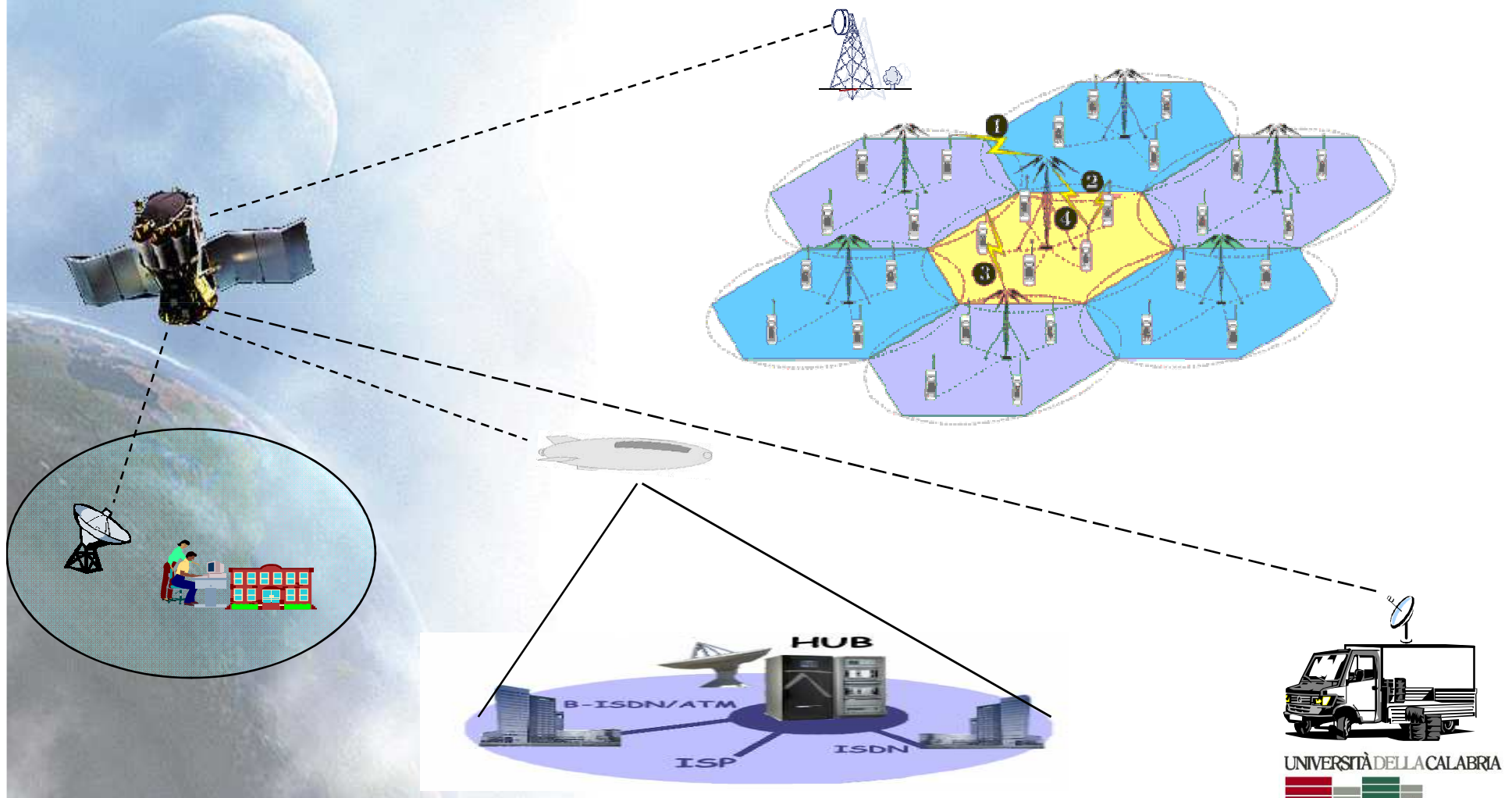
Architettura Scalable CORE (SCORE)

Sommario

- Modello Best-effort
- Nuovi Servizi
- Tecniche e meccanismi per i nuovi servizi
  - Integrated Services
  - Differentiated Services
- Architettura SCORE
  - Dynamic Packet State
  - Algoritmo di Scheduling
  - Algoritmo di Ammissione
  - Prove simulative
  - Osservazioni

Eterogeneità della rete internet



Modello Best-effort

- Oggi Internet fornisce un semplice modello di consegna dei pacchetti
- Tale modello di servizio permette ai routers di essere “stateless”, cioè, eccetto per poche informazioni, di non mantenere lo stato per il traffico
- Ciò rende Internet Scalabile e Robusta
  - Scalabile - perché la complessità dei routers non incrementa con il numero di flussi
  - Robusta – perché ci sono pochi stati da aggiornare per i routers

Nuovi Servizi

- Lo sviluppo di Internet e delle applicazioni multimediali hanno portato ad aver bisogno di servizi più performanti del best-effort:

  - **Servizi Garantiti**: dare la possibilità di garantire parametri prestazionali come banda e ritardo per ogni flusso nella rete

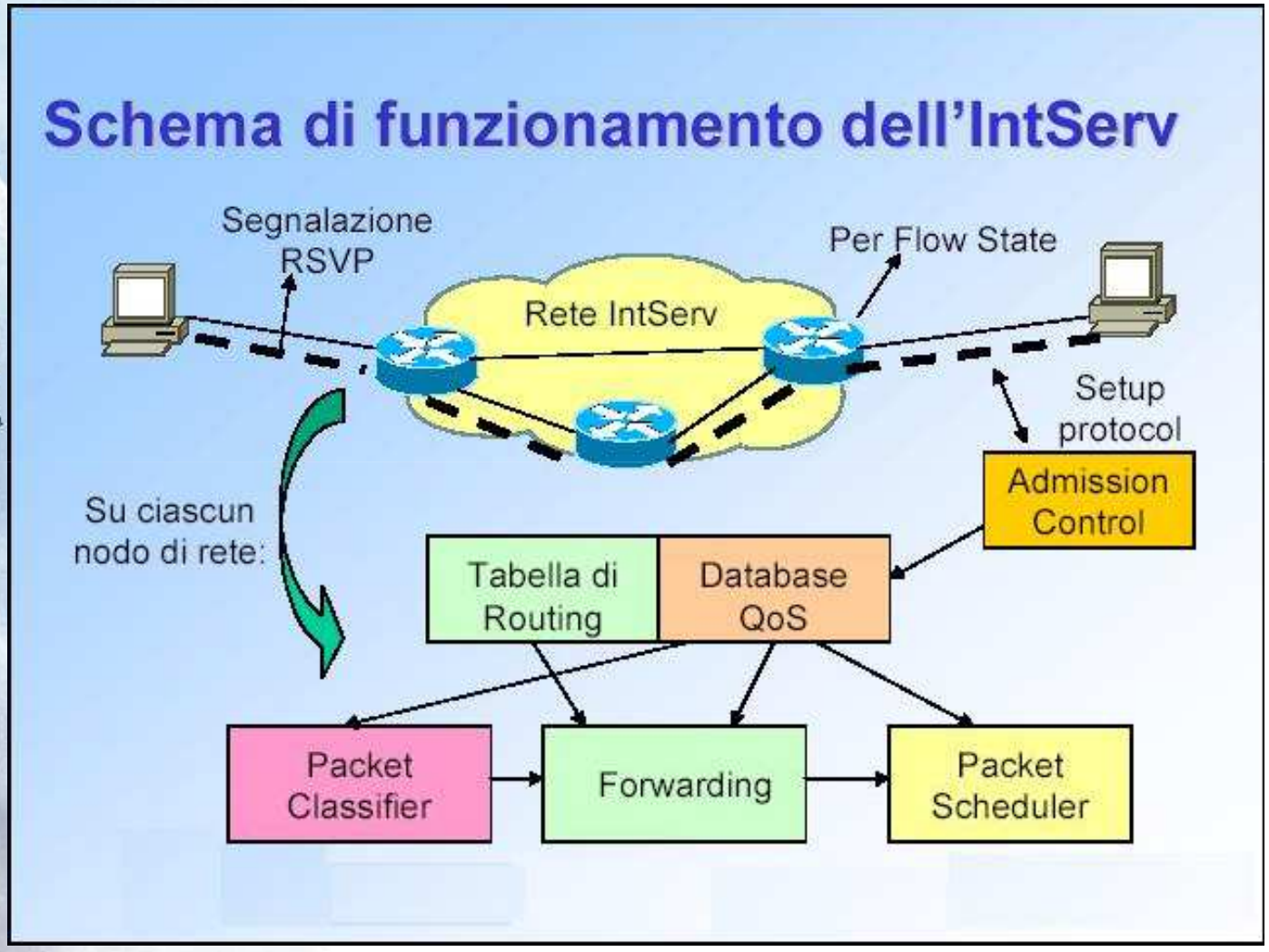
  - **Servizi Differenziati**: che permettono di fornire differenziazione di servizi in base a parametri (banda, rate di perdita,etc.) per aggregati di traffico

Architetture di QoS

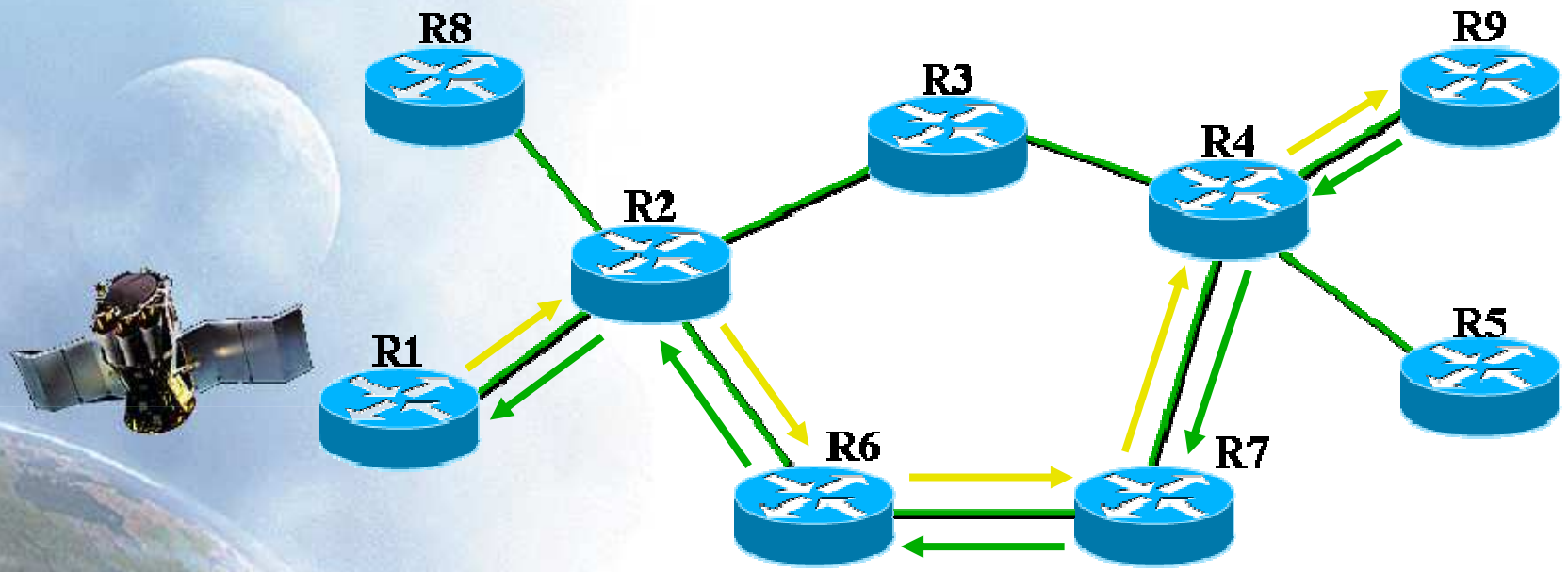
- Sono nate architetture per la Qualità del Servizio (QoS)

- Integrated Services (IntServ) (Statefull)
  - Guaranteed Services (GS)
  - Controlled Load Services (CLS)
- Differentiated Services (DiffServ) (Stateless)
  - Expedited Forwarding (EF)
  - Assured Forwarding (AF)

IntServ


Schema di funzionamento dell'IntServ
Segnalazione RSVP
Per Flow State
Rete IntServ
Setup protocol
Admission Control
Su ciascun nodo di rete:
Tabella di Routing
Database QoS
Packet Classifier
Forwarding
Packet Scheduler

Protocollo RSVP



Setup: Path (R1->R2->R6->R7->R4->R9)

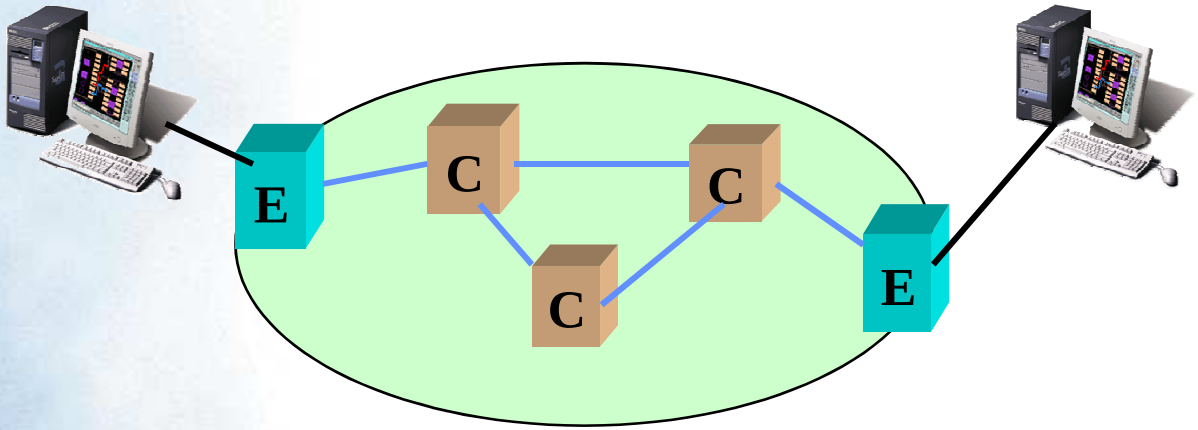
Reply: Resv (R9->R4->R7->R6->R2->R1)

Classi di traffico IntServ

- Guaranteed Services (GS)
  - Quantitativa
  - Bound sul ritardo di accodamento $DB = \frac{b}{B} + \frac{C}{B} + D$
    - C termine dipendente dalla rate
    - D termine indipendente dalla rate

- Controlled Load Services (CLS)
  - Qualitativa
  - Servizio “Better than Best-Effort”
- Best Effort
  - Nessuna garanzia

DiffServ (1)



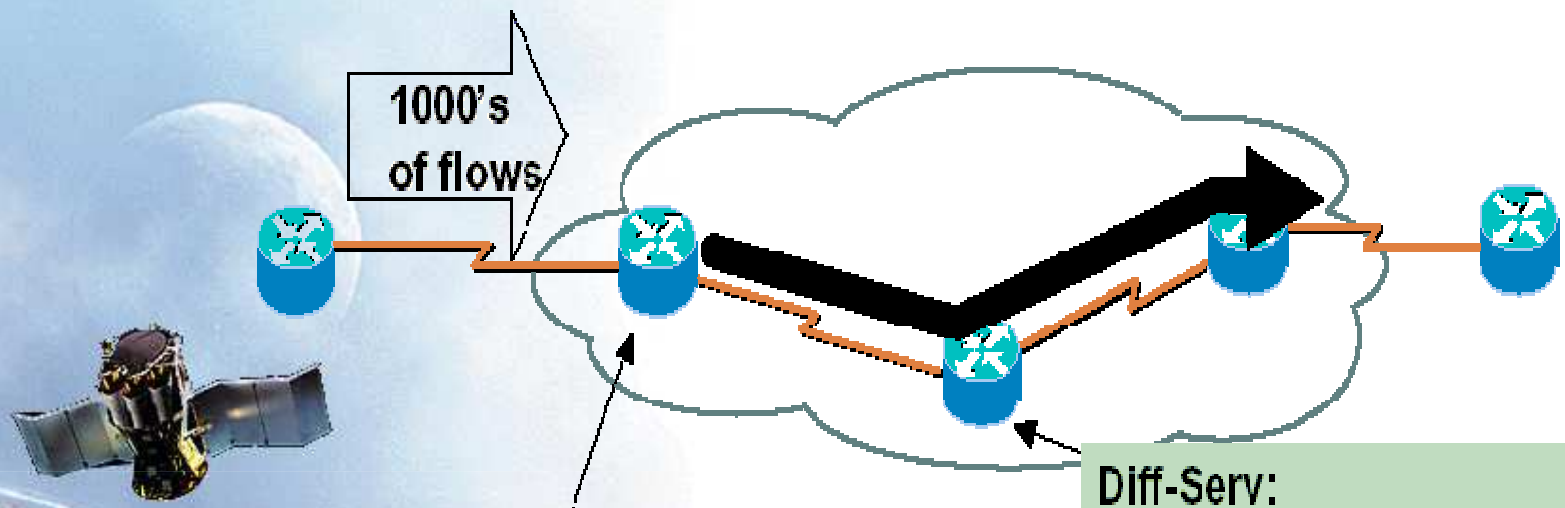
- Edge router

  classificano il traffico in ingresso tramite un campo DS che associa al pacchetto un particolare comportamento di inoltro (PHB)

- Core router

  applicano le strategie di scheduling previste per le diverse classi di traffico

DiffServ (2)



Diff-Serv:

Aggregation on Edge

Many flows associated with a Class (marked with DSCP)

Diff-Serv:

Aggregated Processing in Core

Scheduling/Dropping (PHB) based on DSCP

Classi di traffico DiffServ

- Expedited Forwarding (EF)
  - Rate di uscita dal nodo non inferiore ad una rate configurabile
  - Rate d'uscita indipendente dall'intensità di ogni altro traffico che transita nel nodo
  - Rate di ingresso del traffico forzata ad essere sempre inferiore, o uguale, alla minima rate d'uscita garantita
- Assured Forwarding (AF)
  - Offre differenti livelli di sicurezza di inoltro
  - Per ogni livello è allocata in ogni nodo una certa quantità di risorse
  - Ogni livello ha tre possibili livelli di precedenza di eliminazione

Domanda

- E’ possibile avere il meglio dei “due mondi”, statefull e stateless?
- Fornire servizi “potenti” come quelli forniti da reti statefull
- Utilizzare tecniche scalabili e robuste come quelle usate nelle reti stateless

Soluzione proposta

- Il blocco base della soluzione proposta è il dominio Stateless Core (SCORE)
- Un dominio SCORE è definito come una regione continua nella quale solo gli edge router mantengono lo stato per flusso mentre i core router no
- Dal momento che gli edge usualmente operano a velocità più elevate e gestiscono molti meno flussi dei core router, questa architettura è altamente *scalabile*

Architettura SCORE (1)

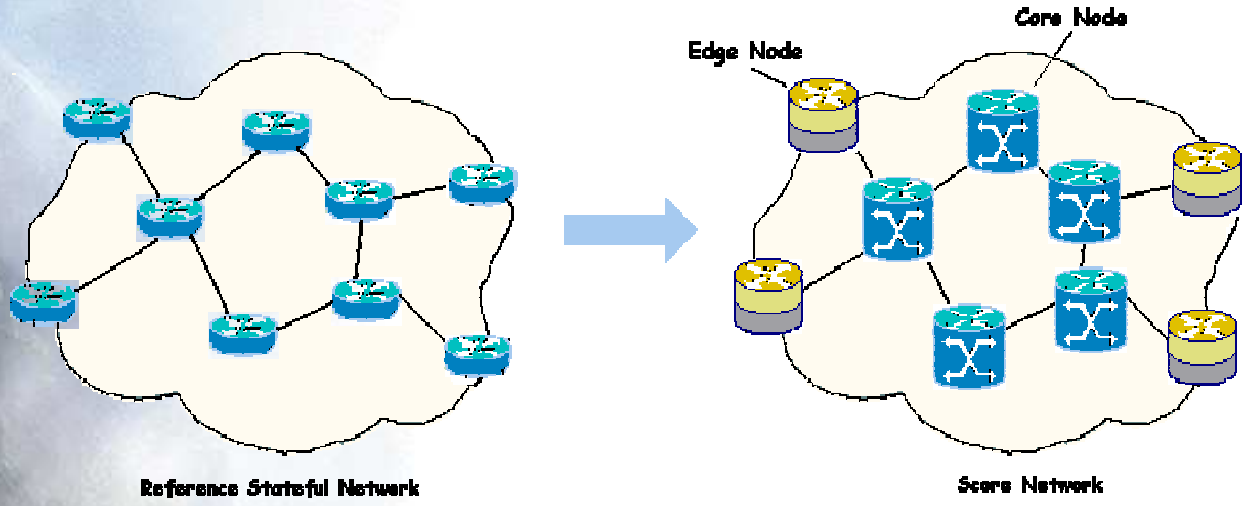
- Per raggiungere l'obiettivo di fornire servizi di rete potenti e flessibili come in una rete stateless è proposto un approccio chiamato "state elimination" e che consiste in 2 passi:

1. Definire una rete di riferimento statefull che implementi i servizi desiderati

2. Approssimare, o se possibile, emulare le funzionalità della rete di riferimento nella rete SCORE

Architetture SCORE (2)

- La rete SCORE ha un'architettura simile a quella Differenziata nella classificazione dei router
- Lo scopo è quello di approssimare i servizi forniti da una rete statefull



Architetture SCORE (3)

- Mostreremo come una rete SCORE può fornire ritardi end-to-end per flusso e garantire la banda come nei servizi IntServ
- Le soluzioni IntServ correnti assumono una rete “statefull” nella quale due tipi di stati per flusso sono necessari:
  - Forwarding state: che è usato dal motore di forwarding per assicurare percorsi di forwarding fissati
  - QoS state: che è usato sia dal modulo di controllo di ammissione nel piano di controllo che dal classificatore e dallo scheduler nel piano di dati

SCORE's problems

- Sebbene la SCORE ha molte vantaggi rispetto alle tradizionali soluzioni statefull essa ancora soffre delle limitazioni di scalabilità e robustezza se comparata con le soluzioni stateless

- La scalabilità è limitata dal fatto che i router sui confini devono essere statefull

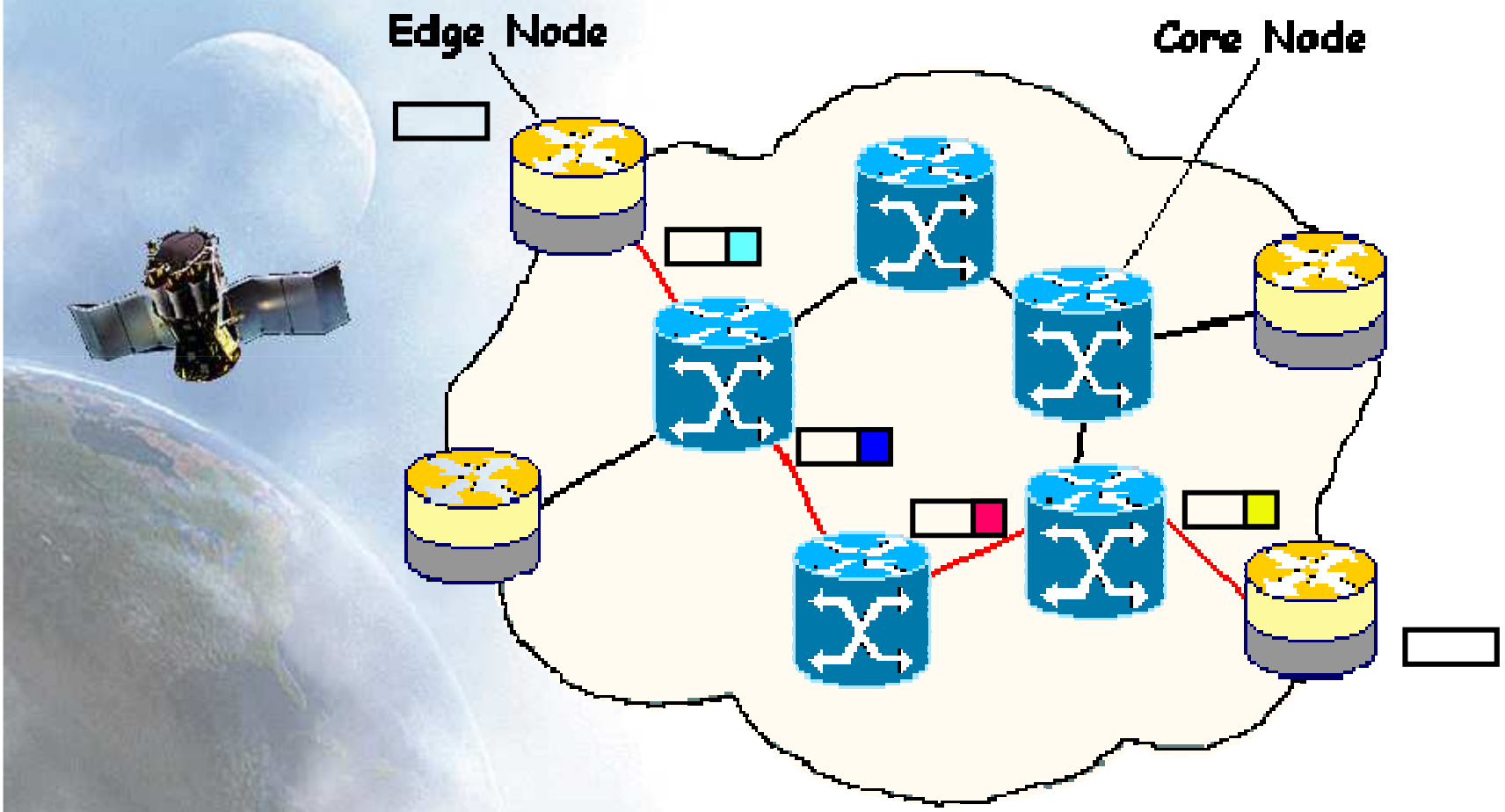
- La robustezza è limitata dalla possibilità che un singolo edge o core router può malfunzionare e inserire informazioni errate nel pacchetto così da impattare sulle prestazioni di tutta la rete

Dynamic Packet State (DPS)

- La tecnica chiave utilizzata per implementare una rete SCORE è la *Dynamic Packet State (DPS)*

  - Ogni pacchetto porta nel suo header alcuni stati che sono inizializzati dagli Ingress Router
  - I router interni processano ogni pacchetto che arriva basandosi sullo stato portato nell' header del pacchetto, aggiornano sia il loro stato sia lo stato nell' header del pacchetto prima di inoltrarlo verso un altro router
  - Alla fine l'egress router rimuove lo stato dall' header del pacchetto

Tecnica DPS



Algoritmi con DPS

- Grazie alla tecnica DPS è possibile utilizzare 2 algoritmi:
  - Uno per il piano dati per ***schedulare*** il pacchetto, chiamato ***Core Jitter Virtual Clock (CJVC)***
  - Uno per il piano di controllo per operare il ***controllo di ammissione (CAC).***

Algoritmo di scheduling

- Il *Core Jitter Virtual Clock (CJVC)* è una variante del *Jitter Virtual Clock (JVC)*

- *Il JVC lavora così:*
  - *Ad ogni pacchetto è assegnato un tempo **eleggibile (e)** e una **deadline (d)** al suo arrivo*
  - *Il pacchetto è trattenuto nel rate controller fin quando non diventa eleggibile*
  - *Lo scheduler ordina l'invio dei pacchetti eleggibili in accordo alla loro deadline*
  - *Per il k-esimo pacchetto del flusso i il suo tempo eleggibile $e_{i,j}^{k}$ e la sua deadline $d_{i,j}^{k}$ al j-esimo nodo sul suo percorso sono calcolati così:*

Jitter Virtual Clock

$$e_{i,j}^{1} = a_{i,j}^{1}$$

$$e_{i,j}^{k} = \max(a_{i,j}^{k} + g_{i,j-1}^{k}, d_{i,j}^{k-1}) \qquad i, j \geq 1, k > 1$$

$$d_{i,j}^{k} = e_{i,j}^{k} + \frac{l_i^k}{r_i} \qquad i, j, k \geq 1$$

- *Dove* l_i^k *è la larghezza del pacchetto*
- r_i *è la rate del flusso*
- $a_{i,j}^{k}$ *è il tempo di arrivo del pacchetto al j-esimo nodo attraversato dal pacchetto*
- $g_{i,j}^{k}$ *, immesso nell'header del pacchetto dal nodo precedente, è l'ammontare di tempo che il pacchetto è stato trasmesso prima della sua deadline, cioè la differenza tra la deadline del pacchetto e il sua attuale tempo di partenza al nodo j-i-esimo.*

Core Jitter Virtual Clock (1)

- Il CJVC è basato sulla tecnica DPS
- L'idea chiave è quella di avere l'ingress router che inserisce i parametri di scheduling in ogni pacchetto
- Il router interno può allora prendere le sue decisioni basandosi su questi parametri, così da eliminare il bisogno di mantenere informazione per singolo flusso
- L'algoritmo ha bisogno di due variabili di stato per ogni flusso i:
  - r_i che è la rate riservata per il flusso i
  - $d_{i,j}^{k}$ che è la deadline dell'ultimo pacchetto del flusso i proveniente dal nodo j

Core Jitter Virtual Clock (2)

- Mentre è facile eliminare r_i non lo è per $d_{i,j}^k$
- La differenza è che la rate è uguale per tutti mentre la deadline è un valore dinamico che è calcolato iterativamente ad ogni nodo
- Però dal momento che $d_{i,j}^k$ è usato solamente in operazioni di massimo possiamo eliminarlo se possiamo assicurare che un altro termine in massimo non è mai più piccolo di $d_{i,j}^k$

- L'idea è quella di usare una variabile slack associata ad ogni pacchetto, denotata con δ_i^k tale che per ogni nodo interno lungo il percorso si abbia:

Core Jitter Virtual Clock (3)

$$\delta_i^1 = 0$$

$$\delta_i^k = \max\left(0, \delta^{k-1} i + \frac{l_i^{k-1} - l_i^k}{r_i} - \frac{e_{i,1}^k - e_{i,1}^{k-1} - l_i^{k-1} / r_i}{h-1}\right) \quad \text{k>1,h>1}$$

- Dove h è un contatore che viene incrementato ad ogni hop e viene calcolato usando l'algoritmo di CAC
- Di seguito è riportato l'algoritmo in pseudocodice

Core Jitter Virtual Clock (4)

- In questo modo il CJVC assicura il tempo eleggibile di ogni pacchetto p_i^k al nodo j non è più piccola della deadline del pacchetto precedente dello stesso flusso al nodo j, i.e. , $e_{i,j}^k \geq d_{i,j}^{k-1}$

- In più, lo schduler Virtual Clock assicura che la deadline di ogni pacchetto è rispettata

- L'esempio di seguito nelle figure (a) e (b) fornisce alcune intuizioni che stanno dietro a questo algoritmo

Figura (a): Jitter Virtual Clock

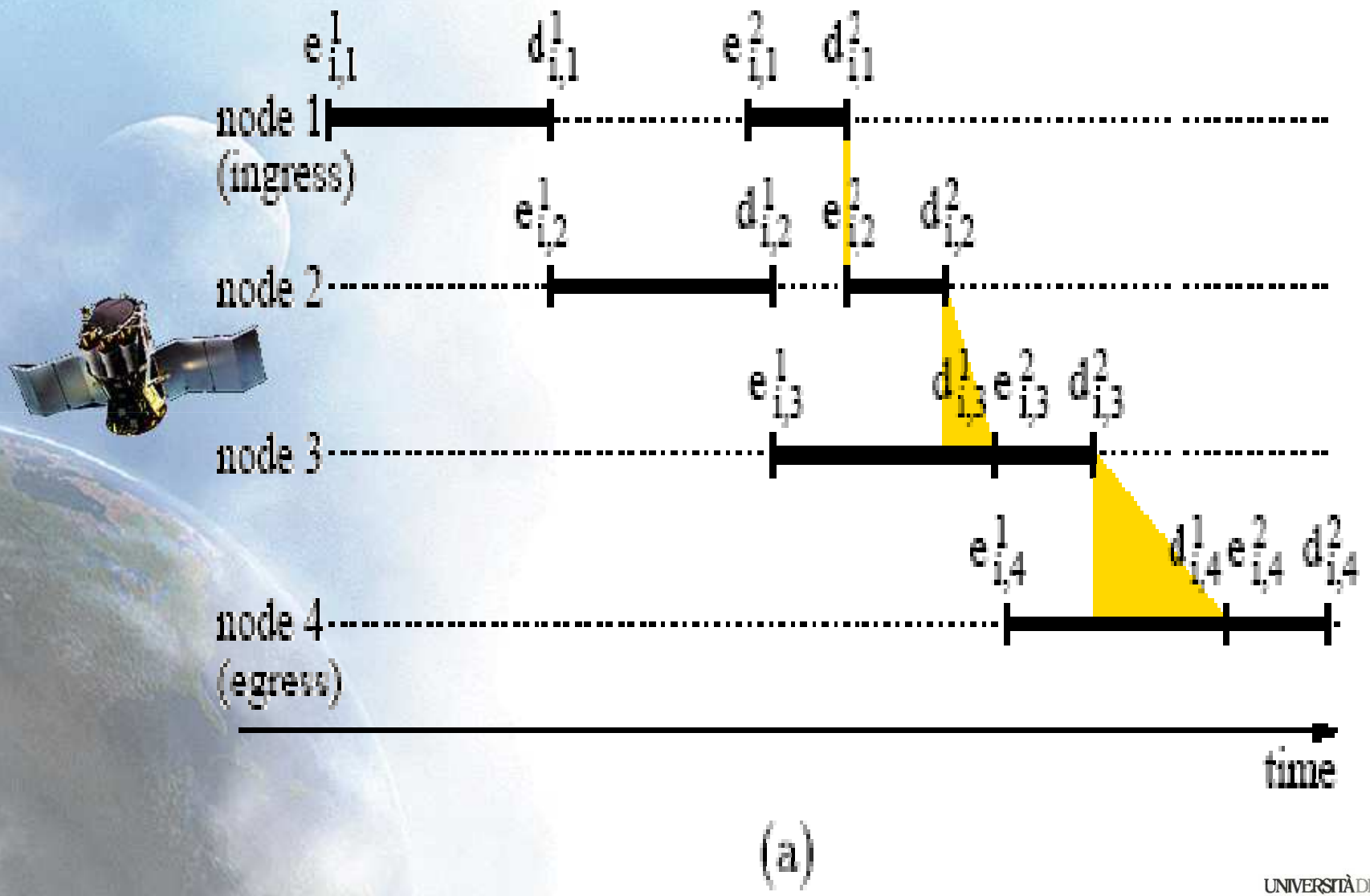
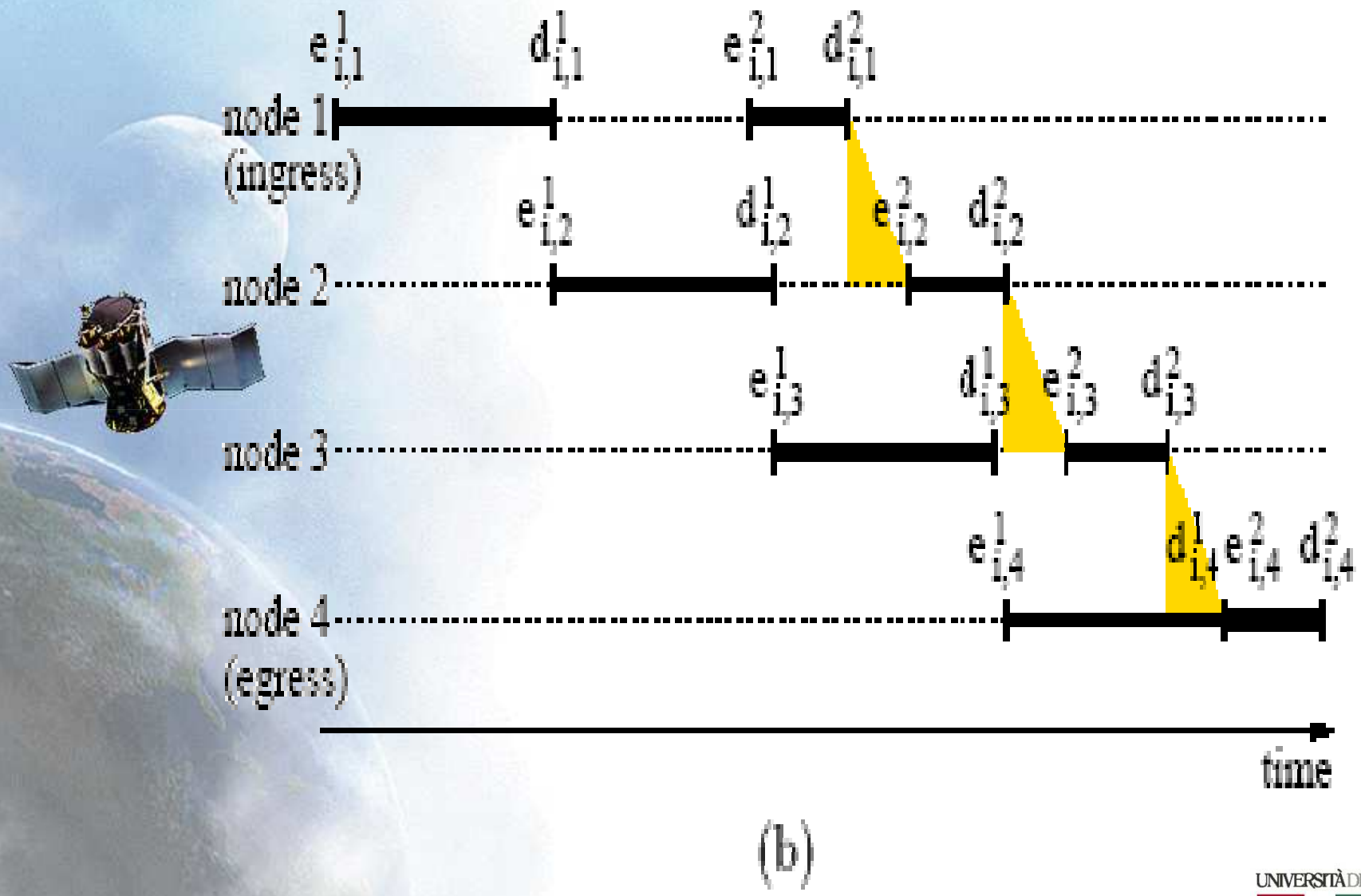


Figura (b): Core Jitter Virtual Clock



(b)

Osservazioni

- L'osservazione base è che con JVC non si tiene conto del ritardo di propagazione, la differenza tra il tempo eleggibile del pacchetto p_i^k al nodo j e la sua deadline al precedente nodo j-1, i.e. , $e_{i,j}^k - d_{i,j-1}^k$, non decresce mai lungo il percorso del pacchetto
- Consideriamo il secondo pacchetto in figura, con JVC la differenza $e_{i,j}^2 - d_{i,j-1}^2$, rappresentato dalla base del triangolo giallo, incrementa in j
- Introducendo la variabile di slack δ_i^k, il CJVC equalizza questi ritardi
- Mentre questo cambiamento può incrementare il ritardo del pacchetto agli hop intermedi non ha effetto sul limite ritardo end-to-end

Core Jitter Virtual Clock: Pseudocodice

ingress node

```
on packet p arrival
    i = get_flow(p);
    if (first_packet_of_flow(p, i))
        e_i = current_time;
        δ_i = 0;
    else
        δ_i = max(0, δ_i + (l_i − length(p))/r_i −
                max(current_time − d_i, 0)/(h − 1)); /* Eq. (5.10) */
        e_i = max(current_time, d_i);
    l_i = length(p);
    d_i = e_i + l_i/r_i;
on packet p transmission
    label(p) ← (r_i, d_i − current_time, δ_i);
```

core/egress node

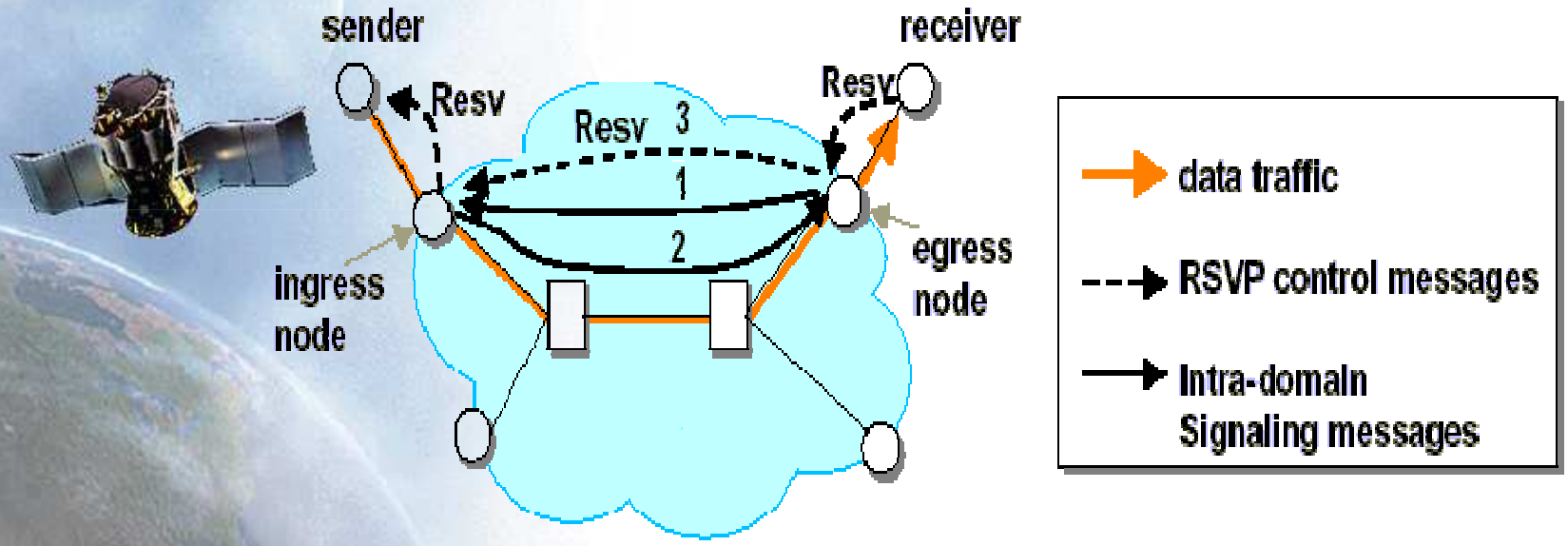
```
    on packet p arrival
        (r, g, δ) ← label(p);
        e = current_time + g + δ; /* Eq. (5.4) */
        d = e + length(p)/r
    on packet p transmission
        if (core node)
            label(p) ← (r, d − current_time, δ);
        else /* this is an egress node */
            clear_label(p);
```

Algoritmo di CAC (1)

- Un componente chiave dell'architettura dei servizi che forniscono garanzie è il controllo di ammissione - Call Admission Control (CAC)
- Lo scopo principale del controllo di ammissione è assicurare che le risorse della rete non siano sovrastimate
- In particolare deve assicurare che la somma delle rate prenotate di tutti i flussi che attraversano un collegamento nella rete non sia maggiore della capacità del collegamento
  - $\sum r_i < C$
- Una nuova richiesta di prenotazione è accettata se passa il test di ammissione lungo tutto il percorso

Algoritmo di CAC (2)

Verrà mostrato un controllo di ammissione per un'architettura distribuita



Controllo di ammissione con RSVP in una rete SCORE

Algoritmo di CAC (3)

- Dal punto di vista del *RSVP* un percorso attraverso un dominio *SCORE* è giusto un collegamento virtuale
- Ci sono due messaggi di base nel *RSVP*: *PATH* e *RESV*
- Questi messaggi sono processati solo dai nodi di frontiera, nessuna operazione è operata dai router interni.
- Appena l'ingress node riceve il messaggio di *PATH*, esso viene semplicemente inoltrato verso l'egress node. Invece, appena l'egress node riceve il messaggio di *RESV* esso manda un messaggio di segnalazione speciale "1" lungo il percorso verso l'ingress node. Appena ricevono il messaggio speciale ogni router interno lungo il percorso opera un test di controllo di ammissione locale.

Algoritmo di CAC (4)

- In più, il messaggio porta un contatore h che è incrementato ad ogni hop e viene utilizzato per il calcolo della variabile *slack* nell'algoritmo di scheduling
- Quando il messaggio "1" raggiunge l'ingress node viene effettuato l'ultimo test e poi viene mandato verso l'egress un nuovo messaggio "2" di reply dopo di che l'egress può inviare il messaggio "3" di *RESV* verso il sender
- Importante è notare che il messaggio di reply può essere mandato anche da un router interno allorquando il test non è andato a buon fine perché in questo caso non è più necessario mandare avanti il test

Algoritmo di CAC (5)

- Dal momento che RSVP usa “raw IP” o UDP per inviare i messaggi di controllo non c’è bisogno di ritrasmissione per i messaggi di segnalazione

- Un messaggio perso non cozza con la semantica RSVP
- Se un sender non riceve un reply dopo un certo timeout, essa semplicemente elimina il messaggio Resv

Ammissione di controllo “Per-Hop” (1)

- La semplice soluzione di incrementare la rate prenotata aggregata (*R*) ogni volta che una nuova prenotazione (*r*) è ammessa e di sottrarre la prenotazione (*r*) allorquando essa è terminata non è abbastanza robusta

- E’ possibile mostrare che questa semplice soluzione non è robusta rispetto a varie condizioni di fallimento come perdita di pacchetti, fallimenti di parziali prenotazioni o crash dei nodi della rete
  - Per gestire perdite di pacchetto, quando un nodo riceve messaggi di set-up o di tear-down, il nodo ha bisogno di sapere se è un duplicato di un messaggio già processato
  - Per gestire fallimenti di parziali prenotazioni un nodo ha bisogno di ricordare quale decisione è stata presa per un flusso al passo precedente

Ammissione di controllo “Per-Hop” (2)

- Tutte le esistenti soluzioni mantengono lo stato di prenotazione per flusso
- Utilizzando la tecnica *DPS* è possibile ridurre significativamente la complessità del controllo di ammissione in ambiente distribuito
- L’obiettivo dell’algoritmo è quello di stimare un upper-bound molto vicino alla rate aggregata
- Usando questo bound nel test di ammissione si evita “over-provisioning”, che è una condizione necessaria per fornire servizi garantiti deterministici

Ammissione di controllo “Per-Hop” (3)

- L’algoritmo usa due tecniche.
  1. un upper-bound conservativo di *R*, denotato con R_{bound} ed aggiornato: $R_{bound} = R_{bound} + r$. E’ importante notare che per far si che rimanga un upper-bound di *R* questo algoritmo non ha bisogno di rilevare messaggio di richiesta duplicati generati o per ritrasmissione in caso di perdita di pacchetti o in caso di parziale fallimento della prenotazione. Il problema ovvio di questo algoritmo è che divergerà da *R*. Quando raggiunge la capacità del collegamento, nessuna nuova richiesta può essere accettato anche se ci può essere capacità disponibile

  2. Per evitare questo problema è introdotto un algoritmo separato per stimare periodicamente la rate prenotata aggregata. Basandosi su questa stima, un secondo upper-bound per *R*, denotato con R_{Cal} , è calcolato ed usato per ri-calibrare

Ammissione di controllo “Per-Hop” (4)

- Un importante aspetto dell’algoritmo della stima è che la discrepanza tra l’upper-bound R_{Cal} e l’attuale rate prenotata R può essere limitata

- La ri-calibrazione diventa quindi scegliere il minimo dei due uppur-bound R_{bound} e R_{Cal}
- R_{Cal} è calcolata a partire da una stima non accurata di R denotata con R_{DPS}
- L’algoritmo di stima è basato sul DPS e non richiede ai core router di mantenere lo stato per flusso

Ammissione di controllo “Per-Hop” (5)

- Questo algoritmo ha importanti proprietà:
  1. E’ robusto in presenza di perdita di pacchetti e fallimenti di parziali prenotazioni
  2. Può sovra-stimare R ma mai sotto-stimare R; questo assicura la semantica dei servizi garantiti, mentre sovra-stimare può portare ad una sottoutilizzazione delle risorse della rete, sotto-stimare può risultare in un “over-provisioning” e quindi ad una violazione delle garanzie richieste
  3. In fine, l’algoritmo di stima si auto-corregge nel senso che una sovra-stima in un precedente periodo sarà corretta nel periodo successivo. Questo riduce enormemente la possibilità di una seria sotto-stima delle risorse

Algoritmo di stima della prenotazione aggregata (1)

- Verrà ora presentato l'algoritmo di stima della rate prenotata aggregata che è performato ad ogni core router
- Verrà descritto come R_{Cal} è calcolato e come è usato per ricalibrare R_{bound}
- L'algoritmo del calcolo di R_{Cal} si propone l'obiettivo di bilanciare due goal:
  - R_{Cal} dovrebbe essere un upper-bound di R
  - Errori di sovra-stima dovrebbero essere corretti e tenuti al minimo

Algoritmo di stima della prenotazione aggregata (2)

- Per calcolare R_{Cal} si inizia con una stima in accurata di R, denotata con R_{DPS} e poi si cerca di correggere questa stima
- La stima di R_{DPS} è calcolata usando la tecnica *DPS*: l'ingress node inserisce lo stato b_i^k nell' header del pacchetto p_i^k :

$$b_i^k = r_i(s_{i,1}^k - s_{i,1}^{k-1})$$

Notation	Comments
r_i	flow i's reserved rate
b_i^k	total number of bits flow i is entitled to transmit during $[s_{i,1}^{k-1}, s_{i,1}^k]$, i.e., $b_i^k = r_i(s_{i,1}^k - s_{i,1}^{k-1})$
$R(t)$	aggregate reservation at time t
$R_{bound}(t)$	upper bound of $R(t)$, used by admission test
$R_{DPS}(t)$	estimate of $R(t)$, computed by using DPS
$R_{new}(t)$	sum of all new reservations accepted from the beginning of current estimation interval until t
$R_{cal}(t)$	upper bound of $R(t)$, used to calibrate R_{bound}, computed based on R_{DPS} and R_{new}

Algoritmo di stima della prenotazione aggregata (3)

- dove $s_{i,1}^{k-1}$ e $s_{i,1}^{k}$ sono i tempi in cui i pacchetti p_i^{k-1} e p_i^{k} sono trasmessi dall'ingress router

- Perciò, b_i^k rappresenta l'ammontare totale di bit che il flusso *i* ha diritto di spedire durante l'intervallo $\left[s_{i,1}^{k-1}, s_{i,1}^{k}\right]$

- Il calcolo di R_{DPS} è basato sulla seguente semplice osservazione:la somma dei calori di b di tutti i pacchetti di un flusso i durante un intervallo è una buona approssimazione del numero totale di bit che hanno il diritto di spedire durante il corrispondente intervallo in accordo con la rate prenotata

Algoritmo di stima della prenotazione aggregata (4)

- Similmente la somma dei valori *b* di tutti i pacchetti è una buona approssimazione del numero totale di bit che tutti i flussi hanno il diritto di spedire durante il corrispondente intervallo

- Dividendo, poi, questa somma per la lunghezza dell'intervallo, ottengo la rate prenotata aggregata

Algoritmo di stima della prenotazione aggregata (5)

- Più precisamente, dividendo il tempo in intervalli di lunghezza: T_W : $(u_k, u_{k+1}]$, k>0. Sia $b_i\ (u_k, u_{k+1})$ la somma dei valori *b* dei pacchetti nel flusso *i* ricevuta durante $(u_k, u_{k+1}]$, e sia $B(u_k, u_{k+1})$ la somma dei valori *b* di tutti i pacchetti durante $(u_k, u_{k+1}]$. La stima è allora calcolata alla fine di ogni intervallo $(u_k, u_{k+1}]$ come segue:

$$R_{DPS}(u_{k+1}) = \frac{B(u_k, u_{k+1})}{u_{k+1} - u_k} = \frac{B(u_k, u_{k+1})}{T_W} \qquad (1)$$

- E da questa stima viene calcolato un upper-bound di $R(u_{k+1})$, denotato con $R_{Cal}\ (u_{k+1})$ come segue:

Algoritmo di stima della prenotazione aggregata (6)

- L'algoritmo su esposto introduce due tipi di inaccuratezza:

1. Ignora l'effetto del delay jitter ed il tempo di inter-dipartenza del pacchetto
2. Non considera l'effetto di accettare o terminare una prenotazione nel mezzo di un intervallo di stima. In particolare l'aver accettato nuovi flussi in un intervallo può risultare in una sotto-stima di R attraverso

Per illustrare questo consideriamo il seguente semplice esempio: non ci sono flussi garantiti su un link fino a che una nuova richiesta con rate *r* è accettata alla fine di un intervallo di stima $(u_k, u_{k+1}]$. Se nessun pacchetto del nuovo flusso raggiunge il nodo prima di u_{k+1} , $B(u_k, u_{k+1})$ sarebbe 0, e così anche $R_{DPS}(u_{k+1})$. Il corretto valore dovrebbe essere invece r.

Algoritmo di stima della prenotazione aggregata (7)

- Presentiamo ora l'algoritmo per calcolare un upper-bound di $R(u_{k+1})$, denotato con $R_{Cal}(u_{k+1})$
- Nel fare ciò teniamo conto di entrambi i tipi di inaccuratezza
- Denotiamo con $Z(t)$ l'insieme delle prenotazioni al tempo t
- Il nostro obiettivo è limitare la prenotazione aggregata al tempo u_{k+1} , i.e.,

$$R(u_{k+1}) = \sum_{i \in Z(u_{k+1})} r_i$$

Algoritmo di stima della prenotazione aggregata (8)

- Consideriamo la divisione di $Z(u_{k+1})$ in due sottoinsiemi:
  - Il sottoinsieme delle nuove prenotazioni che sono accettate durante l'intervallo$(u_k, u_{k+1}]$, denotato con $N(u_{k+1})$
  - E il sottoinsieme contenente il resto delle prenotazioni che sono accettate non più tardi di u_{k+1}

Così possiamo esprimere $R(u_{k+1})$ come

$$R(u_{k+1}) = \sum_{i \in Z(u_{k+1}) \setminus N(u_{k+1})} r_i + \sum_{i \in N(u_{k+1})} r_i \qquad (2)$$

Algoritmo di stima della prenotazione aggregata (9)

- L'idea è poi derivare un upper-bound per ognuno dei due termini a destra dell'espressione e calcolare R_{Cal} come la somma di questi due bound

- Per limitare $\sum_{i \in Z(u_{k+1}) \setminus N(u_{k+1})} r_i$ notiamo che:

$$B(u_k, u_{k+1}) \geq \sum_{i \in Z(u_{k+1}) \setminus N(u_{k+1})} b_i(u_k, u_{k+1}) \qquad (3)$$

- La ragione per cui c'è un segno di disuguaglianza è che quando ci sono dei flussi che terminano durante l'intervallo $(u_k, u_{k+1}]$, i loro pacchetti possono ancora dare contributo a $B(u_k, u_{k+1})$ anche se non appartengono a $Z(u_{k+1}) \setminus N(u_{k+1})$

Algoritmo di stima della prenotazione aggregata (10)

- Ora calcoliamo un lower bound per $b_i\,(u_k, u_{k+1})$
- Dalla definizione, dal momento che

$$i \in \mathrm{Z}(u_{k+1}) \setminus \mathrm{N}(u_{k+1})$$

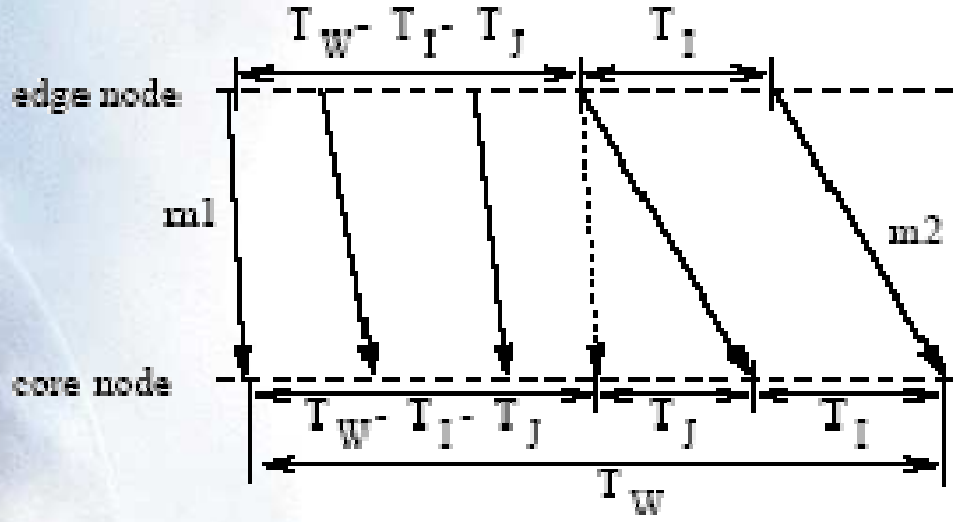
segue che il flusso i contiene una prenotazione durante l'intero intervallo $(u_k, u_{k+1}]$

- Poniamo Ti essere il massimo tempo di inter-dipartenza tra due consecutivi pacchetti di un flusso all'edge node, e poniamo Tj come il massimo delay jitter del flusso, con Ti e Tj molto più piccoli di Tw.

- Consideriamo ora lo scenario presentato nella figura

Algoritmo di stima della prenotazione aggregata (11)

Un core router riceve i pacchetti m1 e m2 appena fuori della finestra di stima. Assumiamo il caso peggiore in cui m1 incorre nel più piccolo ritardo possibile, m2 nel più grande possibile e che l'ultimo pacchetto prima di m2 parta Ti secondi prima, è semplice vedere che la somma dei valori b portati dai pacchetti ricevuti durante l'intervallo di stima dal core node non può essere più piccolo di $r_i(Tw - Ti - Tj)$



Algoritmo di stima della prenotazione aggregata (12)

- Così abbiamo:

$$b_i(u_k, u_{k+1}) > r_i(Tw - Ti - Tj) \qquad \forall i \in Z(u_{k+1}) \setminus N(u_{k+1}) \qquad (4)$$

- Combinando le disequazioni (3) e (4) e l'equazione (1) otteniamo:

$$\sum_{i \in Z(u_{k+1}) \setminus N(u_{k+1})} r_i < \sum_{i \in Z(u_{k+1}) \setminus N(u_{k+1})} \frac{b_i(u_k, u_{k+1})}{Tw(1-f)} \leq \frac{R_{DPS}(u_{k+1})}{1-f} \qquad (5)$$

- dove f=(Ti+Tj)/Tw

Algoritmo di stima della prenotazione aggregata (13)

- Ora, limitiamo il secondo termine del lato destro della equazione (2): $\sum_{i \in \mathrm{N}(u_{k+1})} r_i$
- Per questo, introduciamo una nuova variabile globale R_{new} inizializzata a 0 all'inizio di ogni intervallo $(u_k, u_{k+1}]$ ed aggiornata a $R_{new} + r$ ogni volta che una nuova prenotazione *r* è accettata
- Poniamo $R_{new}(t)$ il valore della variabile al tempo t
- Per semplicità, qui assumiamo che un flusso di cui è accettata la prenotazione durante l'intervallo $(u_k, u_{k+1}]$ diventa attivo non più tardi di u_{k+1}

Algoritmo di stima della prenotazione aggregata (14)

- Allora è facile vedere che:

$$\sum_{i \in \mathrm{N}(u_{k+1})} r_i \leq R_{new}(u_{k+1}) \qquad (6)$$

- La disuguaglianza è valida quando non ci sono richieste di prenotazioni duplicate e nessuna delle nuove prenotazioni accettate termina durante l'intervallo
- Allora definiamo $R_{Cal}(u_{k+1})$ come:

$$R_{Cal}(u_{k+1}) = \frac{R_{DPS}(u_{k+1})}{1-f} + R_{new}(u_{k+1}) \qquad (7)$$

Algoritmo di stima della prenotazione aggregata (15)

- Dall'eq.(2), e dalle diseq.(5) e (6) segue facilmente che $R_{Cal}(u_{k+1})$ è un upper-bound per $R(u_{k+1})$

$$R_{Cal}(u_{k+1}) > R(u_{k+1})$$

- Infine usiamo $R_{Cal}(u_{k+1})$ per ri-calibrare l'upper-bound della prenotazione aggregata, R_{bound} al tempo $(u_k, u_{k+1}]$ come:

$$R_{Bound}(u_{k+1}) = \min\left(R_{Bound}(u_{k+1}), R_{Cal}(u_{k+1})\right)$$

Algoritmo di stima della prenotazione aggregata (16)

- Nella figura seguente è mostrato lo pseudocodice dell'algoritmo di controllo al core node

Per-hop Admission Control

on reservation request r
if $(R_{bound} + r \leq C)$ /* *perform admission test* */
  $R_{new} = R_{new} + r$;
  $R_{bound} = R_{bound} + r$;
  accept request;
else
  deny request;
on reservation termination r /* *optional* */
  $R_{bound} = R_{bound} - r$;

Aggregate Reservation Bound Comp.

on packet arrival p
  $b \leftarrow get\_b(p)$; /* *get b value inserted by ingress (Eq. (5.12))* */
  $L = L + b$;
on time-out T_W
  $R_{DPS} = L/T_W$; /* *estimate aggregate reservation* */
  $R_{bound} = \min(R_{bound}, R_{DPS}/(1-f) + R_{new})$;
  $R_{new} = 0$;

Algoritmo di stima della prenotazione aggregata: Osservazioni (1)

- L'algoritmo di stima usa solo informazioni nell'intervallo corrente. Questo rende l'algoritmo robusto rispetto alle perdite ed alle duplicazioni dei pacchetti di segnalazione dal momento che i loro effetti sono "dimenticati" dopo un intervallo di tempo

- Esempio: se un nodo processa sia il messaggio originale che quello duplicato della stessa richiesta di prenotazione durante un intervallo $(u_k, u_{k+1}]$, R_{bound} sarà aggiornato due volte per lo stesso flusso. Ma, questo aggiornamento errato non si rifletterà nel calcolo di $R_{DPS}(u_{k+2})$ dal momento che il calcolo è basato sui valori di b ricevuti durante l'intervallo $(u_{k+1}, u_{k+2}]$

Algoritmo di stima della prenotazione aggregata: Osservazioni (2)

- Nota che dal momento che $R_{Cal}(u_k)$ è un upper-bound di $R(u_k)$ una semplice soluzione sarebbe usare $R_{Cal}(u_k) + R_{new}$ invece di R_{bound} per performare il test di ammissione durante l'intervallo $(u_k, u_{k+1}]$

- Il problema con questo approccio è che R_{Cal} può sovra-stimare la prenotazione aggregata *R*

Algoritmo di stima della prenotazione aggregata: Osservazioni (3)

- Una possibile ottimizzazione dell'algoritmo per il controllo di ammissione è aggiungere dei messaggi per la terminazione della prenotazione (vedi pseudocodice)
- Questo ridurrà la discrepanza tra l'upper-bound R_{bound} e la prenotazione aggregata *R*
- Comunque per garantire che R_{bound} rimanga un upper-bound di *R* dobbiamo assicurare che un messaggio di terminazione è spedito almeno una volta, i.e., non ci sono ritrasmissioni se il messaggio è perso
- In pratica questa proprietà può essere rafforzata dagli edge router che mantengono lo stato per flusso

Algoritmo di stima della prenotazione aggregata: Osservazioni (4)

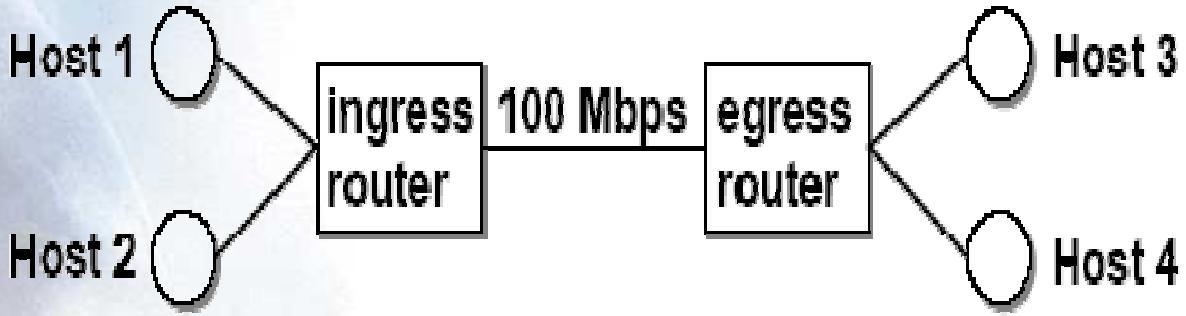
- Infine, per assicurare che il massimo tempo di inter-dipartenza non è più largo di Ti, l'IR può aver bisogno di inviare un pacchetto di "dummy" nel caso non ci siano pacchetti che arrivino per un flusso durante un intervallo Ti

- Questo può essere raggiunto mantenendo all'IR un timer per ogni flusso

- Un'ottimizzazione sarebbe aggregare tutti i "microflussi" tra ogni coppia IR-ER in un unico flusso e calcolare i valori di b sulla base della rate di prenotazione aggregata, e inserire un pacchetto di dummy solo se non ci sono pacchetti del flusso aggregato durante un intervallo

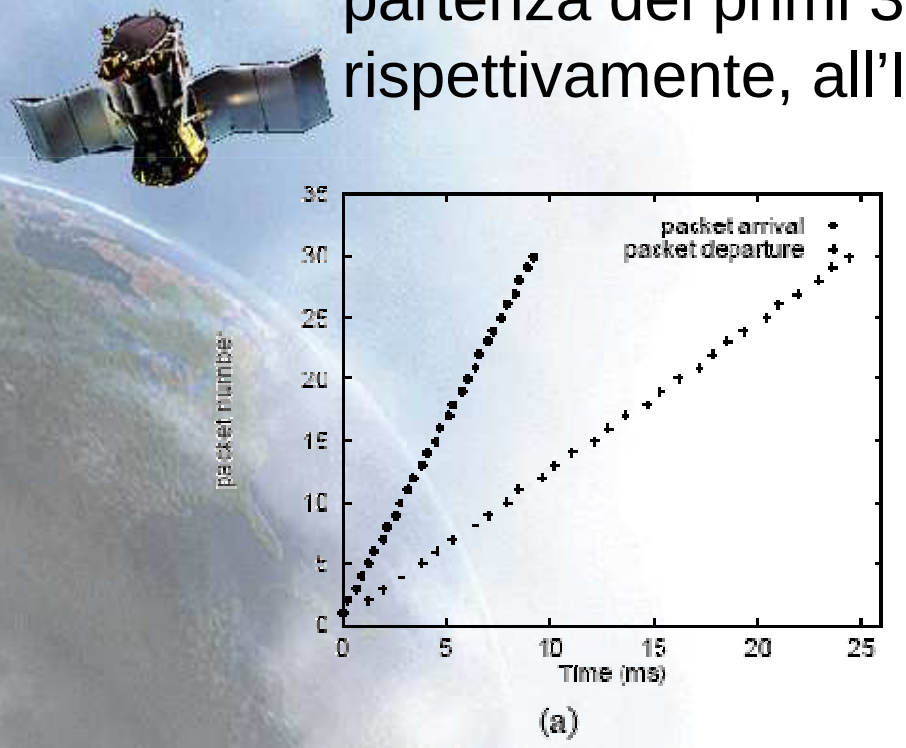
Prove simulative

- Verranno presentati quattro semplici esperimenti
- Tutti gli esperimenti sono effettuati sulla topologia di rete in figura
  - Il primo router è configurato come un ingress router
  - Il secondo è configurato come un egress router ma implementa anche le funzionalità di un core router
- Tutto il traffico è UDP con pacchetti di 1000 bytes senza header

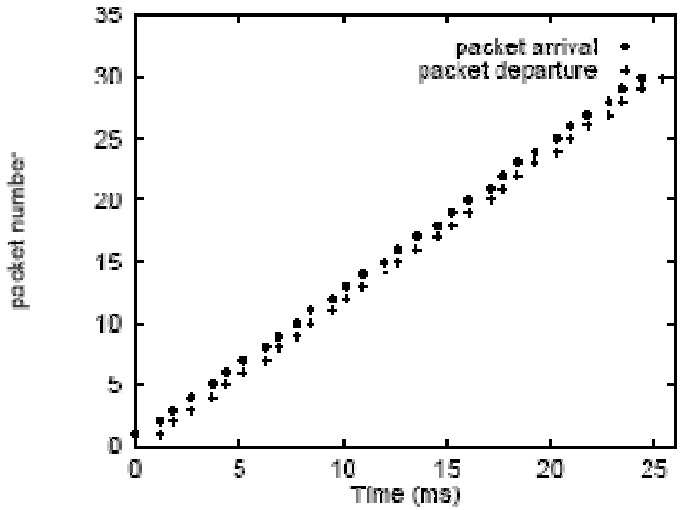


Primo esperimento

- Nel I esperimento viene considerato un flusso tra l'host 1 e l'host 3 che ha una prenotazione 10 Mbps ma invia ad una rate di 30 Mbps
- Le figure (a) e (b) mostrano l'arrivo e la partenza dei primi 30 pacchetti, rispettivamente, all'IR e all'ER



(a)



(b)

Osservazioni

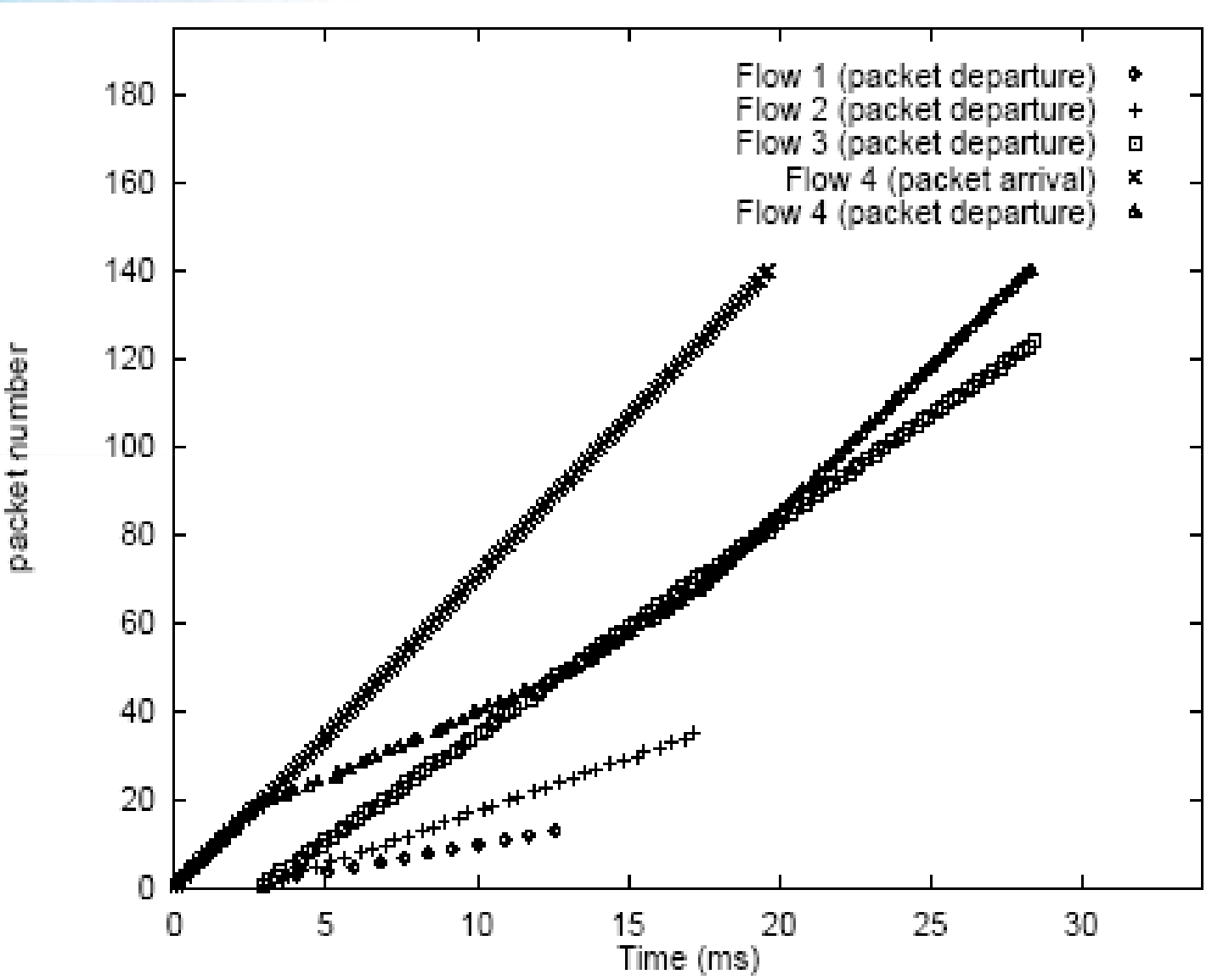
- E’ importante notare:
  - la rate di arrivo all’IR è almeno tre volte la rate di partenza che è quella riservata di 10 Mbps
  - Ciò illustra la natura non conservativa dell’algoritmo di CJVC che forza il profilo di traffico e permette solo 10 Mbps di traffico nella rete

- Un’altra cosa da notare:
  - Tutti i pacchetti incorrono in un ritardo di circa 0.8 ms all’ER.
  - Questo perché sono inviati dall’IR appena diventano eleggibili, e perciò essendo $g \cong l / r$, cioè in numeri 8*1052bits/10Mbps = 0.84ms
  - Quindi i pacchetti saranno trattenuti nel rate controller del prossimo hop, che nel nostro caso è l‘ER

Secondo esperimento

- Si considerano tre flussi garantiti tra l'host 1 e l'host 3 con prenotazioni di 10, 20 e 40 Mbps, in più si considera un quarto flusso tra l'host 2 e l'host 4 trattato come best-effort
- Le rate di arrivo dei primi tre flussi sono leggermente più grandi delle loro prenotazioni mentre quella del quarto flusso è di 60 Mbps
- Al tempo 0 solo il flusso 4 è attivo e dopo 2.8 ms si attivano simultaneamente gli altri 3 flussi
- I flussi 1 e 2 termina rispettivamente dopo 12 e 35 pacchetti
- La figura che segue mostra il tempo di arrivo e di partenza dei pacchetti del flusso 4 best-effort e il tempo di partenza dei pacchetti degli altri 3 flussi

Figura


Flow 1 (packet departure)
Flow 2 (packet departure)
Flow 3 (packet departure)
Flow 4 (packet arrival)
Flow 4 (packet departure)
packet number
Time (ms)
0
20
40
60
80
100
120
140
160
180
0
5
10
15
20
25
30

Osservazioni

- I pacchetti best-effort sperimentano un ritardo molto basso nei primi 2.8 ms
- Appena si attivano i flussi garantiti i pacchetti best-effort sperimentano ritardi molto più lungi mentre i tre flussi ricevono i servizi alla loro rate
- Dopo che i 2 flussi terminano il traffico best-effort può usufruire della banda che rimane

Esperimenti per il CAC (1)

- I successivi esperimenti illustrano l'algoritmo per il controllo di ammissione
  - Il terzo esperimento illustra l'accuratezza della stima della prenotazione aggregata basata su parametro *b* incluso nell'header del pacchetto
  - Il quarto illustra il calcolo del bound per la prenotazione aggregata R_{bound}, quando una nuova prenotazione è accettata o quando una prenotazione termina
- In questi esperimenti si usa un intervallo Tw, di 5 secondi, ed un massimo tempo di inter-dipartenza Ti di 500 ms

Esperimenti per il CAC (2)

- Poiché tutti i pacchetti hanno la stessa dimensione il ritardo sperimentato dall'IR all'ER per i pacchetti di uno stesso flusso è lo stesso, quindi assumiamo Tj=0

- Da qui

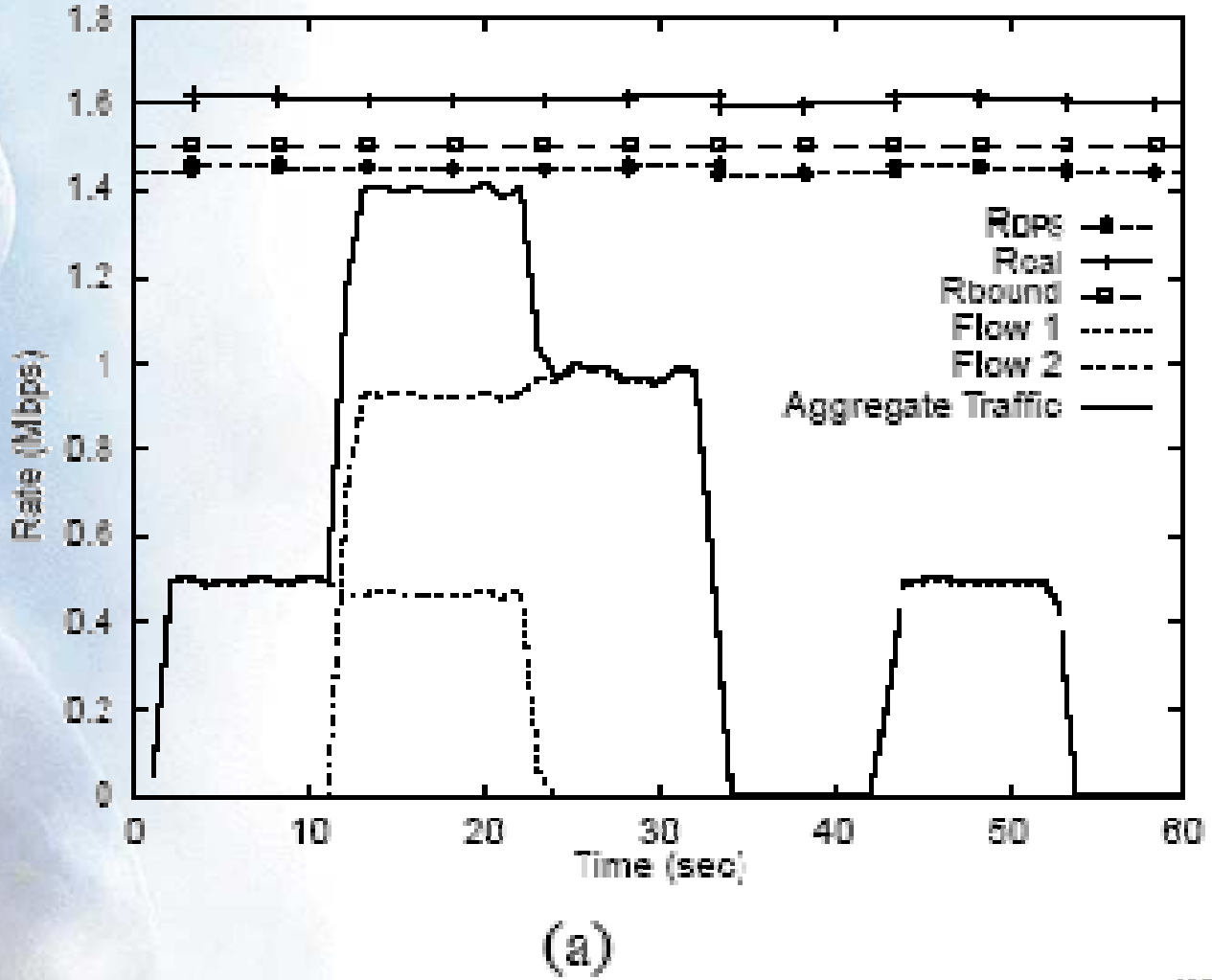
$$f = \frac{T_I + T_J}{T_W} = 0.1ms$$

Da notare che, poiché tutti i pacchetti hanno la stessa dimensione, il valore di δ nell'algoritmo di scheduling è pari a zero

Terzo esperimento

- In questo esperimento si considerano due flussi, il primo con una prenotazione di 0.5 Mbps ed il secondo di 1.5 Mbps
- La figura che segue mostra la rate di arrivo dei flussi singoli così come la rate di arrivo del traffico aggregato
- In più mostra il bound usato per la prenotazione aggregata, R_{bound} , la stima della prenotazione aggregata, R_{DPS} , ed il bound usato per R_{Cal} ricalibrare R_{bound}

Figura (a)

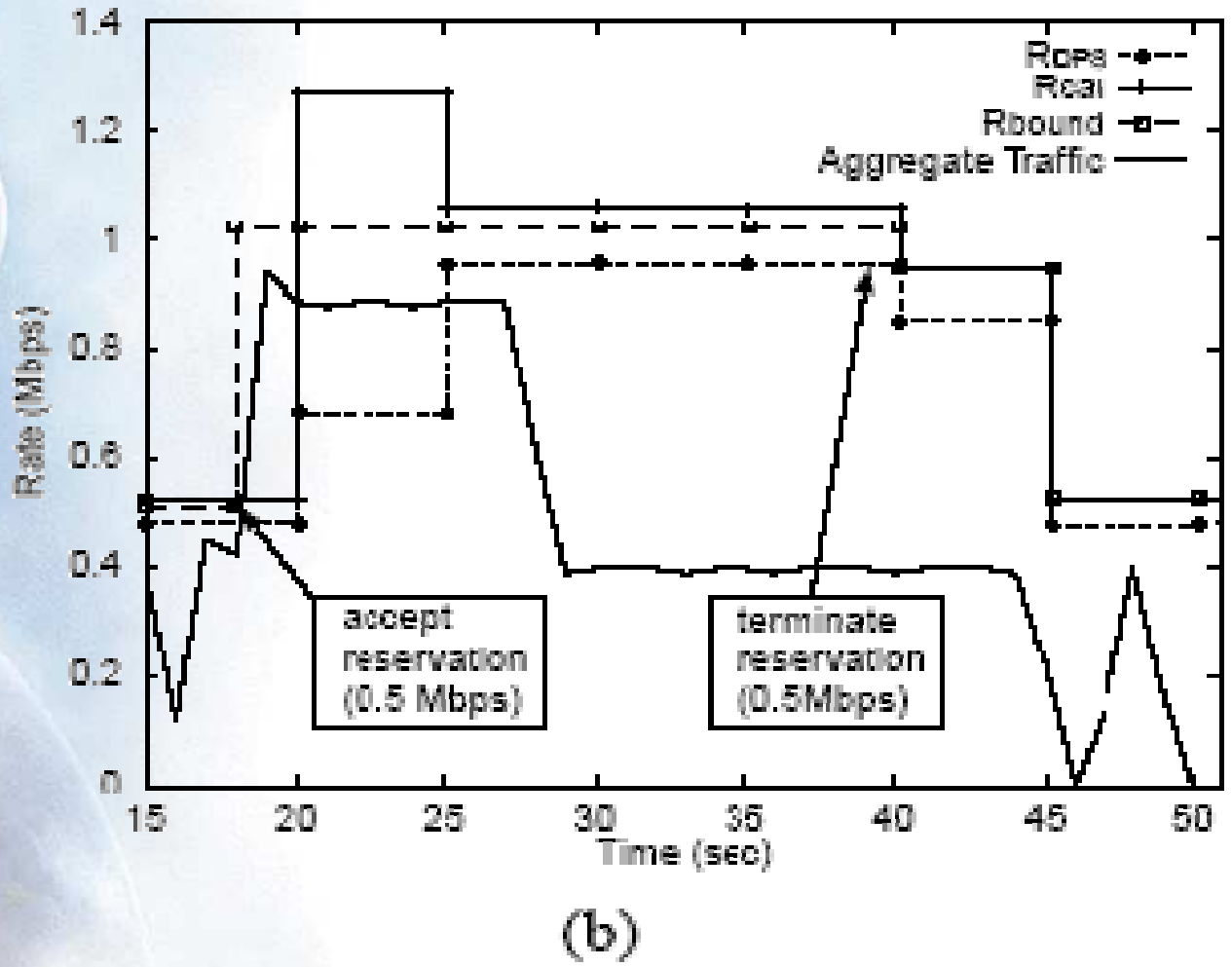

1.8
1.6
1.4
1.2
1
0.8
0.6
0.4
0.2
0
Rate (Mbps)
0
10
20
30
40
50
60
Time (sec)
Rcal
Rbound
Flow 1
Flow 2
Aggregate Traffic
(a)

Quarto esperimento

- In quest'ultimo esperimento si considera uno scenario in cui un flusso di 0.5 Mbps è accettato al tempo 18s e termina a circa 39s

- Anche in questo caso vengono mostrati gli andamenti delle rate R_{bound} , R_{DPS} , R_{Cal}

Figura (b)


1.4
1.2
1
0.8
0.6
0.4
0.2
0
Rate (Mbps)
15
20
25
30
35
40
45
50
Time (sec)
Rops
Real
Rbound
Aggregate Traffic
accept
reservation
(0.5 Mbps)
terminate
reservation
(0.5Mbps)

(b)

Overhead di processamento

- Per valutare l'overhead introdotto da questo algoritmo sono stati effettuati tre esperimenti considerando 1, 10 e 100 flussi
- Si considerano le prenotazioni e le rate di invio uguali per tutti i flussi
- Inoltre la rate di invio aggregata è circa il 20% più grande della rate di prenotazione aggregata
- Nella tabella seguente viene mostrata la media e la deviazione standard per i tempi di codifica e decodifica sia all'IR che all'ER
- I valori sono basati su una misura di 1000 pacchetti

Tabella

	Baseline		1 flow			
			ingress		egress	
	avg	std	avg	std	avg	std
enqueue	1.03	0.91	5.02	1.63	4.38	1.55
dequeue	1.52	1.91	3.14	3.27	2.69	2.81

10 flows				100 flows			
ingress		egress		ingress		egress	
avg	std	avg	std	avg	std	avg	std
5.36	1.75	4.60	1.60	5.91	1.81	5.40	2.33
2.79	3.68	2.30	2.91	2.77	2.82	1.73	2.12

Osservazioni (1)

- Questa implementazione aggiunge meno di 5 µs di overhead per le operazioni di codifica e circa 2 µs per le operazioni di decodifica
- Entrambe le operazioni di codifica e decodifica sono più grandi all'IR che all'ER, poiché l'IR opera su base flusso
- Inoltre, all'aumentare del numero di flussi i tempi di codifica aumentano leggermente, meno del 20%
  - Quindi l'algoritmo è scalabile nel numero di flussi

Osservazioni (2)

- I tempi di decodifica decrescono all'aumentare del numero di flussi
  - Questo perché il rate controller è implementato come una "calendar queue" dove ogni entry corrisponde ad un intervallo di tempo di 128 μs
  - I pacchetti con tempo eleggibile che ricadono nello stesso intervallo sono memorizzati nella stessa entry
  - Quindi, quando il numero di pacchetti è alto, più pacchetti ricadono nella stessa entry
  - e, dal momento che tutti questi pacchetti sono trasferiti durante una solo operazione quando diventano eleggibili, l'overhead per pacchetto decresce.