Universidade Federal de Viçosa
Campus Florestal

Relatório de Trabalho Prático I de Redes de Computadores

Samuel Pedro Campos Sena - 3494
Saulo Miranda Silva - 3475

Objetivo do trabalho

O objetivo deste trabalho foi implementar simulações do funcionamento de diferentes algoritmos de controle de colisão da sub camada MAC em diferentes contextos de número de máquinas.

Implementação (pontos principais)

Nosso trabalho foi implementado em Python 3 devido a facilidade de implementação na linguagem e estruturas de dados auxiliares disponíveis.

Na implementação do Slotted Aloha, foi utilizado vetores para realizar o controle e tratamento de colisões. Cada iteração do laço *while* é equivalente a um slot de tempo. Dentro de cada iteração deste, é realizado verificações de colisões e caso alguma aconteça, um novo tempo é sorteado para as respectivas máquinas que colidem e uma nova iteração se começa. Caso apenas uma máquina deseje enviar no slot, ela é marcada como enviada com sucesso. O algoritmo acaba quando todas enviaram. Ao final da execução do algoritmo, é retornado um vetor de n posições (quantidade de máquinas no canal) contendo o tempo que cada máquina usou para enviar com sucesso, dessa forma o menor tempo é relacionado a primeira máquina que enviou com sucesso, e o maior tempo a última máquina que enviou (logo o tempo necessário para todas enviarem).

Na implementação do CSMA p-persistente com p=1%, há um laço *while* e cada iteração deste é uma passagem de slot. Quando existem colisões, a estação sofre um atraso aleatório entre 1 e n (quantidade de estações). Porém, quando não há colisões e apenas uma estação tenta enviar, esta envia com sucesso. Ao final, os tempos da primeira e a última máquina a enviar são retornados pela função em um vetor de duas posições.

Foi implementado o algoritmo com recuo binário exponencial partindo do algoritmo já implementado CSMA-p com p=1% de persistência. Logo, apenas a forma como o tratamento de colisões era realizado originalmente no código foi alterada para a implementação deste.

No *main.py*, foi implementado um pequeno menu para melhor escolha do algoritmo desejado pelo usuário. Quando cada algoritmo é escolhido, ele é executado 33 vezes em um laço e seus tempos salvos em vetores. Ainda no *main.py*, criamos uma pequena função para o cálculo da média e desvio padrão das

33 execuções de cada algoritmo chamada “CalculosEstatisticos()”. Sempre que a função é referenciada, ela recebe 2 vetores contendo tempos para primeira máquina enviar e tempo para todas as máquinas enviarem, respectivamente. Estes vetores são compostos por valores de cada uma das 33 instâncias de cada algoritmo que deverão ser comparadas nos cenários das tabelas abaixo.

Tabela comparativa, referente ao tempo (em µs) da primeira máquina enviar:

Algoritmo\N	20	40	100
Slotted ALOHA	Média:179.97 Desvio Padrão:113.13	Média:190.83 Desvio Padrão:106.53	Média:186.18 Desvio Padrão:101.45
CSMA p-persistente, p=1%	Média:763.34 Desvio Padrão:343.54	Média:636.12 Desvio Padrão:351.71	Média:695.07 Desvio Padrão:443.77
Algoritmo de recuo binário exponencial	Média:634.56 Desvio Padrão:307.98	Média:816.09 Desvio Padrão:453.62	Média:698.18 Desvio Padrão:364.92

Tabela comparativa, referente ao tempo (em µs) de todas máquinas enviarem:

Algoritmo\N	20	40	100
Slotted ALOHA	Média:3039.41 Desvio Padrão:600.75	Média:6691.68 Desvio Padrão:1088.56	Média:18031.70 Desvio Padrão:2043.33
CSMA p-persistente, p=1%	Média:20517.23 Desvio Padrão:5292.76	Média:22861.57 Desvio Padrão:4964.68	Média:35526.59 Desvio Padrão:4566.99
Algoritmo de recuo binário exponencial	Média:21412.46 Desvio Padrão:6332.58	Média:24005.04 Desvio Padrão:7416.91	Média:33677.18 Desvio Padrão:4883.80

Todo o código fonte se encontra muito bem organizado e comentado. Para executá-lo, execute o seguinte comando na pasta que contém o arquivo *main.py*:

$ python3 main.py

Ou ainda é possível executar com uso do makefile, executando o comando:

$ make