Laboratório Casa 02

Pedro Henrique Dornelas Almeida
Dept. de Engenharia Elétrica (FT-ENE)
Universidade de Brasília (UnB)
Brasília, Brasil
phdornelas.almeida@gmail.com

***Resumo*—Este relatório tem como foco mostrar como se comportam aplicações que competem a banda com outros fluxos, avaliando por meio da política de justiça de Raj Jain se o gerenciamento da fila está sendo justo ou não a cada fluxo.**

***Index Terms*—Fluxos, Justiça, OnOff, PfifoFast, ARED, CoDel, FqCodell, PIE.**

I. Objetivos

O objetivo deste experimento é estudar os algoritmos de AQM(Active Queue Management), que foram criados para contrastar o crescimento descontrolado do tempo de fila dos pacotes ou controle de fluxo. Note que estes algoritmos descartam alguns pacotes para notificar as camadas superiores que há congestionamento de rede e possivelmente diminuir a sua janela de transmissão, descongestionando a fila.

Devemos avaliar e comparar as diferentes estratégias AQM disponíveis para diferentes cenários, no qual iremos aumentar o número de fluxos de pacotes e ver como o algoritmo irá se comportar.

II. Introdução Teórica

Os algoritmos de AQM surgiram para tentar solucionar o problema de *bufferbloat*, que consistia no aumento descontrolado de pacotes enfileirados, pois as camadas superiores queriam utilizar a maior largura de banda possível da rede. Assim, buffers maiores foram criado a fim de comportar mais pacotes na fila, porém, isso aumentou muito a latência, pois os pacotes ficavam muito tempo parados esperando para serem enviados.

Para realizar o experimento devemos entender quais e como funcionam os algoritmos de AQM. Utilizaremos os seguintes: PfifoFast, ARED, CoDel, FqCoDel e PIE. Cada um deles tem uma forma diferente para o gerenciamento das filas, descarte de pacotes, dentre outras diferenças.

A. PfifoFast

Este algoritmo é baseado na política FIFO(*First In, First Out*), em que o primeiro pacote a entrar na fila também é o primeiro a sair da fila. Porém, o PfifoFast conta com 3 filas de prioridades diferentes, enumeradas de 0 a 2, em que a fila 0 é a de menor prioridade e a fila 2 é a de maior prioridade. Os pacotes nas filas de prioridade maior são servidos primeiro, e são classificados em alguma das filas baseado nos quatro bits menos significativos no campo de prioridade. Se alguma fila estiver cheia, o último pacotes que chegou nesta fila será descartado.

B. ARED

O algoritmo ARED(*Adaptative Random Early Detection*) é baseado no algoritmo RED(Random Early Detection), em que este calcula uma probabilidade p para um pacote ser descartado com base no tamanho médio da fila q_{avg} nos buffers analisados. A probabilidade p é zero quando q_{avg} está abaixo de um limite mínimo definido como min_{th}. Esta probabilidade então cresce linearmente quando q_{avg} está entre o limite mínimo (min_{th}) e o limite máximo max_{th}, até chegar em max_p. Assim que q_{avg} é maior que max_{th}, a probabilidade para descartar um pacote se mantém em 1 e os pacotes seguintes que chegarem serão descartados.

Note que os parâmetros max_{th}, min_{th} e max_p devem ser configurados pelo operador, baseado em estudos com a topologia. Já no ARED, esses parâmetros são automaticamente configurados, dado os parâmetros de largura de banda, atrasos, dentre outros. Assim, um mecanismo que o ARED tem é igualar p a max_p de acordo com a carga de tráfego, de forma que q_{avg} se mantenha entre min_{th} e max_{th}.

C. CoDel

Este algoritmo monitora os atrasos dos pacotes para descartá-los, e não mais no tamanho da fila como o ARED. Assim, ele monitora os atrasos reais, e se em determinado tempo de observação configurado todos os pacotes estiverem com o atraso maior que o esperado, o CoDel irá descartá-los. Quando algum pacotes é descartado, o algoritmo programa o tempo em que o próximo pacote será descartado a partir de uma equação que depende do último tempo de pacotes descartados(*lastDropTime*), da quantidade de pacotes descartados(*noOfDrops*), e do intervalo de observação selecionado(*observationInterval*), assim como a seguinte equação:

$$nextDropTime = lastDropTime + \frac{observationInterval}{noOfDrops}$$

D. FqCoDel

O FqCoDel utiliza um mecanismo de agendador de pacotes, e também o algoritmo CoDel para gerenciar suas filas. Este algoritmo organiza os pacotes em até 1024 filas, estas são servidas por uma versão modificada do algoritmo de agendamento DRR(*Deficit Round Robin*). Cada uma das filas criadas é gerenciada pelo algoritmo CoDel, visto na secção anterior. Os pacotes são alocados às filas a partir de uma operação de *hashing* da tupla de 5 elementos, compostos por endereços IP e porta de origem e destino perturbado por um número

aleatório selecionado no tempo que se inicia, além de conter também o protocolo que o pacote carrega. Esta não é a única maneira de ser associar um pacote a uma fila. No geral, o FqCodel possui duas partes principais: 1) o agendador, que utiliza de um algoritmo DRR para escolher qual fila vai ter os seus dados enviados e o 2) algoritmo CoDel para gerenciar o comportamento das filas criadas.

E. PIE

Por usa vez, o algoritmo PIE(*Proportional Integral controller Enhanced*) se assemelha ao ARED no que diz respeito ao descarte de pacotes para prevenir gargalos. Porém, ao contrário do ARED que descarta os pacotes a depender do tamanho da fila, ou do CoDel que descarta a partir de atrasos na fila. Ademais, ele estima utilizando da média móvel suavizada da taxa de drenagem. Assim, a probabilidade p, no algoritmo PIE, para descartar um pacote é atualizada a cada 30 milissegundos. Temos as variáveis de *queueDelay*, que é o valor estimado para o atraso da filam, *refDelay* é o valor de referência para o atraso e *lastQueueDelay* foi o último valor estimado para o atraso na fila. Em seguida, temos também os fatores α e β, que aumentam com o aumento de p. Estes valores se relacionam para que p possa ser descoberto:

$$p = p + \alpha(queueDelay - refDelay) + \beta(queueDelay - lastQueueDelay)$$

Outro fator interessante, este algoritmo evita descartar pacotes quando recebe pequenas rajadas em situações que há pouco congestionamento. Pode-se ver a situação que quando o atraso estimado estiver abaixo da metade do valor alvo por dois intervalos consecutivos de atualização, por 100ms o algoritmo não irá levar p em consideração e aceita todos os pacotes que chegarem na fila.

III. Descrição das Atividades e Análises

Para realizar as atividades solicitadas, foi utilizado um esquema Dumbbell, no formato da figura a seguir:

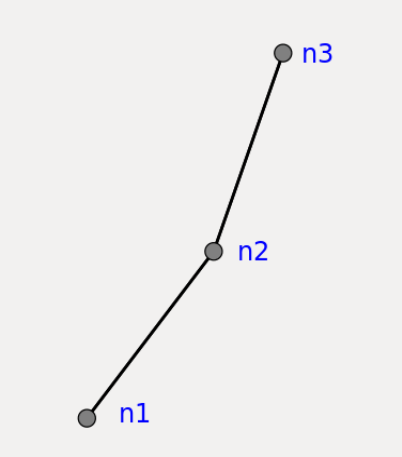

Figura 1. Topologia

As características consistem em ter um link entre o nó 1 e nó 2 que tenha uma banda suficiente para não acontecer gargalo neste link, porém, entre o nó 2 e nó 3 acontecerá um gargalo, pois é um link de menor banda disponível. Dessa maneira, os algoritmos AQM para o gerenciamento da fila estará atuando no nó 2 para amenizar os problemas de gargalo.

Os fluxos criados consistem em um fluxos TCP, um deles partindo do nó 1 e indo ao nó 3 e outro do nó 3 indo ao nó 1, este é o processo denominado *OnOff*, e estes dois fluxos formam um ciclo OnOff. Também podem ser chamados de *upload* e *download*.

O código criado para implementar este cenário contém medidas de RTT que serão utilizadas para alguns cálculos a fim de comparar os diferentes algoritmos AQM quanto a sua eficiência.

QUESTÃO 1

Neste momento, foi necessário executar a simulação para os diferentes algoritmos AQM, sendo eles: PfifoFast, ARED, CoDel, FqCodel e PIE. Para cada simulação pôde ser obtido os gráficos comparativos para os tamanhos da fila de transmissão:

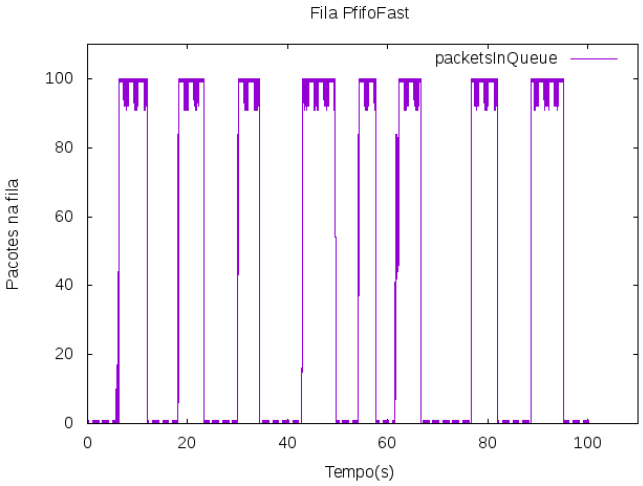

Figura 2. Fila PfifoFast

Note da figura anterior no início da transmissão, como o TCP está se iniciando, ele começa em uma transmissão lenta dos pacotes, assim, o capacidade do enlace entre n2 e n3 ainda é suficiente e a fila se mantém vazia. Porém, com o passar do tempo a taxa transmissão aumenta, fazendo com que a fila aumente de tamanho, até chegar ao seu tamanho máximo de 100 pacotes. Neste momento começa o descarte de pacotes, isso acontece para informar ao remetente que há um congestionamento e alterando sua janela de congestionamento para enviar pacotes conforme a rede tenha capacidade de enviar. Assim, a fila volta a esvaziar, pois n2 consegue dar vazão aos pacotes sem que muitos pacotes cheguem para encher a fila.

Perceba que este processo vai se repetir, pois o algoritmo PfifoFast só descarta pacotes quando sua fila está cheia, logo, esse enchimento da fila e subsequente esvaziamento vão acontecer até que não tenham mais pacotes sendo enviados.

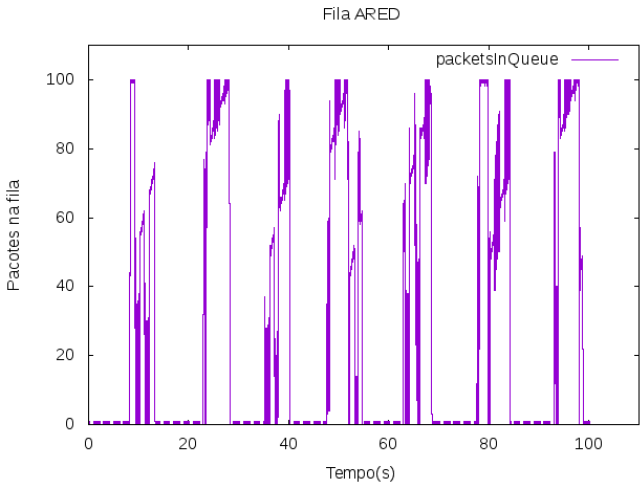

Figura 3. Fila ARED

Já no algoritmo ARED(figura 3), no início acontece o mesmo processo para o algoritmo PfifoFast, porém, a forma como o ARED lida com sua fila é diferente, descartando pacotes preventivamente, ou seja, antes que sua fila fique completamente cheia, assim, note que nos picos o tamanho da fila não se mantém constante no máximo, como o PfifoFast, pois com o descarte preventivo, o remetente altera sua janela de congestionamento.

Note que o ARED altera os parâmetros de descarte conforme o tamanho médio da fila, o mínimo definido e o máximo. Assim, conforme mais pacotes são adicionados a fila o próprio algoritmo se otimiza para que o remetente possa enviar mais dados, desde que fique dentro dos limites para tamanho da fila mínimo e máximo. Este processo torna o envio por parte do remetente mais otimizado, pois consegue uma melhoria na taxa de envio sem descarte dos pacotes quando comparamos ao PfifoFast.

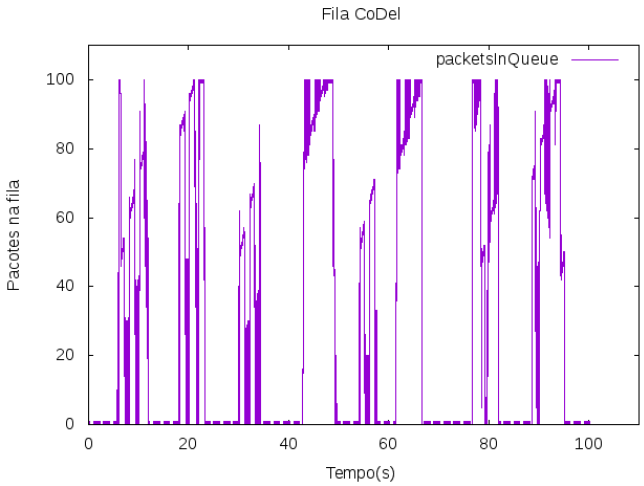

Figura 4. Fila CoDel

Da figura acima, podemos perceber que há uma semelhança no gráfico com o gráfico do ARED, porém, é importante lembrar que o CoDel utiliza de atrasos reais dos pacotes para descartá-los e assim que algum pacote é descartado o próximo pacote a ser descartado terá o tempo agendado para isto. Podemos aproximar para ver melhor o efeito deste algoritmo:

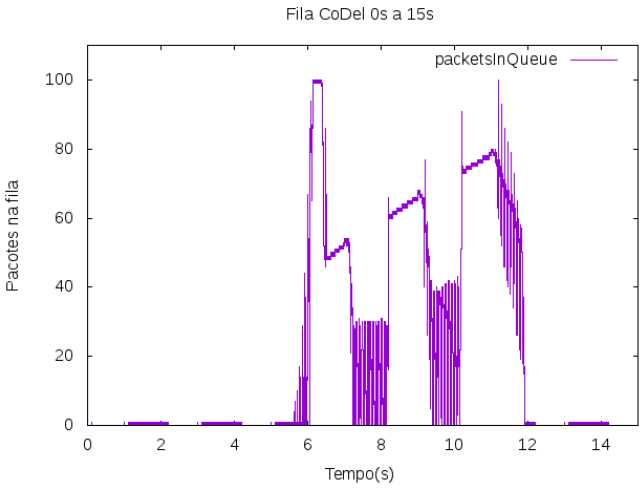

Figura 5. Fila CoDel 0s a 15s

Da figura anterior é possível ver como as variáveis da fórmula são mudadas conforme mais pacotes vão chegando na fila, mudando o intervalo de observação do algoritmo. Note que com o enchimento rápido da fila o descarte de pacotes foi maior, caindo significativamente o tamanho da fila. Porém, após este período, note que o tamanho da fila foi crescendo comportadamente, isso pelas variáveis de próximo descarte, atraso, que foram sendo ajustados e percebendo que a fila poderia ser maior, sem que ela ficasse totalmente cheia.

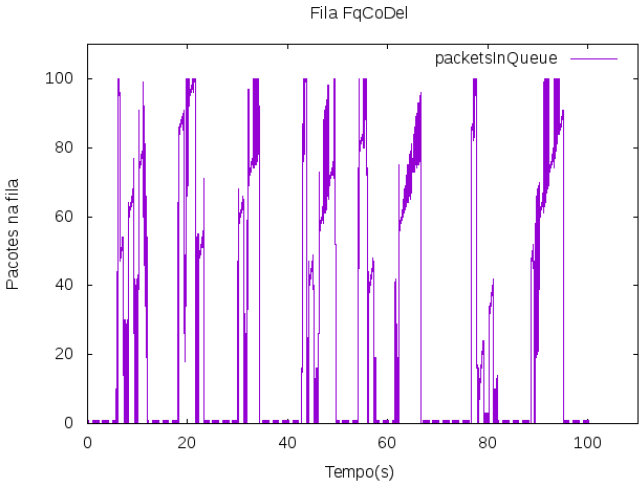

Figura 6. Fila FqCoDel

Note que o algoritmo FqCoDel é bem parecido com o algoritmo CoDel, porém, fazendo o uso de mais filas que são gerenciadas de acordo com um nível de prioridade, servidas

pelo DRR. Assim, para cada uma das filas um algoritmo CoDel estará atuando, medindo atrasos, calculando próximos descartes, intervalos de observação, e todas as outras propriedades.

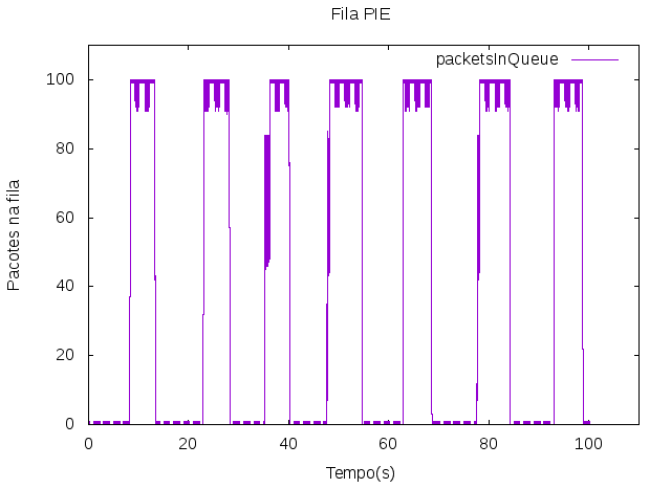

Figura 7. Fila PIE

Por último, não menos importante, temos o algoritmo PIE, este se assemelha ao ARED, por utilizar probabilidade p para o descarte de pacotes e ao CoDel, por considerar atrasos nas filas. Porém, graficamente, podemos obersvar que a fila se parece com a do PfifoFast. Isto se deve ao fato da fórmula empregada para se descartar pacotes, como foi mostrada na introdução teórica.

QUESTÃO 2

Neste momento, foi necessário gerar vários fluxos de dados, pois assim, teremos comparações dos algoritmos AQM que podem ser mais precisos, podendo ser usado o cálculo de Justiça para obter como se comportam tais algoritmos.

O cálculo da Justiça de Raj Jain será utilizado para isto e pode ser calculado da seguinte maneira:

$$J = \frac{\left(\sum_{i=1}^{n} x_i\right)^2}{n \cdot \sum_{i=1}^{n} x_i^2}$$

Em que x_i é o *throughput* do fluxo i, n é o número de fluxos. Assim, teremos um índice J variando de 0 a 1, mais injusto a mais justo, respectivamente. Note que teremos de fazer os cálculos para cada fluxo do ciclo OnOff, ou seja, para *download* e *upload*.

a)

Alterando o código para gerar 2, 5, 10 e 20 fluxos foi criado. Também foi adicionado o código para mostrar taxas de download e upload para cada fluxo, assim, foi possível calcular os valores de justiça para cada um dos algoritmos AQM, assim como nas tabelas a seguir:

Tabela I
JUSTIÇA DE DOWNLOAD

Fluxos	PfifoFast	ARED	CoDel	FqCoDel	PIE
2	0,999999	0,999676	0,999988	0,999161	0,998214
5	0,999996	0,996071	0,998081	0,999950	0,995005
10	0,995316	0,998439	0,991919	0,999729	0,995534
20	0,939249	0,991330	0,988554	0,999815	0,986603

Tabela II
JUSTIÇA DE UPLOAD

Fluxos	PfifoFast	ARED	CoDel	FqCoDel	PIE
2	0,994906	0,997389	0,998638	0,997069	0,998875
5	0,991233	0,945448	0,965636	0,980910	0,950494
10	0,964792	0,939682	0,960366	0,986129	0,964280
20	0,822653	0,935083	0,948595	0,964104	0,944091

b)

Para o algoritmo PfioFast, é possível observar que o algoritmo ficou aceitável até 10 fluxos simultâneos, já para 20 fluxos o seu desempenho foi gravemente afetado. Isto se deve ao fato de que utiliza de 3 filas FIFO, com prioridades diferentes, com isso, é possível perceber que o algoritmo se torna mais injusto conforme o número de fluxos aumenta, pelo fato de que enquanto a fila de prioridade maior tiver pacotes para enviar, as filas de menores prioridades não conseguem dar vazão aos seus pacotes, isto faz com que o descarte desses pacotes de menor prioridade sejam grandes, e para determinados fluxos, que estão sendo classificados como menor prioridade, serão ”injustiçado”para o envio de seus pacotes.

No algoritmo ARED por sua vez, é importante lembrar que possui somente uma fila para envio dos pacotes, ou seja, não precisar alocar prioridades diferentes para os pacotes. Aliado a isso, o ARED descarta os pacotes de maneira preventiva, logo, como não há prioridades diferentes, ele irá descartar pacotes de diferentes fluxos de maneira mais justa, fazendo com que estes diferentes fluxos possam ajustar sua janela de congestionamento, e assim, o algoritmo consegue ser mais justo entre todos os fluxos, pois terão a vazão e respostas do algoritmo ARED parecidas.

Para o algoritmo CoDel, pode-se observar um desempenho parecido, porém levemente melhor que o ARED, principalmente se olharmos para a Justiça de Upload. Isto se deve ao fato de que o CoDel é bem parecido ao ARED, pois também implementa somente uma fila, não precisando classificar prioridades para os pacotes e fluxos, porém, diferente do ARED, o CoDel descarta pacotes preventivamente levando em consideração o atraso na fila. Note que conforme o tamanho médio da fila aumenta, é também provável que o atraso da fila aumente, lopo, era de se esperar que o desempenho destes fossem parecidos.

No FqCoDel podemos ver o melhor desempenho para os índice de justiça calculados, mesmo com um grande número de fluxos. Isso se deve ao fato de que este algoritmo pode utilizar de muitas filas para servir aos fluxos, adotar prioridades, sendo que em cada uma das filas separadamente temos o CoDel atuando, então o FqCoDel irá descartar pacotes das várias

filas, independente da sua prioridade, o que faz os fluxos se ajustarem e assim, o algoritmo fica mais justo, conseguindo servir simultaneamente vários fluxos, sem perder sua qualidade de justiça. O DRR ajuda bastante nesse processo, pois limita a quantidade de *bytes* enviada por fila, assim os fluxos terão uma quantidade mínima de taxa de transferência, o que eleva ainda mais o índice de justiça.

No algoritmo PIE, é possível observar a semelhança com o ARED e o CoDel. Isto se deve ao fato de todos terem somente uma fila, e descartar pacotes preventivamente. O ponto que os diferem é a maneira como estes algoritmos descartam pacotes. O PIE descarta seus pacotes por atrasos estimados na fila, logo, era de se esperar que fosse bem parecido com o CoDel, que descarta pacotes por atrasos reais na fila, daí podemos tirar a semelhança dos resultados obtidos.

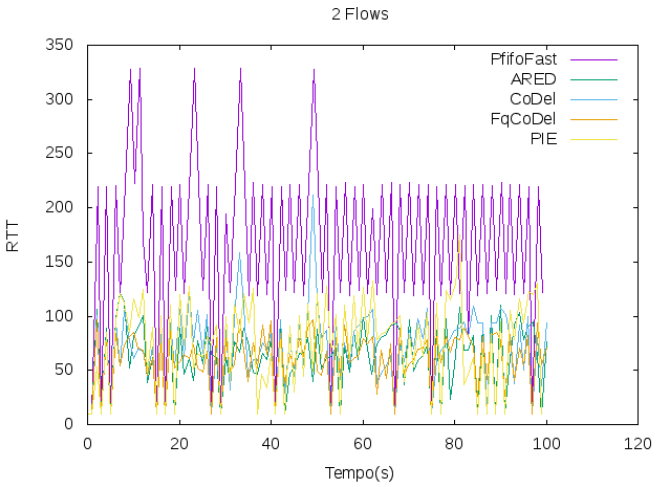

Figura 9. RTT para 2 Fluxos

QUESTÃO 3

Neste momento foi possível montar gráficos comparativos dos RTTs a partir de um arquivo de saída da própria simulação(AQM-v4PingRtt.txt). Com este arquivo foi possível obter as métricas necessárias para gerar os gráficos e medianas para comparar os algoritmos.

a)

Para este item, foi utilizado o gnuplot para gerar o gráfico comparativo entre os mecanismos AQM para a mesma quantidade de fluxos. O código feito para montar estes gráficos foi da seguinte maneira:

```
set terminal png
set output '2flows.png'

set title "2 Flows"
set xlabel "Tempo(s)"
set ylabel "RTT"

set autoscale

plot "../PfifoFast_2Flow/PfifoFast-v4PingRtt.txt" using 1:2 with lines title "PfifoFast", \
     "../ARED_2Flow/ARED-v4PingRtt.txt" using 1:2 with lines title "ARED", \
     "../CoDel_2Flow/CoDel-v4PingRtt.txt" using 1:2 with lines title " CoDel", \
     "../FqCoDel_2Flow/FqCoDel-v4PingRtt.txt" using 1:2 with lines title "FqCoDel", \
     "../PIE_2Flow/PIE-v4PingRtt.txt" using 1:2 with lines title "PIE"


```

Figura 8. Código .plt

Apenas alterando a pasta para os diferentes fluxos, foi possível obter os gráficos das figuras 9, 10, 11, 12.

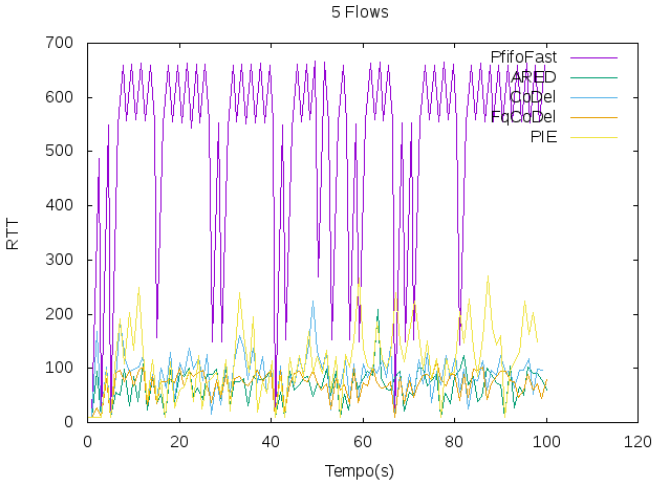

Figura 10. RTT para 5 Fluxos

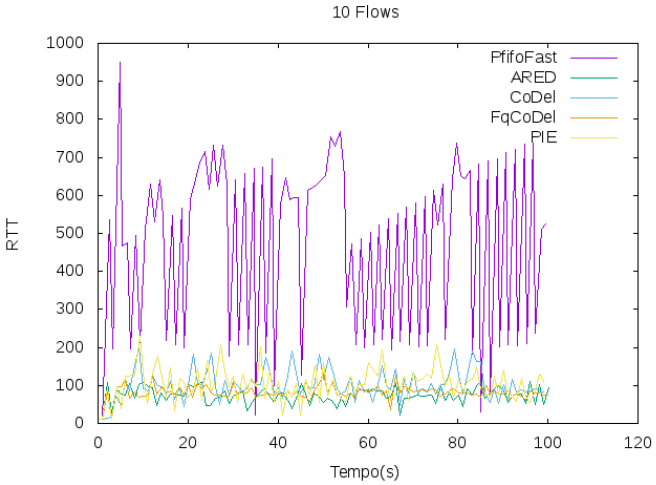

Figura 11. RTT para 10 Fluxos

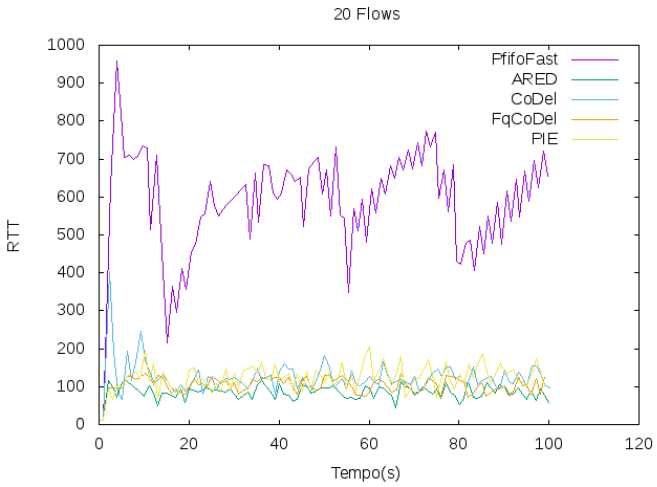

Figura 12. RTT para 20 Fluxos

Das figura acima conseguimos ver claramente alguns comportamentos, o primeiro deles é que o algoritmo PfifoFast é o algoritmo com pior desempenho em relação ao RTT, isso deve ao fato de que ele permite que sua fila se encha totalmente, para só depois descartar pacotes, isso faz com que mais pacotes fiquem mais tempo na fila, aumentando o atraso dos pacotes que acabam entrando na fila quando esta está cheia.

Para os outros algoritmos, ARED, CoDel, FqCodel e PIE, o RTT é bem semelhante, mesmo que haja algumas diferenças, essa diferença não é muita, e na prática, eles já fazem um papel bem mais eficiente que o PfifoFast, melhorando bastante os atrasos causados na fila. Isto porque os mecanismos para gerência dessas filas avisam as camadas superiores que estão congestionadas pelo simples fato de descartarem pacotes antes de sua fila encher completamente, logo, as camadas superiores alteram sua janela de congestionamento, mantendo a fila comportada, assim, os algoritmos dão vazão aos pacotes que chegam mais rapidamente, isso aumenta bastante a eficiência no RTT desses pacotes.

b) Em seguida, para calcular a mediana dos RTTs coletados também no arquivo citado no item anterior, realizei um código em python para que fosse possível coletar tais medianas a partir destes arquivos (Figura 13).

Então, foi possível obter os seguintes dados:

Tabela III
MEDIANA RTT

Fluxos	PfifoFast	ARED	CoDel	FqCoDel	PIE
2	218,5	66	77	68	83
20	607	85	118	105	123

Note da tabela que temos uma diferença significante para os diferentes tipos de mecanismos AQM. Para o PfifoFast note que a mediana é muito maior que para os outros algoritmos, o que nos confirma o que já foi visto no experimento como um todo, que este é o menos eficiente para gerenciamento ativo de fila, e com o RTT, não seria diferente.

```python
import statistics

def mediana(x):
    return statistics.median(x)

def rtt(df):
    f = open(df,'r')
    lines = f.readlines()
    rtt = []
    for line in lines:
        line = line.split()
        rtt.append(int(line[1]))
    return rtt

pfifofast_2flow = mediana(rtt('PfifoFast_2Flow/PfifoFast-v4PingRtt.txt'))
ared_2flow = mediana(rtt('ARED_2Flow/ARED-v4PingRtt.txt'))
codel_2flow = mediana(rtt('CoDel_2Flow/CoDel-v4PingRtt.txt'))
fqcodel_2flow = mediana(rtt('FqCoDel_2Flow/FqCoDel-v4PingRtt.txt'))
pie_2flow = mediana(rtt('PIE_2Flow/PIE-v4PingRtt.txt'))


pfifofast_20flow = mediana(rtt('PfifoFast_20Flow/PfifoFast-v4PingRtt.txt'))
ared_20flow = mediana(rtt('ARED_20Flow/ARED-v4PingRtt.txt'))
codel_20flow = mediana(rtt('CoDel_20Flow/CoDel-v4PingRtt.txt'))
fqcodel_20flow = mediana(rtt('FqCoDel_20Flow/FqCoDel-v4PingRtt.txt'))
pie_20flow = mediana(rtt('PIE_20Flow/PIE-v4PingRtt.txt'))

# labelx = ['PfifoFast','ARED','CoDel', 'FqCodel', 'PIE']
# labely = ['2 Flows', '20 Flows']

print('PfifoFast_2Flow = ' + str(pfifofast_2flow))
print('ARED_2Flow = ' + str(ared_2flow))
print('CoDel_2Flow = ' + str(codel_2flow))
print('fqcodel_2Flow = ' + str(fqcodel_2flow))
print('pie_2Flow = ' + str(pie_2flow))
print('pfifofast_20Flow = ' + str(pfifofast_20flow))
print('ared_20Flow = ' + str(ared_20flow))
print('codel_20Flow = ' + str(codel_20flow))
print('fqcodel_20Flow = ' + str(fqcodel_20flow))
print('pie_20Flow = ' + str(pie_20flow))
```

Figura 13. Código para Mediana de RTTs

Pode-se perceber que em relação aos outros, o algoritmo ARED foi o melhor em termos de controle ao RTT, pois com 2 ou 20 Fluxos, ou seja, “melhor” e “pior” caso testados ele ainda se manteve melhor que os outros, sendo que no “pior”, o desempenho significantemente melhor que nos outros algoritmos. Isto nos mostra que o gerenciamento de fila pelo tamanho médio da mesma é uma boa opção para quem deseja ter tempos de RTT melhores.

Os algoritmos CoDel e FqCoDel, apesar de guiar o descarte de pacotes a partir de atrasos reais medidos, não fazem um bom gerenciamento para evitar o aumento do RTT, assim como o PIE, que mesmo parecendo com o ARED e o CoDel, ainda manteve o problema do algoritmo CoDel para que o RTT não aumente.

IV. CONCLUSÃO

Com a realização deste experimento foi possível entender o funcionamento e comparar o desempenho dos algoritmos de AQM: PfifoFast, ARED, CoDel, FqCoDel e PIE. Foi possível ver como um link pode ser o gargalo de uma rede, e também estratégias para que os fluxos não sejam gravemente afetados por filas e atrasos grandes no percurso.

Foi possível compará-los a nível de Justiça do tráfego, em relação a latência causada, tudo isso em função do aumento das filas e do número de fluxos existentes. Percebe-se que o pior algoritmo em termos de desempenho foi o PfifoFast, pois em todos os quesitos apresentou uma menor eficiência. No que tange a Justiça dos algoritmos, o melhor desempenho apresentado foi do FqCoDel, que apresentou os melhores

índices e se mostrou ser mais justo quando se tem muitos fluxos sendo servidos. Já no quesito latência, o melhor foi o ARED, com um desempenho significativamente melhor, que poderia ser parâmetro para escolher este em cenários que precisem a menor latência possível, já que o desempenho deste em termos de Justiça foi bem aceitável.

Com isso, conclui-se o estudo sobre algoritmos AQM e seu funcionamento.

REFERÊNCIAS

[1] *Roteiro do Lab. Casa 02*, https://aprender3.unb.br/mod/resource/view.php?id=690281

[2] J. F. Kurose e K. W. Ross, *Computer Networks: A Top-Down Approach. (5th ed.). Pearson Addison-Wesley, 2009.*

[3] Documentação NS3: https://www.nsnam.org/documentation/

[4] Documentação FlowMonitor https://www.nsnam.org/docs/models/html/flow-monitor.html

[5] Documentação WireShark https://www.wireshark.org/docs/