

Corso di Reti di Calcolatori

Anno accademico 2007–2008

Note integrative sugli algoritmi di *Traffic Shaping*

Mauro Brunato

Versione 1.0 — 9 dicembre 2007

Sommario

Queste note, molto sintetiche, contengono esclusivamente alcune integrazioni al contenuto del Tanenbaum discusse a lezione.

Vogliamo esaminare brevemente, ma in modo quantitativo, il comportamento di un algoritmo *Leaky Bucket*. Seguono alcune indicazioni sulla realizzazione dell'algoritmo *Token Bucket* e un confronto fra i comportamenti dei due algoritmi.

1 L'algoritmo Leaky Bucket

L'algoritmo Leaky Bucket (Secchio bucato) è facilmente esprimibile con un paragone idraulico (figura 1, parte sinistra): un flusso in ingresso, intenso a piacere, dev'essere moderato entro una portata predefinita; l'eventuale eccedenza dev'essere mantenuta in un tampone (buffer) che la rilascia a una velocità predefinita. Se il tampone si riempie, parte del flusso viene perduta.

Supponiamo che l'algoritmo sia realizzato in un componente di rete con un ingresso che riceve dati provenienti da una macchina e un'uscita controllata dai seguenti parametri, visibili nella parte destra di figura 1:

- Un buffer d'uscita di capienza massima M_{\max} ;
- Una velocità massima d'uscita B_{\max} .

Le regole di inoltro sono semplici: i dati in ingresso vengono inseriti nel buffer, che funziona come una coda FIFO; se il buffer è pieno, i dati vengono scartati. Finché ci sono dati in coda, questi vengono prelevati alla velocità massima consentita in uscita, B_{\max} .

Un'assunzione molto comune nel tipo di analisi che ci apprestiamo a fare è il cosiddetto *modello fluido*: i dati sono visti come una quantità continua che fluisce attraverso la rete, come acqua in un condotto. Tale approssimazione è accurata se le unità di discretizzazione, tipicamente pacchetti, sono molto piccole rispetto alla banda di trasmissione.

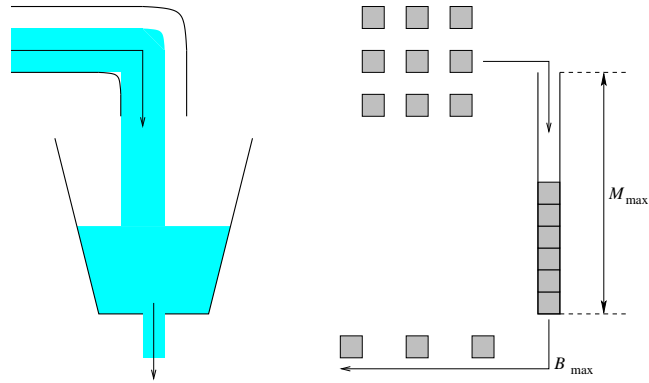


Figura 1: L'algoritmo Leaky Bucket

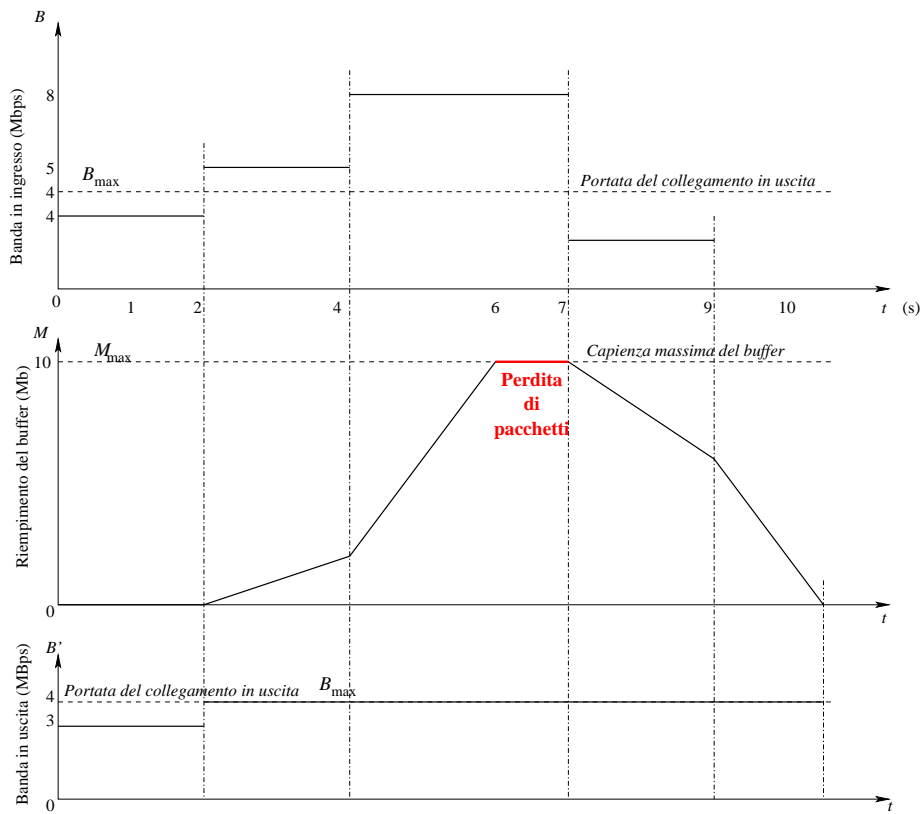


Figura 2: Sviluppo dell'esempio proposto

1.1 Un problema eemplificativo

Supponiamo, ad esempio, che un router riceva, da una linea di ingresso, un flusso di dati che evolve nel tempo come indicato nella parte superiore di figura 2: il flusso è di 3Mbps per i primi 2 secondi, poi cresce a 5Mbps per altri 2 secondi, raggiunge un picco di 8Mbps mantenendolo per 3 secondi, scende a 2Mbps per 2 secondi, poi si interrompe. Il router ha una velocità di uscita $B_{\max} = 4\text{Mbps}$, e possiede una coda da 8Mb.

Vogliamo studiare l'evoluzione nel tempo dello stato del buffer, evidenziando in particolare eventuali trabocchi con perdita dei dati, e il flusso dei dati in uscita.

I primi due secondi. Per i primi due secondi, non ci sono problemi: la coda viene svuotata man mano che arrivano pacchetti (il flusso in ingresso al secchio è inferiore alla portata del buco sul fondo, quindi tutta l'acqua defluisce non appena arriva). Segue che la coda resta vuota, (grafico nel mezzo in figura 2), e il flusso in uscita eguaglia quello in ingresso (grafico in basso di figura 2).

Da 2 a 4 secondi. Non appena il flusso in ingresso sale al di sopra di B_{\max} , il buffer comincia a riempirsi. La velocità di riempimento è ovviamente pari alla differenza fra il flusso entrante e il flusso uscente, risulta quindi di 1Mbps. Dopo due secondi, il buffer conterrà dunque 2Mb. Si noti che il flusso di uscita è pari a B_{\max} . In generale, se il buffer non è vuoto il flusso in uscita è sempre pari al massimo consentito.

Da 4 a 7 secondi. La velocità di riempimento del buffer sale a 4Mbps (il divario fra flusso entrante e flusso uscente); dopo 2 secondi, il buffer raggiunge la sua piena capacità. A quel punto, il router comincia a scartare pacchetti. Il flusso in uscita rimane stabile a B_{\max} .

Da 7 a 9 secondi. Il flusso in ingresso è inferiore a quello in uscita. Il buffer si svuota alla velocità di 2Mbps. La velocità in uscita è sempre il massimo.

Da 9 a 11.5 secondi. Il flusso in ingresso è interrotto. Il buffer si svuota alla velocità massima, B_{\max} , corrispondente al flusso di uscita. Una volta che il buffer è vuoto, il flusso in uscita ovviamente si interrompe.

2 L'algoritmo Token Bucket

L'algoritmo Token Bucket (secchio di gettoni) serve ad obbligare il sistema a mantenere una velocità di trasmissione media limitata, pur consentendo occasionalmente dei picchi.

Idealmente, per spedire un pacchetto il sistema deve possedere un *gettone* (token). I gettoni in possesso del sistema sono raccolti in un "secchio", e nuovi gettoni vengono generati con un periodo prefissato. Il secchio ha una capacità massima: non è possibile mettere da parte un numero arbitrariamente grande di gettoni.

Variabile	Tipo	Significato
<i>massimo</i>	Parametro	Massimo numero di gettoni che il secchio può contenere
<i>periodo</i>	Parametro	Intervallo di tempo fra successive generazioni di gettoni
<i>gettoni</i>	Variabile globale	Numero di gettoni attualmente contenuti nel secchio
<i>pacchetto</i>	Oggetto locale	Pacchetto da spedire

1. **procedura** inizio
2. $\left[\begin{array}{l} \textit{gettoni} \leftarrow 0 \\ \text{timer}(\textit{crea_token}, \textit{periodo}) \end{array} \right.$
3. $\left[\begin{array}{l} \textit{gettoni} \leftarrow 0 \\ \text{timer}(\textit{crea_token}, \textit{periodo}) \end{array} \right.$
4. **procedura** *crea_token*
5. $\left[\begin{array}{l} \text{se } \textit{gettoni} < \textit{massimo} \\ \textit{gettoni} \leftarrow \textit{gettoni} + 1 \end{array} \right.$
6. $\left[\begin{array}{l} \text{se } \textit{gettoni} < \textit{massimo} \\ \textit{gettoni} \leftarrow \textit{gettoni} + 1 \end{array} \right.$
7. **procedura** *invio* (*pacchetto*)
8. $\left[\begin{array}{l} \text{attendi}(\textit{gettoni} > 0) \\ \textit{gettoni} \leftarrow \textit{gettoni} - 1 \\ \text{passa } \textit{pacchetto} \text{ al livello inferiore} \end{array} \right.$
9. $\left[\begin{array}{l} \text{attendi}(\textit{gettoni} > 0) \\ \textit{gettoni} \leftarrow \textit{gettoni} - 1 \\ \text{passa } \textit{pacchetto} \text{ al livello inferiore} \end{array} \right.$
10. $\left[\begin{array}{l} \text{attendi}(\textit{gettoni} > 0) \\ \textit{gettoni} \leftarrow \textit{gettoni} - 1 \\ \text{passa } \textit{pacchetto} \text{ al livello inferiore} \end{array} \right.$

Figura 3: La versione asincrona dell'algoritmo

3 La versione asincrona dell'algoritmo

È facile immaginare l'algoritmo come fosse composto di due diversi processi (figura 3). Il primo, gestito da un timer, invoca periodicamente la procedura di aggiunta di un gettone al secchio. Il secondo, invocato quando c'è un pacchetto da spedire, attende la disponibilità di un gettone e lo usa.

4 La versione sincrona

Un'implementazione che non comporta inutili incrementi asincroni della variabile *gettoni* è presentato in figura 4. In questo caso, il numero di gettoni in possesso del sistema non è sempre aggiornato; una variabile contiene l'istante di ultimo aggiornamento, mentre il numero di gettoni da aggiungere al secchio è calcolato in base al tempo trascorso dall'ultimo aggiornamento.

5 Confronto fra i due algoritmi

Per quanto i due algoritmi siano applicabili in campi abbastanza diversi (il primo in un componente di rete, il secondo nella macchina client), è possibile confrontarne il comportamento in figura 5. Si suppone che il sistema generi due raffiche di dati che superano la capacità B_{\max} della linea di uscita.

Un traffic shaper di tipo Leaky Bucket si limita a "spalmare" il flusso su un tempo maggiore grazie al buffer, come si vede nella parte inferiore di figura 5. Si noti che, se non ci sono perdite dovute al riempimento del buffer, l'area sottesa

Variabile	Tipo	Significato
<i>massimo</i>	Parametro	Massimo numero di gettoni che il secchio può contenere
<i>periodo</i>	Parametro	Intervallo di tempo fra successive generazioni di gettoni
<i>ultimo_invio</i>	Variabile globale	Istante in cui l'ultimo pacchetto è stato inviato
<i>gettoni</i>	Variabile globale	Numero di gettoni attualmente contenuti nel secchio
<i>pacchetto</i>	Oggetto locale	Pacchetto da spedire
<i>t</i>	Variabile locale	Istante in cui il pacchetto arriva al livello rete

```

1. procedura inizio
2.   [ ultimo_invio ←  $-\infty$ 
3.   [ gettoni ← 0

4. procedura invio (pacchetto)
5.   [ t ← tempo attuale
6.   [ gettoni ← gettoni + (t - ultimo_invio) / periodo
7.   [ se gettoni > massimo
8.   [   gettoni ← massimo
9.   [ se gettoni = 0
10.  [   attendi (ultimo_invio + periodo - t)
11.  [ altrimenti
12.  [   gettoni ← gettoni - 1
13.  [   passa pacchetto al livello inferiore
14.  [   ultimo_invio ← tempo attuale

```

Figura 4: La versione sincrona dell'algoritmo

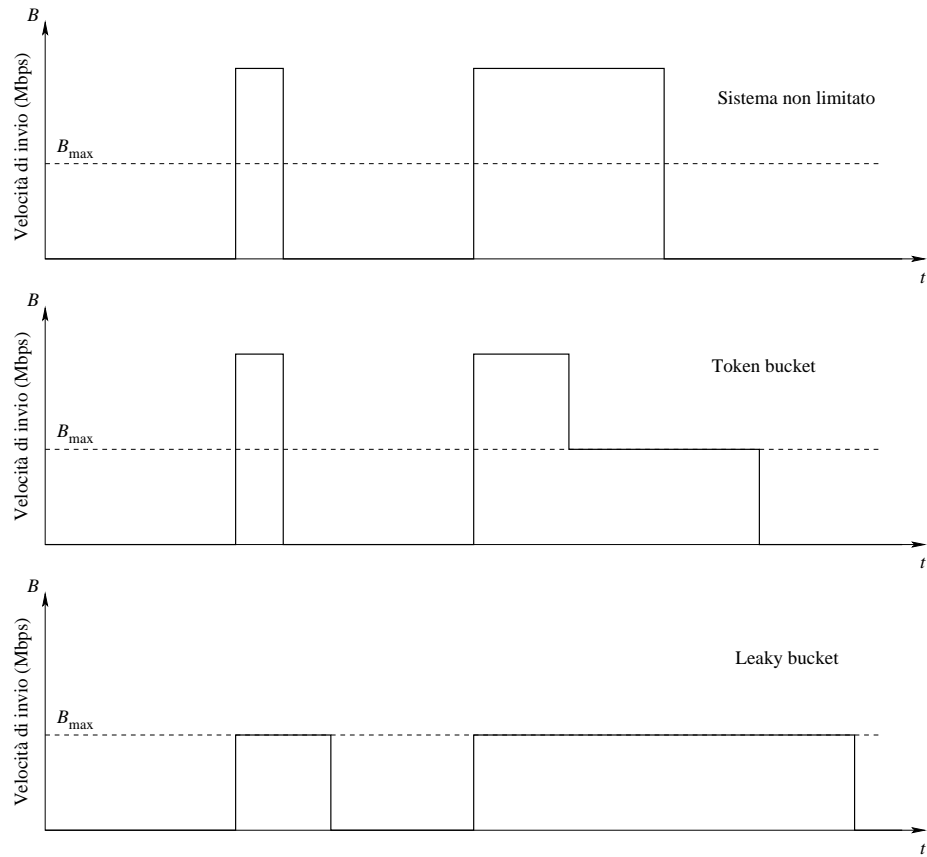


Figura 5: Confronto fra i comportamenti dei due algoritmi

al grafico in basso (rappresentante la quantità di dati trasmessi) è uguale a quella del grafico in alto.

Per quanto riguarda il grafico intermedio di figura 5, notiamo che l'algoritmo Token Bucket ha accumulato abbastanza gettoni da poter gestire la prima raffica senza necessità di limitarla. Per la seconda raffica i gettoni bastano solo fino a un certo momento; in seguito, la velocità di trasmissione eguaglia la velocità di generazione dei gettoni e si prolunga nel tempo.