Guilherme Cano Lopes

Simulador de Roteador DOCSIS

Brasil

Dezembro de 2016

Resumo

Este projeto tem por objetivo o desenvolver um programa capaz de simular a transmissão de dados com base no protocolo DOCSIS (Data Over Cable Service Interface Specification). O programa simula o fluxo *downstream* entre um nó de CMTS (Cable Modem Termination System), localizado no *headend* de uma operadora de CATV (*Cable Access Television*), e *cable modems* de seus assinantes. O sistema foi modelado como um simulador de eventos discretos baseado em teoria das filas, onde é possível obter informações sobre o tempo médio de espera na fila, a utilização, quantidade de dados transmitidos, volume de dados ou pacotes descartados e características dos sistemas de *traffic shapping*. O software foi elaborado na linguagem Python.

Palavras-chaves: Simulador. DOCSIS. Filas.

Sumário

Introdução

O protocolo DOCSIS (3.0/3.1) consiste no padrão comercial de acesso à internet via cabeamento.

Algumas aplicações de tempo real, como *streamming* de vídeo, jogos on-line ou VOIP (*Voice Over IP*) são sujeitas à atrasos que podem prejudicar a experiência do usuário. Portanto, para estudar estes problemas, é possível simular o comportamento do TCP (*Transfer Controle Protocol*) por meio de um modelo baseado em teoria das filas (Volkers; Barakat; Darcie, 2010). Com isso, também é possibilitada a análise de metodologias de controle de tráfego que visam ditar a taxa da transmissão nas filas da forma como foi acordada entre a operadora e os assinantes.

1 Desenvolvimento do projeto

No CMTS (*Cable Modem Termination System*), existem 50 nós em que são conectados um conjunto de até 300 CMs. E em entre estes nós, 24 possuem banda de aproximadamente 800Mbps, enquanto os outros 16 tem cerca de 400Mbps. A figura 1 ilustra o fluxo *downstream* que está sendo modelado.

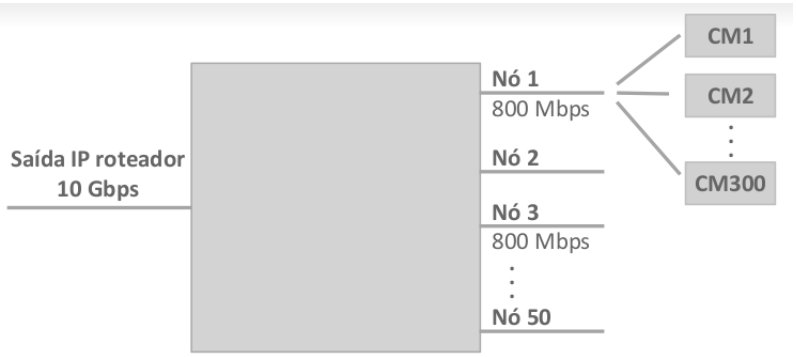

Figura 1 – Ilustração do funcionamento da transmissão entre o CMTS(*Cable Modem Termination System*) e os CMs.

Fonte: Material de aula.

A representação de um nó do CMTS é apresentada na figura 2, onde os canais que compõem um nó são representados por um número de filas FIFO (*First In First Out*). Considerou-se que cada *link* CMTS-CM é uma fila do tipo M/M/1/k, sendo que M significa que tanto o processo de chegada quanto o de serviço são Markovianos, e k representa o tamanho do *buffer* do roteador. Ou seja, o número máximo de pacotes que a fila suporta.

1.1 Geração de eventos discretos

Neste trabalho, a taxa de chegada no nó é estimada por Poisson e a taxa de serviço de cada assinante é exponencial. Isto significa que os intervalos de chegada entre os pacotes variam de acordo com uma distribuição exponencial 1.1:

$$F_x(X_i) = 1 - e^{-\lambda x_i} \tag{1.1}$$

E que pelo método da inversa, um intervalo é dado pela equação 1.2:

$$x_i = -\frac{1}{\lambda} ln(1 - r_i) \tag{1.2}$$

Sendo:

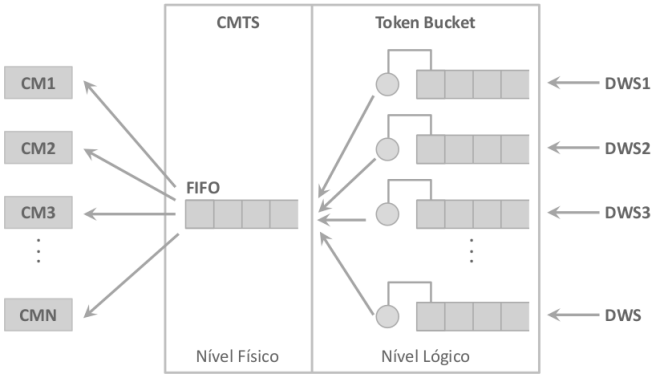

Figura 2 – Representação da transmissão entre nó do CMTS e CMs por meio de filas com taxa de transmissão gerenciadas por *leaky bucket*.

Fonte: Material de aula.

r_i : um número pseudoaleatório entre 0 e 1.

λ : A taxa de ocorrência de um evento no tempo.

Para a geração de valor para o intervalo entre eventos utilizou-se a função *expovariate* da biblioteca *random* do python:

```
import random

def rGen(lmb):
    return random.expovariate(lmb)
```

Para o modelo dos canais, criou-se uma classe para definir objetos filas do tipo FIFO. Ou seja, o primeiro pacote a chegar será o primeiro a ser extraído da fila, e outra classe para o registro dos pacotes.

```
class Queue:
    def __init__(self, maxSize):
        self.items = [] #Itens atualmente na fila
        self.discarted = [] #Itens descartados
        self.log = [] #Registro dos itens que passaram pela fila
        self._maxSize = maxSize

    def isEmpty(self):
        return self.items == []

    def enqueue(self, item):
        if self.size() < self._maxSize:
            self.items.insert(0,item)
        else:
```

```
15              self.discard(item)
16
17      def dequeue(self):
18          return self.items.pop()
19
20      def size(self):
21          return len(self.items)
22
23      def discard(self, item):
24          self.discarted.append(item)
25
26      def discSize(self):
27          return len(self.discarted)
28
29      def register(self, item):
30          self.log.append(item)
31
32      def logSize(self):
33          return len(self.log)
34
35  class Pacote:
36      def __init__(self, tChegada, tInicioServico, tServico):
37          self.tChegada = tChegada
38          self.tInicioServico = tInicioServico
39          self.tServico = tServico
40          self.tFimServico = tInicioServico + tServico
41          self.tAtraso = tInicioServico - tChegada
```

1.2 Controle de fluxo (*traffic shapping*)

Cada assinante possui uma limitação da taxa de transmissão *downstream*. Para este simulador, foram feitas as seguintes considerações para o fluxo de entrada:

- Os 800 Mbps são compartilhados igualmente entre os n assinantes. Isto significa que a capacidade de transmissão de cada *link* é reduzida conforme o número de assinantes no nó.
- Há uma população infinita de pacotes que vem do CMTS para os CMs.
- Considerou-se um tamanho médio de pacotes de 188 Bytes ou 1504 bits. Baseado na dimensão de pacotes mpeg.

A taxa de transferência de pacotes entre nós do CMTS e cada CM é controlada por meio de um algoritmo denominado *leaky bucket*. Este algoritmo funciona com base no modelo de tráfego por processos envelope, uma forma simplificada de estimar a quantidade de dados transferidos em um determinado período. Estes modelos podem ser determinísticos ou probabilísticos, e essencialmente definem uma reta que garante um limite superior à quantidade de dados que passam pelo canal no tempo. Limitando a capacidade de transmissão do *link*.

O algoritmo *Token Bucket* consiste em um sistema de controle de chegada de pacotes em uma fila. Dois parâmetros são essenciais para a configuração deste algoritmo: A taxa de enchimento do "balde" e sua quantidade máxima de tokens. Estes parâmetros estão diretamente associados à velocidade de transmissão contratada pelo assinante e a rajada (*burst*) de dados permitida naquele canal.

A figura 3 representa o funcionamento deste controlador. Os novos pacotes de dados que chegam em uma determinada fila consomem *tokens* que garantem sua passagem. Quando não houver mais *tokens*, os pacotes que chegarem são descartados. Deste modo, a transmissão fica limitada à taxa de enchimento e o fluxo é controlado.

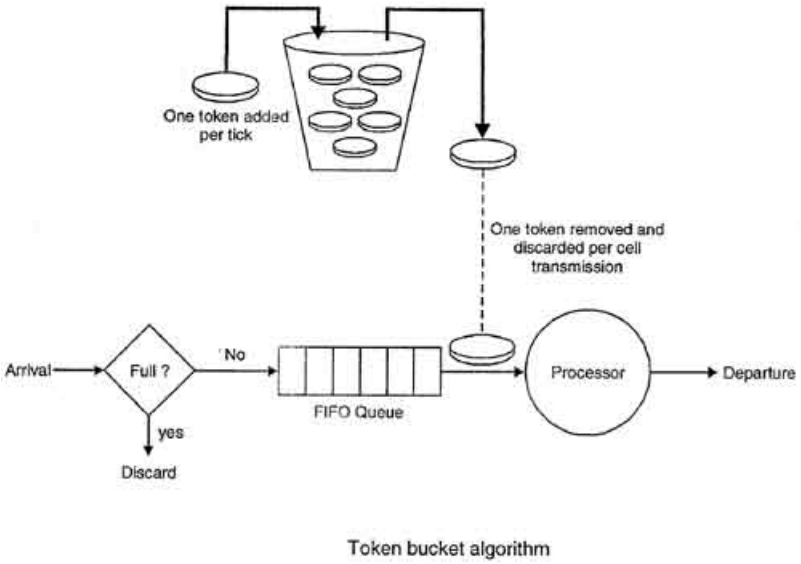

Figura 3 – Ilustração do algoritmo *token bucket*.

O algoritmo utilizado é apresentado a seguir:

```
1 class LeakyBucket(object):
2
3     def __init__(self, tokens, fill_rate, time):
4         self.capacity = float(tokens)
5         self._tokens = float(tokens)
6         self.fill_rate = float(fill_rate)
7         self.timestamp = time
```

```
8
9       def consume(self, tokens, time):
10          self.timestamp = time
11          if tokens <= self._tokens:
12              self._tokens -= tokens
13          else:
14              return False
15          return True
16
17      def get_tokens(self, time):
18          now = time
19          if self._tokens < self.capacity:
20              delta = self.fill_rate * (now - self.timestamp)
21              self._tokens = min(self.capacity, self._tokens +
                    delta)
22          self.timestamp = now
23          return self._tokens
```

1.3 Simulador

Por fim, a função de simulação foi elaborada. E é apresentada a seguir:

```
1   import pylab as pl
2
3   def simQueue(n, tSimulacao = False, txDadosLeaky = False, mu =
        False, k = False, B=False, txNodeMax = False, pkgSize = False):
4       if not tSimulacao:
5           tSimulacao = 1
6       if not txDadosLeaky:
7           txDadosLeaky = float(input("Digite a taxa de dados do
                assinante (Mbps):\n"))
8       if not B:
9           B = 5 #Quantidade ãpadro de tokens/limite de buffering
10      if not mu:
11          mu = float(input("Digite a taxa de çservio da fila (Mbps)
                :\n")) #(Mbps)
12      if not txNodeMax:
13          txNodeMax = 800/n #Capacidade do link (Mbps)
14      if not pkgSize:
15          pkgSize = 188 #bytes
16      if not k:
17          k = 60 #Valor arbitrario para o tamanho da fila
```

```
18
19      #Conversoes:
20      txDadosLeaky = txDadosLeaky*1024*1024
21      txNodeMax = txNodeMax*1024*1024 #Taxa ámxima de ãtransmisso
           downstream de um ón:
22      pkgSize = pkgSize*8
23      lmbdLeaky = txDadosLeaky/pkgSize
24      #Limite de êtransferncia downstream de um ón.
25      lmbdNode = txNodeMax/pkgSize #Taxa de chegada ámxima do ón.
26
27      #logs
28      downloaded = []
29      tLog = []
30      qsLog = []
31      qdLog = []
32      qlLog = []
33
34      t = 0
35      queue = Queue(k)
36      leaky = LeakyBucket(B, lmbdLeaky, t)
37
38      while t<tSimulacao:
39          tLog.append(t)
40          qsLog.append(queue.size())
41          qdLog.append(queue.discSize())
42          qlLog.append(queue.logSize())
43
44          if queue.logSize() == 0:
45              tChegada = rGen(lmbdNode)
46          else:
47              tChegada = tChegada + rGen(lmbdNode)
48              if queue.size() >0 and tChegada > queue.items[0].
                   tFimServico:
49                  downloaded.append(queue.dequeue())
50
51          if leaky.consume(1,t):
52              if queue.size() == 0:
53                  tInicioServico = tChegada
54              else:
55                  tInicioServico = max(tChegada, queue.items[0].
                       tFimServico)
56              tServico = rGen(mu)
```

```
57              queue.enqueue(Pacote(tChegada, tInicioServico,
                    tServico))
58              queue.register(Pacote(tChegada, tInicioServico,
                    tServico))
59          else:
60              queue.discard(Pacote(tChegada, 0, 0))
61          t = tChegada
62          leaky.get_tokens(t)
63
64      atrasoVec = [i.tAtraso for i in queue.log]
65      mediaAtraso = sum(atrasoVec)/len(atrasoVec)
66      utilizacao = lmbdLeaky/mu
67
68      print("\nResultados:")
69      print("Taxa de ãtransmisso downstream para um ón do CMTS (bps)
            : %.2f"%(txNodeMax))
70      print("Taxa de ãtransmisso downstream para um ón do CMTS (
            Pacotes/s): %.2f"%(txNodeMax/pkgSize))
71      print("\nTaxa de chegada de pacotes óaps traffic shaping (bps)
            : %.2f"%(txDadosLeaky))
72      print("Taxa de chegada de pacotes óaps traffic shaping (
            Pacotes/s): %.2f"%(lmbdLeaky))
73      print("\nTaxa de çservio do CM (bits/segundo): %.2f"%(mu*1504)
            )
74      print("Taxa de çservio do CM (pacotes/segundo): %.2f"%(mu))
75      print("\nNumero de pacotes óaps traffic shapping: %d"%(queue.
            logSize()))
76      print("Volume de dados óaps traffic shapping: %.2f"%(queue.
            logSize()*pkgSize))
77      print("\nNumero de pacotes baixados: %d"%(len(downloaded)))
78      print("Volume de dados baixados(bits): %.2f"%((len(downloaded
            )*pkgSize)))
79      print("\nNumero de pacotes descartados: %d"%(queue.discSize()
            ))
80      print("Volume de dados descartados: %.2f"%(queue.discSize()*
            pkgSize))
81      print("\çãnUtilizao da fila: %f"%(utilizacao))
82      print("\nAtraso émdio na fila: %.5f s"%(mediaAtraso))
83
84      #%pylab inline
85      ppt = pl.axes() #Plot de pacotes na fila durante a çãsimulao
86      ppt.plot(tLog, qsLog)
```

```
87      ppt.set_title('úNmero de pacotes na fila x tempo')
88      ppt.set_xlabel('t (s)')
89      ppt.set_ylabel('úNmero de pacotes')
```

2 Metodologia

Apenas uma fila foi simulada detalhadamente neste trabalho. No entanto, a influência dos outros canais é representada por n, que representa o número de assinantes com quem a capacidade do nó (800 Mbps) é igualmente dividida. Portanto, para 10 assinantes utilizando a capacidade do nó, os pacotes chegam a uma taxa de 80 Mbps, ou aproximadamente 55775 pacotes por segundo. Que posteriormente será limitada pelo *token bucket*

Para aquisição de resultados e teste do simulador, realizou-se três testes em condições diferentes de utilização.

```
1 #lambda/mu muito elevada
2 simQueue(10,1,15, 1)
3
4 #lambda/mu igual a 1
5 simQueue(10,1,15, 15)
6
7 #lambda//mu << 1
8 simQueue(10,1,15, 45)
```

3 Resultados

Os resultados para estes casos específicos são apresentados a seguir. Para o primeiro caso, com utilização elevada, foram obtidas:

- Taxa de transmissão downstream para um nó do CMTS (bps): 83886080.00
- Taxa de transmissão downstream para um nó do CMTS (Pacotes/s): 55775.32
- Taxa de chegada de pacotes após traffic shaping (bps): 15728640.00
- Taxa de chegada de pacotes após traffic shaping (Pacotes/s): 10457.87
- Taxa de serviço do CM (bits/segundo): 1048576.00
- Taxa de serviço do CM (pacotes/segundo): 697.19
- Numero de pacotes após traffic shapping: 10462
- Volume de dados após traffic shapping: 15734848.00
- Numero de pacotes baixados: 646
- Volume de dados baixados(bits): 971584.00
- Numero de pacotes descartados: 55036
- Volume de dados descartados: 82774144.00
- Utilização da fila: 15.000000
- Atraso médio na fila: 0.00667 s

O estado da fila é visível na figura 4.

Para o segundo caso, com utilização igual a 1, foram obtidos os resultados:

- Taxa de transmissão downstream para um nó do CMTS (bps): 83886080.00
- Taxa de transmissão downstream para um nó do CMTS (Pacotes/s): 55775.32
- Taxa de chegada de pacotes após traffic shaping (bps): 15728640.00
- Taxa de chegada de pacotes após traffic shaping (Pacotes/s): 10457.87
- Taxa de serviço do CM (bits/segundo): 15728640.00

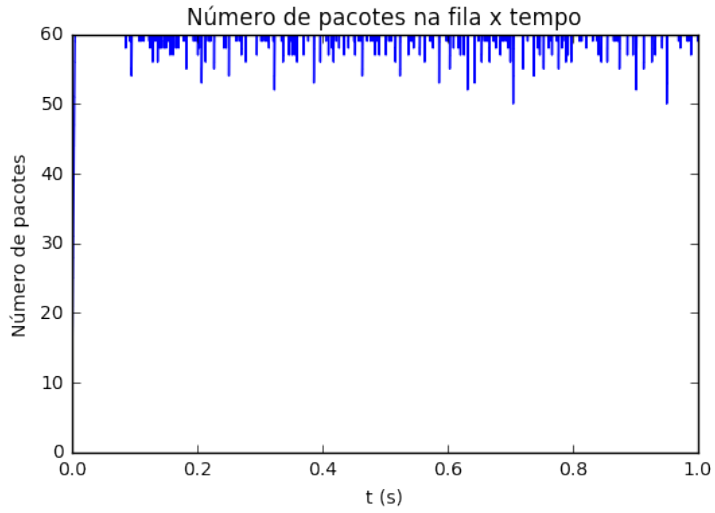

Figura 4 – Pacotes na fila durante a simulação 1.

Fonte: Produzido pelo autor.

- Taxa de serviço do CM (pacotes/segundo): 10457.87
- Numero de pacotes após traffic shapping: 10462
- Volume de dados após traffic shapping: 15734848.00
- Numero de pacotes baixados: 8599
- Volume de dados baixados(bits): 12932896.00
- Numero de pacotes descartados: 47112
- Volume de dados descartados: 70856448.00
- Utilização da fila (lmbdLeaky/mu): 1.000000
- Atraso médio na fila: 0.00019 s

O estado da fila para este segundo é apresentada na figura 5.

E por fim, para o terceiro caso:

- Taxa de transmissão downstream para um nó do CMTS (bps): 83886080.00
- Taxa de transmissão downstream para um nó do CMTS (Pacotes/s): 55775.32
- Taxa de chegada de pacotes após traffic shaping (bps): 15728640.00
- Taxa de chegada de pacotes após traffic shaping (Pacotes/s): 10457.87
- Taxa de serviço do CM (bits/segundo): 47185920.00

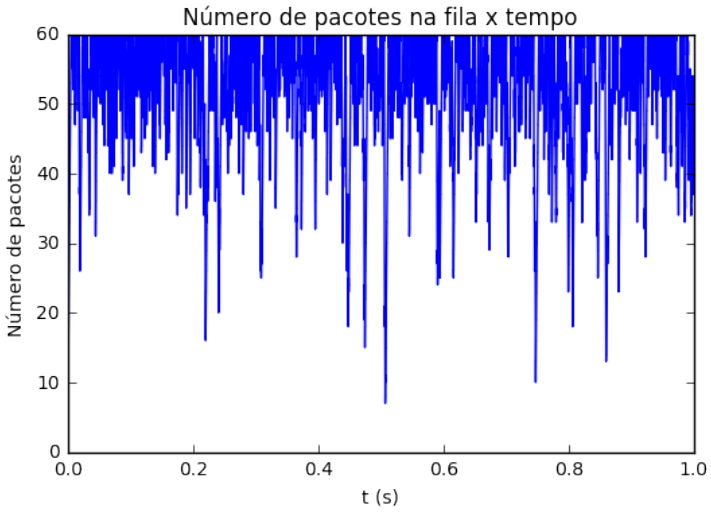

Figura 5 – Pacotes na fila durante a simulação 2.

Fonte: Produzido pelo autor.

- Taxa de serviço do CM (pacotes/segundo): 31373.62
- Numero de pacotes após traffic shapping: 10462
- Volume de dados após traffic shapping: 15734848.00
- Numero de pacotes baixados: 10460
- Volume de dados baixados(bits): 15731840.00
- Numero de pacotes descartados: 45135
- Volume de dados descartados: 67883040.00
- Utilização da fila (lmbdLeaky/mu): 0.333333
- Atraso médio na fila: 0.00000 s

O estado da fila para este segundo é apresentada na figura 6.

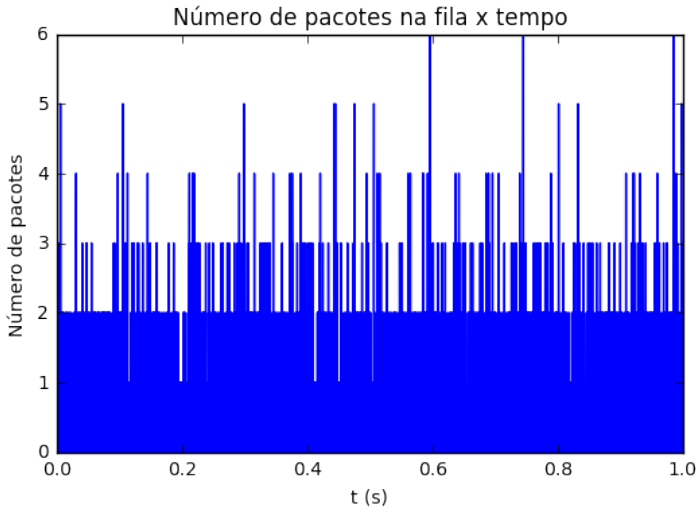

Figura 6 – Pacotes na fila durante a simulação 3.

Fonte: Produzido pelo autor.

4 Conclusão

O simulador desenvolvido representou o comportamento de uma fila Markoviana entre um número de assinantes. O sistema de *traffic shaping* funcionou da forma esperada, de acordo com os parâmetros do processo envelope, que mantiveram a *taxa de download* do assinante em um valor limitado. Com algumas modificações neste simulador, é possível utilizar dois objetos *leakyBucket* com parâmetros distintos e que entram em funcionamento durante dois intervalos. Concretizando um dual token bucket.

Foi possível notar nas simulações a variação do estado da fila no tempo. Sendo que, nos casos de utilização elevada, o tempo de espera na fila se mostrou também elevado, e a quantidade de pacotes no *buffer* permanecia próximo à capacidade (no caso igual a 60). Isto mostra que além do *traffic shapping*, é interessante utilizar sistemas de gerenciamento ativo de filas para aprimorar a performance da internet. Por fim, estes tipos de controladores podem ser adaptados para este simulador de eventos discretos em futuros trabalhos.

Referências

Volkers, L.; Barakat, N.; Darcie, T. A Simple DOCSIS Simulator. *IEICE Transactions on Communications*, v. 93, p. 1268–1271, 2010. Citado na página 5.