✓ Dimensione pacchetti untagged: 100÷1500 Bytes
- ✓ Rate flussi tagged/untagged: 1Mb/s (code scariche), 1.5Mb/s untagged (code piene)
- ✓ Quanti uguali e minimi per assicurare rate di 1Mb/s tagged e untagged

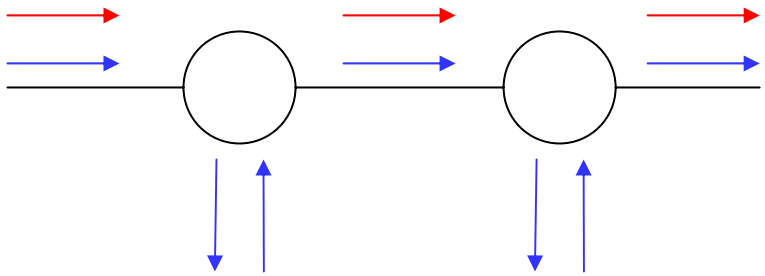
Primo scenario: Delay



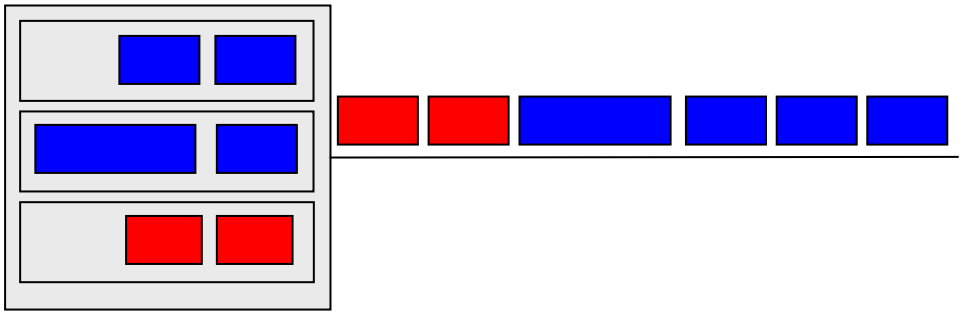
Numero di nodi

Delay medio

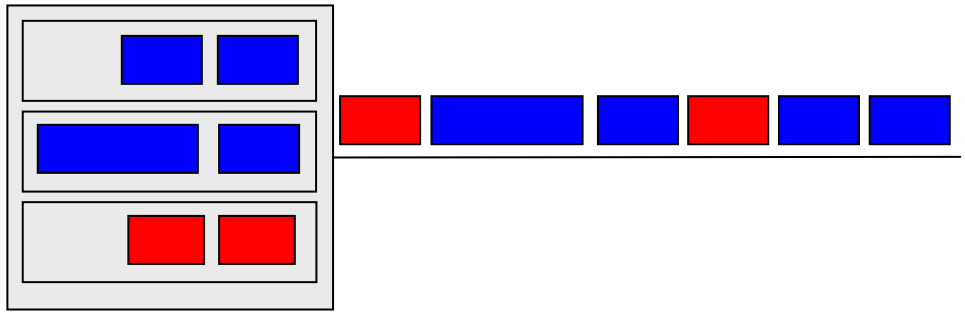
✓ Crescita lineare del delay



✓ Ritardo minore rispetto al DRR

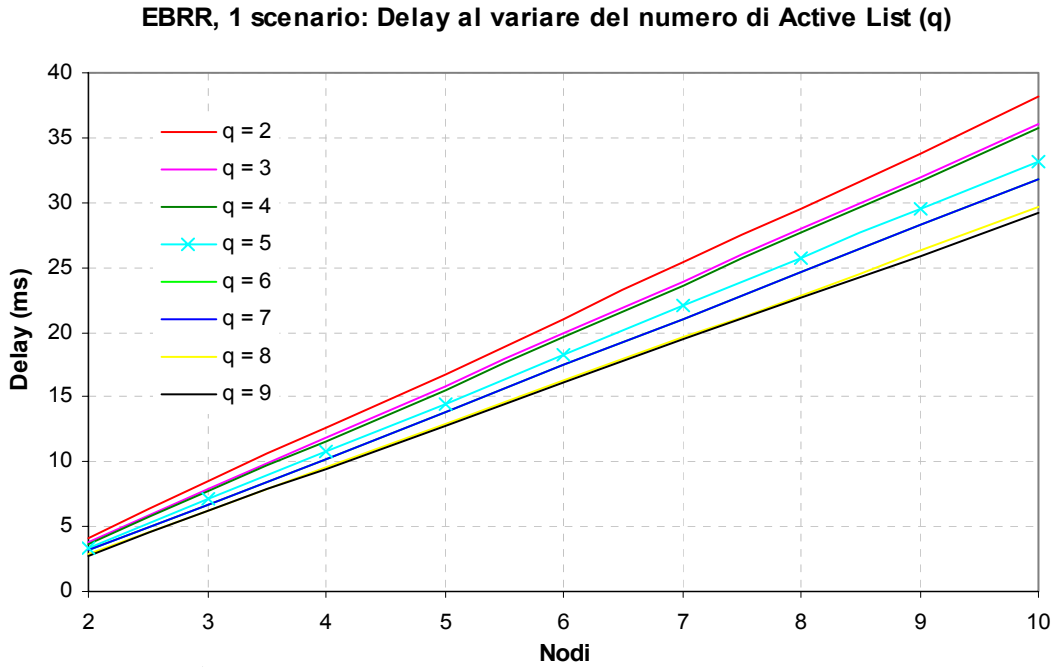


DRR



EBRR

Primo scenario: Delay



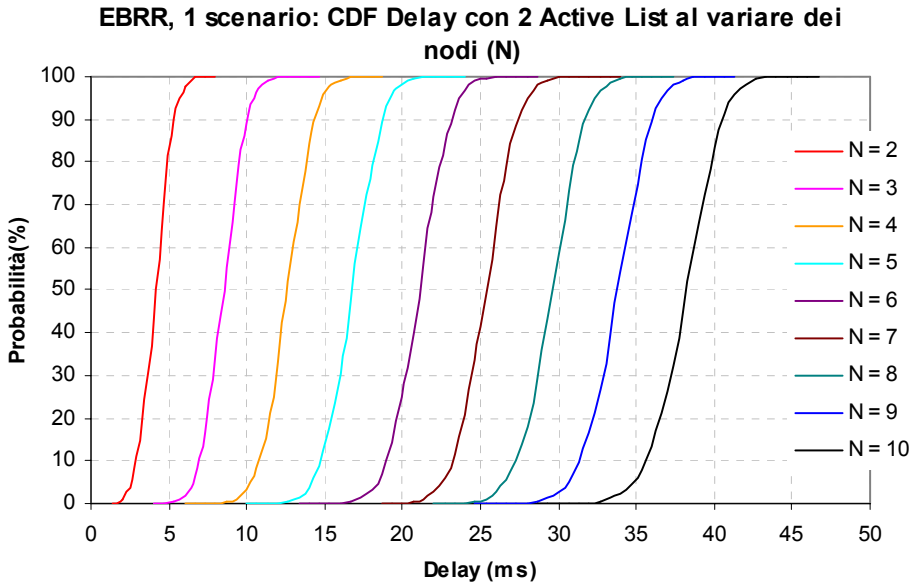
↑ Numero di active list ↓ Delay medio

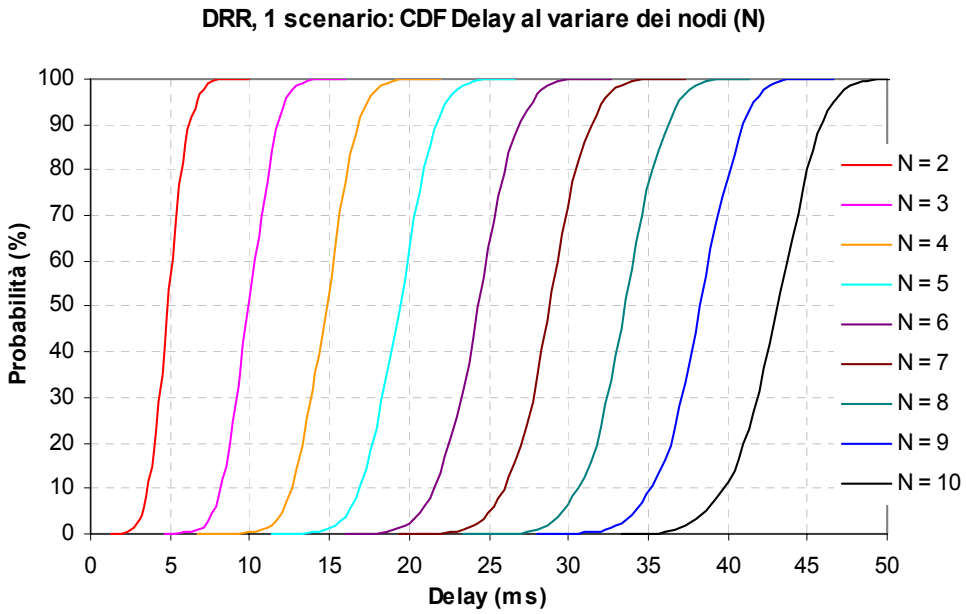
Tagged Flow Flow 1 Flow 3	Flow 4 Flow 2

Tagged Flow	Flow 3	Flow 4 Flow 2	Flow 1

✓ Il delay diminuisce ulteriormente se il quanto è sottomultiplo intero della dimensione del pacchetto

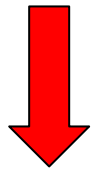
Primo scenario: Delay





L'andamento delle distribuzioni conferma quanto detto:

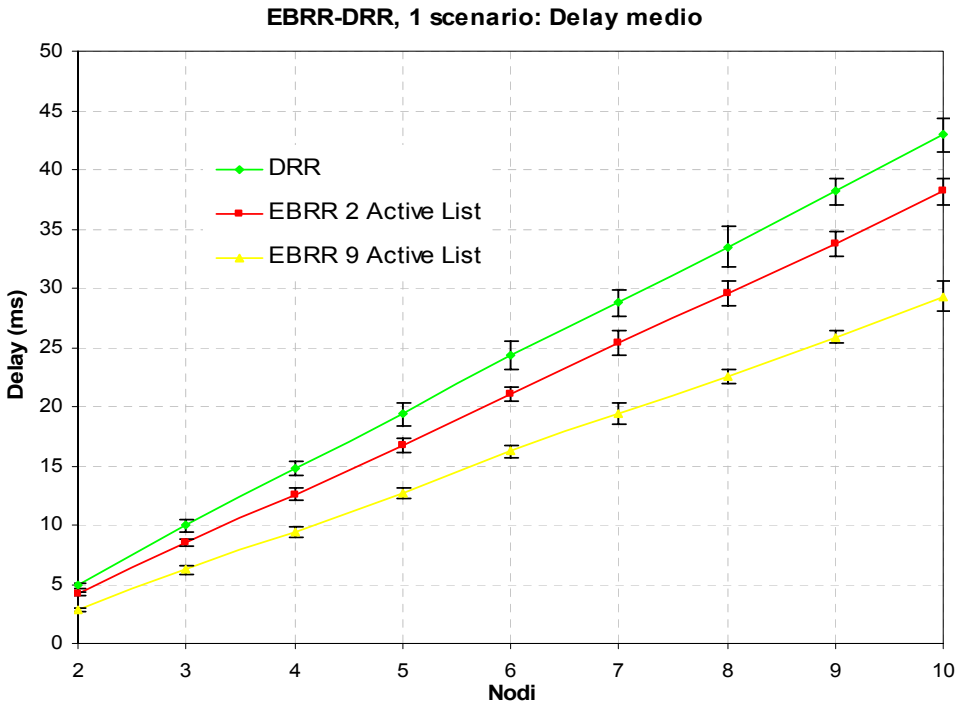
Crescita lineare del delay

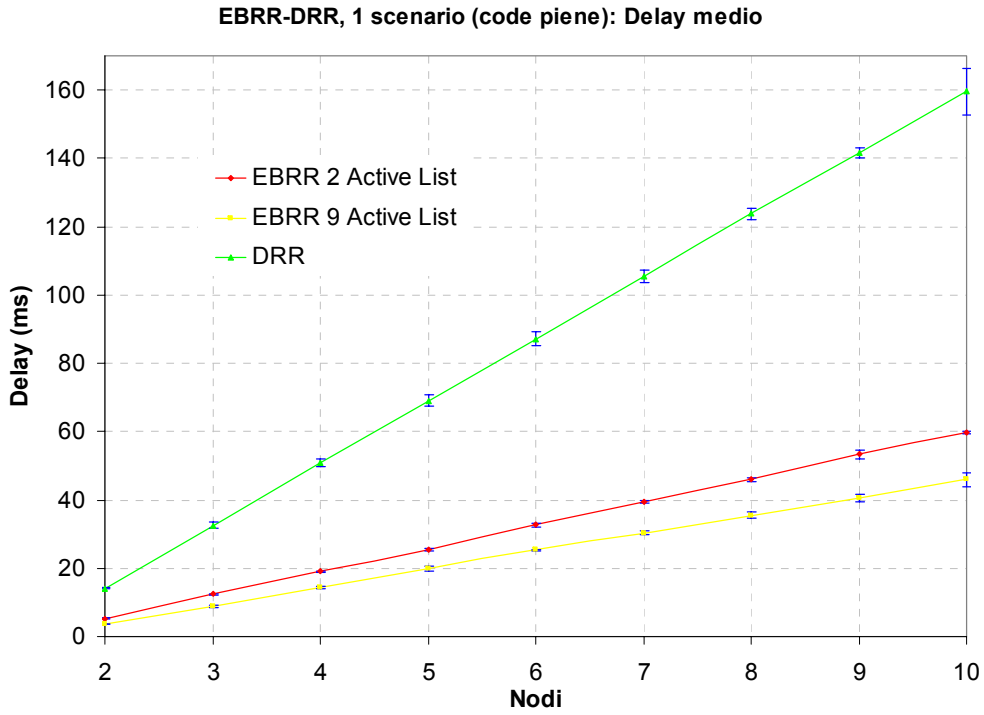


Curve equispaziate

Simmetria rispetto al valor medio

Primo scenario: Delay

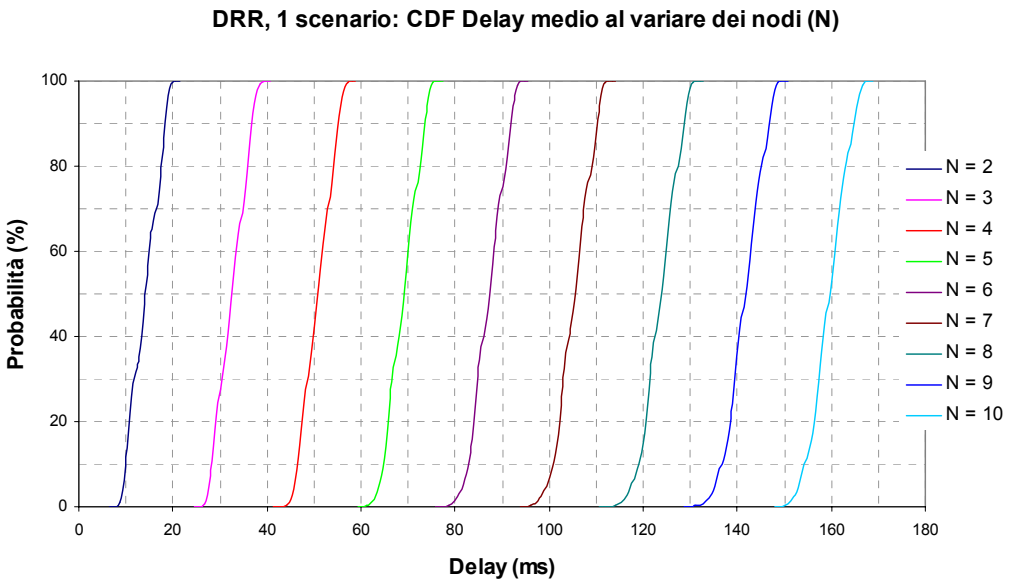


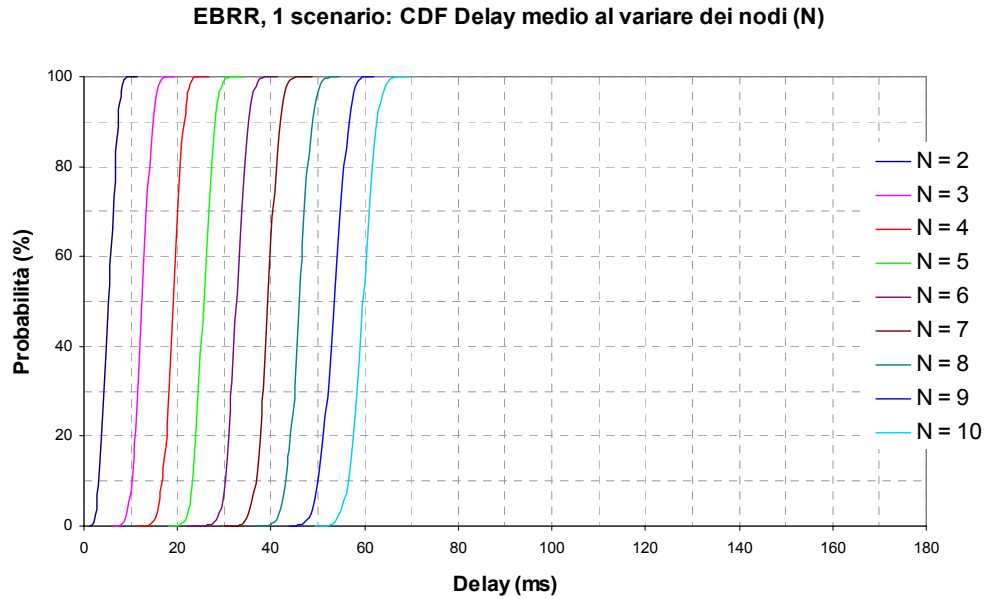


Code piene ➡ flussi untagged continuamente backlogged

- ✓ delay medio più alto
- ✓ la crescita si mantiene lineare con l'aumentare dei nodi
- ✓ più active list migliorano ulteriormente le prestazioni

Primo scenario: Delay

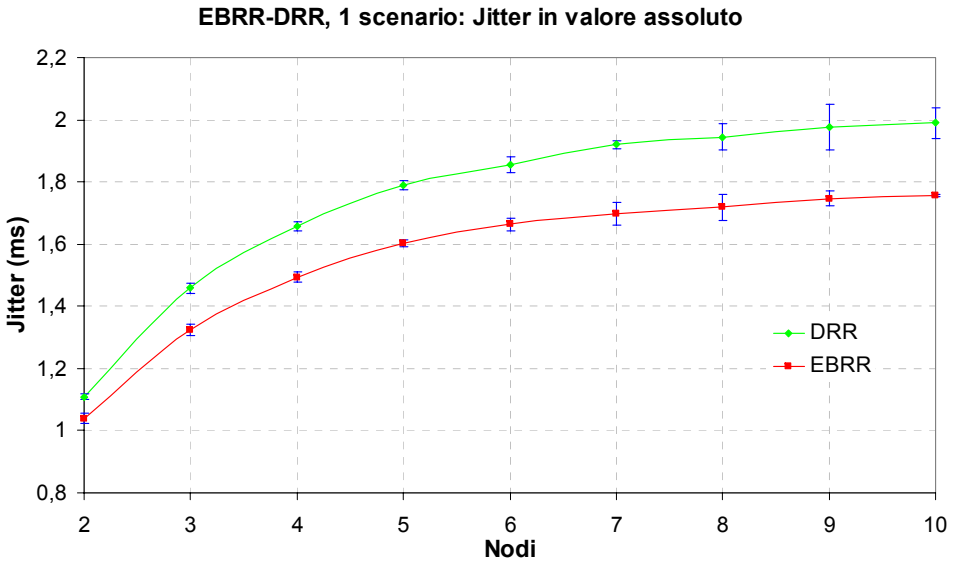


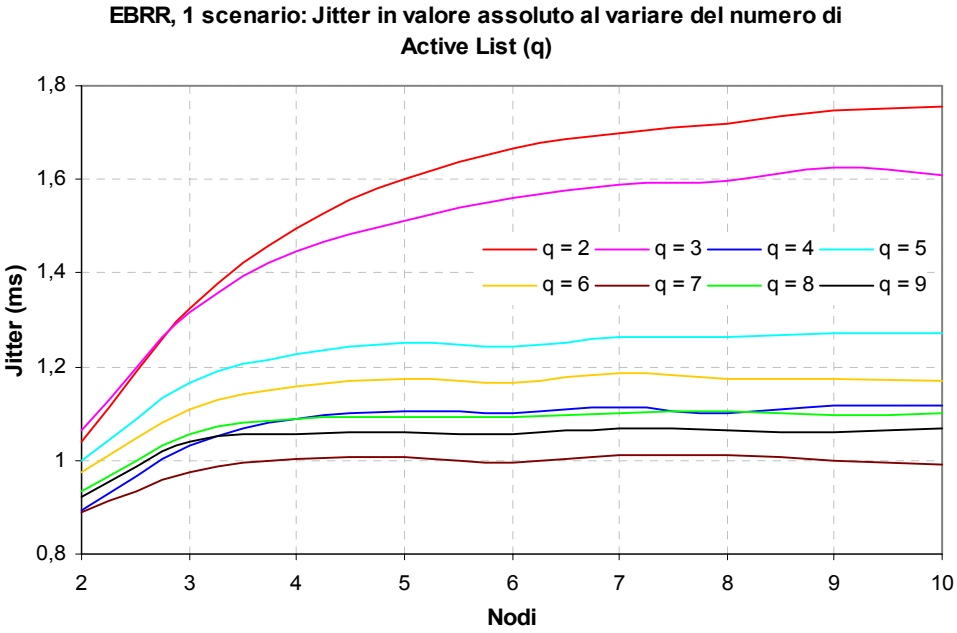


Distribuzioni cumulative:

- ✓ Il maggior ritardo introdotto dal DRR si evidenzia più chiaramente
- ✓ Si mantiene lo stesso andamento visto nel caso di sistema scarico

Primo scenario: Jitter





Andamento del Jitter

✓ All'aumentare dei nodi, la crescita è via via minore

✓ L'EBRR trasmette un pacchetto alla volta per ogni flusso in lista

Minor jitter rispetto al DRR

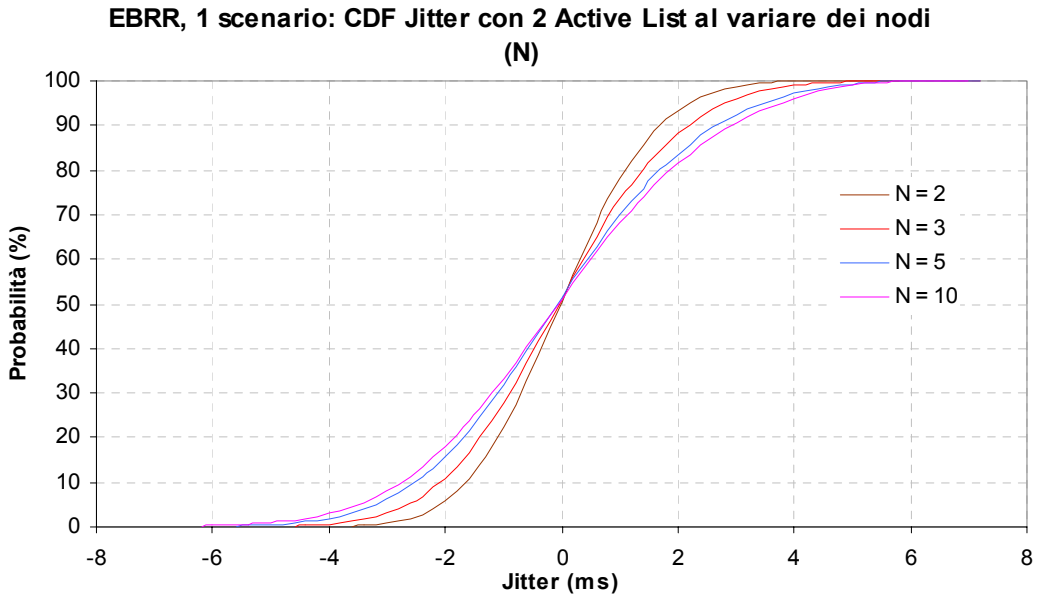
✓ La diminuzione del jitter all'aumentare del numero di active list non è lineare

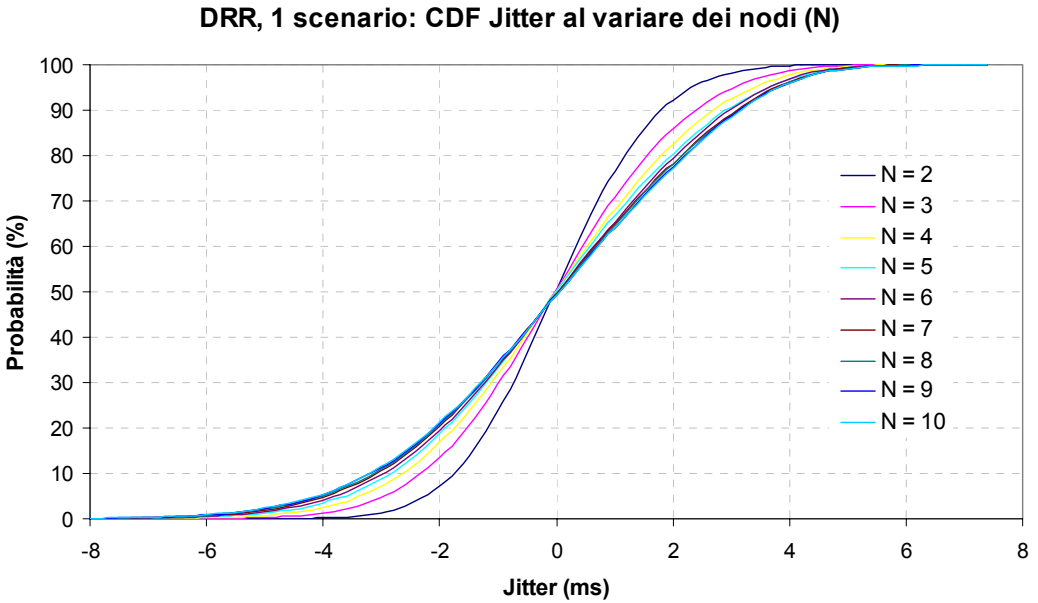
✓ Se $$\frac{L_T}{\phi_T} = \left\lfloor \frac{L_T}{\phi_T} \right\rfloor$$

il tagged flow trasmette sfruttando l'intero quanto senza resti

Invio più regolare e minor jitter

Primo scenario: Jitter

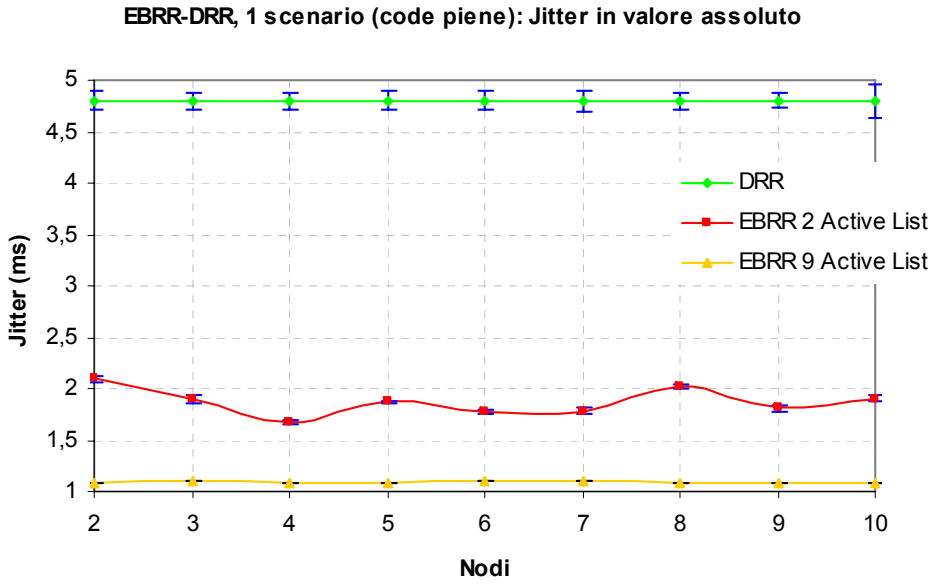


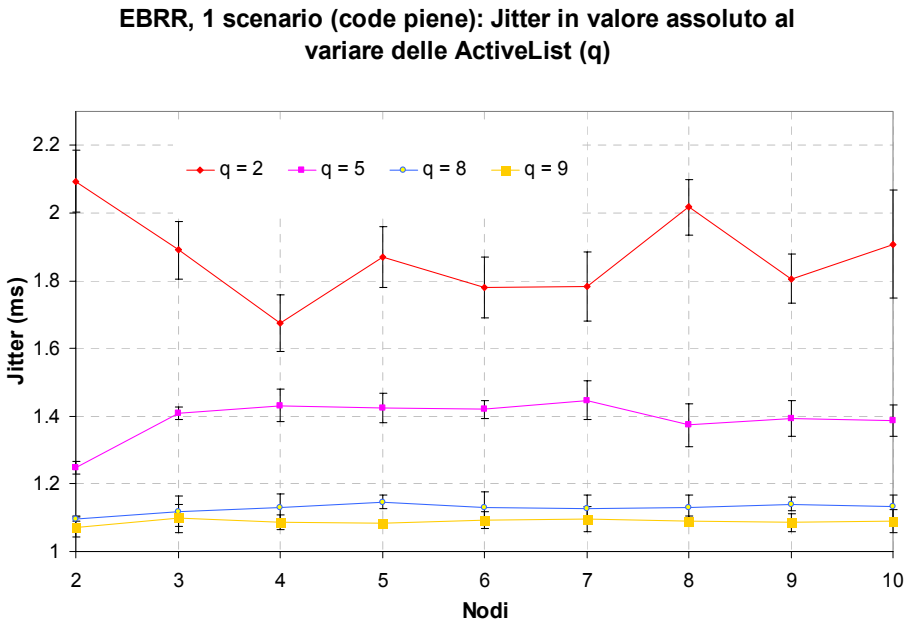


✓ L'andamento individuato attraverso la media del valore assoluto, viene confermato dalla distribuzione media

✓ La distribuzione di probabilità del jitter conferma quanto detto in precedenza, presentando curve sempre più ravvicinate al crescere dei nodi

Primo Scenario: Jitter





Code piene

✓ Flussi untagged sempre backlogged → ✓ Jitter quasi costante

✓ L'EBRR trasmette un solo pacchetto alla volta → ✓ Jitter minore rispetto al DRR

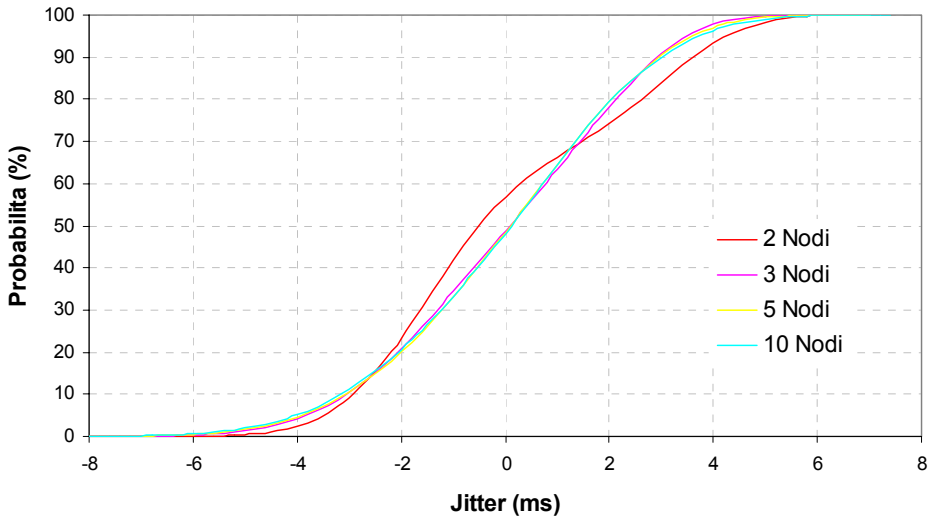
Come già osservato:

✓ ulteriore miglioramento all'aumentare del numero di Active List

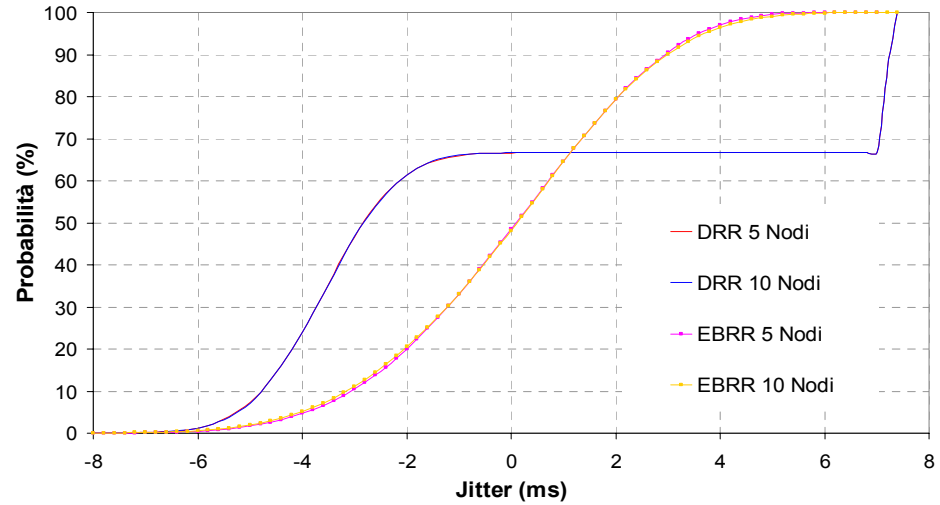
✓ situazione particolare se il quanto è sottomultiplo intero della dimensione del pacchetto

Primo Scenario: Jitter

EBRR, 1 scenario : CDF Jitter con 2 ActiveList al variare dei nodi (code piene)



EBRR-DRR, 1 scenario (code piene): CDF Jitter al variare dei nodi



Code piene

✓ Per lo scheduler EBRR, le curve mantengono lo stesso andamento rispetto al caso standard e si sovrappongono

✓ Nel DRR si notano discontinuità dovute ai pacchetti che viaggiano in burst

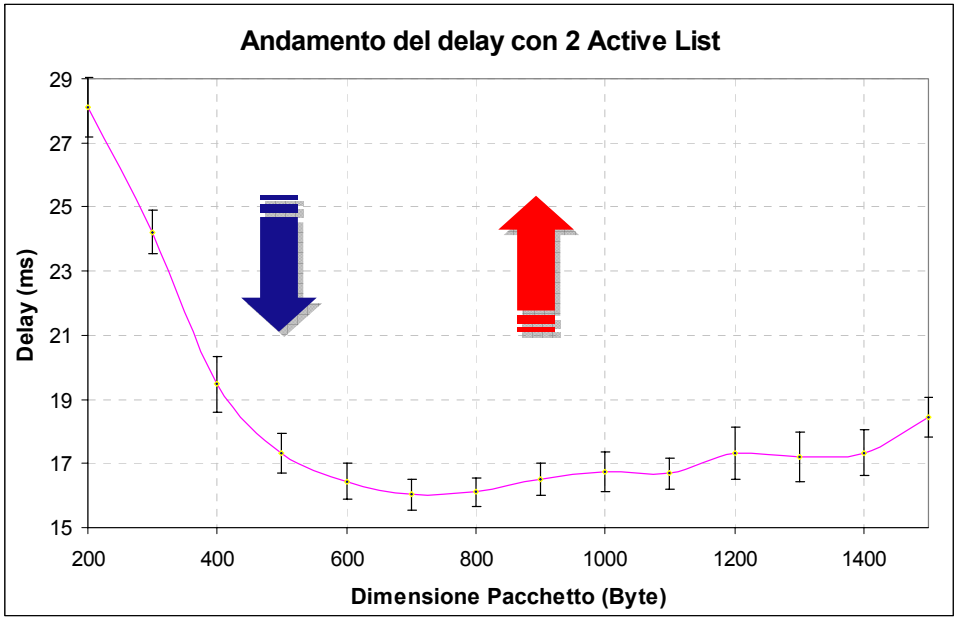
✓ Le caratteristiche del burst non cambiano all'aumentare del numero di nodi

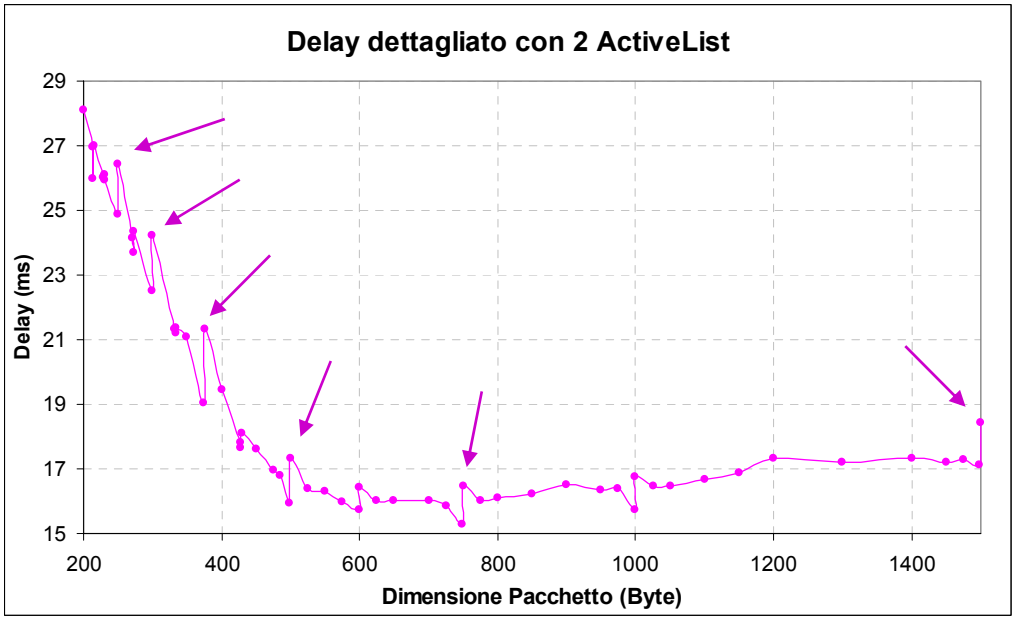
Secondo scenario

Secondo scenario:
Prestazioni al variare della dimensione dei pacchetti del Tagged Flow

- ✓ Nodi: 5
- ✓ Active List: 2 ÷ 9
- ✓ Dimensione pacchetti tagged: 200÷1500 Bytes
- ✓ Dimensione pacchetti untagged: 100÷1500 Bytes
- ✓ Rate flussi tagged/untagged: 1Mb/s (code scariche), 1.5Mb/s untagged (code piene)
- ✓ Quanti uguali e minimi per assicurare rate di 1Mb/s tagged e untagged

Secondo scenario: Delay





1. Per L <750 byte circa:

   Rate fisso
   Pacchetti piccoli
   → Tasso di arrivo in coda elevato
   → Più pacchetti sperimentano attesa prima della trasmissione
   → Delay medio alto

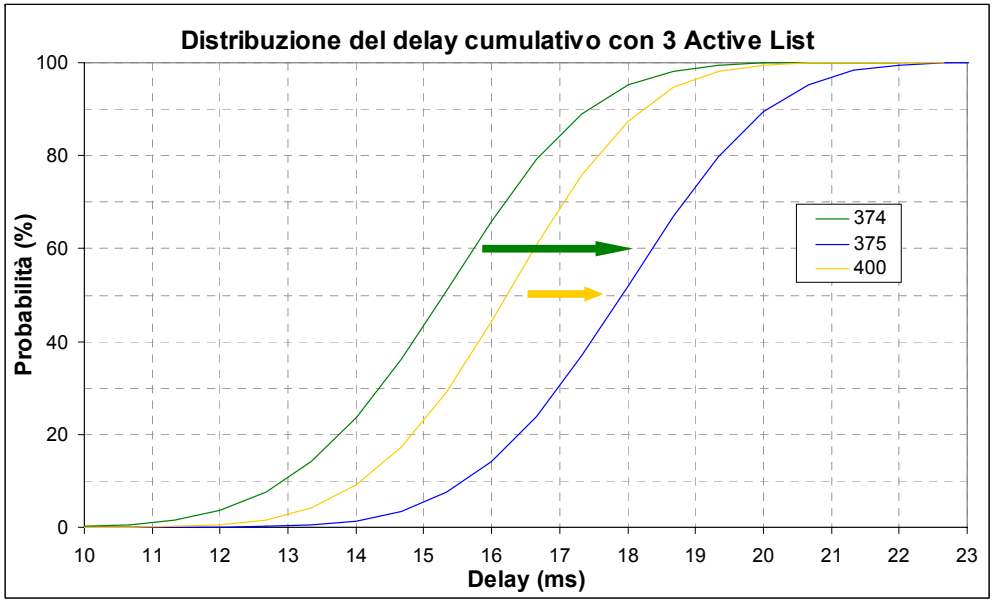
2. Per L >750 byte circa: il quanto permette meno trasmissioni nella stessa AL

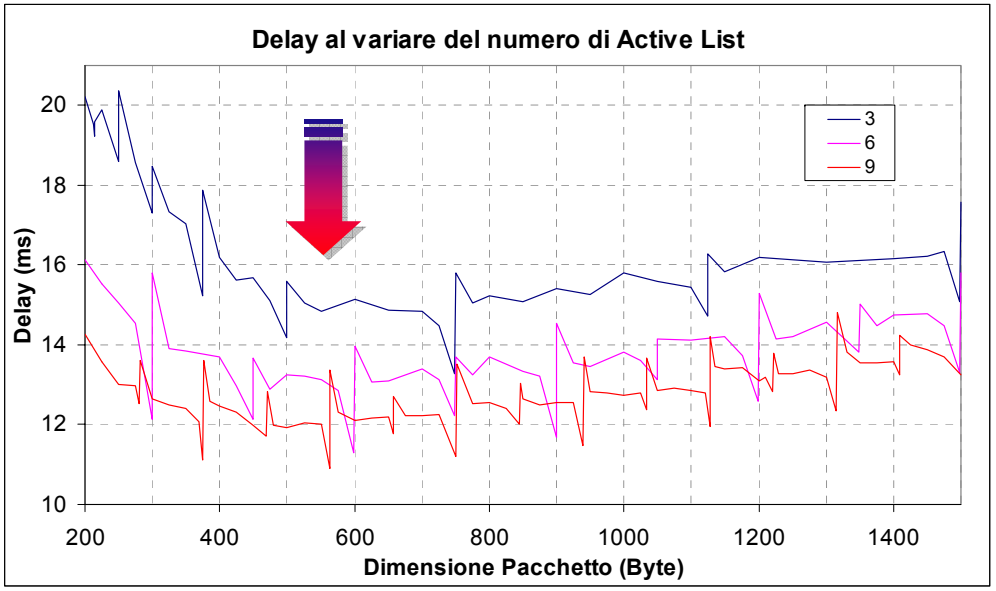
   Cambio AL → Aumento del delay

3. Per L sottomultiple del quanto: il credito residuo è nullo

   Cambio frequente di AL → Brusco aumento del delay → Punti di massimo

Secondo Scenario: Delay





✓Le CDF nei punti di discontinuità:

es. AL=3, quanto=750 bytes; con L=375 bytes la curva della distribuzione è molto più a destra sia di quella di 374 che di quella di 400

↓

Quanto/L = credito residuo nullo

Cambio frequente AL → Maggior delay

Più AL → Delay minore

Flussi distribuiti in più liste

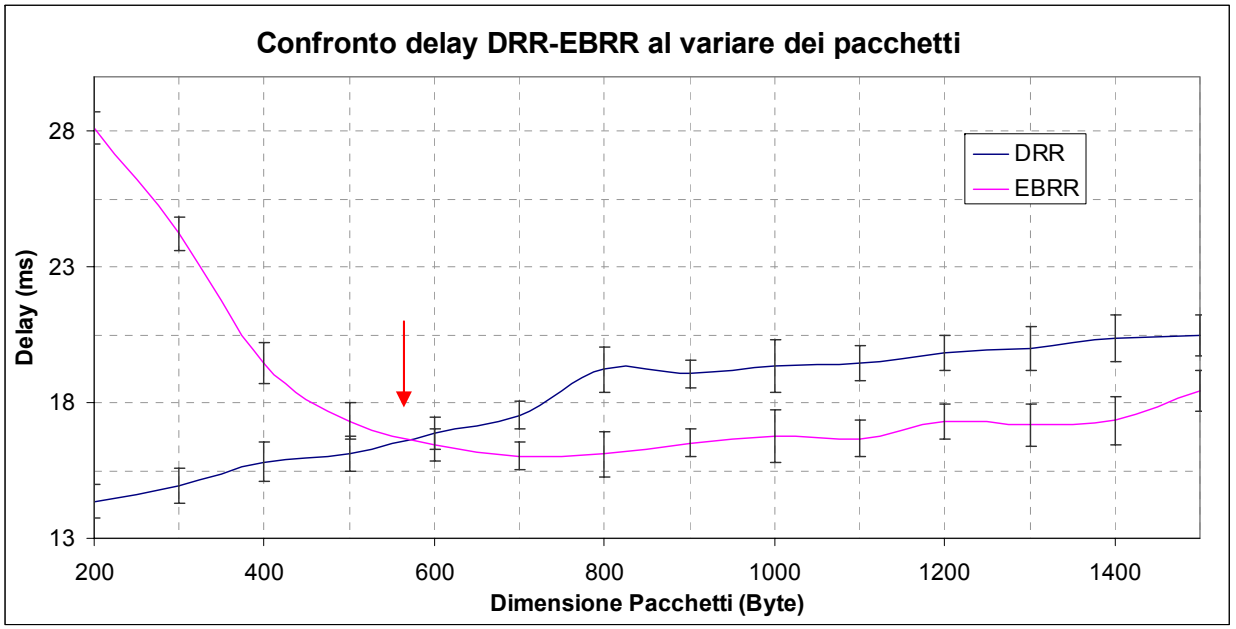
↓

Il tagged avrà sempre meno flussi davanti

↓

Il tagged aspetta meno

Secondo scenario: Delay



✓ Per L <600 byte circa: tanti pacchetti piccoli.

EBRR) tutti inviano un pacchetto alla volta

DRR) tutti possono trasmettere in burst

→ Delay DRR < Delay EBRR

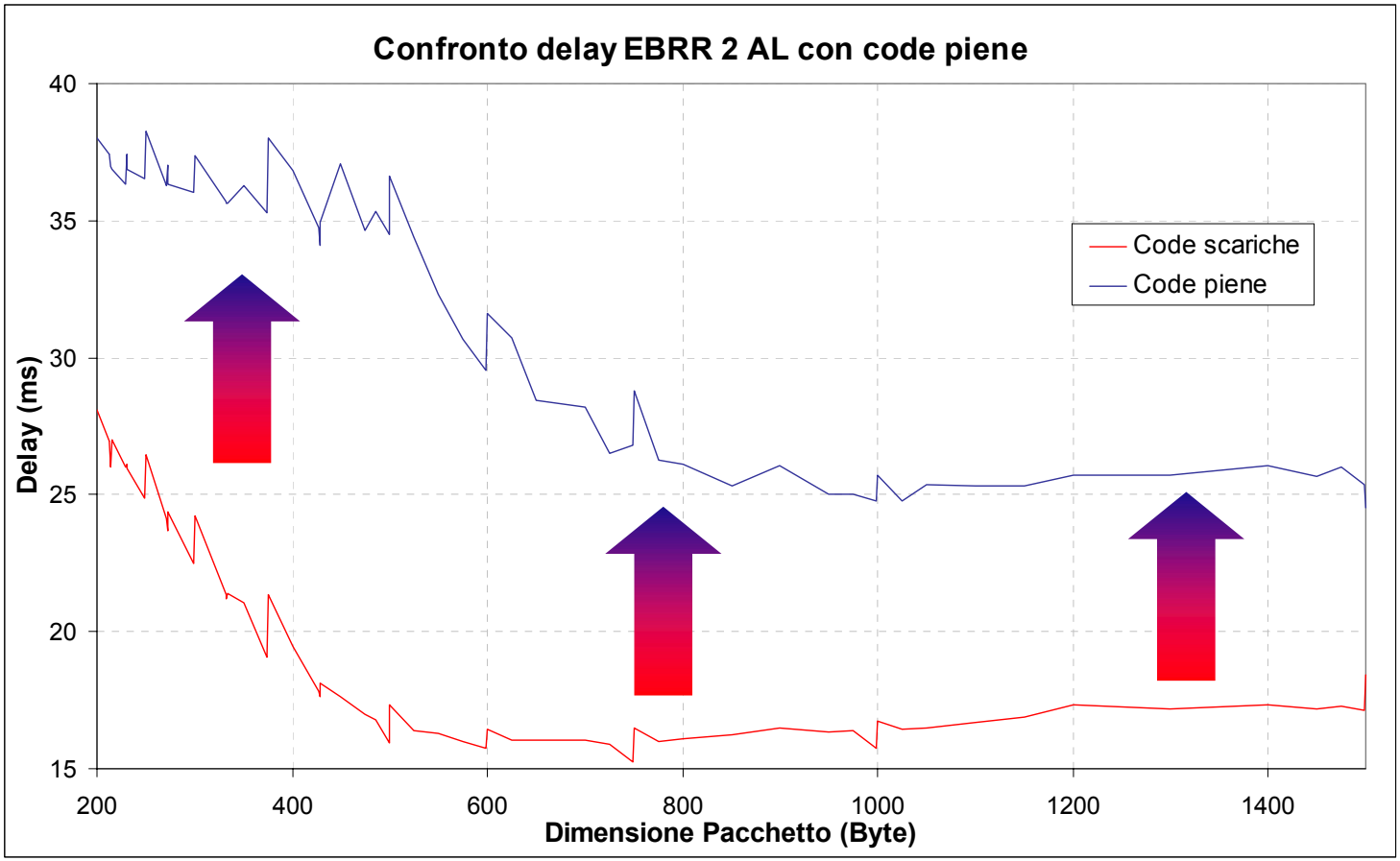
✓ Per L >600 byte circa: pochi pacchetti grandi.

EBRR) tutti trasmettono un pacchetto alla volta

DRR) il tagged può inviare burst piccoli o un pacchetto, gli altri inviano come prima (tagged svantaggiato)

→ Delay DRR > Delay EBRR

Secondo scenario: Delay

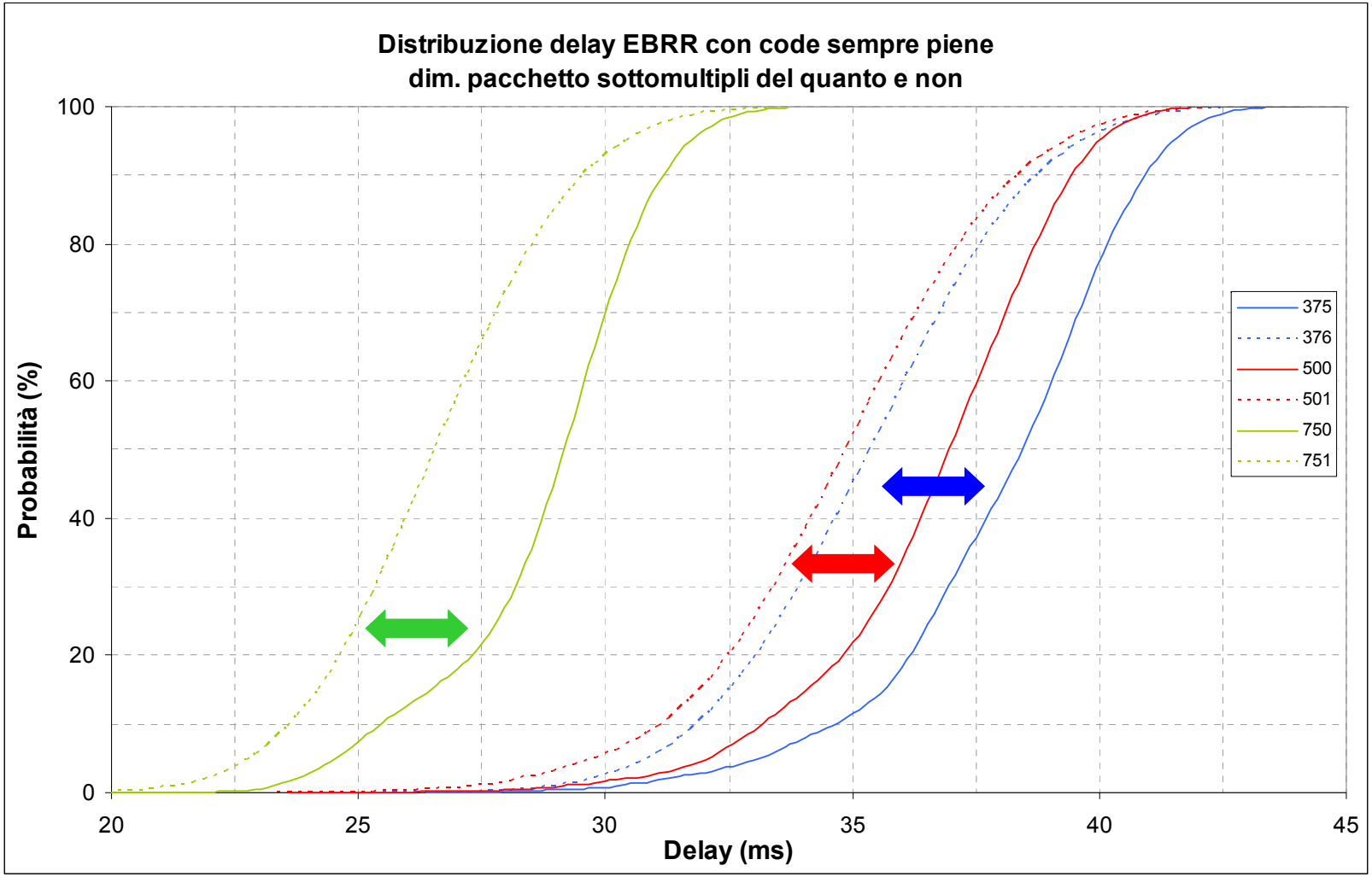


✓Code sempre piene:

Dopo l'invio di un pacchetto tagged l'attesa media aumenta perché tutti gli altri flussi hanno sempre qualcosa da trasmettere

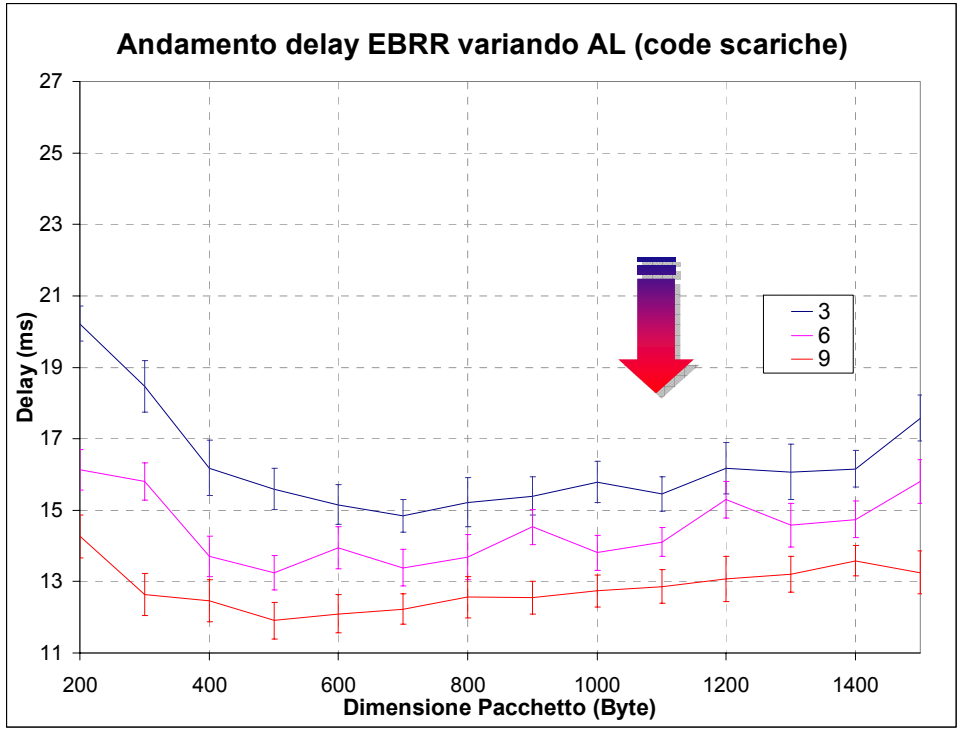
✓L'andamento resta simile al precedente

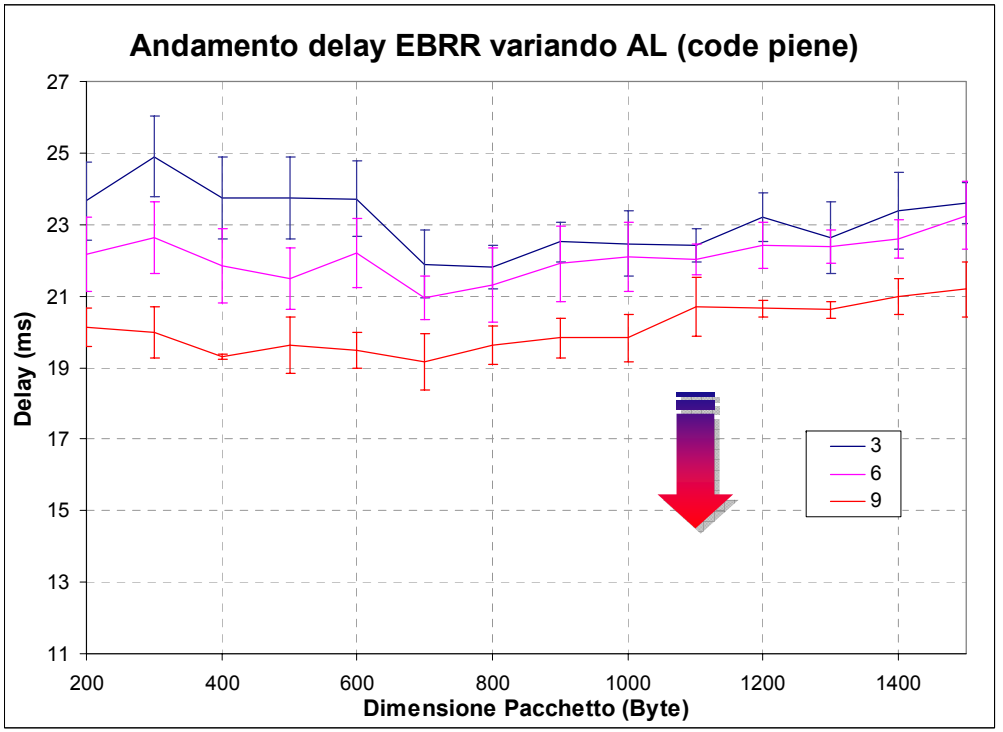
Secondo scenario: Delay



- ✓ Anche nel caso di code sempre piene si verifica il fenomeno dei punti di massimo nei sottomultipli del quanto per lo stesso motivo, come evidenziano le distribuzioni

Secondo scenario: Delay

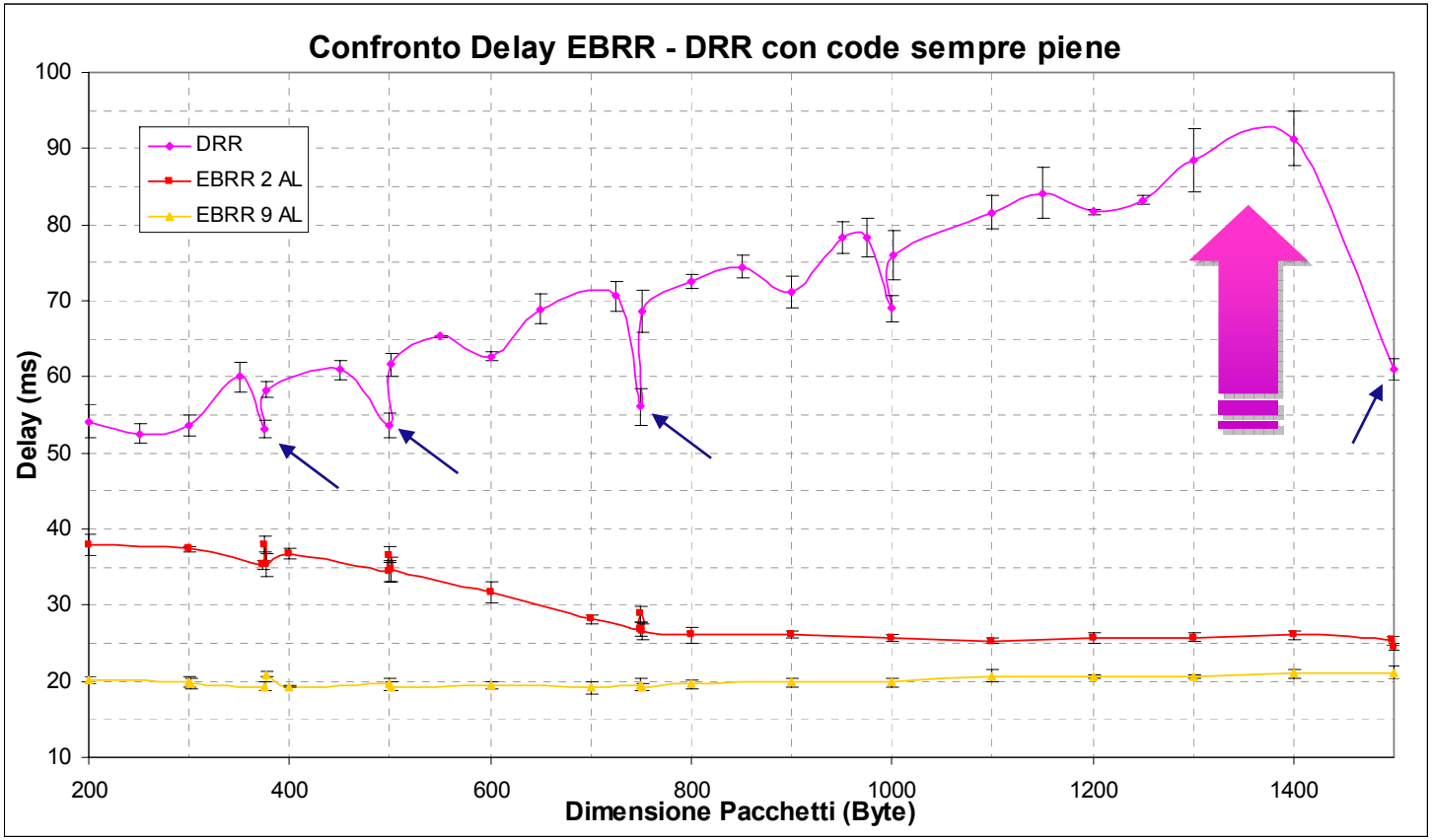




✓ Anche con code sempre piene il delay tende a diminuire all'aumentare del numero di AL

Active List → Delay Medio

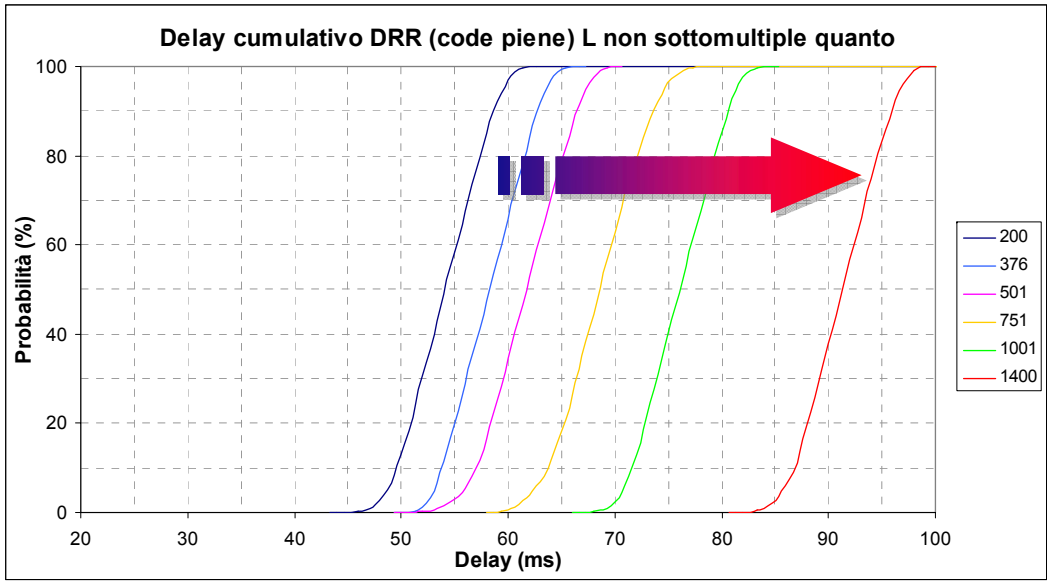
Secondo scenario: Delay

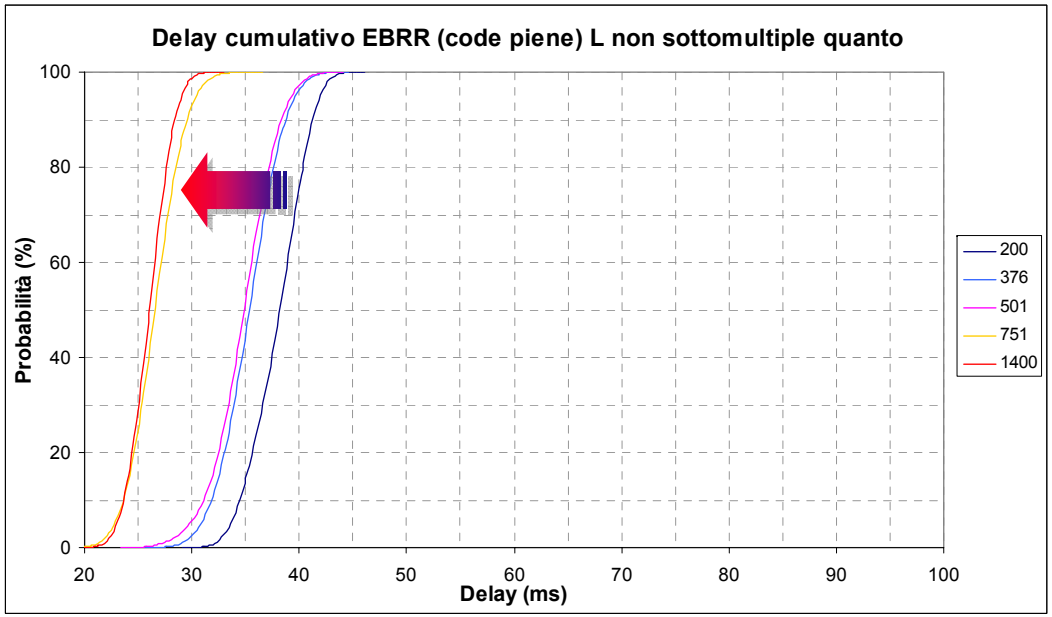


Confronto a code piene:

- ✓ Delay medio: DRR > EBRR
- ✓ Andamento delay: DRR crescente, EBRR variazioni sempre minori al crescere AL
- ✓ Punti sottomultipli: DRR minimi, EBRR massimi

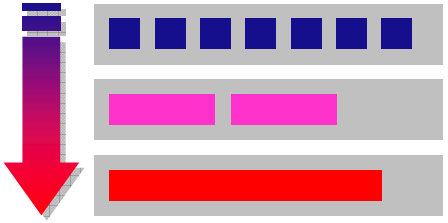
Secondo scenario: Delay





DRR:

- ✓ Quanto fisso
- ✓ L crescente
- ✓ Burst più piccoli

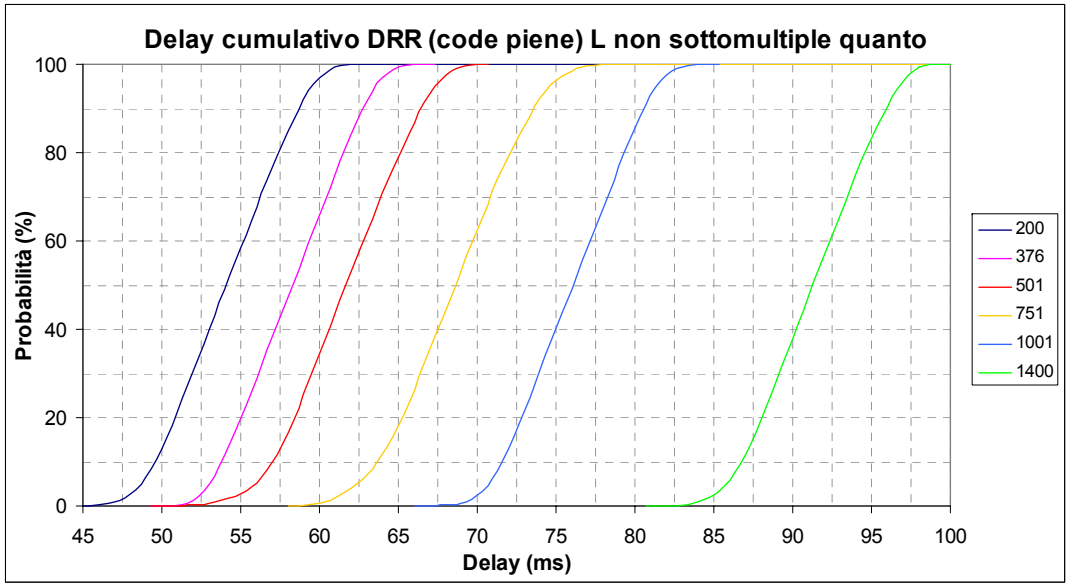


- ✓ Altre code sempre piene

EBRR:

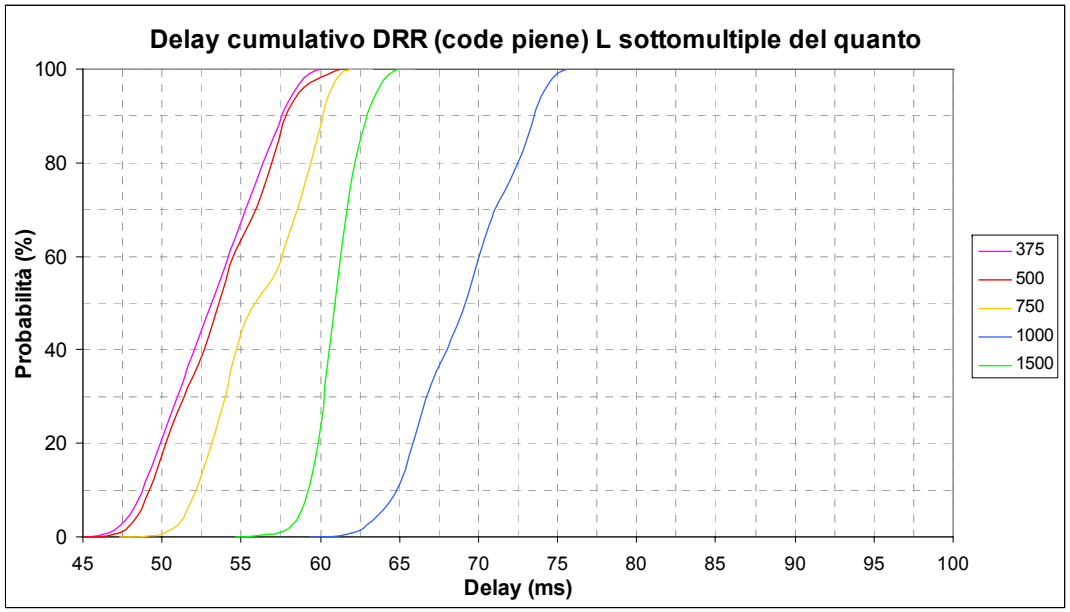
- ✓ Delay più basso e tende a distribuirsi in un intervallo di tempi minore

Secondo scenario: Delay



DRR con Quanto/L frazionario:

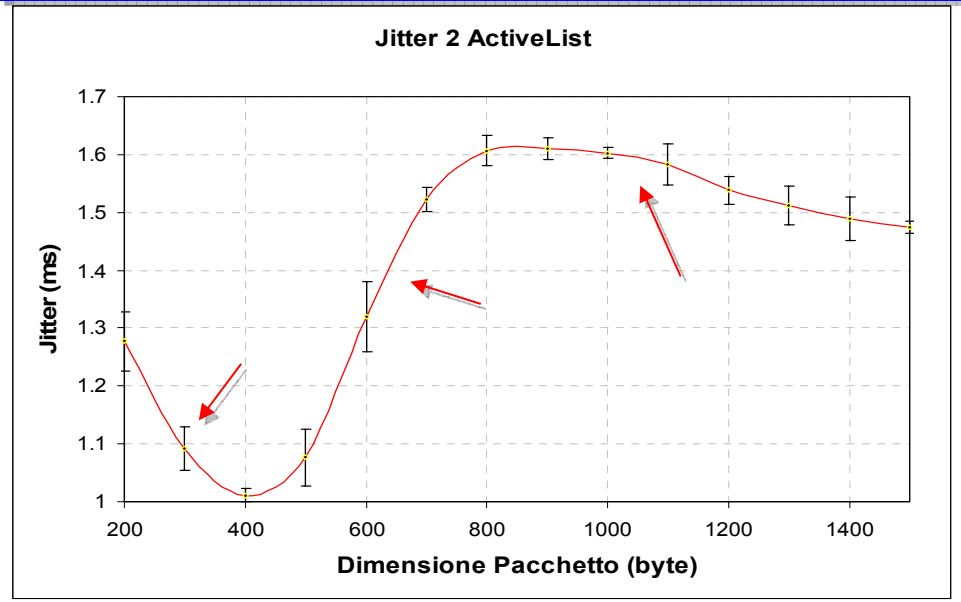
- ✓ Tagged quasi sempre backlogged
- ✓ Attesa media elevata: tutti gli altri flussi trasmettono prima di lui
- ✓ DC presenta spesso resti

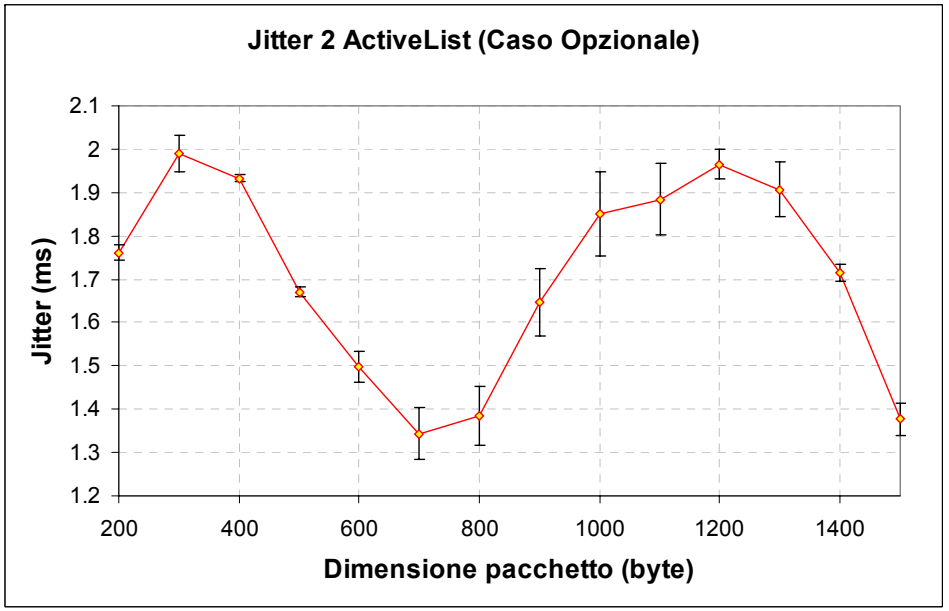


DRR con Quanto/L intero:

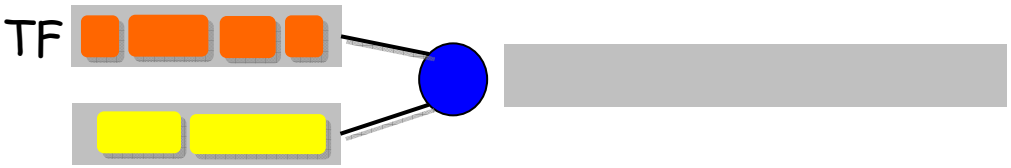
- ✓ Tagged riesce a svuotare meglio il buffer dei pacchetti in coda
- ✓ Decremento DC con meno resti

Secondo Scenario: Jitter





Trasmissione pacchetti piccoli



200 400 ↓ **Jitter:** cambio AL poco frequente

400 750 ↑ **Jitter:** i resti portano cambi più frequenti

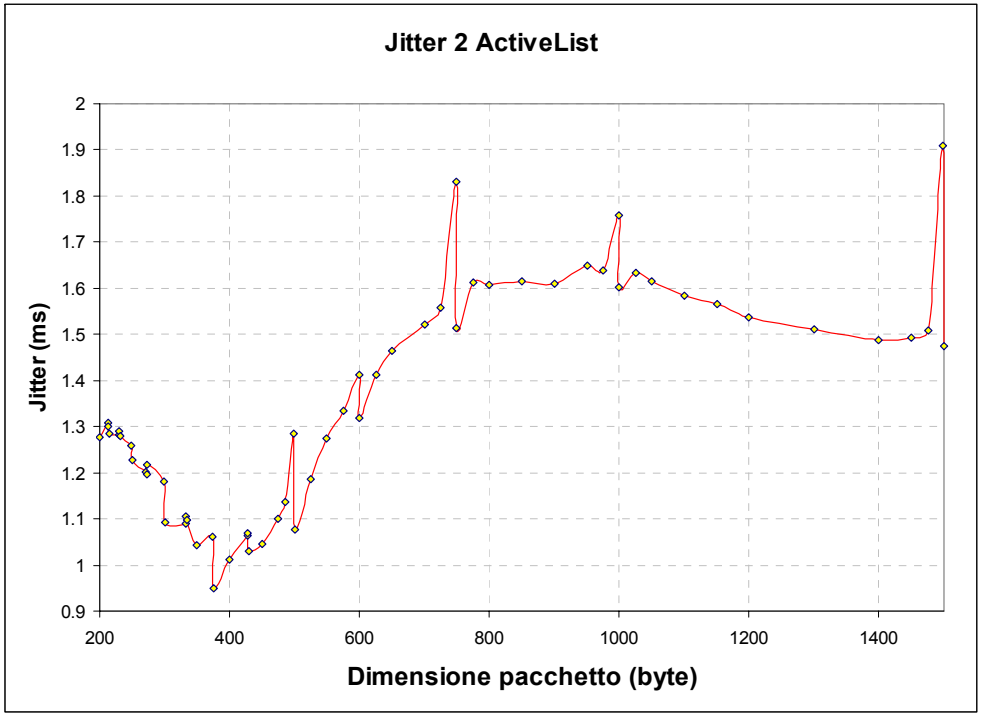
750 1500 ↓ **Jitter:** maggiore regolarità

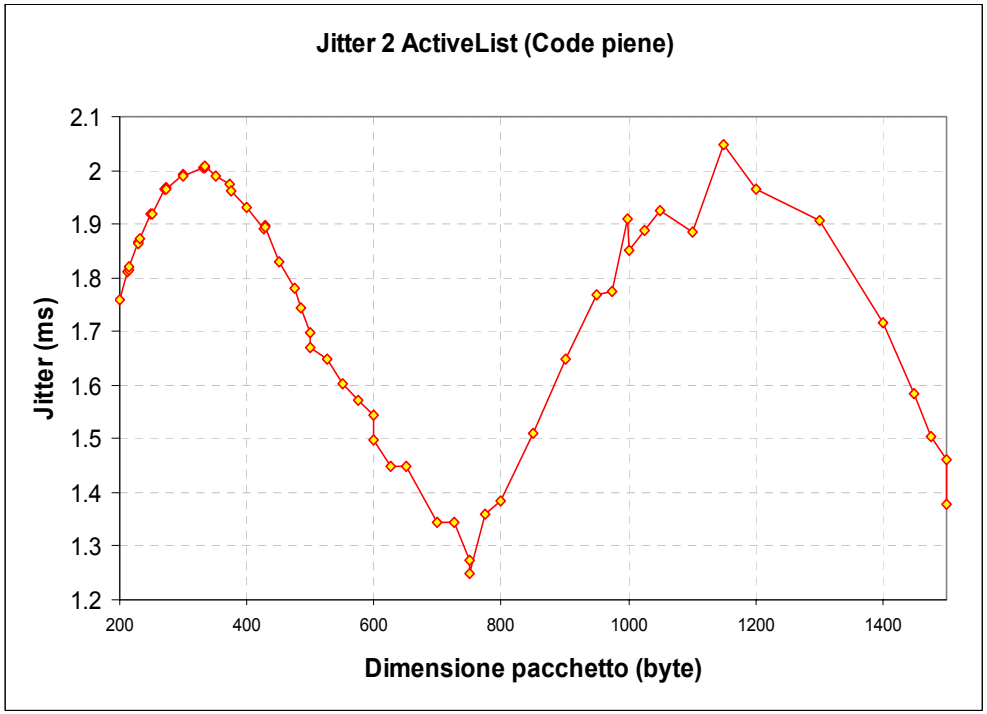
Code sempre piene:

✓ Punti minimo in presenza di sottomultipli del quanto

✓ Punti di massimo quando il resto permette la trasmissione del maggior numero di pacchetti

Secondo Scenario: Jitter

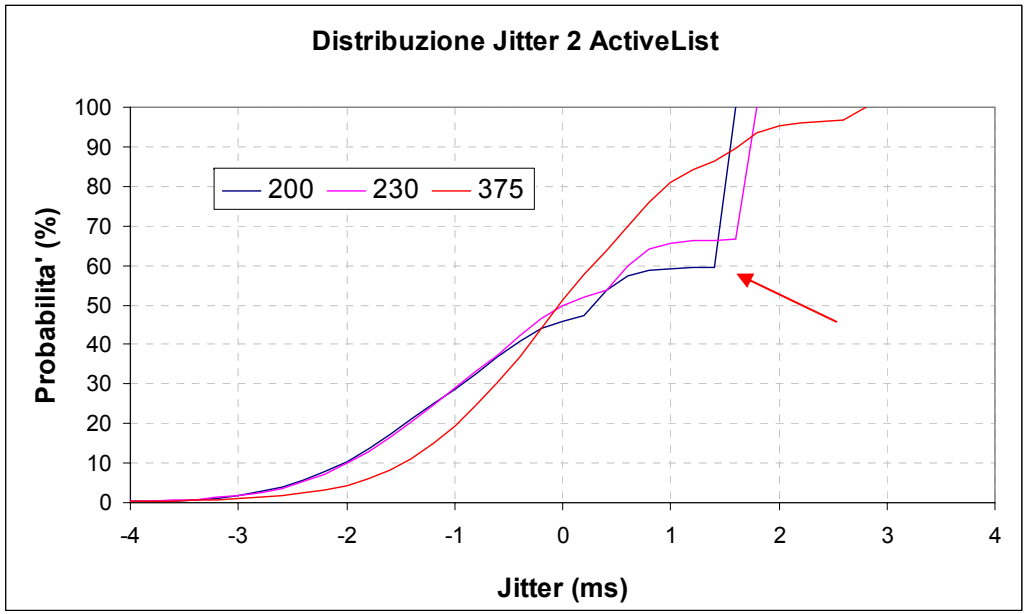




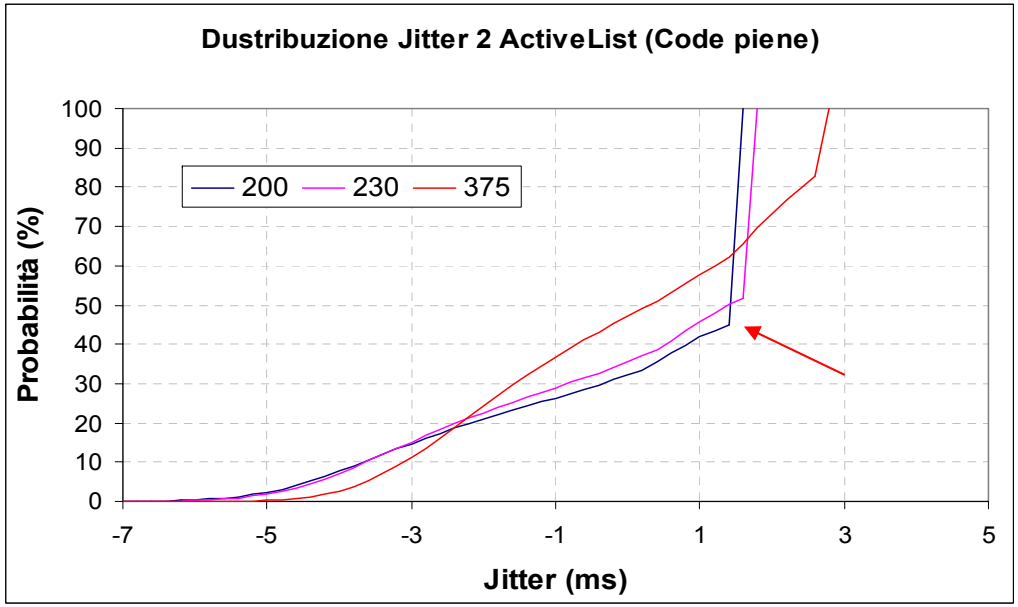
- Se quanto/L intero ➡ credito residuo nullo
  - ➢ **Jitter minimo**
- Se quanto/L frazionario ➡ credito residuo
  - ➢ **Jitter ↑**

- Picchi meno evidenti
  - ➢ Tempi di interarrivo più uniformi

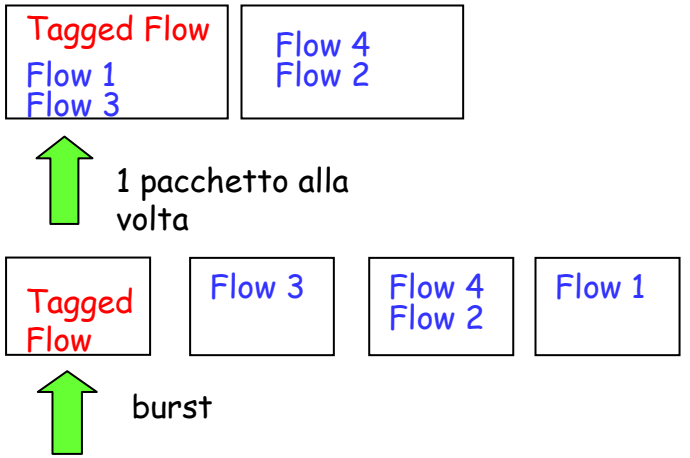
Secondo Scenario: Jitter



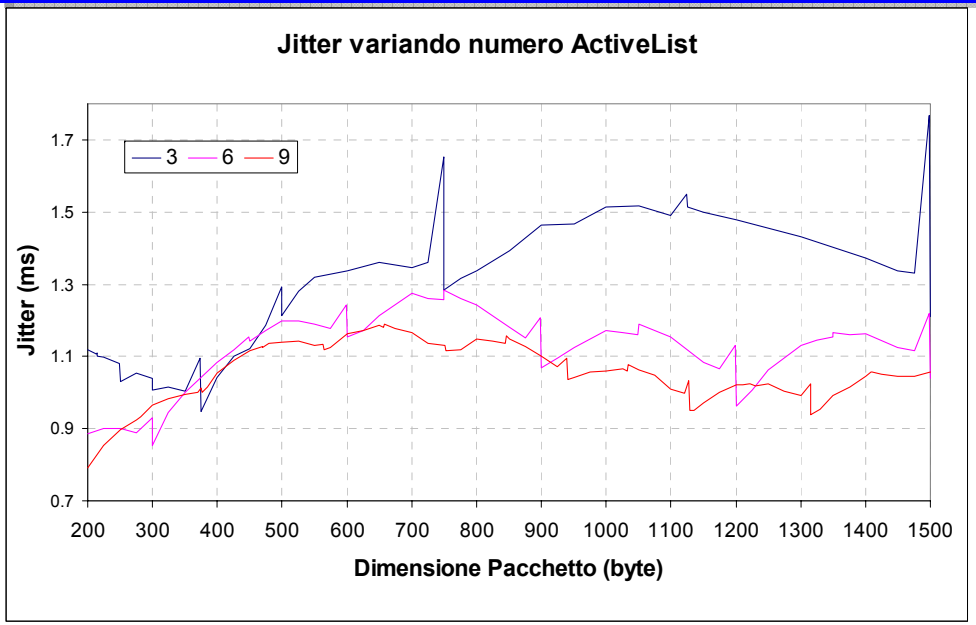
✓ L < 750 ⟶ trasmissione di burst



✓ code piene ⟶ burst maggiori

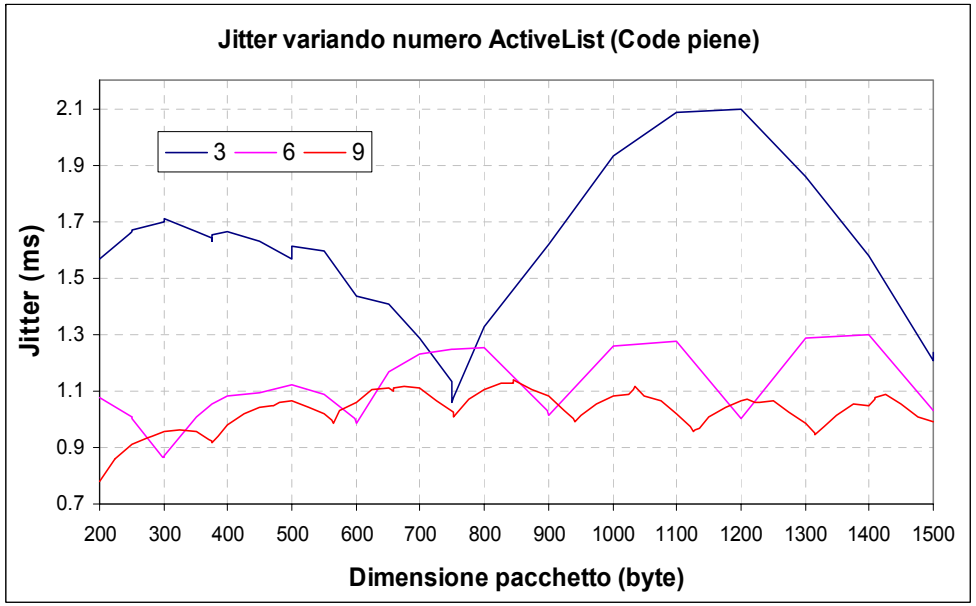


Secondo Scenario: Jitter



✓↑ ActiveList ↓ Jitter

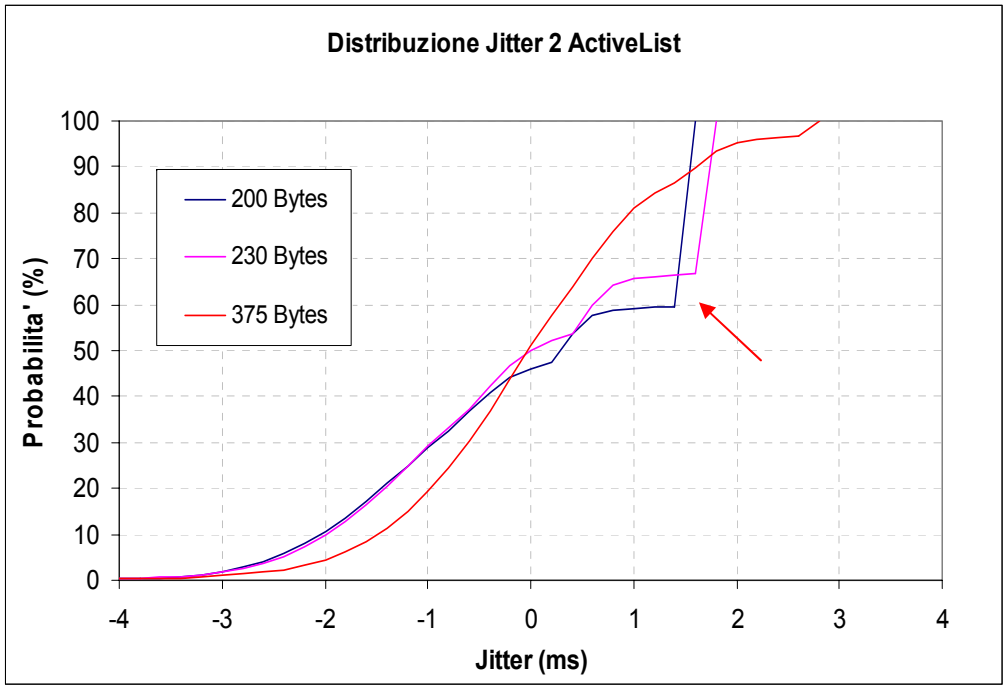
- Lo scheduler è più fair per un numero maggiore di ActiveList

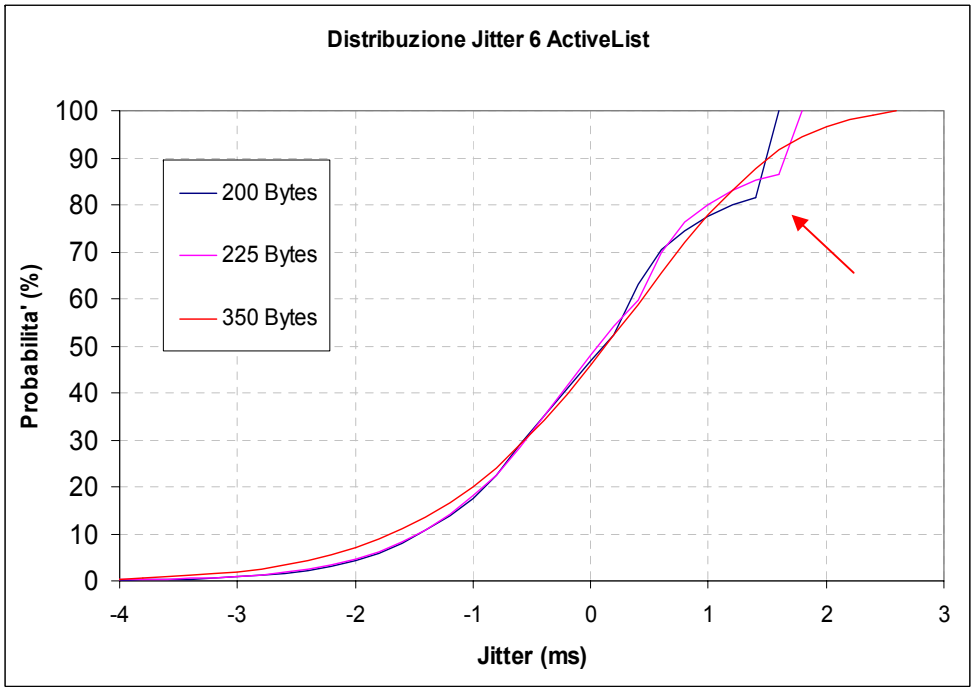


✓↑ ActiveList e Quanto/L intero

- **Jitter minimo**

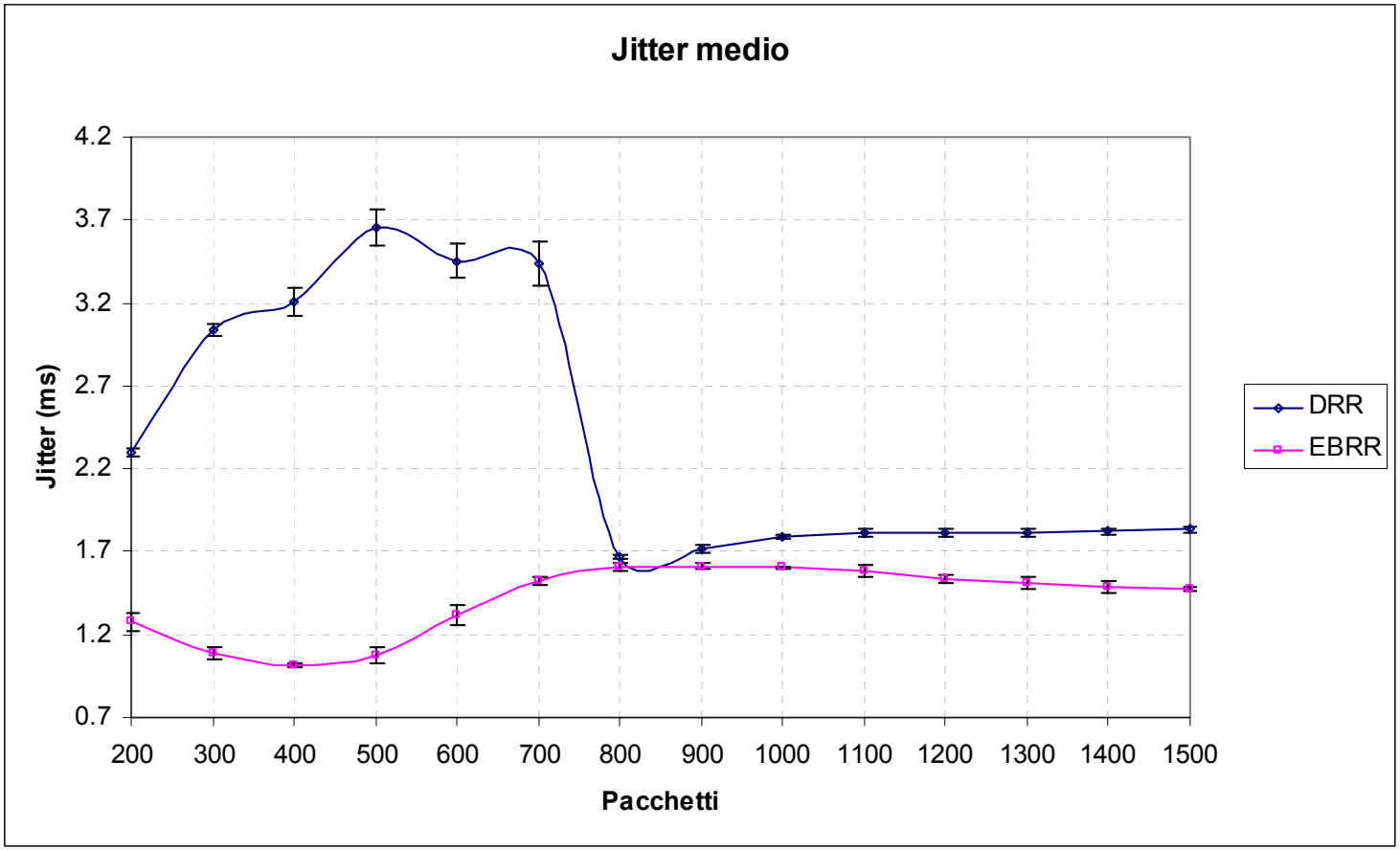
Secondo Scenario: Jitter





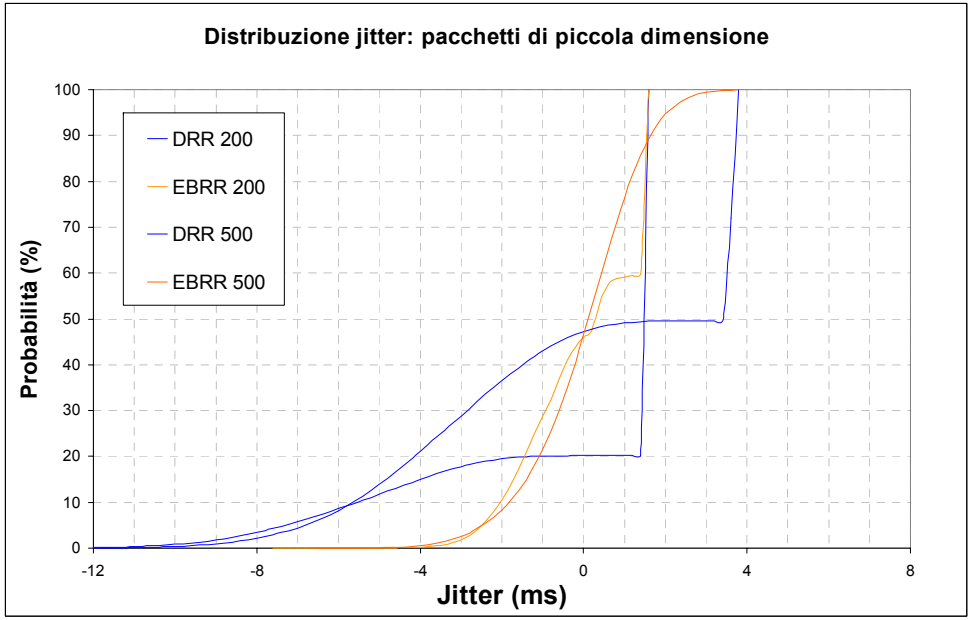
✓All'aumentare del numero di ActiveList la conseguente diminuzione del quanto, porta ad una minore probabilità di formazione di burst

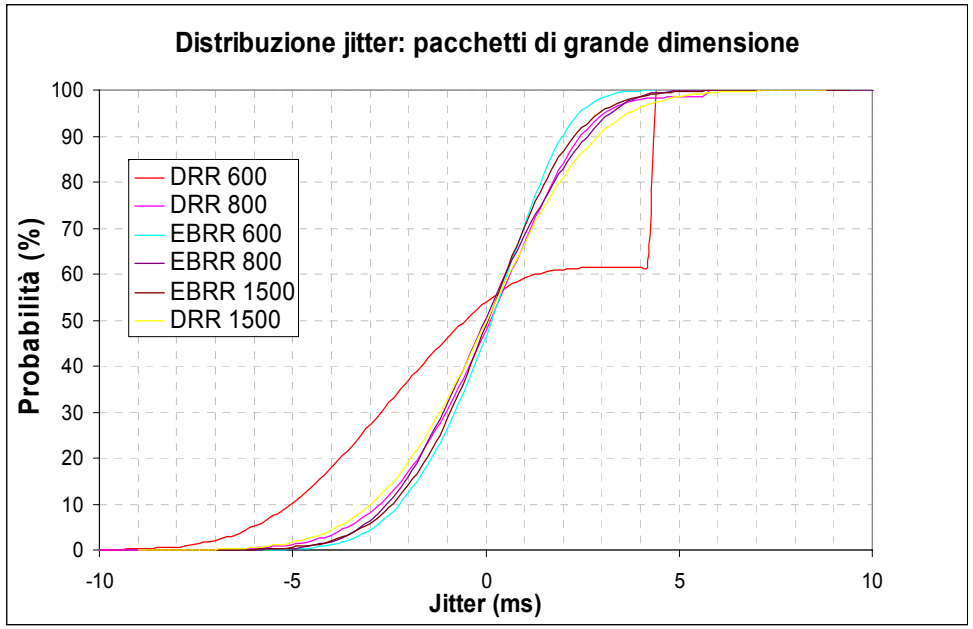
Secondo scenario: Jitter



- ✓ ↑ Dimensione pacchetti ↑ Jitter
  - I burst introducono una alta variabilità
- ✓ L > quanto/2 ⟹ variabilità piccola
  - In un round trasmetto al più 2 pacchetti

Secondo scenario: jitter

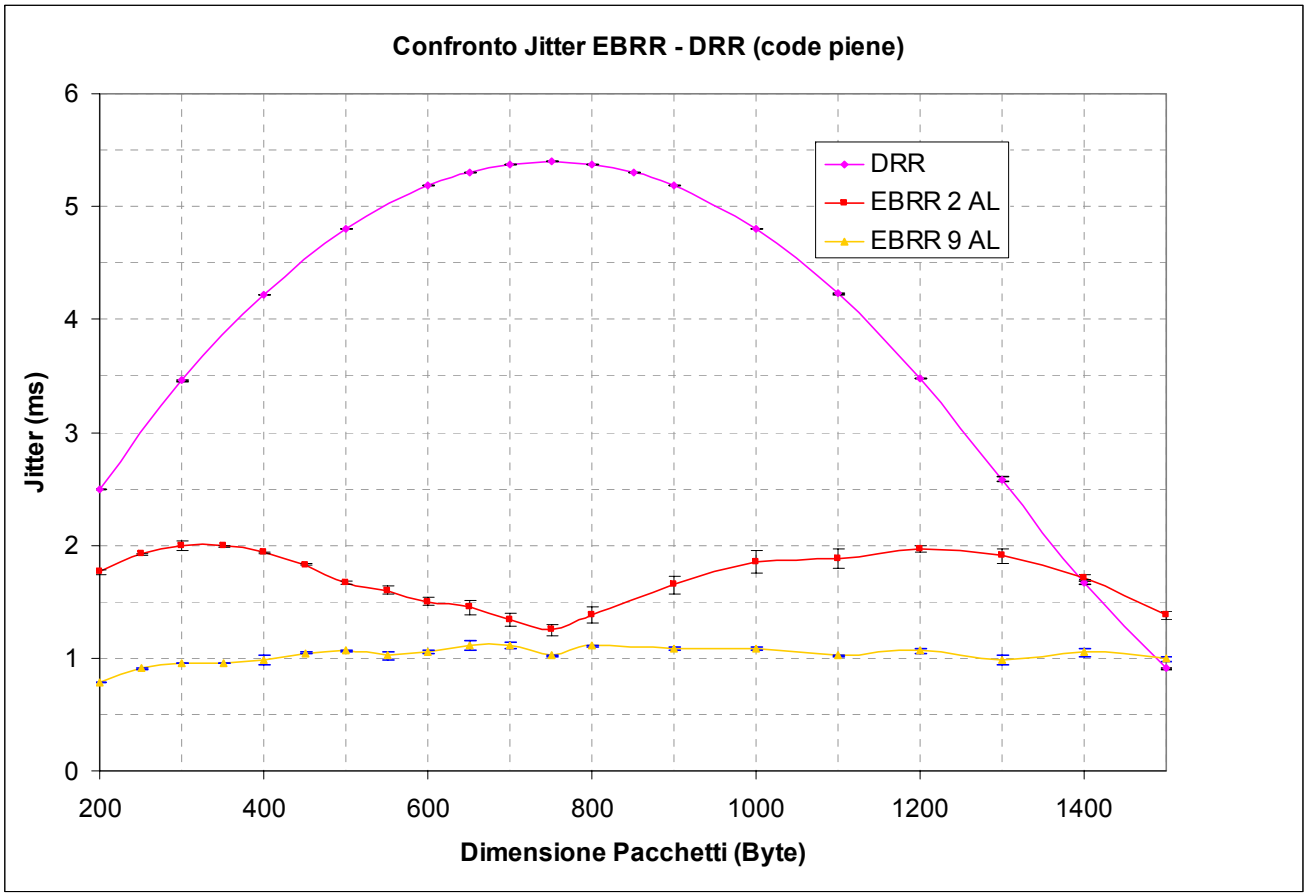




- ✓ DRR ➡ maggior pacchetti in burst
- ✓ ↑Dimensione Pacchetti ↑variabilità jitter
  - ➢ Minori pacchetti in burst, ma separati da un tempo maggiore

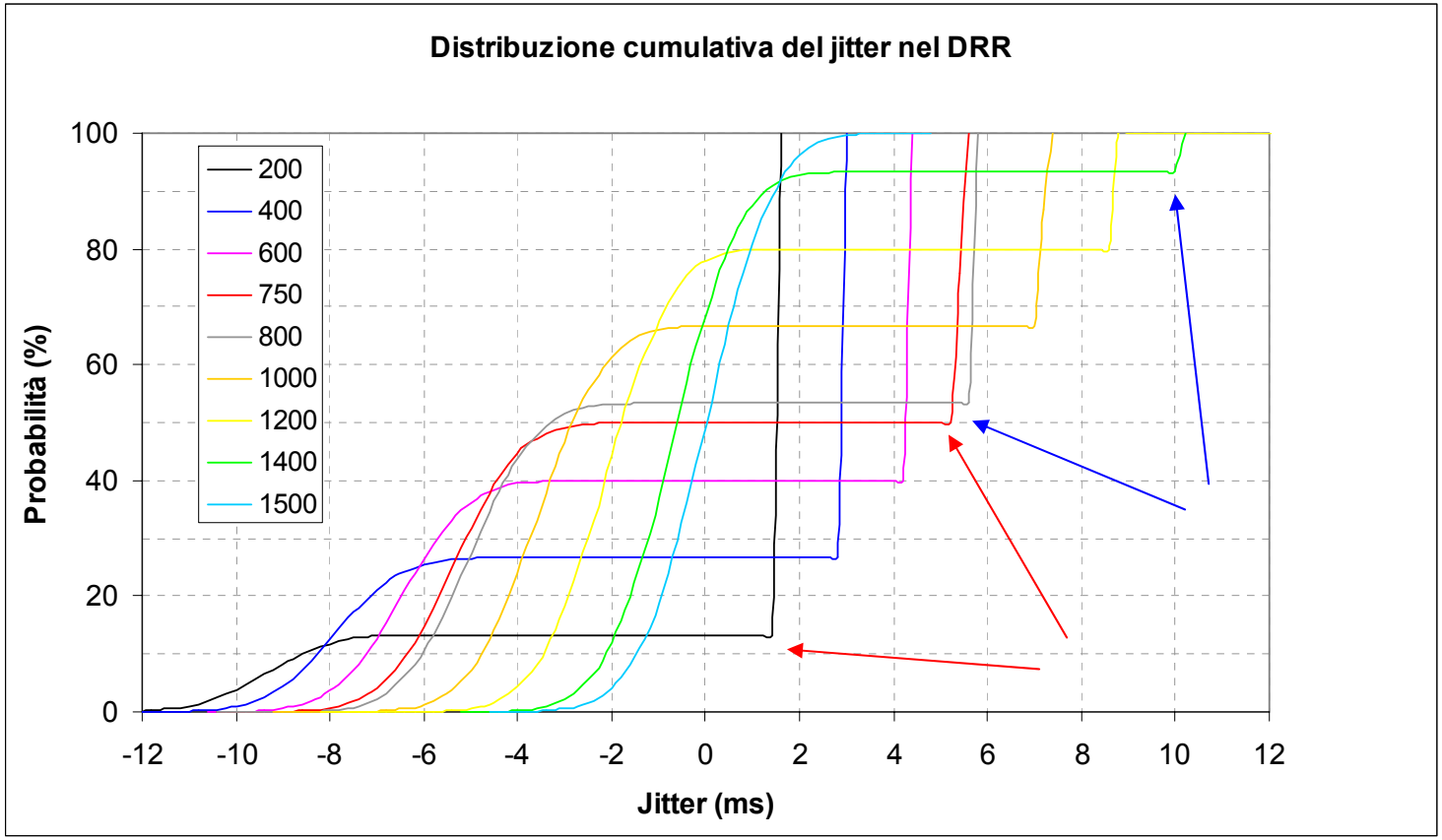
✓ Con pacchetti di grandi dimensioni il jitter accumulato dai due scheduler è simile

Secondo scenario: Jitter



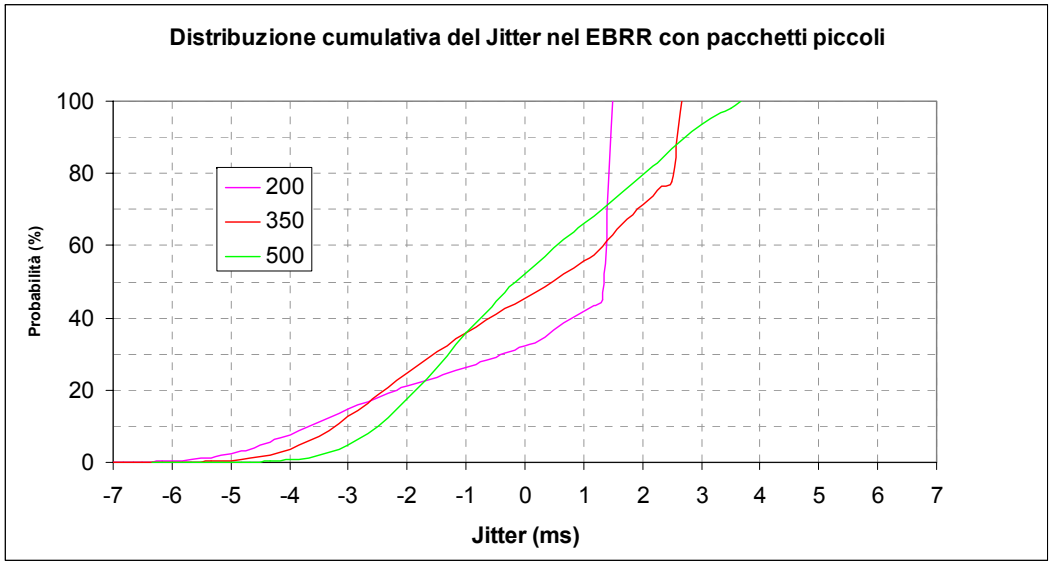
- ✓ Se L = quanto/2
  - ➢ Massimo Jitter DRR
  - ➢ Minimo Jitter EBRR.

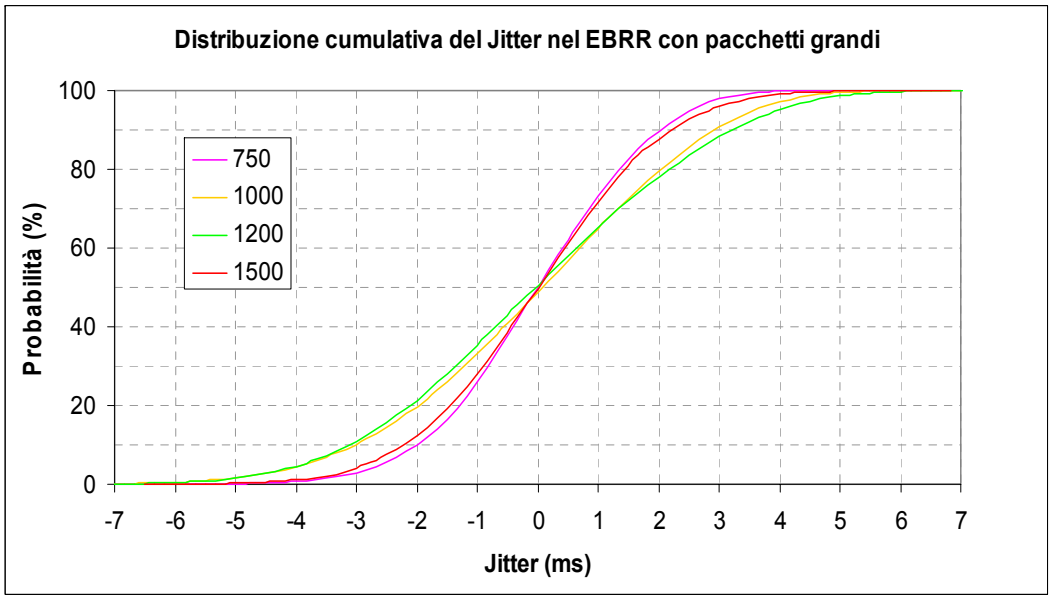
Secondo scenario: Jitter



- ✓ L < 750
  - ➢ ↓Pacchetti interni al burst ↑Jitter
- ✓ L > 750
  - ➢ ↑Pacchetti interni al burst ↓Jitter

Secondo scenario: Jitter





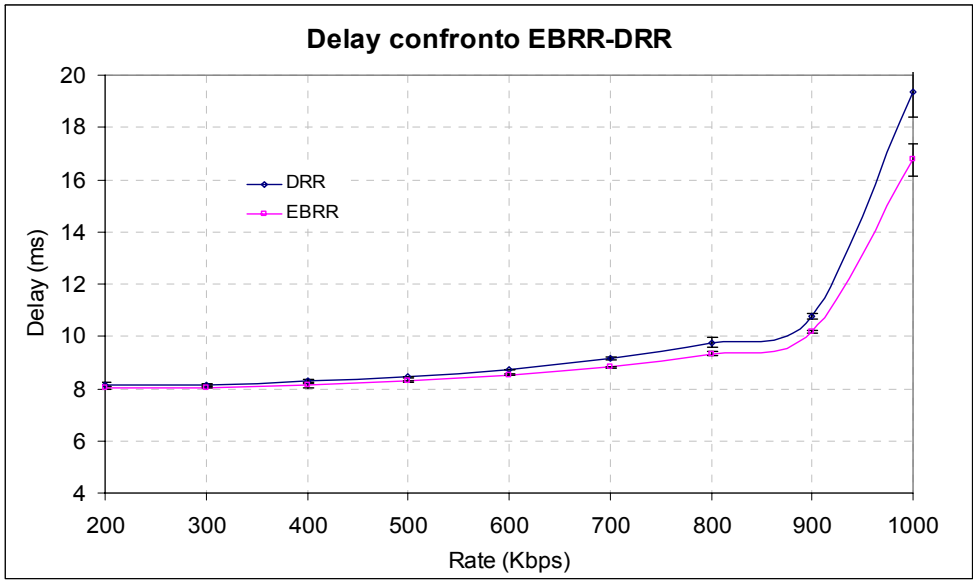
- ✓ L'EBRR non permette l'invio di burst veri e propri tuttavia con code sempre piene e pacchetti piccoli può capitare che se ne possa trasmettere più di uno in una stessa active list, quando cioè gli altri flussi sono schedulati per essere serviti più avanti
- ✓ I grafici esaltano proprio tale comportamento mostrando come con valori maggiori di 750 bytes gli andamenti siano continui non presentando "burst".
- ✓ In 750 si verifica il minimo perché si avrà un cambio periodico di active list ogni 2 pacchetti regolarizzando le differenze dei tempi di interarrivo.

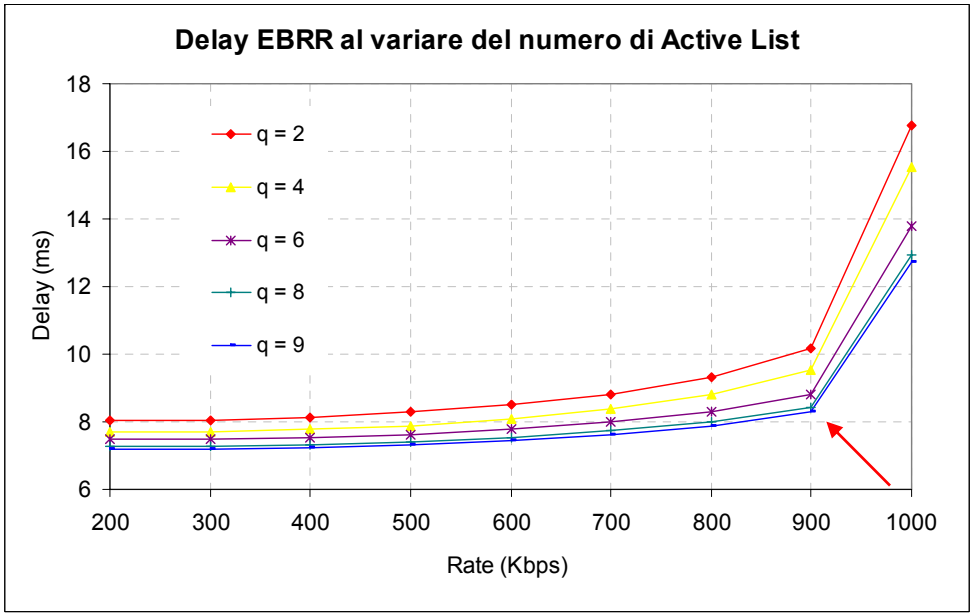
Terzo Scenario

Terzo scenario: prestazioni al variare del rate di invio

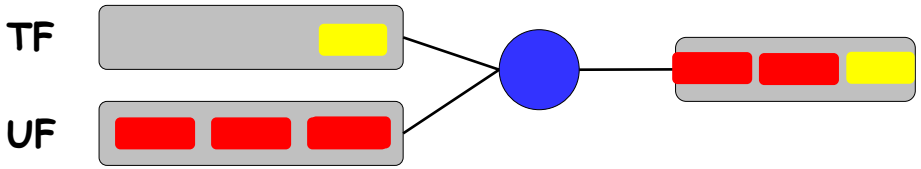
- ✓Nodi: 5
- ✓Active List: 2 ÷ 9
- ✓Rate flusso tagged: [200, 1000] Kbit/s
- ✓Rate flussi untagged: 1Mb/s (code scariche), 1.5Mb/s (code piene)
- ✓Dimensione pacchetti tagged: 1000 Bytes
- ✓Dimensione pacchetti untagged: 100÷1500 Bytes
- ✓Quanti uguali e minimi per assicurare rate di uscita di 1Mb/s tagged e untagged

Terzo scenario: Delay

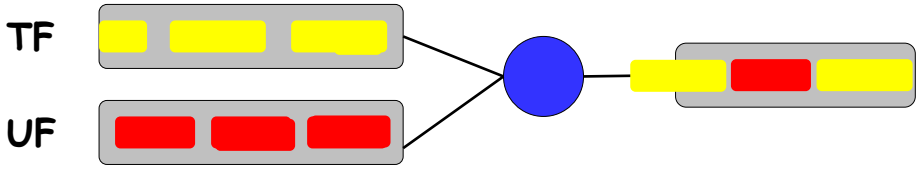




- ✓ Comportamento EBRR simile al DRR
  - ✓Rate molto basso

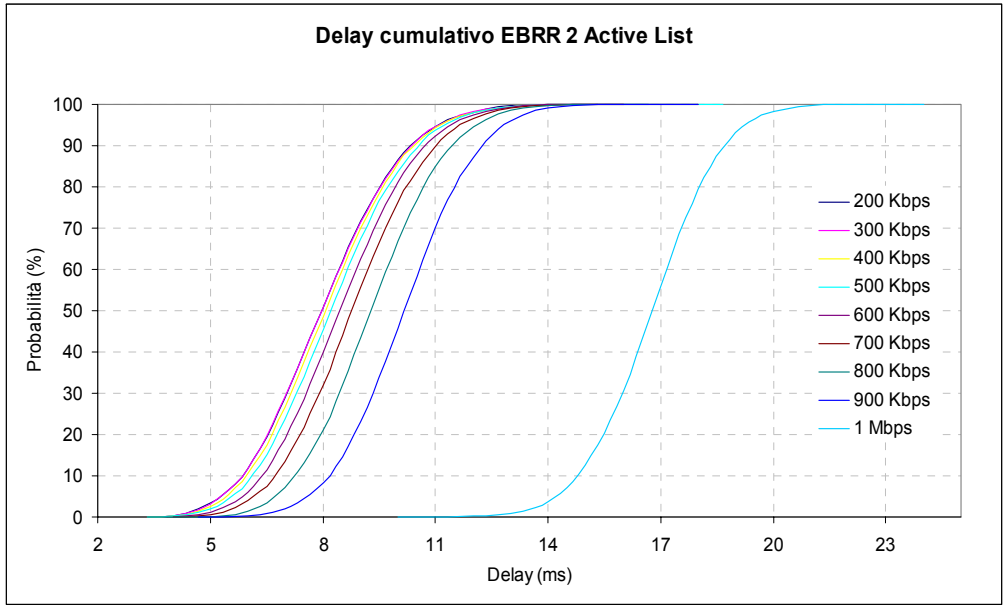


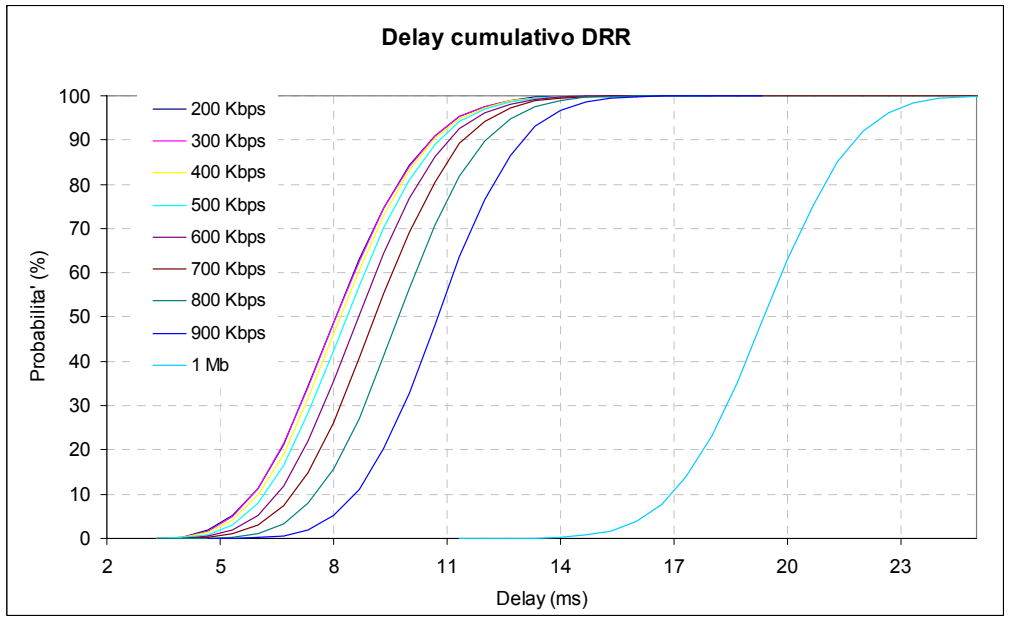
  - ✓Aumentare del Rate



- ✓ ↑ numero di ActiveList ↓ delay

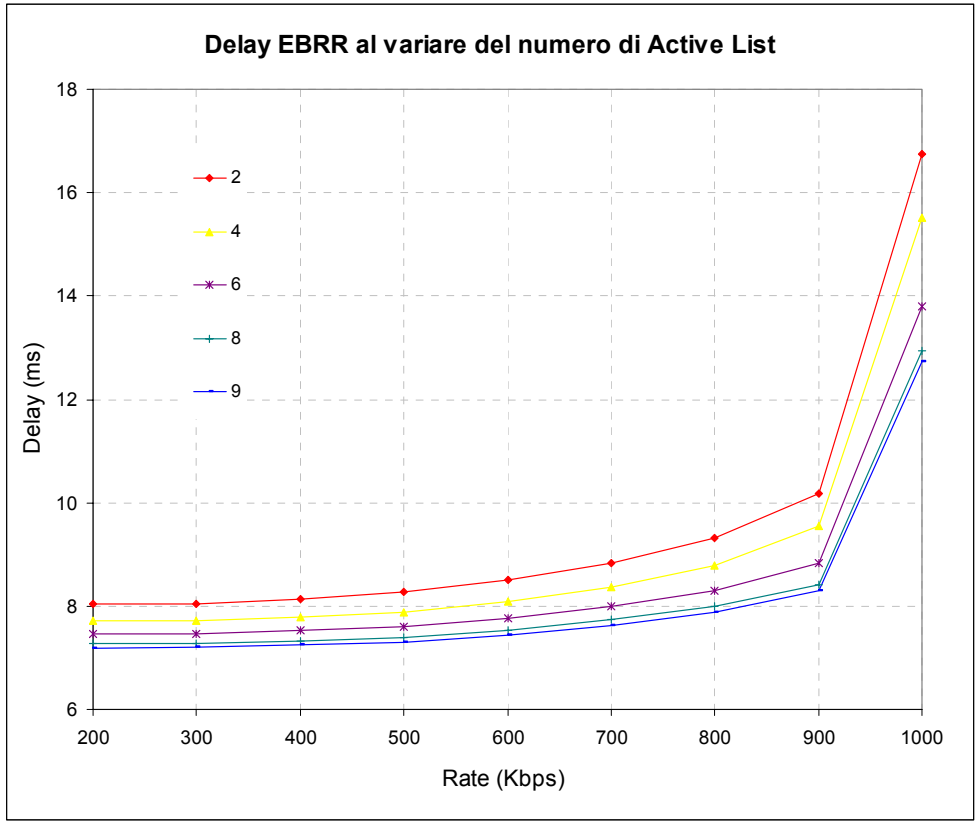
Terzo Scenario: Delay

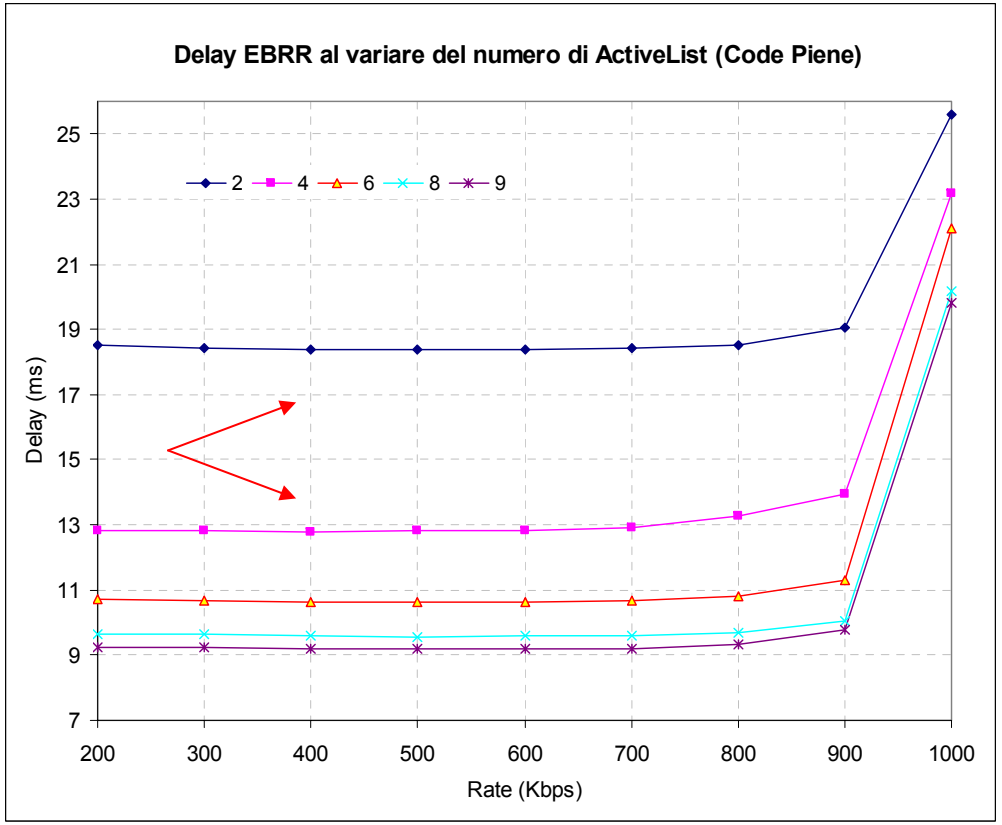




- ✓ Ritardo dei pacchetti dipendente da:
  - numero flussi untagged
  - riempimento delle code dei flussi untagged
- ✓ Andamento simile EBRR - DRR

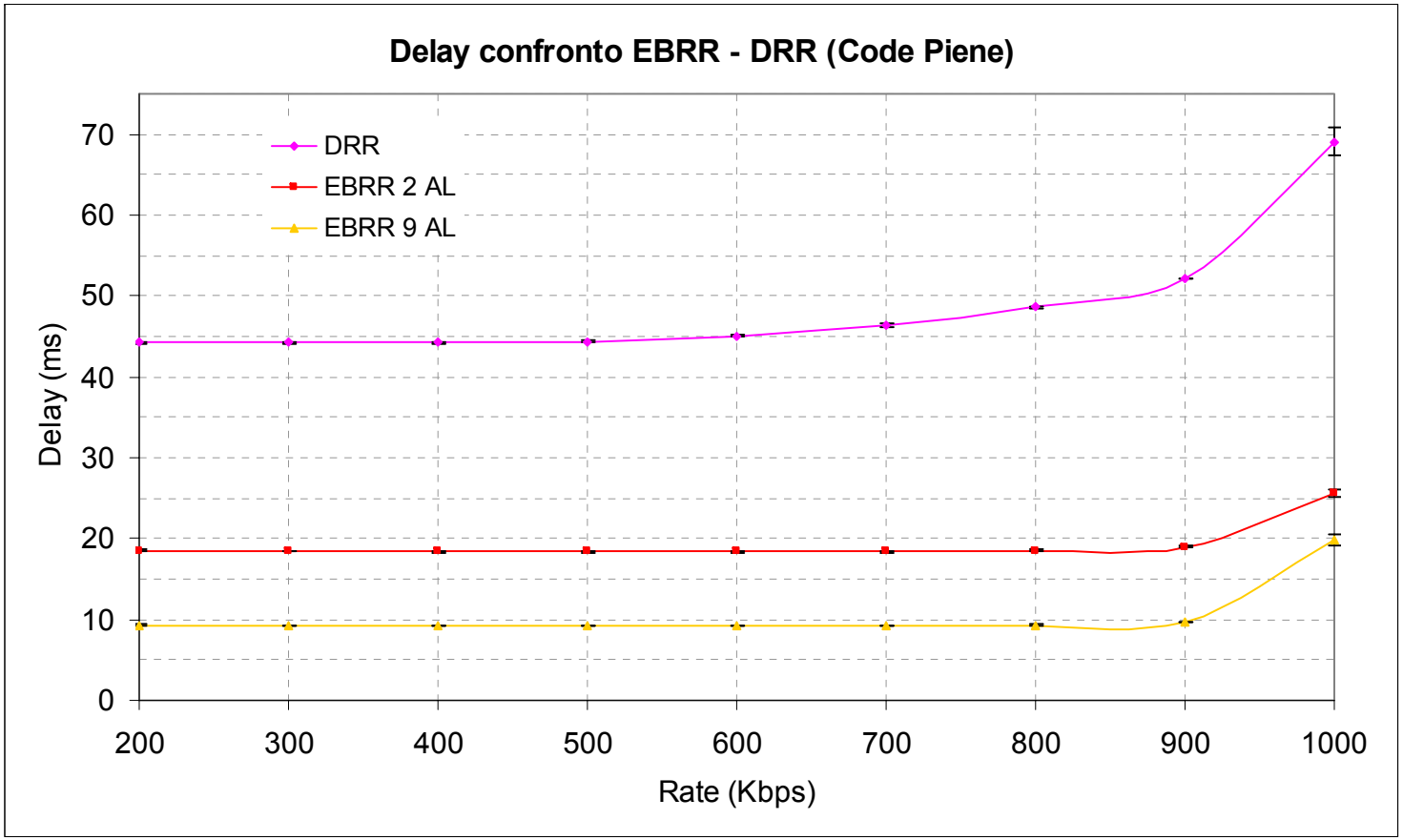
Terzo Scenario: Delay





- ✓ Code Piene
  - ↑ numero di ActiveList ↓ delay
  - maggiore spaziatura fra le curve
  - distribuzione su piu' ActiveList → guadagno maggiore

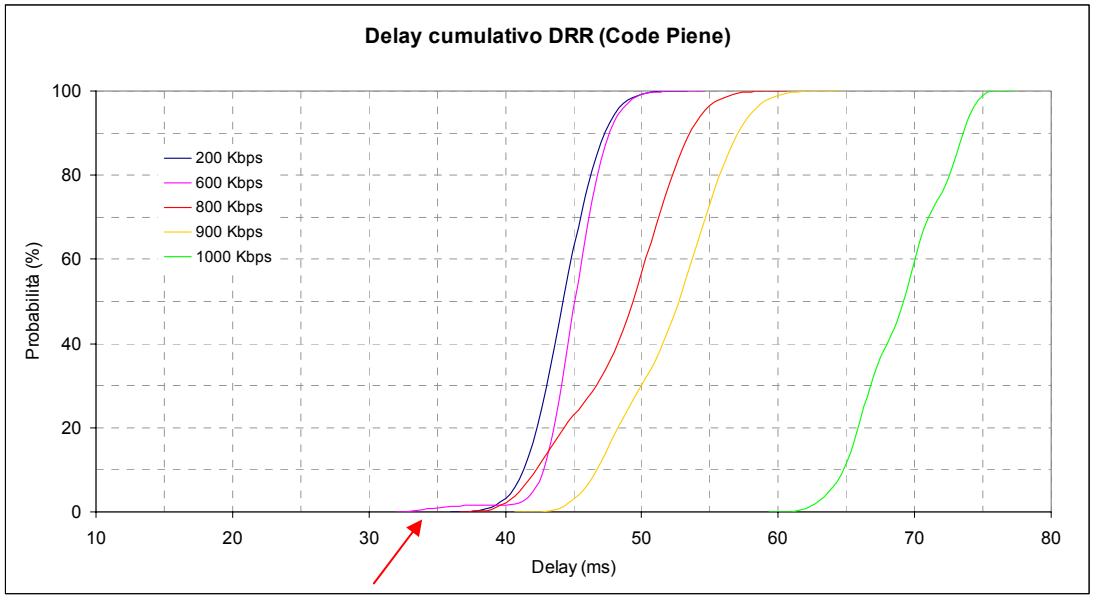
Terzo scenario: Delay

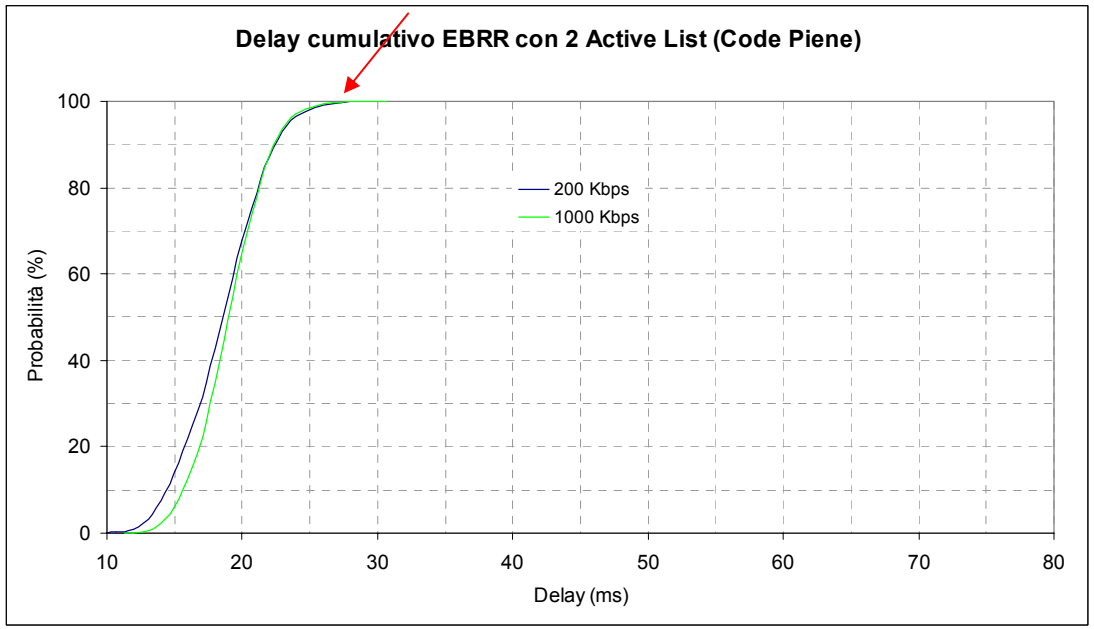


✓ Andamento simile al caso con code vuote:

- delay maggiore in entrambi gli scheduler → maggior carico sul canale
- nel DRR i flussi trasmettono l'intero quanto → delay più alto

Terzo scenario: Delay





✓ DRR:

- 200-600 curve vicine ➡ andamento costante
- ↑ rate , curve piu' distanziate

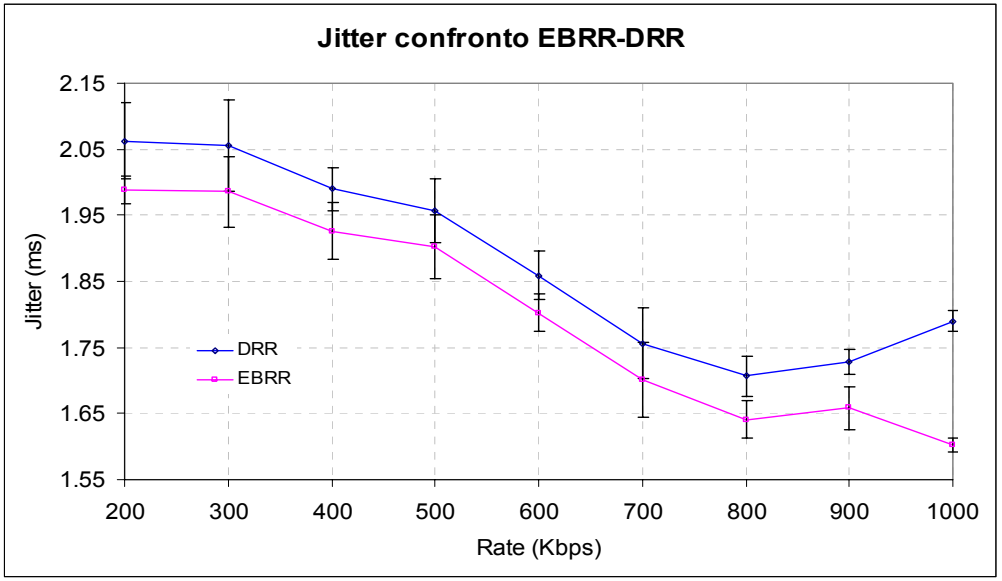
✓ EBRR:

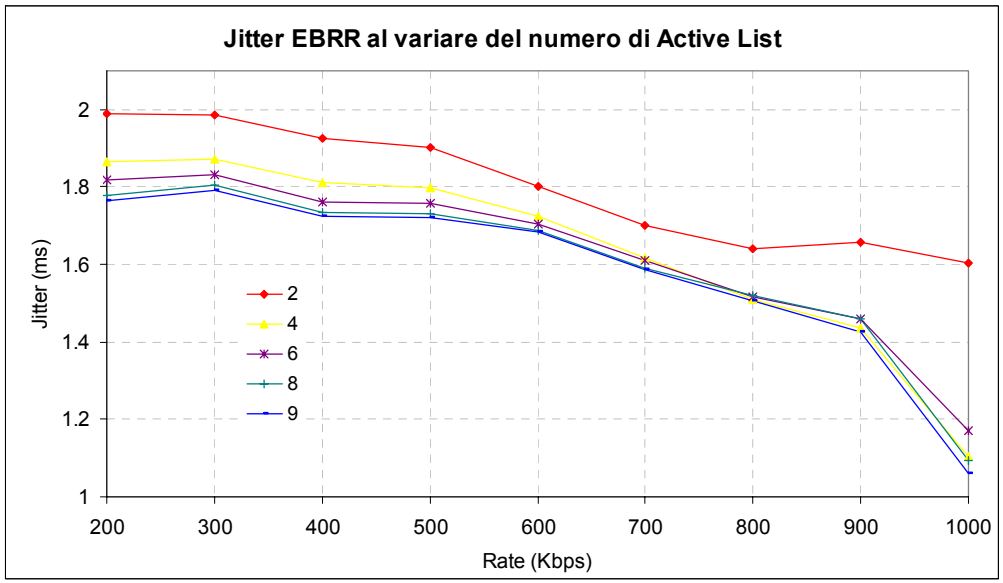
- curve meno distanziate

⬇

comportamento costante per intervallo di rate molto maggiore

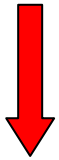
Terzo scenario: Jitter





✓ EBRR-DRR: Andamento ↓

- jitter diminuisce quando il TF è nello stato backlogged con più frequenza

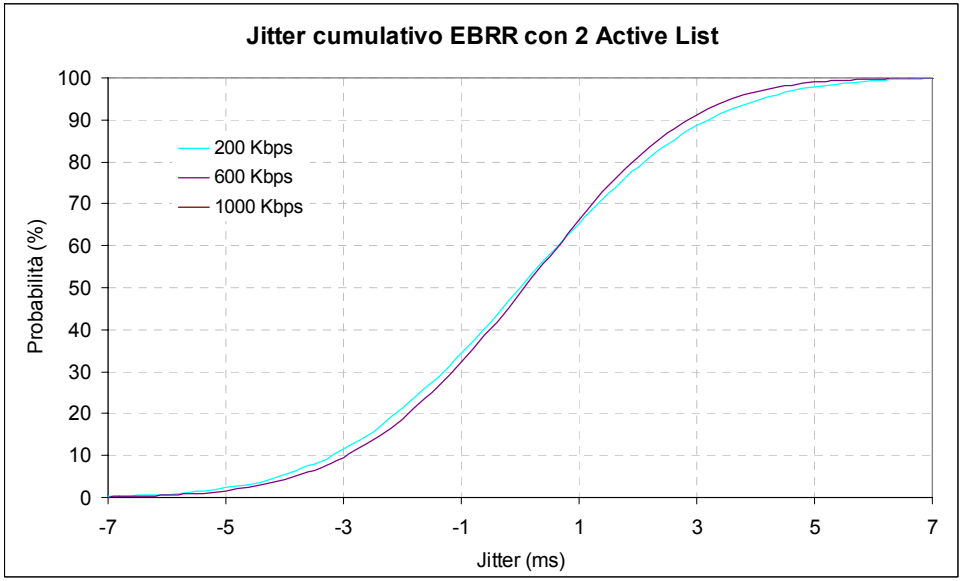


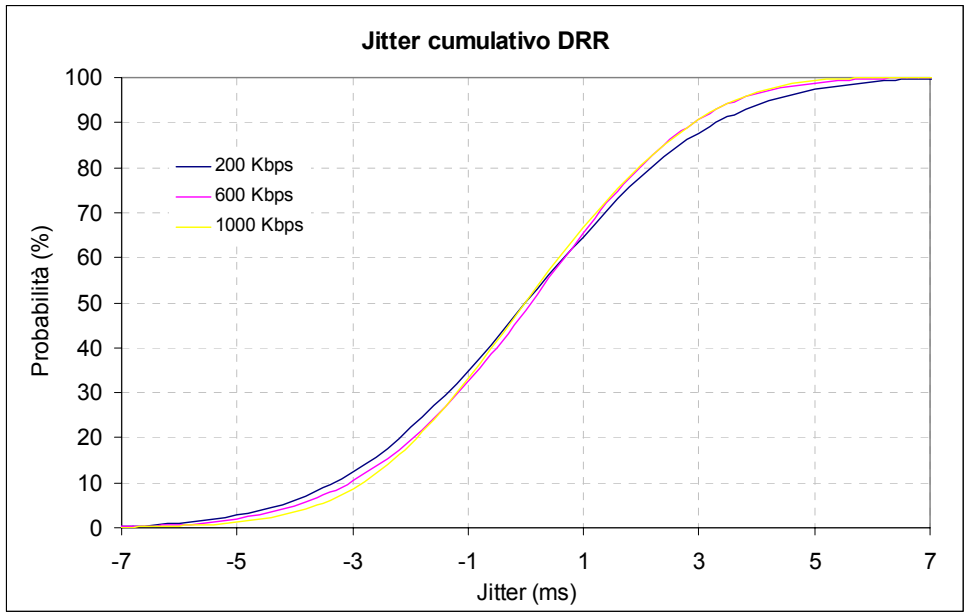
- ↑ rate ➡ TF è sempre meno idle

✓ ↓ jitter ↑ numero di ActiveList

- è più probabile che al passaggio allo stato backlogged, il TF trovi meno flussi davanti

Terzo scenario: Jitter

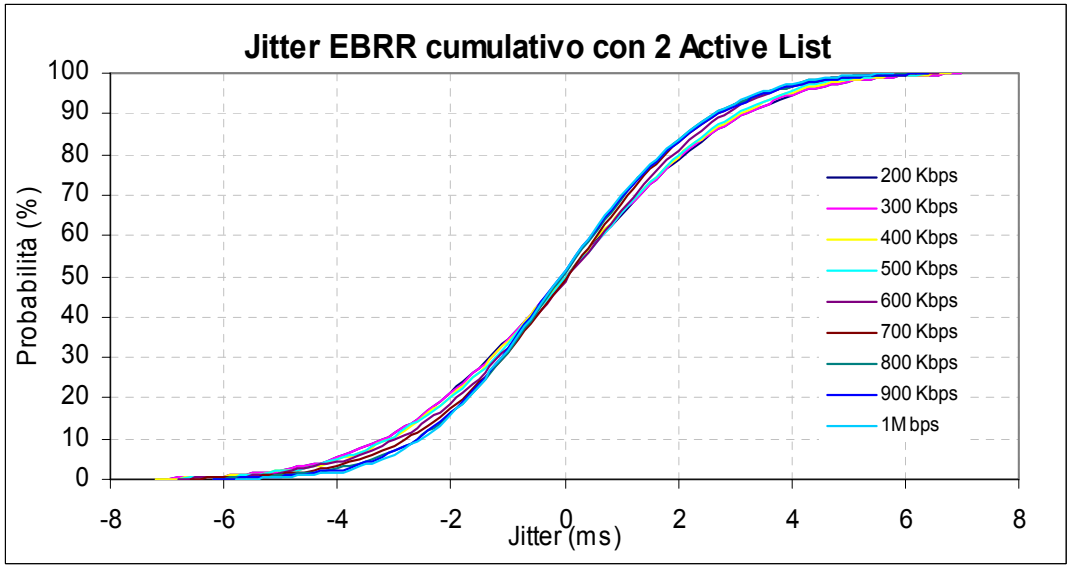


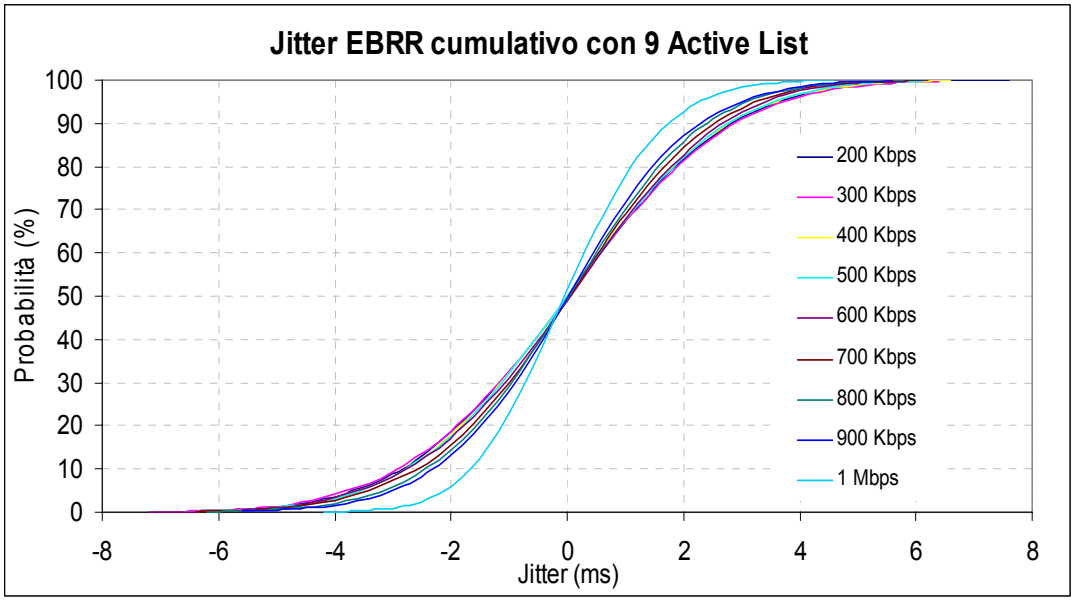


- ✓ EBRR-DRR:
  - Rate di invio garantito dai quanti >= rate di arrivo dei pacchetti in coda

  - non ci sono burst

Terzo Scenario: Jitter





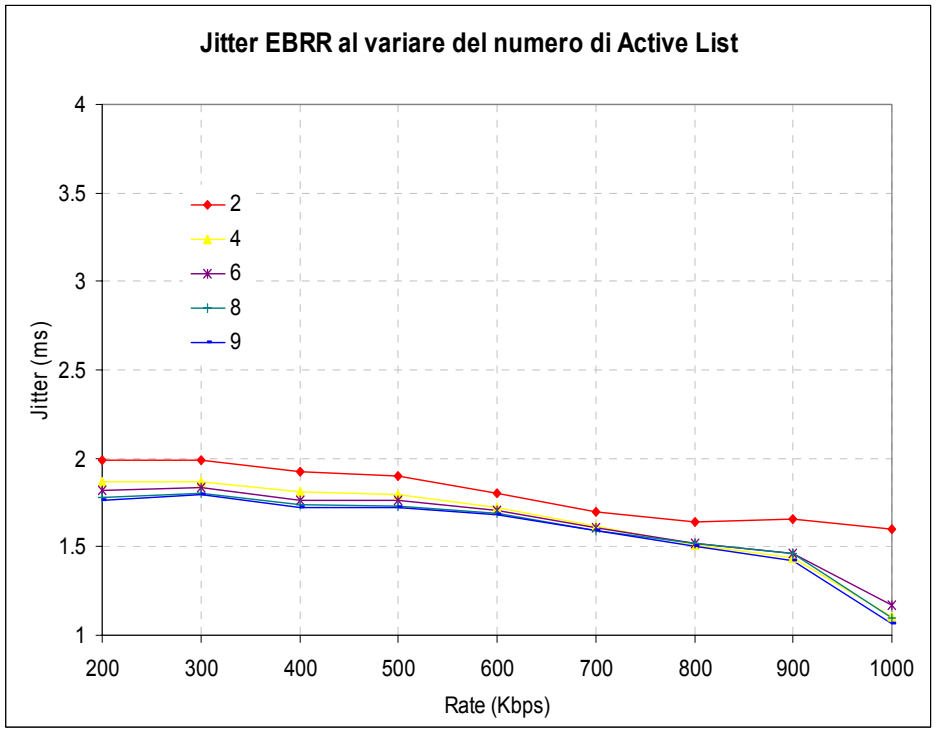
✓ Distribuzioni simmetriche intorno allo zero

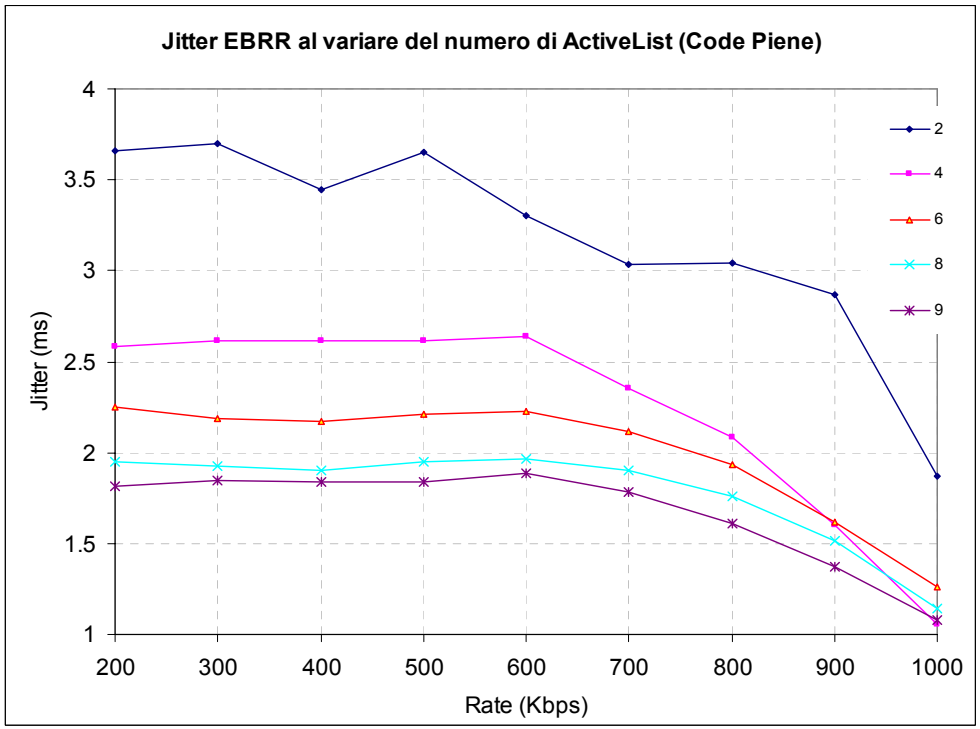
✓ Per rate di invio tale da saturare la banda a disposizione:

↑ numero Active List ↓ variazione Jitter

Aumentare il numero di Active List migliora prestazioni scheduler

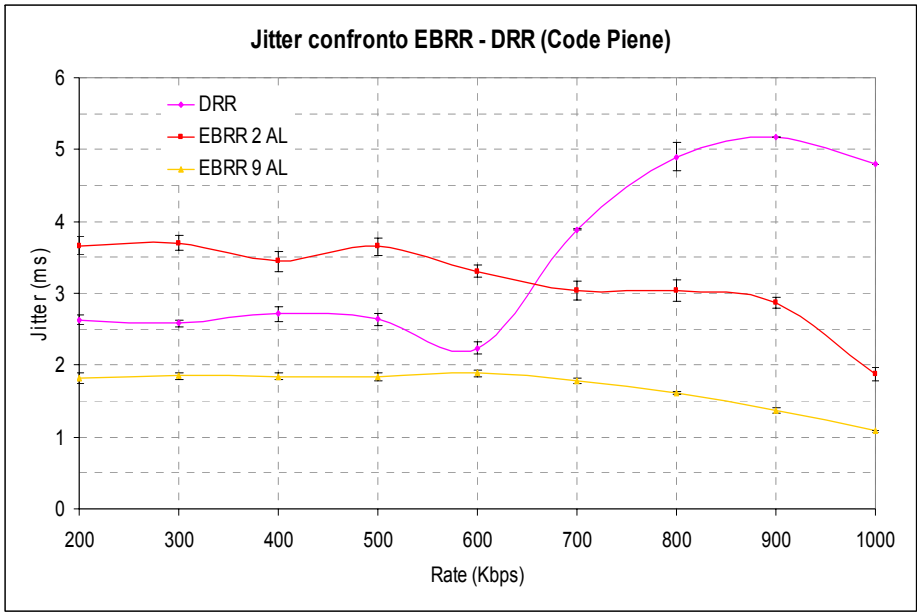
Terzo Scenario: Jitter

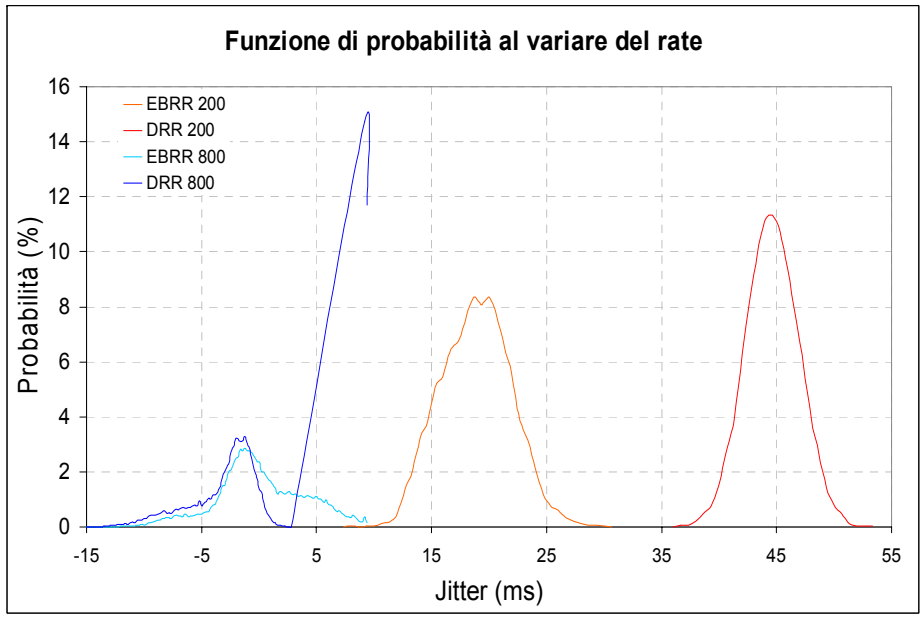




- ✓ Code piene: ↑ numero di Active List ↓ jitter
- ✓ Jitter code piene >> Jitter code vuote
  - code piene dei flussi untagged ➡ il TF avrà davanti in media un numero maggiore di flussi e per questo molto piu' variabile
  - code vuote dei flussi untagged ➡ il TF avrà davanti in media un numero minore di flussi e per questo molto meno variabile

Terzo scenario: Jitter





- ✓ DRR:
  - Jitter quasi costante per Rate < 600Kbps
  - Per Rate > 600 Kbps ↑ Jitter

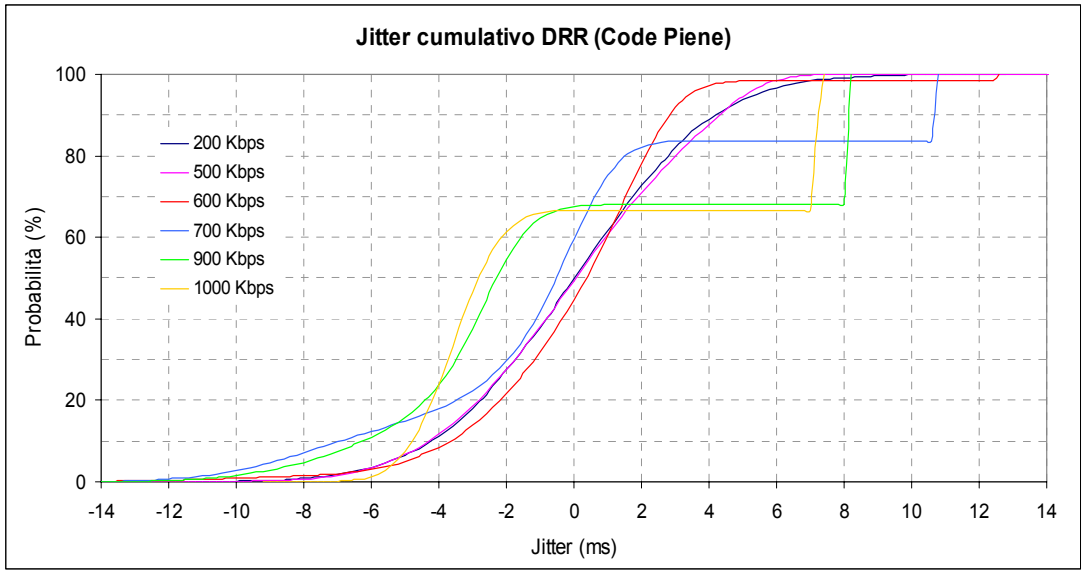
  ↓

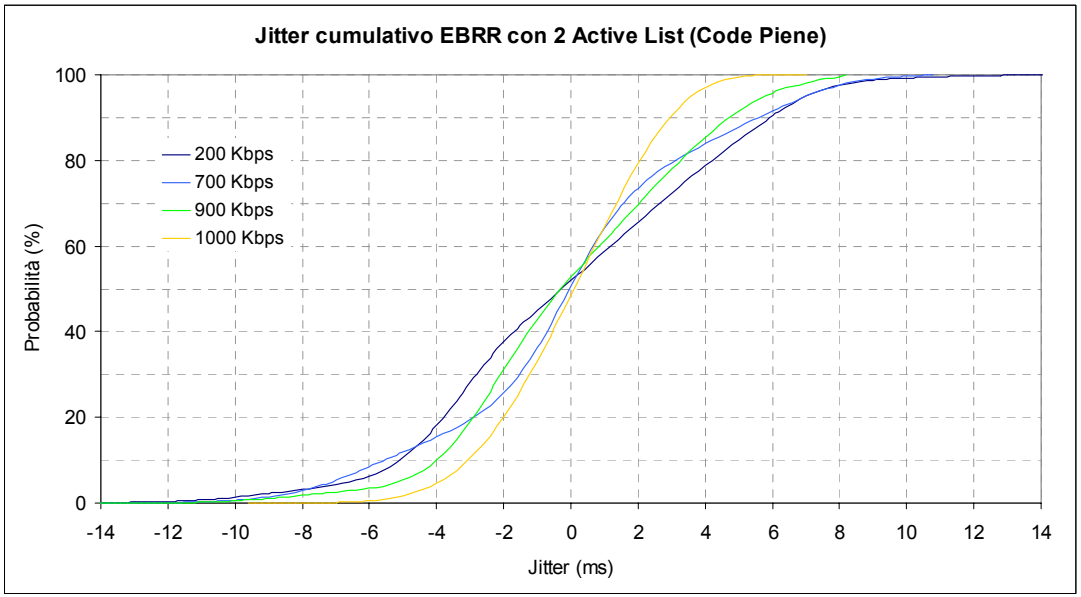
  Fenomeno legato alla generazione di burst
- ✓ EBRR:
  - Andamento decrescente

  ↓

  Trasmissione di un solo pacchetto alla volta
- ✓ Fino a 650 Kbps circa migliori prestazioni DRR
  - Assenza di Burst per il TF
  - Il TF in EBRR aspetta sempre tutti i flussi UF, che trasmettendo un solo pacchetto alla volta di dimensione variabile introducono una maggiore variabilità del ritardo

Terzo scenario: Jitter





✓ DRR:

- Rate>=600 Kbps aumenta la probabilità di spedire burst

Aumento del Jitter

✓ EBRR:

- No discontinuità
- Lieve variazione di pendenza all'avvicinarsi di 1 Mb

Algoritmo EBRR

Conclusioni

Conclusioni

- ✓ Nel DRR, per avere una complessità computazionale di O(1) per la selezione del successivo flusso da servire in un round, deve verificarsi che

$$\phi_i \geq L_{i_{\max}}$$

- ✓ Nell'EBRR, si ottiene comunque una complessità O(1). La dimensione del quanto può essere perciò regolata a piacimento, permettendo di variare la granularità con cui vengono serviti i pacchetti, fermo restando i vincoli imposti dal rate e dal numero di ActiveList

- ✓ Una maggiore granularità permette di avvicinarsi al comportamento di un sistema fluido, con vantaggi in termini di fairness. Dimensionare opportunamente il quanto in base alla dimensione dei pacchetti, porta inoltre una maggiore efficienza nell'utilizzo del quanto a disposizione in un round, e quindi un miglioramento sia in termini di delay che di jitter

- ✓ Un maggior numero di ActiveList permette ai flussi di distribuirsi su più liste. Ciò aumenta la probabilità che, nel round in cui è eleggibile, un flusso sia l'unico in lista e possa perciò essere servito immediatamente

Conclusioni

- ✓ Di contro, un maggior numero di ActiveList porta un carico computazionale maggiore; un quanto piccolo inoltre penalizza i flussi con pacchetti mediamente più grandi
- ✓ Il DRR permette l'invio di più pacchetti in burst nel round corrente, se il DeficitCounter del flusso lo permette
- ✓ Nell'EBRR, dopo aver trasmesso un pacchetto, un flusso viene rimesso in coda in fondo alla lista dei flussi eleggibili per la trasmissione nel round corrente: ciò comporta, a parità di flussi in lista, una minor probabilità di formare burst di pacchetti e un ulteriore vantaggio in termini di fairness

Conclusioni

Grazie per l'attenzione