## A Report On

# SACD : An Active Queue Management Strategy for Congestion Control

**Submitted in partial fulfillment of the course:**

**CS F366 - LAB ORIENTED PROJECT**


**BIRLA INSTITUTE OF TECHNOLOGY & SCIENCE, PILANI**

**November 2019**

Submitted by: -

**Aviral Sethi : 2016B3A70532P**

**Sanjeet Malhotra : 2016B4A70601P**

Submitted to: -

**Dr. Virendra S. Shekhawat**

*Assistant Professor,*
*Department of*
*Computer Science & Info. Systems*

# TABLE OF CONTENTS

# Abstract

Queue management and congestion control are very important to the robustness and fairness of the Internet. Through this report we propose a new queue management algorithm, termed as **SACD** for active queue management. It is based on the well-known RED algorithm and two recently proposed algorithms namely CHOKED and SAC algorithm.

The algorithm is designed considering the powers of both CHOKeD and SAC and combining them in a way to achieve better control over the number of flows , partitions of the queue and CHOKe packet picking strategy.

The results show the superiority of the algorithm over the standard algorithms in some of the scenarios in terms of overall throughput and safeguarding tcp against udp sources. However, extensive testing and development of the algorithm and its parameters especially ‘alpha’ is required to confirm the validity of the proposed algorithm.

# Literature Review

- **Introduction:**

- Congestion management is a fundamental problem in networking because the way to achieve cost effective and scalable designs is by sharing the network infrastructure.
- In general it is the shared resource which needs to be managed. With this, a question arises as to what shared resource is in our interest here. Let us look at it through the perspective of the router.

- **Routers and Congestion Control:**

- The model at any router is for each packet to be processed and forwarded to one of the several possible output links at a rate determined by the rated bandwidth of the said link. In the meantime, more packets arrive at the router possibly from different flows. The router tries to accommodate these packets in its buffers, but if there isn't enough space some of the packets are dropped and hence they are lost.
- So there are majorly two resources at our disposal:
    - **Link Bandwidth** : How to maintain a fair allocation to different arriving flows at different rates.
    - **Queue Space** : When to drop a packet, when the queue is full or sometime before. And which packet should be dropped if the queue is full, the incoming one, the last one or any random packet.
- Having a proper queue management strategy thus defines as to how much delay and loss packets over the network will suffer.

- **Buffer Management and Active Queue Management:**

- Buffer management schemes at routers decide as when a packet should be dropped / marked (in case of ECN enabled packets) and which packet should be dropped for that instance. The simplest of the strategies is **Drop Tail** which accommodates packets until the queue is full, after which all incoming packets are dropped until queue is available again.
- This has several drawbacks as it does not signal congestion early enough which leads to bursts of packets being dropped and also leads to a high average queue size which

propagates to higher queueing delays that might not be reasonable. Nevertheless, still a lot of the internet today is governed by this policy owing to its simplicity in incorporation.

Nonetheless, many queueing algorithms have been proposed over the years which have helped in much better congestion control and hence improved performance. We will discuss a few of them in the next section.

## 1. RED

- This is the most widely adopted and successful techniques for router's active queue management is Random Early Detection (RED), recommended by the Internet Research Task Force (IRTF).RED solves full queue and lockout, drawbacks of Drop Tail (Braden et al, 1998). RED calculates exponentially weighted moving average queue size. In RED queue length is marked with one of the 2 thresholds: minimum and maximum threshold. If the average queue size is less than the minimum threshold, RED does not mark or drop any arrived packet. But if the average queue size is greater than the maximum threshold, RED drops all arrived packets. If the average queue size is between maximum and minimum thresholds, RED marks arriving packets with certain probability. This probability helps to control the average queue size through marking of packets (Floyd and Jacobson, 1993).

  **Limitations:**

- TCP gets starved due to the presence of unresponsive flows as RED has no mechanism to penalize UDP/unresponsive flows.

## 2. BLUE

- BLUE is an active queue management algorithm to manage congestion control by monitoring packet loss and link utilization history instead of queue utilization. BLUE keeps a single probability, Pm, to mark (or drop) packets. If the queue is regularly dropping packets due to buffer overflow, BLUE increases Pm by amount d1 as explained below, thus increasing the rate at which it sends back congestion notification or drops packets. Conversely, if the queue is empty or if the link is idle, BLUE reduces its marking probability. This effectively allows BLUE to "learn" the accurate rate it needs to send back congestion notification or dropping packets.

- The three typical parameters of BLUE are d1, d2, and freeze_time where d1 determines the amount by which Pm is increased when the queue overflows, while d2 determines the amount by which Pm is decreased when the link is idle. Freeze_time is an important parameter that determines the minimum time interval between two successive updates of Pm. This allows the changes in the marking probability to take effect before the value is updated again. Based on those parameters. The basic blue algorithms can be summarized as follows:

| **Upon link idle event:** | **Upon packet loss event:** |
|---|---|
| if ((now-last_update)>freeze_time)<br>Pm = Pm-d2;<br>Last_update = now; | if ((now–last_updatte)>freeze_time)<br>Pm = Pm+d1;<br>last_update = now; |

**Limitations :**

- Although it guarantees fairness among TCP flows but it does not give fair share to TCP flows if UDP/unresponsive flows are involved. So, not fair when unresponsive flows are involved.

## 2.1 SFB

- This was proposed as a modification to BLUE algorithm in order to safeguard the responsive flows from unresponsive flows. The algorithm goes as follows: SFB maintains NxL accounting bins. The bins are organized in L levels with N bins in every level. Along with this SFB maintains L independent hash functions ,each associated with one level of the accounting bins. Each hash function maps a flow,vis its connection ID(Source address, Destination address, Source port ,Destination port, Protocol) into one of the N accounting bins in that level. The accounting bins are used to keep track of queue consumption statistics of packets belonging to a specificr bin.
- The following figure depicts the complete algorithm:

```
B[l][n]: L × N array of bins(L levels, N bins per level)
Enque()
Calculate hash function values h0,h1,...,hL-1;
Update bins at each level
For i =0 to L-1
If(B[i][hi].QLen> bin_size)
B[i][hi].Pm += delta;
Drop packet;
Else if (B[i][hi].Qlen ==0)
B[i][hi].Pm - = delta;
Pmin = min(B[0][h0].Pm...B[L][hL].Pm);
If(Pmin==1)
Ratelimit();
Else
Mark/drop with probability Pmin;
```

Figure 1: SFB Algorithm

Unlike ,Stochastic Fair Queueing (SFQ) where the hash functions map flows into separate queues SFB uses hash function for accounting purposes only.

- **Limitations :**
- It uses bloom filters to identify a flow to be as UDP and thereby penalize it. But since it is not possible to make an ideal bloom filter with no false positives it also classifies a TCP as UDP in some cases.

## 3. CHOKe

- Pan et al., (2000) proposed a technique called CHOKe (CHOose and Keep for responsive flows, CHOose and Kill for unresponsive flows) to solve RED problems. CHOKe does not keep records or states of unresponsive flow, and it is a completely stateless. Working mechanism of CHOKe is given as follows: it calculates exponential weighted moving average based average queue length as in RED. CHOKe also defines the same thresholds as given in RED, minimum threshold $T_{min}$ and a maximum threshold $T_{max}$ . No packet is dropped if the average queue size $Q_a$ is less than $T_{min}$ and if the average queue size is greater than all packets are dropped. If $T_{min} < Q_a < T_{max}$ , CHOKe draws a packet from the queue randomly and compare it with an arriving packet. If both have the same flow id, it drops both (arrived packet and packet from queue). If flow ID's of both does not match, arrived packet is dropped with RED's defined probability (Pan et al., 2000).

**Limitations :**

- CHOKe (basic CHOKe) cannot handle multiple UDP flows of same or different rates. One of the flow may get starved. It is unfair among UDPs.

## 3.1 Adaptive CHOKe

- Adaptive CHOKe is a variant of basic CHOKe (described above), with the slight variation that instead of picking just one drop candidate for comparison with the incoming packet we pick multiple packets. Number of drop candidates are decided dynamically each time depending upon average queue length. Procedure to calculate average queue length and drop probability is same as basic CHOKe. To determine the number of drop candidates, whole queue is divided into multiple regions and if average queue length lies in r-th region then 2r random packets are chosen as drop candidates. Rest of the dropping strategy is exactly the same as in basic CHOKe.

**Limitations :**

- Adaptive CHOKe can not be considered fully dynamic as the criteria for determining the number of drop candidates does not take into account the number of UDP flows present. Distribution of packets of unresponsive flows is also not taken into account as we uniformly pick so determined packets from queue but UDP packets are present at the rear end of the queue.

## 3.2 SAC

- Jiang et al., (2003) proposed a router based AQM solution for the unfairness problem, named as Self-Adjustable CHOKe (SAC). Jiang et al., (2003) revealed that CHOKe although penalize unresponsive flows, but still it is not fair enough. They describe two shortcomings of CHOKe: one is unfairness among UDP flows and another one, the number of selected candidates for newly arrived packet's comparison. SAC working mechanism is given as: it divides region between 0 and into k number of regions. If queue size is in between maximum and minimum threshold, say in region named , then number of candidates will be selected for comparison and will be dropped if matched. The value of k is described by two parameters P and R. R is the probability that arriving packet is UDP. And P is the probability that arriving packet and randomly packet selected from the queue has same flow id. UDP and TCP flows are treated differently in

the case of SAC, for TCP on congestion notification; SAC behaves like original CHOKe (Jiang et al, 2003).

**Limitations :**

- According to the SAC algorithm, we search for packets with same flow id (in case of a match) as the matching ones from the rear end of the queue and do what was needed to be done with the packets picked from the rear end. This re-searching is an expensive overhead. Moreover, distribution of packets has not been taken into account properly as all partitions are of equal size and number of packets picked from the partition does not capture the distribution of packets in the partition to a complete extent.

## 3.3 CHOKeD

- CHOKeD is a stateless AQM approach as it needs no record keeping for number of flows residing in the queue. CHOKeD is motivated by CHOKe (Pan et al., 2000) which is a much better approach than traditional AQM approaches RED and Drop Tail (Floyd and Jacobson, 1993) to prevent friendly flows like TCP from unfriendly or unresponsive flows like UDP flows. CHOKeD methodology is based on three components:
- 1. Queue regions
- 2. Drop candidate packet's selection from queue, i.e., called dynamic drawing factor,
- 3. Location of drop candidate packets from the queue.

CHOKeD slices the queue into two equal size regions, named as rear and front regions. Firstly, it draws packets from the rear queue region and match with the flow id of . The value of the number of packets(Dr) to be picked is determined by Eq 3.3. If all or any of the packets from the queue has the same flow id, same as of the arriving packet , CHOKeD drops all matched packets and also the arriving packet . In this comparison, if none of candidate packets from the queue has the same id as that of, CHOKeD returns all packets back to the queue and draws the packets from the front region for comparison, before admitting the arriving one. The value of Df is computed using Eq 3.4. Now CHOKeD compares, candidate packets, drawn from the front queue region, with flow id of arriving packet, if any or all packets have same flow id as of the arriving packet , arriving packet and all matched packets are dropped. In the event of no matching at this moment, the candidate packets are restored back to the queue, but the incoming packet may still be dropped with a probability that depends on the degree of congestion.

$$D_i = Round\left(\frac{(Q_c * \sqrt{B})}{(T_{max} - T_{min}) * ln(B)}\right) \quad (3.2)$$

$$D_r = D_i \quad (3.3)$$

$$D_f = Round\left(\frac{D_i}{2}\right) \quad (3.4)$$

Figure 2: CHOKeD Equations

**Limitations :**

- Percentage of UDP packets not considered.
- Distribution considered in a 2-step manner.

## Our Proposed Algorithm: SACD

- **Idea and Motivation**
- Plenty of research has been done in the area of AQM as a solution for the unfairness problem. The main objective of this research is to protect responsive flows from unresponsive flows and allocate fair bandwidth among all flows. Router based AQM solutions must have to be simple in terms of memory and time complexity, because, only stateless AQM systems are encouraged to deploy over the network routers (Adams, 2013). CHOKe (Pan et al., 2000) suggests simple and stateless solution for the unfairness problem. CHOKe gives an innovative idea of drawing single candidate packet from the queue to compare with incoming packets. CHOKe outperforms as compared with traditional AQM schemes like RED and BLUE.

- This project is motivated by two of the CHOKe versions CHOKeD and SAC to penalize unresponsive flows and ensure fairness with minimal memory and time complexity.
- Where SAC gives us a reasonable mechanism to divide the current queue into regions for packet selection, CHOKeD proposed a mechanism to decide as to how many packets are needed to be picked from the front and back region.

- We build our algorithm with the idea that why not divide the selection process among k regions as given by SAC so as to get a much better capture of the distribution of unresponsive flows over the current queue.
- That is give a **k-step** function instead of **2-step** function for distribution of packets , as given by CHOKeD.

## DESCRIPTION

- **Parameters :**
  - **p : as in SAC**
  - **r : as in SAC**
  - **wp , wr , wk : as in SAC**
  - **$Q_c$ : Current queue length**
  - **q_lim : Buffer size/max queue size**
  - **j : the partition of the queue in which we are present currently**
  - **Dja == Dka : number of packets given as** $( Qc * \sqrt{q\_lim} * (1-1/2^{(j/2)})^{-1} )/ 2*(th_{max}-th_{min})*\ln(q\_lim)$
  - **alpha : Number of partitions to be considered at once**

- One of the major parameters of our algorithm is 'alpha' which tells us as to how many regions to be picked at once before considering to break out of any further picking if a match is found.
- The basic algorithm is as follows :
- **If arriving packet is UDP:**
  - Calculate the current partition in which we are currently using the value of 'K' as devised from the SAC algorithm with a few tweeks to incorporate in our algorithm and store it in 'j'.
  - Calculate Dja as given above. Select Dja packets from 'jth' partition and then as we travel to next partition make Dja = Dja/2;
  - Once 'alpha' partitions are covered and at least one match is found, drop the incoming packet and all the matched packets and break out of the loop.

    - If no match was found continue with a new set of alpha partitions until we reach the front of the queue i.e. to the 0th partition.
    - If still there is no match then drop the packet with probability as devised by red if qavg is less than max threshold else drop the incoming packet..
- **Else :**
    - Continue as in red.

- **Pseudo Code**
    - The appendix at the end gives a detailed pseudocode for our algorithm. It is built with SAC algorithm as a base and incorporating our additional features over it.
    - The code was implemented in NS-2 simulator for its testing and feasibility analysis.

## Simulation and Results

The following three algorithms were tested against the following four test scenarios :

**Algorithms :**

1. **RED**
2. **CHOKe**
3. **SACD**

**Scenarios/ Test Cases :**

The scenarios considered for simulation of our algorithm have been chosen due to the following reasons-

1. UDP = 1 & TCP = 1 is chosen to test whether our algorithm is able to save responsive flows from unresponsive flows.
2. UDP = 2 & TCP = 2 is chosen to test whether our algorithm is able to save responsive flows from unresponsive flows and further to test whether fairness is achieved among flows of same type i.e. among TCP/ UDP .
3. UDP = 2 & TCP = 3 is chosen to again test fairness achieved by our algorithm among responsive and unresponsive flows and to test whether responsive flows

were saved in the case when number of TCP flows is not equal to number of UDP flows.

We considered three cases for the same:

a. **CASE-I** : Both UDP of same rate with same packet size.
b. **CASE-II** : Both UDP of same rate but different packet size **(More Closer To Reality)**
c. **CASE-III :** UDP-1 with rate 1mbps and UDP-2 with rate 6mbps

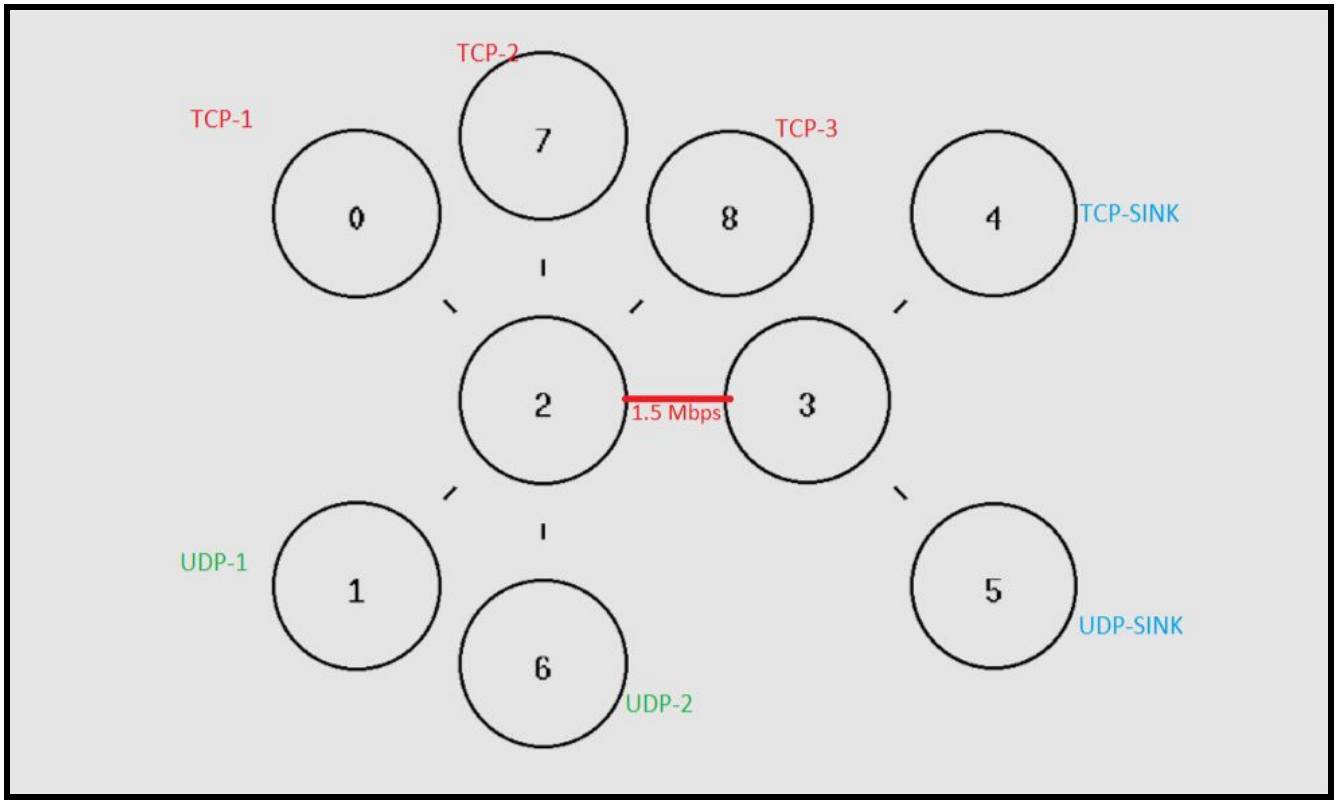


***Figure 3 : Topology used for testing***

**TABLE COMPRISING OUR TESTING SCENARIOS BELOW:**

| S.No | Bottleneck link Bandwidth | Buffer Capacity | No of UDP sources | No of TCP sources | Udp rate same | Alpha |
|---|---|---|---|---|---|---|
| 1 | 1.5 Mbps | 15000 bytes | 1 | 1 | same | j/3 |
| 2 | 1.5 Mbps | 15000 bytes | 2 | 2 | same | j/3 |
| 3.a | 1.5 Mbps | 15000 bytes | 2 | 3 | same | j/4 |
| 3.b | 1.5 Mbps | 15000 bytes | 2 | 3 | Same (diff packet size) | j/4 |
| 3.c | 1.5 Mbps | 15000 bytes | 2 | 3 | Diff (1 and 6) | j/4 |

**Table 1 : Scenarios used for testing**

## Results

Given below are the results for the scenarios as listed above. Explanation and observation are listed after each set of graphs.

- **ALL THROUGHPUT VALUES IN kbps**

# TOTAL THROUGHPUT

1. **UDP:TCP = 1:1**

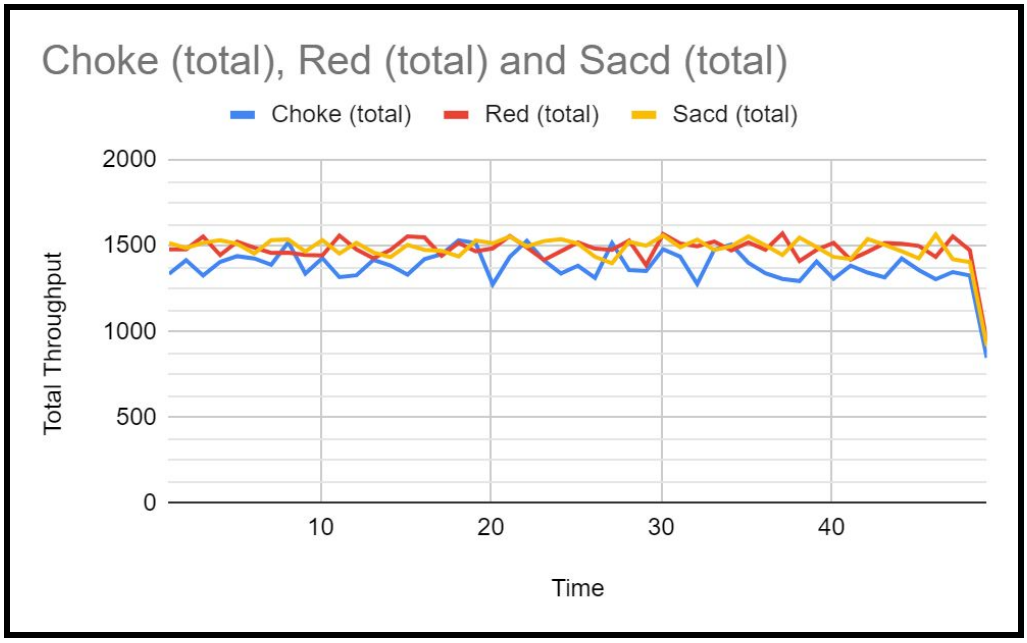


***Figure 4: Total Throughput for 1:1 (in kbps)***

2. **UDP:TCP = 2:2**

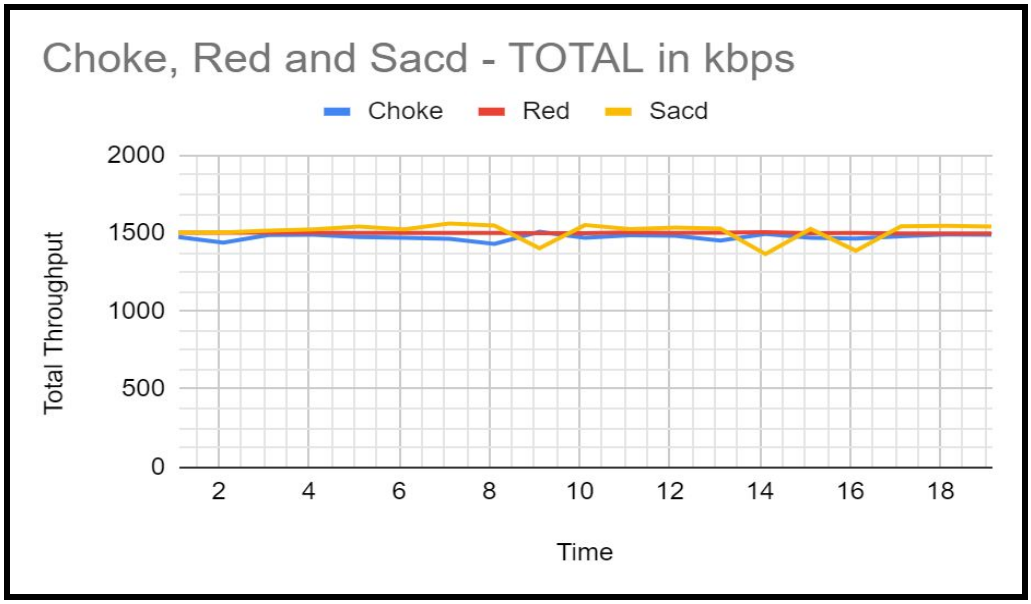


***Figure 5: Total Throughput for 2:2 (in kbps)***

3.

a. **UDP:TCP = 2:3**

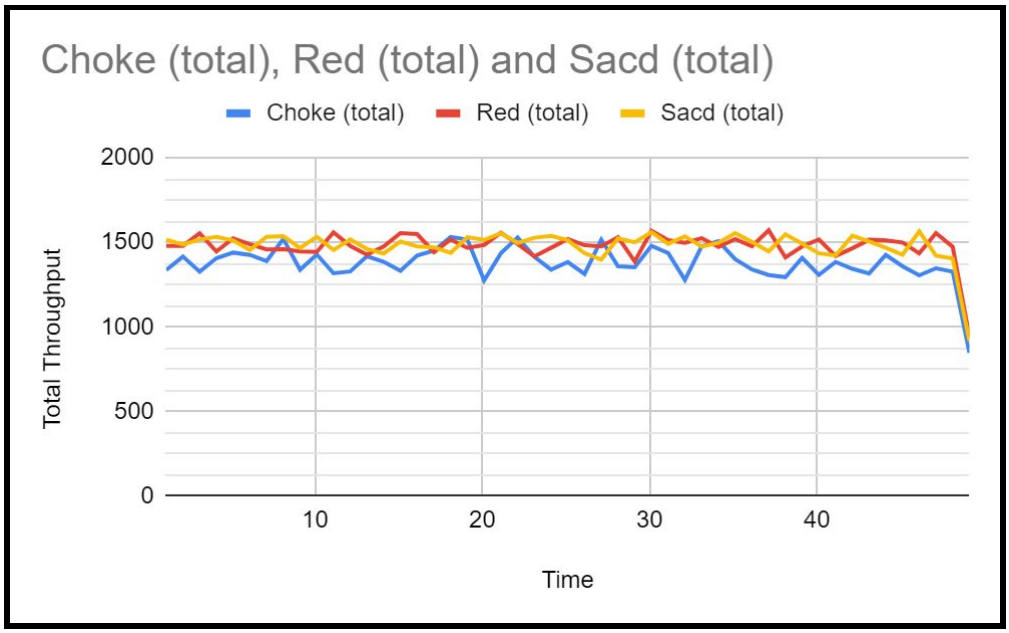


***Figure 6: Total Throughput for 2:3 (in kbps)***

As can be seen from the graphs above the total throughput was slightly larger than both RED and CHOKE or almost equal for our proposed algorithm in all the 3 cases used for testing.

This can be attributed to the fact that our algorithm was able to save the tcp sources to a larger extent and also algorithm was developed with the aim to reduce the incorrect penalization of udp sources. So the overall effect is increased throughput.

# INDIVIDUAL FLOW COMPARISON

## TCP and UDP FLOW COMPARISON IN CASE OF SINGLE UDP

1. **UDP:TCP = 1:1**

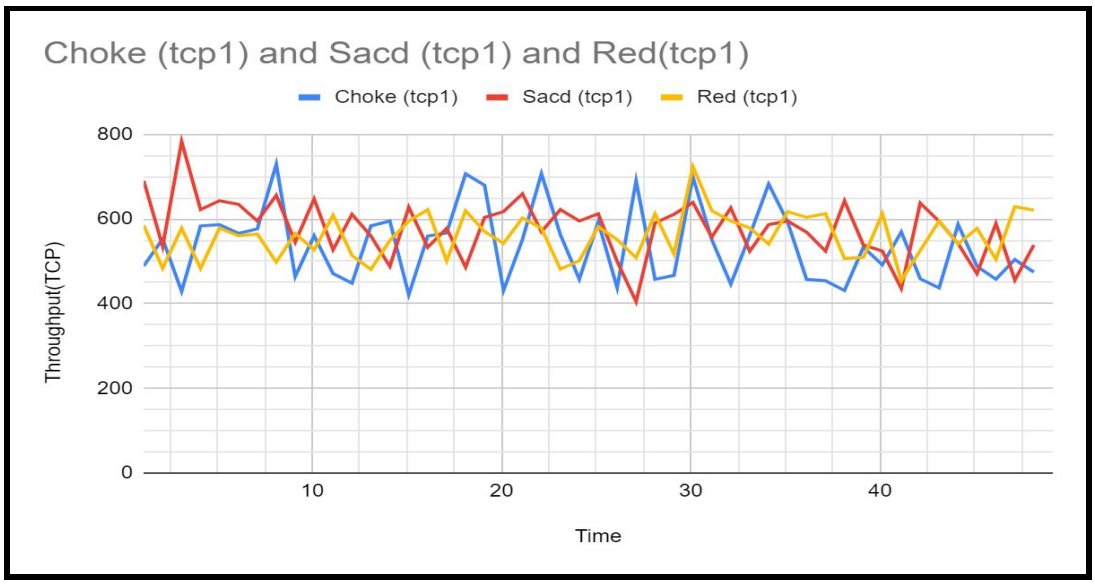


***Figure 7 : TCP throughput***

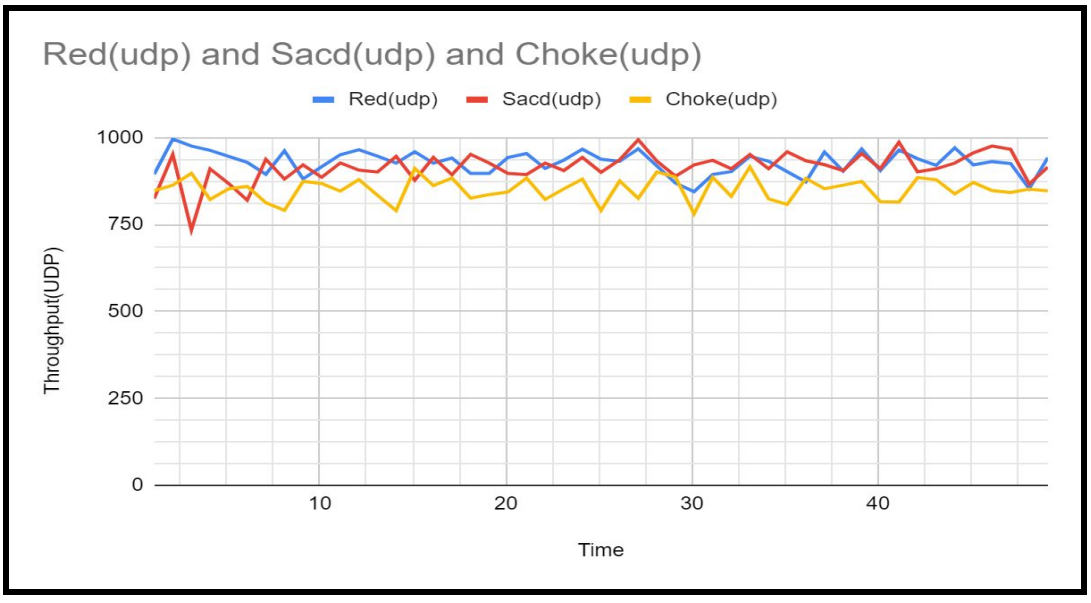


***Figure 8 : UDP throughput***

When a single UDP is present then we know that even CHOKe performs well and thereby our algorithm has a similar performance as with CHOKE.

The following are the average throughput values for the three algorithms:

| FLOW | RED | CHOKE | SACD |
|---|---|---|---|
| **TCP** | 561.139812 | 537.640625 | 577.9792292 |
| **UDP** | 928.377958 | 851.780291 | 915.2600471 |

***Table 2: Average Throughput Values in kbps***

Our algorithm performed better than CHOKe in terms of fairness as it slightly less penalized udp flow and also with a slightly more throughput for tcp than RED and CHOKE.

**Hence, for ideal case i.e. of 1:1 our algorithm performed better than CHOKe and RED.**

## TCP FLOW COMPARISON IN CASE OF MULTIPLE UDP's : Scenario 3.a

**UDP-1 : 1 mbps** **UDP-2 : 1 mbps**

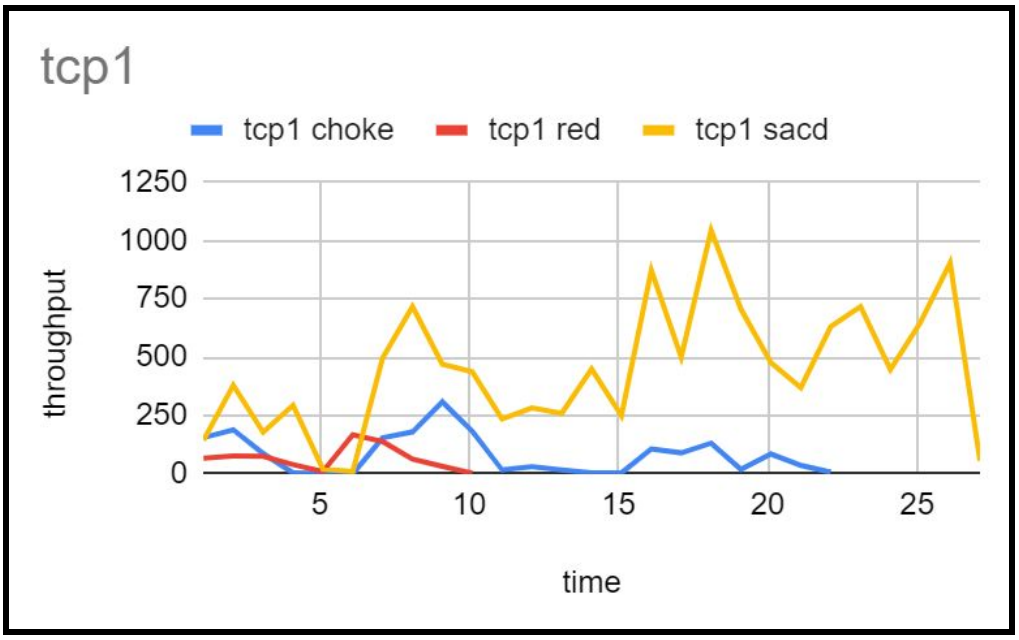


***Figure 9: TCP1 throughput***

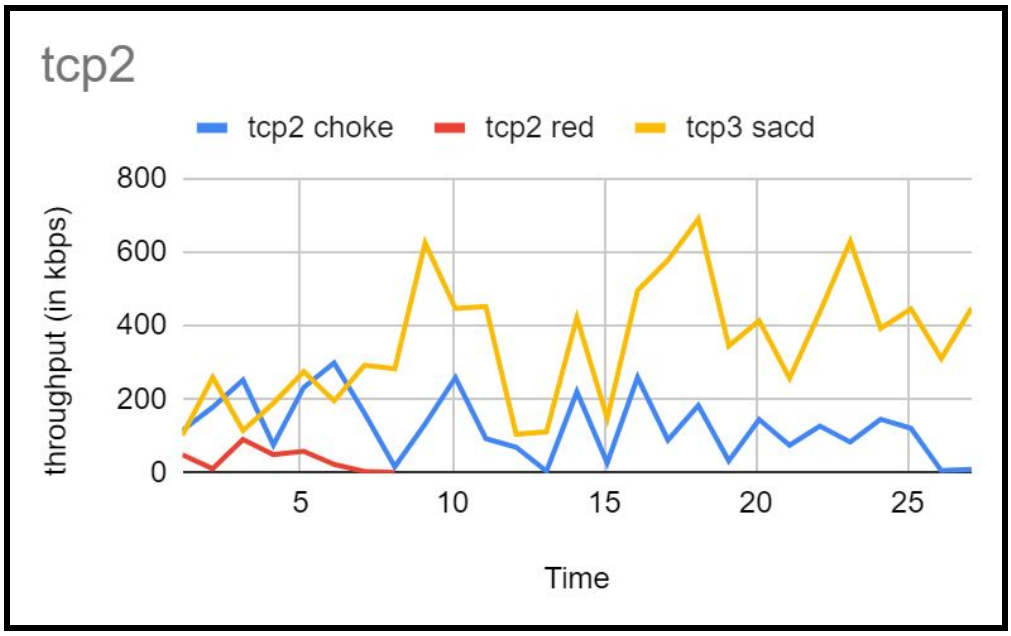


***Figure 10 : TCP2 throughput***

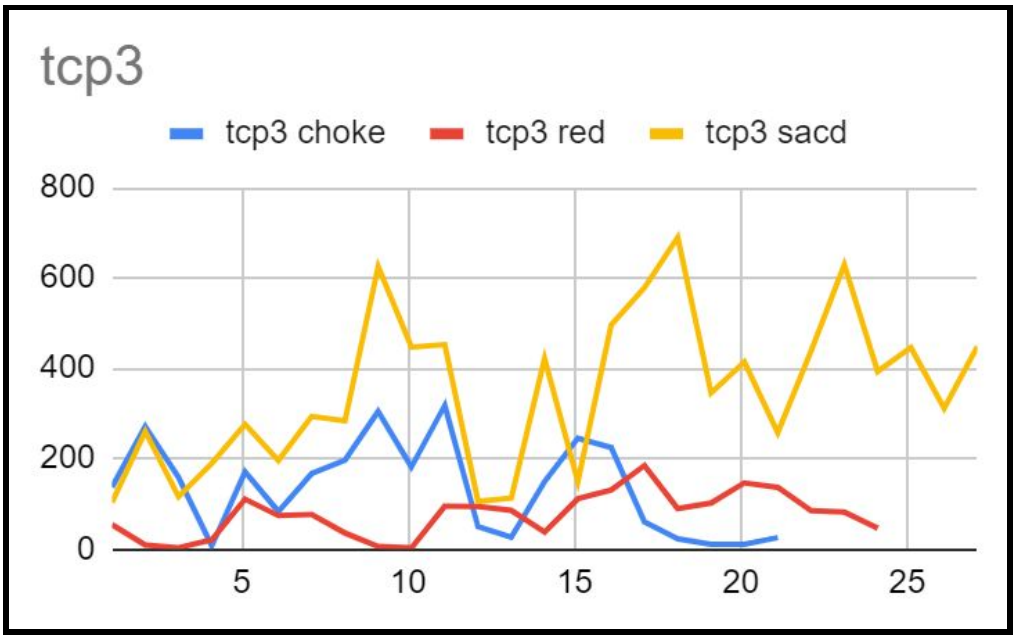


***Figure 11: TCP3 throughput***

Our algorithm was much better at guarding the TCP from UDP flows as can be seen from the TCP throughput values.

If we see the individual TCP flows we can see that they were also fair amongst themselves, all of them lying in 400-700 range on average.

## UDP FLOW COMPARISON IN CASE OF MULTIPLE UDP's of diff rate: Scenario 3.c

**UDP-1 : 1 mbps** **UDP-2 : 6 mbps**

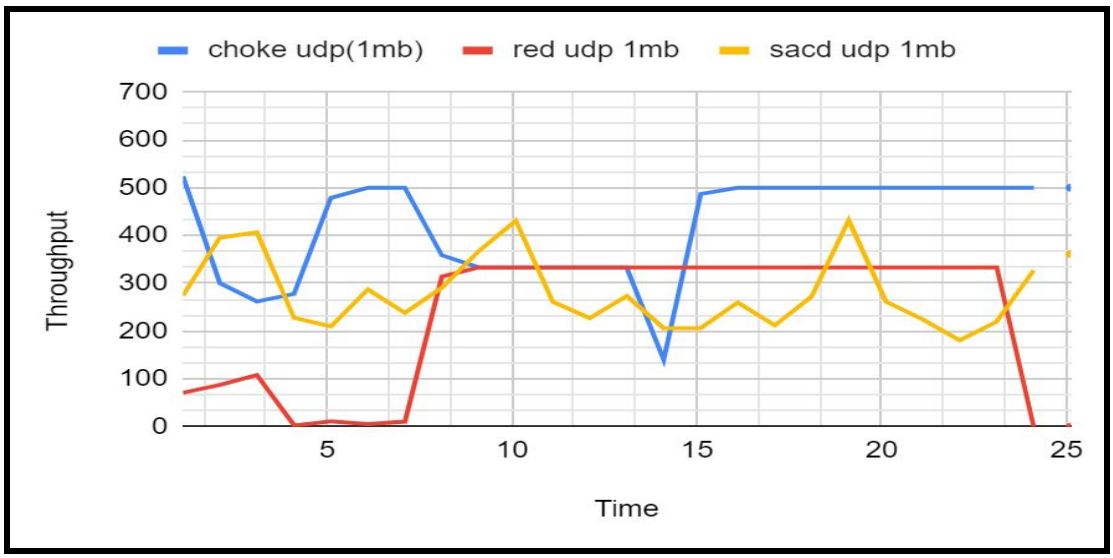


***Figure 12: UDP1 throughput***

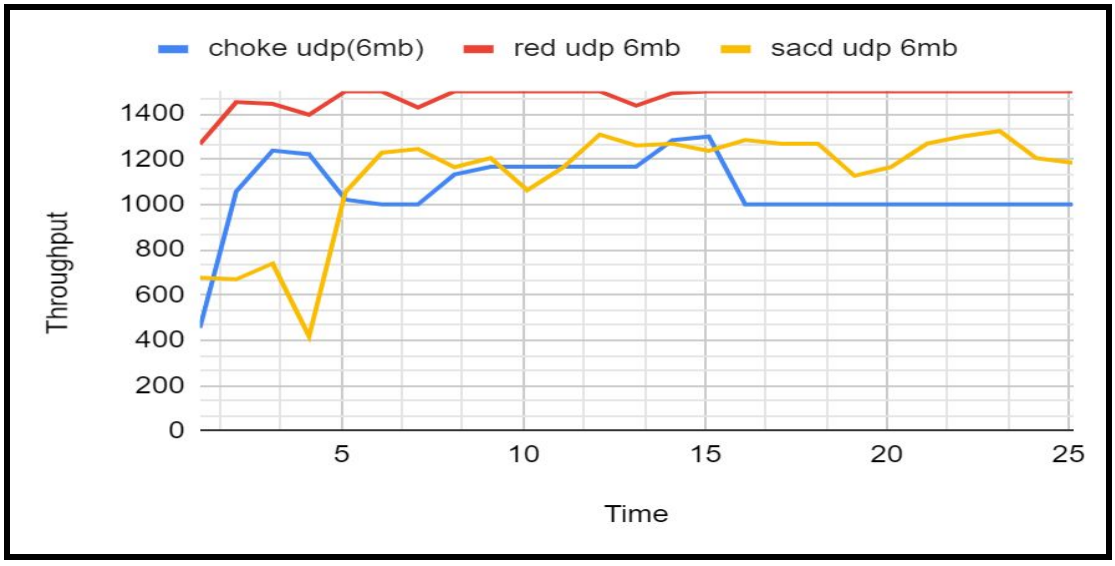


***Figure 13: UDP2 throughput***

| | RED | CHOKE | SACD |
|---|---|---|---|
| **UDP1 (1 mbps)** | 215.8337754 | 419.82336 | 282.20692 |
| **UDP2 (6 mbps)** | 1476.628 | 1061.82596 | 1124.0704 |
| **Ratio** | **0.146166655** | **0.3953786928** | **0.2510580476** |

***Table 3: Average throughput values to compare fairness***

As we can observe from the graphs and table 3 above CHOKe is fairer than SACD which is fairer than RED as closeness of throughput ratio to **1** is proportional to fairness of an AQM. Here, CHOKe has higher ratio than SACD which has higher ratio than RED. Hence, our proposed algorithm has this problem though we developed SACD to achieve fairness among UDPs. Therefore, our algorithm requires more intensive tuning of the parameters to avoid unfair penalization of UDP flows.

# Conclusion

The given report summarises the idea and methodology of the proposed algorithm i.e. **SACD**. It shows that the algorithm works better than both red and choke in some aspects like total throughput and TCP throughput in the testing topologies/scenarios as considered. However, there are a few problems that needs to be addressed w.r.t the algorithm , the same is detailed in the future work section below in order to further improve our algorithm..

# Future Work

The proposed algorithm can be further improved after fine tuning of its parameters especially **alpha**. A formula may be devised to globally change alpha depending on the situation based on p,r,k etc. values at any given instance so that penalization of udp flows is more accurate which is a problem as seen in the results section in scenario 3.c.

Devising strategies for the initialization of parameter ‘k’ i.e number of partitions.

Another possibility could be that once we reach the starting half of the queue we should stop the process of picking for udp and shift to the similar strategy as in red algorithm.

More rigorous and exhaustive scenario testing in order to confirm the validity of the algorithm and also comparing it with a few of the other available algorithms.

# References

[1] Pan, R., Prabhakar, B. and Psounis, K. (2000), ―CHOKe-a stateless active queue management scheme for approximating fair bandwidth allocation ‖ , INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE, IEEE, Vol. 2, pp. 942–951.

[2] Jiang, Y., Hamdi, M. and Liu, J. (2003), ―Self adjustable CHOKe: an active queue management algorithm for congestion control and fair bandwidth allocation ‖ , Computers and Communication, 2003.(ISCC 2003). Proceedings. Eighth IEEE International Symposium on, IEEE, pp. 1018–1025.

[3] Manzoor et al. CHOKeD: A Fair Active Queue Management System

[4] Ao Tang, Jiantao Wang, and Steven H. Low  Understanding CHOKe: Throughput and Spatial Characteristics IEEE/ACM TRANSACTIONS ON NETWORKING, VOL. 12, NO. 4, AUGUST 2004

[5] rfc6864 , rfc3697

[6] Minakshi Mehta, Gaurav Deep ,COMPARISON OF ACTIVE QUEUE MANAGEMENT IN NS2,Asian Journal of Computer Science And Information Technology

[7] Wu-chang Feng,Dilip D. Kandlur,Debanjan Saha and Kang G. Shin-- Stochastic Fair Blue: A Queue Management Algorithm for Enforcing Fairness

[8] Wu-chang Feng,Dilip D. Kandlur,Debanjan Saha and Kang G. Shin-- BLUE: A New Class of Active Queue Management Algorithms

[9] 6.829 Router Based Congestion Control - Lecture 9 MIT

# Appendix

- **Pseudo Code for our algorithm**

## Pseudo Code

---

**Procedure SACD**{
P :- 1 // variables required
R :- 1
tempk :- init
K :- init
q_avg :- as in RED
Alpha :- // Hyperparameter
wp = 0.002
wr = 0.002
wk = 0.0005

for each arriving packet{
calculate the avg. queue size;
If (**TCP**){
R :- (1-wr)*R;
if(minth <= q_avg <= maxth){
calculate prob Pa;
with Pa drop arriving packet;
}
else if(maxth <= q_avg)
drop arriving packet // FORCED drop
}
else{ // packet is UDP
R :- (1-wr)R + wr
found :- false
if(q_avg <= maxth){
calculate partition number;
j :- int(q_avg/maxth) * k;
Dja :- ( Qc * sqrt(q_lim) * $(1-1/2^{(j/2)})^{-1}$ )/ 2*($th_{max}$-$th_{min}$)*ln(q_lim);
for i=j to 1 in steps of alpha{
for k=0 to alpha-1{
Dr = Dja;

```
            while(Dr>0){
                draw packet pk' from buffer(partition);
                if(ID(pk')=ID(pk)){
                    drop pk';
                    found = true;
                    if(Dja-Dr==1) p:-(1-wp)*p+wp;
                }
                Dr=Dr-1;
            }
            Dja = Dja/2;
            if(found==true){
                    p = wp + (1-wp)*p;
                    break;
            }
            else{
                    p = (1-wp)*p;
            }
        }
    }
    if(found==false){
        p :- (1-wp)*p;
    }
    if(found==true && minth<=q_avg){
        drop incoming packet;
    }
    else if(q_avg>=minth){
        calculate prob pa and drop packet with pa;
    }
}
else{ // q_avg >= maxth
    calculate Dka // Same as Dja;
    for i = j to 1 in steps of alpha{
        for k =0 to alpha-1{
            Dr = Dka;
            while(Dr>0){
                Dr = Dr-1;
                Draw packet pk' from buffer;
                if(ID(pk')=ID(pk)){
                    drop pk';
                    found = true;
                    if(Dka-Dr==1) p:-(1-wp)*p+wp;
                }
            }
```

```
                }
                Dka = Dka/2;
                if(found==true){
                    p = wp + (1-wp)*p;
                    break;
                }
                else{
                   p = (1-wp)*p;
                }
            }
            if(found==false)
                p :- (1-wp)*p;
            drop incoming packet;
        }
    }
    tempk :- (1-wr)tempk + wk*2*(sqrt(R))/p**0.75;
    k :- (int)(tempk+0.5);
  }
}
```

---

## Excel comparing various AQMs : HERE

| Name of the algorithm | Problem handled (w.r.t basic choke) | Basic Methodology | Key parameters | Throughput (TCP) | Throughput (UDP) | Fairness (Overall) | Fairness (TCP) | Fairness (UDP) |
|---|---|---|---|---|---|---|---|---|
| CHOKe (Choose and Keep for responsive flows, Choose and Kill for unresponsive flows) | Unfairness of RED | choose one packet from queue and other one is arriving packet. compare their flow ids | T(max), T(min), Q(avg), p(b) | near fair share in case in of single udp | penalized in case of multiple udp flows | fair than BLUE, RED | throughput is near fair share | in single udp flow it is fair, not in multiple flows |
| M-Choke | penalization of udp flows in case of multiple udp flows | rather than choosing 1 packet choose m-1 packets | T(max), T(min), Q(avg), p(b), m | near fair share even with multiple udp flows | penalized if multiple udp flows vary a lot in number | fair than CHOKe | throughput near fair share | if number of udp flows vary than fairness is not achieved |
| Adaptive Choke | dynamic version of M-CHOKe | rather than fixing m, before starting we take it as a dynamic variable | T(max), T(min), Q(avg), p(b), K, f(R(k)) | near fair share even with multiple udp flows | penalized if multiple udp flows vary a lot in number in terms of their bandwidth | fair than M-CHOKe | throghput near fair share | if bandwidth of multiple udp flows vary a lot then fairness is not achieved |
| Self- Adjustable CHOKe | extra researching overhead | For TCP flow - RED, For UDP flow - Divides region into k segments which are computed dynamically and picks i packets when in region i | K - no. of divisions , Q(avg), T(max),T(min) , P & R- which determines number of udp flows | near fair share even with multiple udp flows | near fair share | fair than M-CHOKe | throughput near fair share | fair than M-Choke |
| CHOKeD | selecting packets uniformly from queue can be a problem as udp packets are probable to be present at rear end than front end | divide whole queue till the point to which it is filled, in two regions and first try selecting drop candidates from rear end then from front end | T(max), T(min), Q(avg), p(b), Q(c), B, D(r), D(f) | near fair share even with multiple udp flows | near fair share but not dynamic to accomodate multiple udp flows | fair than M-CHOKe | throghput near fair share | not dynamic enough |

## Hierarchy of AQMs based on our study

As we go down the hierarchy of algorithms complexity and fairness values increase.

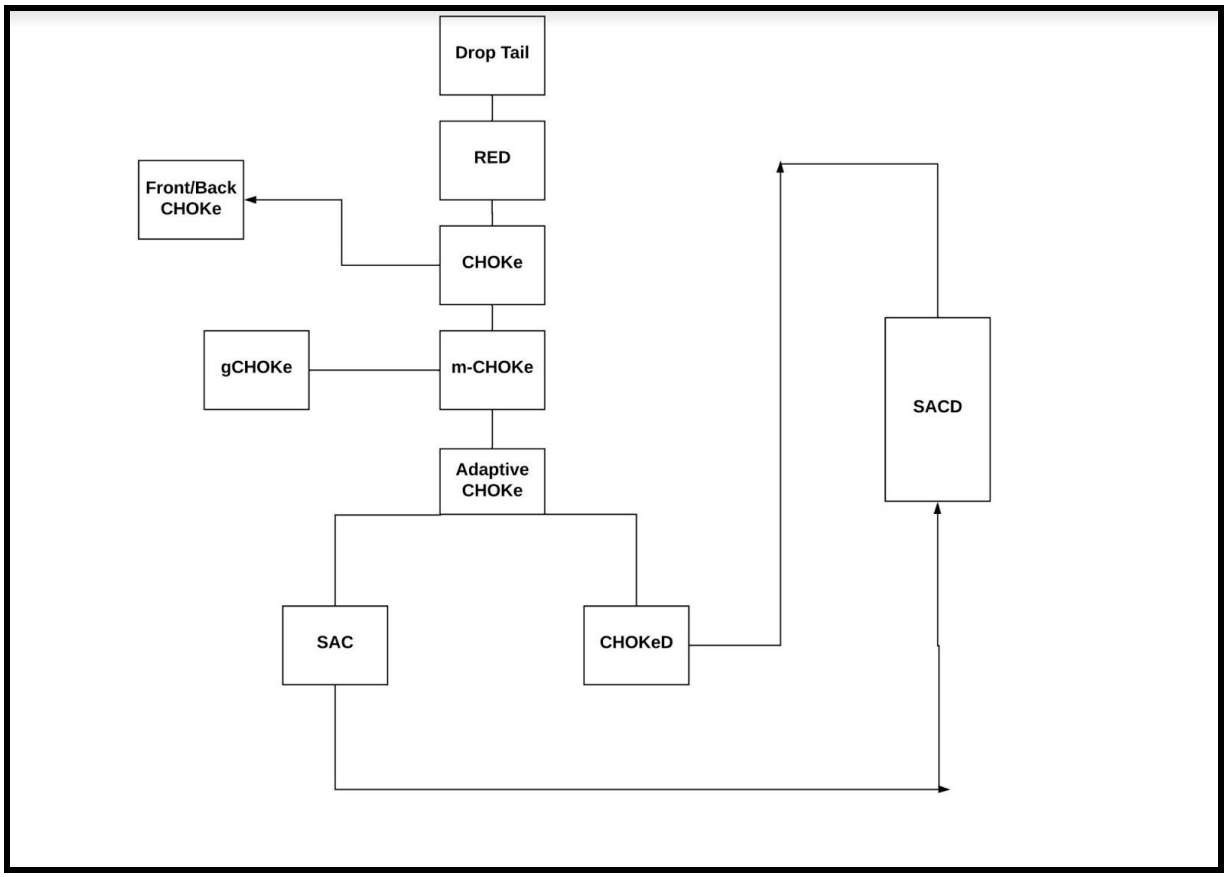


*Figure 13 : Chart showing hierarchy of algorithms*

---

END