A Congestion Control Algorithm for Improving Internet Traffic Control

Ramachandra.V.Pujeri[1] , Uma.K.Gurav[2] , S.N.Sivanandam[3]

Abstract :

With the development of multiple services on Internet, the dominant Internet congestion control scheme, namely the TCP end-to-end control [16], becomes insufficient. The RED algorithm [14] to detect incipient congestion early and to convey congestion notification to the end hosts, in order to allow them to reduce their transmission rates before queue overflow and packet loss occur using active queue management [15]. On-line mechanism for adaptively changing the parameters according to the observed traffic. This algorithm, called Adaptive RED. This paper also presents completely different architecture, which avoids congestion [1], prevents packet loss, and regulates unresponsive traffic in a uniform fashion. No per-flow state information is maintained in routes while the fairness is guaranteed. The new architecture takes a node-to-node approach advocating close coordination among the core router, the edge router, and the host. Employing both router and host mechanisms, it has three main elements, the generic congestion control protocol (GCCP), the early congestion detector, and the rate controller. The revised version of RED, called GCCP-capable RED. Routers and hosts show its advantages over pure end-to-end control. This paper presents one specific algorithm for tailoring RED [14] parameters to the input traffic.

Index Terms: Internet, Congestion control, Node-to-node, Early Congestion Notification (ECN), Packet loss, Unresponsive traffic, GCCP, RED.

1. INTRODUCTION

The main objective of this paper is to avoid congestion and to improve network performance and resource utilization. Random Early Detection (RED) and enhanced RED called Adaptive RED algorithm are compared with the implementation of the RED algorithm under Generic Congestion Control. Architecture called GCCA-capable RED.

In this two methods we are detecting the congestion [8] using Active Queue Management[15]where the parameter for congestion is average queue length. Also the differences between end-to-end and node-to-node congestion control are explained.

In order to enhance the performance of the above algorithms a new method is proposed. In this new method depending on the number of connections active the threshold values of average queue length are changed. This method is implemented under GCCA. The performance of this algorithm is compared with above methods. The main performance indicators considering here are average queue length, throughput, and the number of packets dropped. The higher the throughput higher

[1]Ramachandra.V.Pujeri, Assistant Professor, is with the Department of CSE, PSG College of Technology, Coimbatore, TamilNadu India.
(e-mail: sriramu_vp@yahoo.com).

[2]Uma.K.Gurav, UG Student, is with the Department of CSE, Lokamanya Tilak Institute of Technology, New Mumbai, India.

[3]S.N.Sivanandam, Professor and Head, , is with the Department of CSE, PSG College of Technology, Coimbatore, TamilNadu India.

the performance and the lower the packets dropped higher the performance. The system structure of the GCCA is shown in fig1. Basic idea of GCCA is to control congestion through coordination among network routers and end systems rather than relying on end-to-end control. Control mechanism is de-coupled from particular flows. Thus it is scalable from service to service. Figure 1 illustrates how GCCA works. A host, S, and three routers, R1, R2 and R3, of a sample network are shown.

GCCA is designed to be transparent to network services. For this purpose, in the router the architecture is hidden beneath the operating system interface. Unlike the backpressure link-by-link or hop-by-hop flow control , however, it is not in the link layer but in

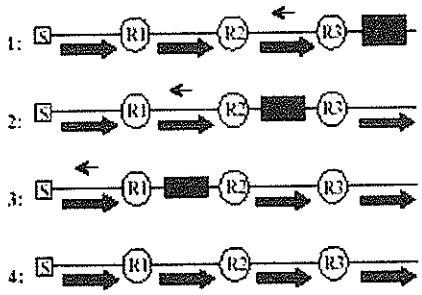

Figure 1: Process of Congestion Alleviation

the network layer. Thus it can work across different types of physical networks. The system structure for GCCA in the router has three components, the GCCP, the congestion detector, and the rate controller. The **GCCP** [4] defines the message format for congestion status information notification. The congestion detector monitors the packet queue and detects incipient congestion. The rate controller controls the output rate of the packet queue. It acts according to the congestion [2] status information received from the downstream nodes. It also relies on some host functions to complete its control. The host part is important because only with it can GCCA manage unresponsive traffic.

A. Congestion control approach and methods

It is important to avoid high packet loss rates in the Internet. When a packet is dropped before it reaches its destination, all of the resources it has consumed in transit are wasted. In extreme cases, this situation can lead to congestion collapse. Loss rates are especially high during times of heavy congestion [2]. So, detection of congestion before it occurs avoids above problem. In this section different congestion control algorithms are explained.

1) Active Queue Management

The idea behind active queue management [6] is to detect incipient congestion *early* and to convey congestion notification to the end hosts, in order to allow them to reduce their transmission rates before queue overflow and packet loss occurs. One form of active queue management being proposed by the IETF for deployment in the network is RED (Random Early Detection) [14]. RED maintains an exponentially weighted moving average (EWMA) of the queue length which it uses to detect congestion. RED detects increases in the average queue length and uses it to determine whether or not to drop or ECN-mark a packet.

2) RED Algorithm

One form of active queue management being proposed by the IETF for deployment in the network is RED (Random Early Detection). RED maintains an exponentially weighted moving average (EWMA) of the queue length which it uses to detect congestion. RED detects increases in the average queue length and uses it to determine whether or not to drop or ECN-mark a packet. As the figure shows, when the average queue length exceeds a minimum threshold, packets are randomly dropped or marked with a given probability. The probability that a packet arriving at the RED queue is either dropped or marked depends on, among other things,

the average queue length and an initial probability parameter.

$$avg_Q = (1-W_Q)avg_Q + W_Q Q$$

$$P_d = \begin{cases} 0 & avg_Q < \min_{th} \\ p_{\max}\dfrac{avg_Q - \min_{th}}{\max_{th} - \min_{th}} & \min_{th} \le avg_Q < \max_{th} \\ 1 & \max_{th} \le avg_Q \end{cases}$$

As Figure 2 shows, the calculated marking/dropping probability is a linear function of the average queue length. The probability is 0 when the average queue length is less than or equal to $\min_{th}$ and linearly increases to $\max_p$ when the average queue length approaches a maximum threshold ($\max_{th}$). When the average queue length exceeds $\max_{th}$, all packets are dropped or marked

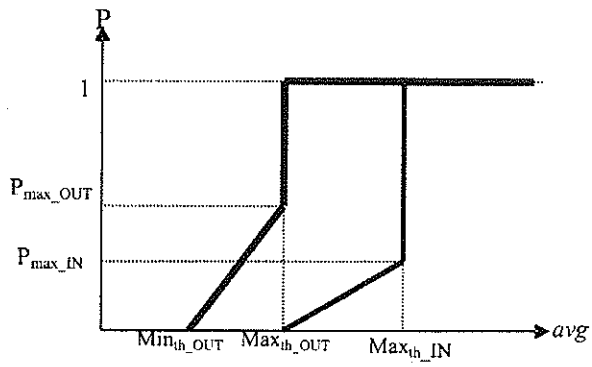

Figure 2: The marking/dropping behavior of RED

3) Adaptive RED

One of the inherent weaknesses of RED and some of the other proposed active queue management schemes [3] is that congestion notification does not directly depend on the number of connections multiplexed over the link. More aggressive early detection is needed when a large number of flows are active in order to avoid packet loss and deterministic congestion notification. Similarly, less aggressive early detection is needed when a small number of flows are active in order to prevent underutilization.

```
Every Q(ave) Update:
        if ( min_th < Q(ave) < max_th)
                status = Between;
        if (Q(ave) < min_th && status != Below)
                status = Below;
                max_p = max_p/α;
        if (Q(ave) < max_th status != Above)
                status = Above;
                max_p = max_p/β;
```

Figure 3: Adaptive RED algorithm

The idea behind this algorithm is to infer whether or not RED should become more or less aggressive by examining the average queue length behavior. If the average queue length continually crosses back and forth over Q(ave), then the early detection mechanism is being too aggressive. If the average queue length continually crosses back and forth over $\max_{th}$, then the early detection mechanism is not aggressive enough. Given the behavior of the average queue length, the algorithm [10] then adjusts its value of $\max_p$ accordingly. For this algorithm, $\max_p$ is simply scaled by constant factors of ± and ² depending on which threshold it crosses.

4) GCCA capable-RED

The congestion detector need detect the liability of congestion earlier. It warns upstream nodes of the danger before it really happens. In addition to early detection, the congestion detector should also guarantee the fairness between flows. At the same time, per-flow computation should be avoided. RED is designed as a router congestion avoidance mechanism to work with a transport-layer congestion control protocol such as TCP. It detects incipient congestion by calculating the average queue size, and notifies connections of the danger by marking or dropping packets with a certain probability. It does not make per-flow computation. Another advantage of this is

that the probability that a particular connection is marked or dropped is proportional to that connection's share of the bandwidth through that router. Nevertheless, RED is only based on average enough to prevent packet loss. It is possible that the instantaneous queue length increases, which makes it not enough to prevent packet loss. It is possible that the instantaneous queue length increases very fast because of a sudden burst while the average queue length keeps small.

```
Name: GCCP-capable RED
for each packet arrival
calculate the instantaneous queue increment inc
if inc > s T
send a GCCP CONGESTION message
set status as CONGESTION
return
calculate the average queue size avg
if minth <= avg < maxth
calculate probability pa
with probability p :
send a GCCP CONGESTION message
set status as CONGESTION
else if maxth <= avg
send a GCCP CONGESTION message
set status as CONGESTION
else if avg <= minth
if status is CONGESTION
send a GCCP NORMAL message
set status as NORMAL
return
```

Figure 4:GCCA-capable RED algorithm

So we add another dimension for congestion detection, that is, the increasing speed of queue size. At every instant, we calculate the change of the instantaneous queue size. If it is above a threshold, s_T, a GCCP congestion message is sent to involved upstream nodes. Otherwise, normal RED is followed. Whenever a packet is to be marked or dropped according to RED, we send a GCCP CONGESTION message to the direct upstream node where the packet came from. The packet itself is queued just like others rather than dropped. Packet loss is thus avoided. The revised version of RED, called GCCP-capable RED, is given in figure 4. In the algorithm we use $(avg <= min_{th})$ as the criteria for the NORMAL status. The GCCA control loop will try to keep the average queue size below min_{th} all along. Therefore, in order to maintain the link utilization at an acceptably high level, min_{th} should be larger than that in RED. On the other hand, $(min_{th}$- $min_{th})$ can be smaller. This is because of two reasons. First, the max_{th} determines the average queue delay to some extent. A less max_{th} makes a shorter queue delay. Second, the one-hop feedback control loop in GCCA is much shorter than the roundtrip loop for end-to-end control. A large $(max_{th}$- $min_{th})$ is not necessary.

2. DESIGNING ISSUES

A. Proposed algorithm

Active queue management and end host congestion control algorithms can be designed to effectively eliminate packet loss in congested networks. In particular, an adaptive RED mechanism [14] that is cognizant of the number of active connections and the use of bandwidth-based linear increases can both provide significant benefits in terms of decreasing packet loss and increasing network utilization. There are several ways in which these mechanisms can be extended. We present one specific algorithm for tailoring RED parameters to the input traffic. There are many other potential alternatives for doing so. For example, the RED queue could actually keep track of the number of active connections and change its aggressiveness accordingly.

Another mechanism would be to have the RED queue infer the number of connections present by the rate at which the average queue length changes and have it then

adapt its parameters accordingly. It may also be possible to adapt other RED parameters instead of Q(avg) to optimize performance. For example, one could adaptively change inter-packet drop probabilities or the RED threshold values depending on the input traffic[1]. Finally, in extremely congested networks for which setting max_p to 1 is not sufficient, it may be possible to make the marking/dropping even more aggressive by having the marking probability change as a non-linear function of the average queue length. When a max_p setting of 1 is insufficient, the marking function can assume a non-linear shape allowing the marking to become even more aggressive. So, in this algorithm every time the load of the network is calculated. The load is inferred from the average queue length at the routers. Depending on the load the threshold values are changed. After calculating the threshold values Adaptive-RED algorithm under GCCA is used. The GCCA is used b because of high response and node-to-node control. The detailed algorithm is shown in figure 5.

RED algorithm is implemented in end-to-end congestion control and on node-to-node congestion control, which is under GCCA. The propped algorithm is implemented under GCCA. All these algorithms are simulated with the following network scenario as shown in figure 6.. The simulation is done in C. Figure 7 shows the average queue lengths of the three algorithms. Figure 8 shows the throughput of the three algorithms. Among all the three the proposed algorithm is showing high throughput. Figure 9 shows comparisons of number of packets dropped due to congestion. The GCCA-capable RED and the proposed algorithm are showing great reduction in the number of packets dropped. But, when proposed algorithm is compared with GCCA-capable RED, the GCCA-capable RED is leading to under utilization of resources.

```
Name: GCCP-capable RED
for each packet arrival
calculate the instantaneous queue increment inc
if inc > s T
send a GCCP CONGESTION message
set status as CONGESTION
return
calculate the average queue size avg
if minth <= avg < maxth
calculate probability pa
with probability p :
send a GCCP CONGESTION message
set status as CONGESTION
else if maxth <= avg
send a GCCP CONGESTION message
set status as CONGESTION
else if avg <= minth
if status is CONGESTION
send a GCCP NORMAL message
set status as NORMAL
return
```

Figure 6. Proposed Algorithm

3. DISCUSSION AND RESULTS

It is focused on solving two extremely important challenges to today's Internet: supporting An explosion in the number of users and supporting a myriad of new applications which require more out of the network than the best-effort service that the Internet currently provides. To this end, a number of modifications to the basic congestion control and queue management algorithms of the Internet have been examined.

More specifically, it has shown that even with ECN, current active queue management mechanisms such as RED are ineffective because they are not sensitive to the level of congestion in the network. To address this problem, an adaptive modification to RED which allows it to manage congestion more effectively has been developed, implemented and evaluated. In addition, it

has demonstrated the inherent weakness in all current active queue management mechanisms in that they rely on queue lengths to do congestion control.

So, a generic node-to-node congestion control architecture for Internet is presented. It avoids congestion, prevents packet loss, and manages unresponsive traffic in a uniform fashion. Main techniques are described, namely, the GCCP, the early congestion detection, and the light rate control. The scheme combines the node-to-node control with the end-to-end control to reach global optimal performance.

GCCA is for the time scale form one to several hops. It is used to prevent packet loss. The end-to-end control is for the time scale from one to several round trips. It reduces the source rates and eliminates the danger of congestion completely.

So combination of GCCA and ECN [3] is a good way for global congestion control. When incipient congestion is detected, it is notified along both backward and forward paths. In the scheme, both congestion detection and rate control are executed at queue level, which avoid maintaining per-flow state information in the router. This stateless processing consumes very small overhead, and brings simplicity and efficiency in network control. Compared with the TCP end-to-end control, it speeds up response to congestion. As a network layer control scheme, it can work across different kinds of physical networks, and free applications from taking care of rate adaptation. The architecture is scalable to new services and compatible with existing protocols. To have the maximum network utilization in addition to congestion avoidance a new algorithm is proposed. It is providing high throughput compared to other methods. That is because of dynamic changes of queue threshold values depending on the load of the network.

4. IMPLEMENTATION AND RESULTS

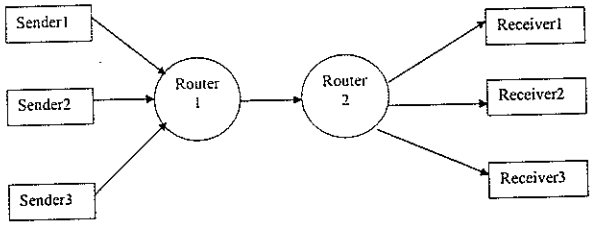

Figure6. Experimental Network Scenario

Maximum sending rates of senders:

Max. sending rate of sender1 = 32MBps
Max. sending rate of sender2 = 32MBps
Max. sending rate of sender3 = 32MBps

Maximum sending rates of Routers:

Max. sending rate of router1 = 40MBps
Max. sending rate of router2 = 40MBps

Series1———→ Adaptive RED
Series2 ——-→ GCCA-RED

Maximum Queue size at Routers:

Max. Queue size at router1 = 30MB
Max. Queue size at router2 = 30MB

The graphs for the above three algorithms with respect to average queue length, throughput, and number of packets dropped are shown in fig.

Series3 ——-→ Proposed Algorithm

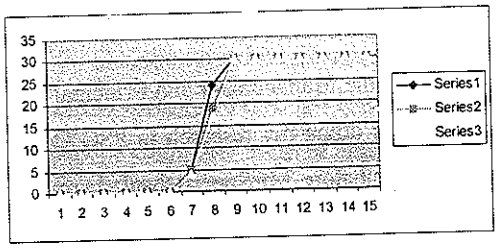

Figure 7.Comparison of average Queue length

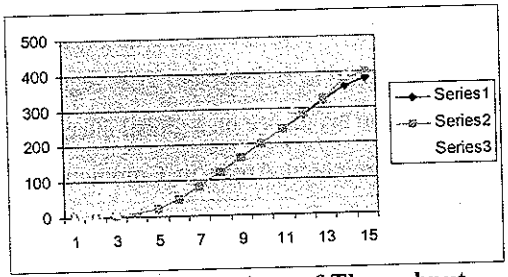

Figure 8.Comparison of Throughput