CO323
Computer Communication Networks II
Lab 03: Token Bucket Algorithm

Gihan Chanaka Jayatilaka
E/14/158

05-04-2018

---

1 Task 01

```
$ javac MainProgram.java
$ java MainProgram TTL totPKT PKTps ran Qcap BKTcap TKNps
```

TTL	The time to live for the experiment in seconds.
totPKT	The total number of packets to be generated by the packet generator.
PKTps	The number of packets to be generated per second
ran	True or False for the packet generation speed to be random. Only the non random case is implemented.
Qcap	The maximum capacity of the queue at the receiving interface of the router.
BKTcap	The maximum token capacity of the bucket.
TKNps	The number of tokens generated per second.

1.1 Example 01

1.1.1 Input

```
$javac MainProgram.java
java MainProgram 10 1000 100 False 20 20 100
```

1.1.2 Output

```
********************
Finished setting up.
Starting clock
********************
0:0:10   PKT-GEN :Created packet 0
0:0:10   QUEUE   :Enqueud packet 0
0:0:10   TKN-GEN :Generated a token. Total tokens in bucket=0
0:0:10   TKN-BKT :Added to bucket. Now capacity=1
0:0:10   TKN-BKT :Used a token. Total=1
0:0:10   ROUTER  :Forwarded the packet 0



0:10:0   TKN-BKT :Used a token. Total=1
0:10:0   ROUTER  :Forwarded the packet 999
********************
End of experiment
********************
0:10:0   PKT-GEN :TOTAL PACKETS GENERATED = 1000
0:10:0   ROUTER  :Forwarded total packets = 1000
```

2 Task 02

The experiment is done on **1 kb (kilobit)** packets. Since we need to limit the outflow of packets to **40kbps** we should make sure that not more than $\frac{40kbs^{-1}}{1kb} = 40s^{-1}$ packets are forwarded from the router.

Consider a case where,
No of tokens per seconds is **TKNps** and the bucket holds **BKTcap** tokens. If the router has not received any packets in a while and the bucket is full we have the highest forwarding rate in case of a burst. Then,

$$\begin{aligned} \text{Rate of data transfer} &= TKNps + BKTcap \\ 40kbs^{-1} &= TKNps + BKTcap \end{aligned}$$

2.1 Example

We can choose,

$$\begin{aligned} TKNps &= 10 \\ BKTcap &= 30 \end{aligned}$$

2.1.1 Input

```
$ javac MainProgram.java
$ java MainProgram 50 1000 100 False 20 30 10
```

3 Task 03

3.1 The problem

The purpose of the token bucket algorithm is to give an advantage for burst traffic conditions.
The tokens are collected in idle times. Having more tokens is an indication of the router being idle over a long period of time. In such a scenario, when a burst of traffic happens it is acceptable to forward the packets faster because if we average the effect of the particular router over time, it is not congesting the network.
This also helps to reduce the packets dropped in the queues in time of burst traffic.

But the problem with token bucket algorithm is that it generates tokens in a constant speed and the bucket has a fixed capacity. Consider the following cases,

- The router is idle for 10s and a burst occurs
- The router is idle for 10min and a burst occurs

There might be no difference in how the token bucket algorithm handles these cases if the bucket is filled before 10s idle time. But intuitively, if the bursts occur once every 10min they deserve more priority than the once that occur every 10s.

3.2 Proposed solution

The naive solution seems to be increasing the bucket size so that more tokens will be collected in 10min than 10s. But we cannot go on increasing the bucket capacity indefinitely.

A non linear token generation rate is a viable solution. In case of an idle time the first few tokens will be generated fast but then the generation speed will decay when the number of tokens in the bucket increases. This solution can also be implemented easily by introducing a probability for a particular token to be added into the bucket.

Consider a situation where the bucket has i tokens, and the $(i+1)^{th}$ token t_{i+1} is generated and being pushed to the bucket. The probability of it being added to the bucket $p(t_{i+1})$ should be given by

$$p(t_{i+1}) = e^{-ki}$$

$$k > 0$$

i is NOT the imaginary number

3.2.1 Choosing parameters

For this to work, k should be chosen.
Since our program takes $b \leftarrow BKTcap$ as an input it could be used to determine k
Whatever the k value we choose,for the 0^{th} packet there is 1.00 probability as

$$p(t_0) = e^{k \times 0} = 1.00$$

It is suitable to choose k in a way that

$$p(t_{b-1}) > 0.75$$

Here 0.75 is a handpick threshold value. This makes sense because we can expect the tokens to go into the bucket by at least 0.75 probability until it reaches the input bucket capacity (BKTcap).

$$p(t_{b-1}) = 0.75$$
$$e^{-k(b-1)} = 0.75$$
$$\underline{\underline{k = \frac{ln(0.75)}{-(b-1)}}}$$

3.3 Implementation

The implementation is found in the **modified/TokenBucket.java**. To run it,

```
$ mkdir backup
$ mv TokenBucket.java backup/TokenBucket.code
$ mv modified/TokenBucket.code TokenBucket.java
$ rm *.class
$ javac MainProgram.java
$ java MainProgram 50 1000 100 False 20 30 10
```

To revert back to the original,

```
$ mv backup/TokenBucket.code TokenBucket.java
```