# SECURING DATA IN COMMUNICATION NETWORKS

**[1]P. Anitha, [2]Dr.C.Chandrasekar,**

**[1]Assistant Professor / MCA, KSR College of Engineering, Thiruchengode**

**Email-Id : psp03ster@gmail.com**

**[2]Director / MCA, KSR College of Engineering, Thiruchengode**

***Abstract***

*The main objective of congestion control is to keep the system running pretty close to its rated capacity, even when faced with extreme overload. This is achieved by restricting users that are allowed service. In this paper we perform a "Securing Data In Communication Networks" has been developed for handling congestion in the network. The fundamental philosophy behind the internet is expressed by the scalability argument: no protocol, mechanism, or service should be introduced into the Internet Perhaps the best example of the internet philosophy is TCP congestion control, Which is implemented primarily through algorithms operating at end systems. As a result of its strict adherence to end to end congestion control, the current internet suffers from main maladies: congestion collapse from undelivered packets. End-to-end congestion control algorithms alone, however, are unable to prevent the congestion collapse and unfairness created by applications that are unresponsive to network congestion. To address these maladies, the propose and investigate a novel congestion avoidance mechanism called network border patrol (NBP). NBP entails the exchange of feedback between routers at the borders of a network in order to detect and restrict unresponsive traffic flows before they enter the network, thereby preventing congestion within the network.*

## 1. Introduction

The fundamental philosophy behind the Internet is expressed by the scalability argument: no protocol, mechanism, or service should be introduced into the Internet if it does not scale well. A key corollary to the scalability argument is the end-to-end argument: to maintain scalability, algorithmic complexity should be pushed to the edges of the network whenever possible. Perhaps the best example of the Internet philosophy is TCP congestion control, which is implemented primarily through algorithms operating at end systems. Unfortunately, TCP congestion control also illustrates some of the shortcomings of the end-to-end argument. As a result of its strict adherence to end-to-end congestion control, the current Internet suffers from two maladies: congestion collapse from undelivered packets, and unfair allocations of bandwidth between competing traffic flows.

The first malady congestion collapse from undelivered packets - arises when bandwidth is continually consumed by packets that are dropped before reaching their ultimate destinations. Nagle assigned the term "congestion collapse" in 1984 to describe a network that remains in a stable congested state. At that time, the primary cause of congestion collapse was inefficient use of retransmission timers by TCP sources, which led to the unnecessary retransmission of delayed packets.

The second malady unfair bandwidth allocation to competing network flows - arises in the Internet for a variety of reasons, one of which is the existence of applications that do not respond properly to congestion. Adaptive applications (e.g., TCP-based applications) that respond to congestion by rapidly reducing their transmission rates are likely to receive unfairly small bandwidth allocations when competing with unresponsive applications. The Internet protocols themselves can also introduce unfairness. The TCP algorithm, for instance, inherently causes each TCP flow to receive a bandwidth that is inversely proportional to its round-trip time. Hence, TCP connections with short round-trip times may receive unfairly large allocations of network bandwidth when compared to connections with longer round-trip times. The impact of emerging streaming media traffic on traditional data traffic is of growing concern in the Internet community. Streaming media traffic is unresponsive to the congestion in a network, and it can aggravate congestion collapse and unfair bandwidth allocation.

## 2. Related Work

The maladies of congestion collapse from undelivered packets and of unfair bandwidth allocations have not gone unrecognized. Some have argued that there are social incentives for Multimedia applications to be friendly to the Network, since an application would not want to be held responsible for throughput degradation in the Internet. Nevertheless, unresponsive UDP flows are becoming disturbingly frequent in the Internet, and they are an example that the Internet control congestion or to operate fairly.

The output ports of ingress routers are also enhanced in NBP. Each output port contains a flow classifier, per-flow traffic shapers (e.g**., leaky buckets**), here I am going to use T**oken bucket** algorithm a feedback controller, and a rate controller (see Fig. 4). The flow classifier classifies packets into flows, and the traffic shapers limit the rates at which packets from individual flows enter the network. The feedback controller receives backward feedback packets returning from egress routers and passes their contents to the rate controller. It also generates forward feedback

packets that are transmitted to the network's egress routers. To prevent congestion collapse, the rate controller adjusts traffic shaper parameters according to a TCP-like rate-control algorithm, and the rate-control algorithm used in NBP is described later in this section.

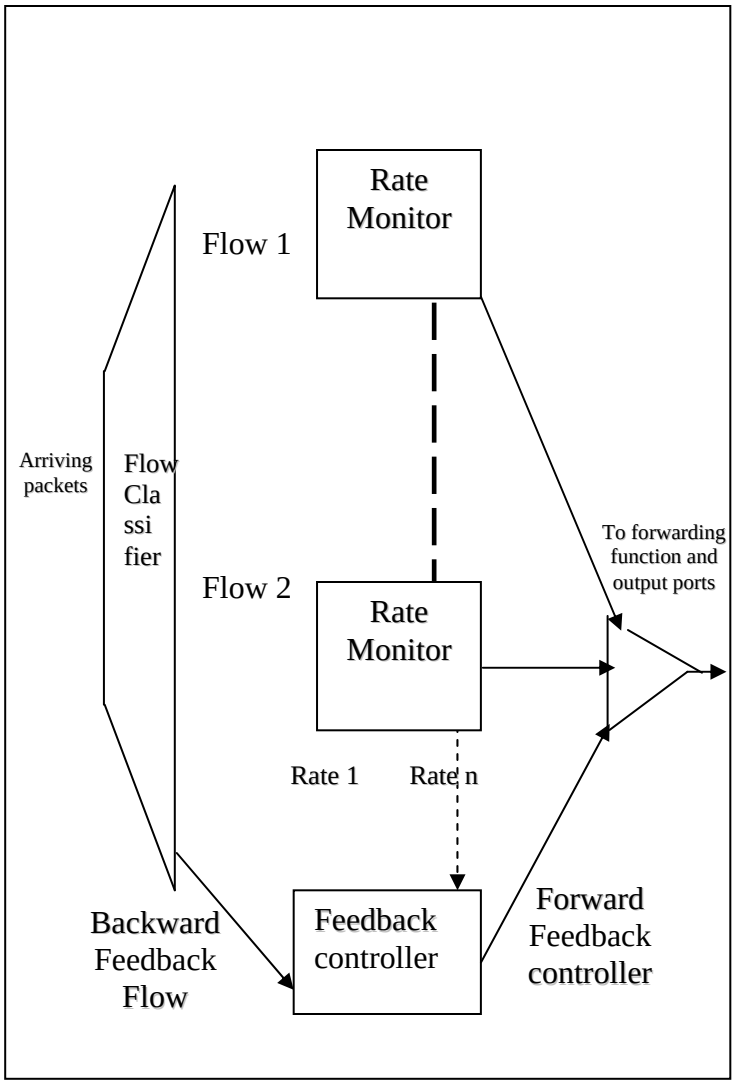


**Fig: 1 Input Port of an NBP egress router**

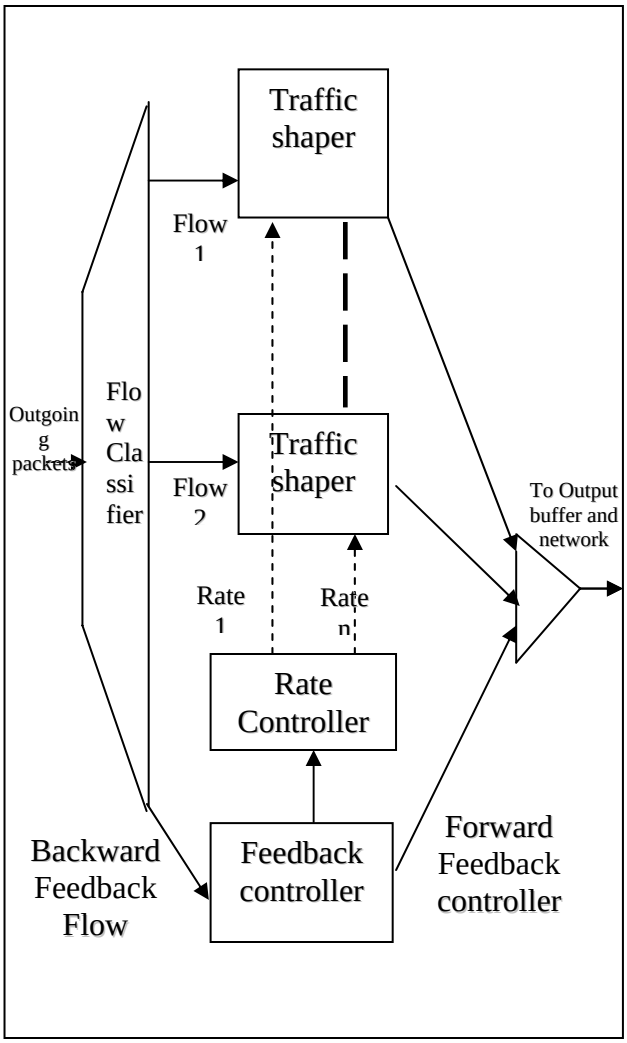


**Fig: 2 Output Port of an NBP ingress router**

**3.Token Bucket implementation**

The implementation of the basic token bucket algorithm is just a variable that counts tokens. The counter is incremented by one every ΔT and decremented by one whenever a packet is sent. When the counter hits zero, no packets may be sent. In the byte-count variant, the counter is incremented by k bytes every ΔT and decremented by the length of each packet sent.

There are two versions, One packet per token, and One byte per token. Implementation is quite simple: the counter counts tokens: increment by one at every $\Delta T$ decrement the counter by one when a packet is sent, Stop transmission when the counter reaches 0.

Input: 25 MB/sec for 40 msec. Leaky bucket output rate: 2BM/sec; Capacity of the bucket: 1MB Draw the traffic trace at the output of the leaky bucket.

Q2. Token Bucket Capacity: 25MB/sec Input burst: 25 MB/sec for 40 msec. Capacity of the bucket: 500 KB = 0.5MB Token arrival rate $\rho$ = 2 MB/sec. Draw the traffic trace at the output of the token bucket.

The Token Bucket Algorithm provides a different kind of traffic shaping than that of the leaky bucket algorithm. The leaky bucket algorithm does not allow idle

hosts to save up permission to send large bursts later. The token bucket algorithm does allow saving, up to the maximum size of the bucket, *n*. This property means that bursts of up to *n* packets can be sent at once, allowing some burstiness in the output stream and giving faster response to sudden bursts of input. The Token Bucket Algorithm responds to the burst size and speeds up the output when large bursts come in. Tokens are generated at every $\Delta T$. Hosts can save tokens for later use. Token bucket throws away tokens not packets. Implementation is quite simple:

The counter counts tokens: increment by one at every $\Delta T$. Decrement the counter by one when a packet is sent. Stop transmission when the counter reaches 0.

The new token are adding to the bucket at rate of r tokens/sec, the maximum token can be accumulated is b bytes. If the bucket is full, the incoming tokens will be thrown away. The Token Bucket (TB) profile contains three parameters are Average Rate, Peak Rate, Burst Size.

**3.1 Token Bucket In Differentiated Services**

The operation of the Token Bucket in different services can be defined for as follow :

Arriving packets of L bytes are conforming (immediately processed) if there are at least L tokens in the bucket (one token= one byte). If the current number of accumulated tokens b' is less than the arriving number of packets L, L-b' packets are nonconforming. Packets are allowed to the average rate in bursts up to the burst size, as long as they do not exceed the peak rate, at which point the bucket is drained. If there are no packets to be transmitted, tokens can be accumulated up to size of b. The rest of tokens will be thrown away.

Conforming and non-conforming functions are depends on SLA agreement. If the packets are non-conforming the following actions can be taken. The packet may be thrown away. The packet may be re-marked in a particular way. The packet can be buffered (by inserting a buffer between the inflow and the decision point) and not released until sufficient number of tokens arrive in the bucket. These functions are applied to different classes in different ways.

**3.4 Burst in Token Bucket**

In fact it is slightly complicated to compute the time length, $S$, of output burst:

$$S = C/(M-\rho)$$

$S$ : Burst length; $C$: is the token bucket capacity; $\rho$ : Token arrival rate; $M$: Maximum output rate. To curb the output of token bucket from being too bur sty, a leaky bucket with a leaky rate, $\rho_l$, is put after the token bucket: $\rho < \rho_l < \rho_n$ where $\rho_n$ is the maximum rate of the network.

The algorithm can be conceptually understood as follows:

A token is added to the bucket every 1 / *r* seconds. The bucket can hold at the most *b* tokens. If a token arrives when the bucket is full, it is discarded. When a packet (network layer PDU) of *n* bytes arrives, *n* tokens are removed from the bucket, and the packet is sent to the network. If fewer than *n* tokens are available, no tokens are removed from the bucket, and the packet is considered to be *non-conformant*. The algorithm allows bursts of up to *b* bytes, but over the long run the output of conformant packets is limited to the constant rate, *r*. Non-conformant packets can be treated in various ways:

They may be dropped. They may be enqueued for subsequent transmission when sufficient tokens have accumulated in the bucket. They may be transmitted, but marked as being non-conformant, possibly to be dropped subsequently if the network is overloaded

This document tries to explain basic token bucket model (some document use leaky bucket), its parameters and its usage in diffserv environment. Token bucket represents the Policing function of Traffic Conditioning Block of diffserv. A token bucket flow is defined by (r,b), r denotes the rate at which tokens(credits) are accumulated and b is the depth of the token pool(in bytes).

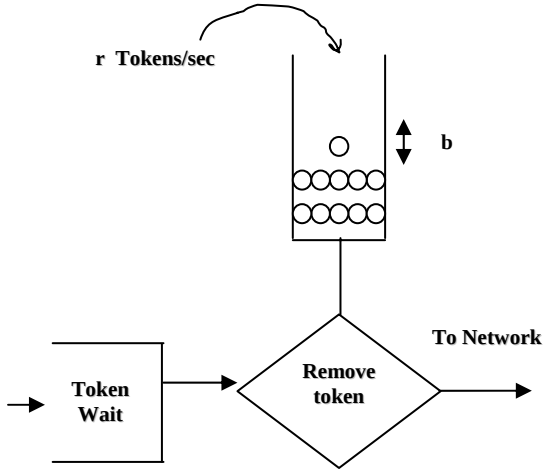


**Fig:4 Token bucket model**

## 4. Conclusion

This research paper successfully entails the exchange of feedback between routers at the borders of a network in order to detect and restrict unresponsive traffic flow before they enter the network, thereby preventing congestion within the network.

## 5. References


[1] S. Floyd and K. Fall, "Promoting the use of end-to-end congestion control in the internet," *IEEE/ACM Trans. Networking*, vol. 7, pp. 458–472, Aug. 1999.

[2] J. Nagle, "Congestion control in IP/TCP Internetworks," Internet Engineering Task Force, RFC 896, Jan. 1984.

[3] V. Jacobson, "Congestion avoidance and control," *ACM Comput. Commun. Rev.*, vol. 18, no. 4, pp. 314–329, Aug. 1988.

[4] (1999, Jan.) Real Broadcast Network

[5] (1999, Jan.) Real Video Technical White Paper. Real Networks Inc. [Online].
Available: http://www.real.com/devzone/library/whitepapers/overview.html

[6] D. Hong, C. Albuquerque, C. Oliveira, and T. Suda, "Evaluating the impact of emerging streaming media applications on TCP/IP performance," *IEEE Commun. Mag.*, vol. 39, no. 4, pp. 76–82, Apr. 2001.