Token Ring

Gian Lopez, Nei Cardoso, Nicolas Curço

O Token ring é um protocolo de redes que utiliza uma topologia lógica de anel, utiliza um símbolo (token), que funciona ao circular em na lógica de anel onde as estações precisam aguardar a sua recepção para poderem transmitir. A partir daí, a transmissão é realizada durante uma pequena janela de tempo e apenas pelas que possuem o token.

Desenvolvida pela IBM em meados dos anos 1980, as redes baseadas em Token Ring estão escassas visto que este protocolo apresenta diversas desvantagens em relação às redes Ethernet, o custo para montar uma rede Token Ring é muito mais elevado e sua velocidade de transmissão é limitada a 16 Mbps, enquanto que a Ethernet permite 100 Mbps.

No entanto, apesar de utilizada em redes mais antigas e de médio e grande porte, o Token Ring possui vantagens em relação à Ethernet. A topologia lógica de anel utilizada pelo protocolo é praticamente imune a colisões de pacote. Também, o diagnóstico de problemas e a solução deles é mais simples, visto ser utilizado nessas redes obrigatoriamente hubs inteligentes.

1. *Introdução*

1.1. Introdução

1.1.1. O programa simula uma rede em anel, utilizando o protocolo UDP para a transmissão de mensagens entre as máquinas, somente uma mensagem é transmitida cada vez que o Token é recebido.

1.2. Token

1.2.1. Uma determinada máquina gera o token a primeira vez o envia à próxima máquina do anel, caso a máquina que recebe o token não tenha dados para transmitir, o token será enviado para a próxima máquina do anel e assim sucessivamente. Caso a máquina tenha pacotes aguardando envio, a primeira mensagem é retirada da fila de mensagens e é enviada para a máquina à sua direita, seguindo a ordem do anel até chegar na máquina de destino. O token é um pacote contendo apenas a string de caracteres “1234” (utf-8).

1.3. Mensagem

1.3.1. A mensagem tem 3 estados *ok*, *erro* e *nãcopiado*, se a mensagem não foi visualizada quer dizer que a máquina destino não existe

naquela rede, se devolve erro significa que a mensagem precisa ser reenviada, se devolve ok significa que a mensagem foi enviada com sucesso. A mensagem é no formato de texto é identificada pela sequência numérica ‘2345’ no primeiro campo. Pacotes seguem o formato: """id;statuscopy;origin;destination;msg"""

2. Implementação

2.1. Classes

2.1.1. Criamos a classes *Computer* e *Packet*. A classe *Computer* possui métodos referentes ao envio e recebimento de pacotes na rede e a criação e modificação de instâncias da classe Packet. A classe Packet possui apenas métodos de autogerenciamento como o construtor, o equivalente de um to_string em Python e um método que verifica se um packet é um token.

2.2. Computer

2.2.1. O método construtor da classe Computer recebe o IP e porta do próximo computador, o seu apelido, IP e porta onde ele ouvirá por pacotes e um boolean controlando se ele é o primeiro computador. Se ele for o primeiro computador ele cria um token e envia para o computador à sua direita logo no começo

2.2.2. No método start, o programa faz todas as operações lógicas necessárias para descobrir como tratar os pacotes recebidos, quando passar o token etc.

2.2.3. Os outros métodos são apenas para reciclagem de código e são chamados dentro do start.

2.3. *Packet*

2.3.1. Construtor da classe Packet:

```
def __init__(self, packet_type: str, has_been_read: str,
             origin_nick: str, dest_nick: str, text: str):
    self.packet_type = packet_type
    self.origin_nick = origin_nick
    self.dest_nick = dest_nick
    self.text = text
    self.has_been_read = has_been_read
```

2.3.2. A classe Packet me permite me referir aos atributos de um pacote de forma mais intuitiva do que índices numa lista. Também implementa um método que retorna uma byte_stream dos seus dados para que eu possa transmitir pela rede.

3. Conclusão

3.1. Arquitetura

3.1.1. O uso de uma arquitetura orientada a objetos foi extremamente útil para nós, pois permitiu excelente reciclagem de código.

3.1.2. Nossas maiores dificuldades envolveram entender como estruturar nossas classes e a professora proveu dicas úteis para nós sempre que pedimos.

3.1.3. Python3.5 foi uma ótima escolha, pois é uma linguagem muito expressiva (pouco código, muita função) e possui um ambiente interativo (REPL) muito propício para a prototipação e debugging.

3.2. A Rede

3.2.1. Não rodamos nenhum tipo de benchmark na nossa aplicação e não implementamos outras para testar contra ela, mas acreditamos que ela irá ter uma perfomance relativamente

4. Refs:

- Link to VCS repo: github.com/giancluciano/token_ring
- Wikipedia.org
- docs.python.org