Trabalho Prático de Redes de Computadores

Relatório

Laboratório de Simulação de Conexões Cliente-Servidor e Análise de Métricas de Desempenho de Protocolos de Comunicação

Trabalho apresentado na disciplina 2035015 - Redes de Computadores

Sob orientação do Professor Edelberto Franco Silva

Departamento de Ciência da Computação
Universidade Federal de Juiz de Fora
Minas Gerais - Brasil
2022.2

Sumário

1 Especificação técnica do trabalho - Laboratório de Testes

O presente documento tem por objetivo detalhar o desenvolvimento do laboratório implementado durante a atividade prática da disciplina 2035015 - Redes de Computadores.

Foi implementado um laboratório de rede que, ao aproveitar de algumas funções disponíveis no kernel Linux, gera tráfego em 3 diferentes protocolos: TCP, UDP e ICMP.

1.1 Modelagem do problema

O problema proposto consistiu em construir um ambiente que possibilitasse o envio de tráfego utilizando o padrão cliente-servidor e de forma que fosse possível posterior reprodução e análise do experimento.

Assim, foi implementado um *script* com o objetivo de padronizar e automatizar a execução dos testes já considerando as nuances enumeradas no parágrafo anterior e que, com as devidas adaptações e/ou configurações, pudesse ser portátil para outros ambientes computacionalmente distintos.

1.2 Implementação do *script*

A implementação do *script* se deu por meio da linguagem *Shell Script*[1]. O código fonte foi dividido em funções de forma a torná-lo o mais legível possível e também, como efeito secundário, tornou-se mais clara a sequência de execução das etapas do código.

Além de verificar se possui acesso de super usuário (necessário devido à captura de pacotes feita na interface de rede), o *script* começa sua execução configurando parâmetros e iniciando serviços do ambiente de teste do lado do servidor. Após esse *setup* inicial, as ações do cliente são iniciadas, de forma que o envio de tráfego dos 3 protocolos (TCP, UDP e ICMP) ocorra, ou com o envio de *dataframes* contendo um *payload* de $32bytes$ (no caso dos dois primeiros), ou com o envio de *dataframes* padrão para a execução de um teste de conectividade (conhecido como *"ping"* - no caso do último). Ao final, os serviços são encerrados e a captura do tráfego salva em disco para consultas posteriores. Como pode ser visto na seção de exemplos do Roteiro da atividade, todo o processo de execução conta com mensagens informativas de "alto nível" informando o *status* da etapa atual.

1.3 Análise do experimento

Dado o enunciado do problema, após as etapas de implementação e execução do *script*, também foi feita uma análise do experimento realizado. Em cada uma das 4 configurações de parâmetros de ambiente foram feitas coletas de dados estatísticos. Os dados foram tabulados e então gráficos foram construídos para representá-los. Mais detalhes dessa análise podem ser vistos na Seção 2.

1.4 Dificuldades encontradas

A principal dificuldade enfrentada durante a execução da solução para o problema proposto foi relacionada à geração de pacotes de tráfego. A função *sendData* (por meio da variável $numOfPackets$) é a responsável por controlar a quantidade de pacotes que serão gerados por cada etapa de execução. Entretanto, por um motivo que não foi possível descobrir a tempo de entregar a atividade, ao se adicionar as perdas propositais de pacotes, como é possível ver nos gráficos da Seção 2.1 o número de pacotes enviados era reduzido de forma mais ou menos proporcional. Esse comportamento observado possivelmente pode ter ocorrido por conta de alguma funcionalidade do *kernel* Linux que afetou a execução do controle da fila de envio, ou pelo fato de que foi utilizada a interface virtual de rede de *loopback* da própria máquina onde ocorreram os testes.

2 Resultados obtidos

Esta seção contém alguns dos principais resultados obtidos durante a realização das atividades e tarefas deste trabalho.

2.1 Gráficos construídos

Um dos requisitos do enunciado é de que fossem construídos ao menos 3 gráficos representando a vazão, o número de pacotes enviados pelo cliente, e o número de pacotes recebido pelo servidor.

Cada um dos gráficos contém os dados (representados em escala) obtidos para cada um dos 4 testes. Cada teste foi marcado com uma cor diferente e representa uma das configurações em relação ao parâmetro de descarte proposital de pacotes (sendo a perda representada pela porcentagem escrita na legenda). Os resultados também foram agrupados por protocolo a fim de facilitar comparações diretas entre os diferentes parâmetros configurados.

No gráfico da Figura 1 estão representados os valores obtidos para a vazão de pacotes (medida em kilo *bits* por segundo).

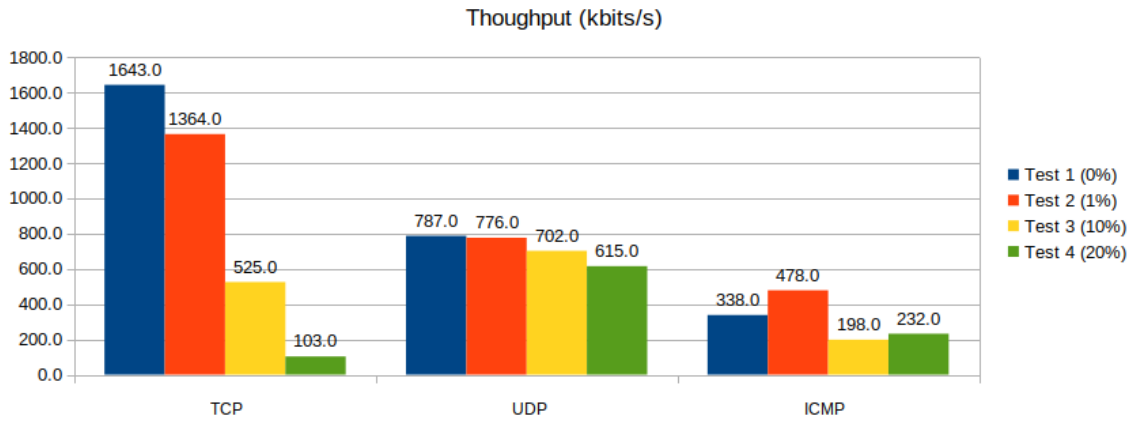

Figura 1: Gráfico apresentando a vazão medida em cada um dos 4 testes e protocolos

Um resultado importante que é possível de notar ao se observar esse gráfico, é o de que, dentre os algoritmos testados, o TCP foi o que teve a vazão mais severamente impactada a medida que a perda de pacotes aumentava. Isso provavelmente ocorreu por conta do algoritmo de controle de tráfego que está acoplado ao TCP, que deve ter interpretado os descartes como sinais de congestionamento do enlace de saída.

Já quando se analisa o UDP, é observável que a vazão decresce a medida que a perda de pacotes aumenta. Observa-se que a taxa de decrescimento é fortemente relacionada à taxa de descarte. Ao calcular-se a taxa de decrescimento entre o teste 1 (sem perdas) e os demais testes, obtemos os seguintes resultados: 1.14%, 10.80% e 21.86%, respectivamente. Provavelmente esse comportamento foi observado pois o UDP não realiza controle de tráfego, então, a medida que os pacotes foram sendo descartados, reduziu-se a vazão praticamente à mesma taxa.

Por fim tem-se o protocolo ICMP. A vazão observada ao se considerar todos os testes não parece seguir um padrão que se relacione diretamente com a taxa de descarte. Um ponto a ser notado aqui é o de que, por padrão, o utilitário de "ping" envia um pacote a cada segundo, porém para os testes esse parâmetro foi configurado para um pacote a cada 0.02 segundos. Somando-se a essa pequena modificação, variações na carga do sistema hospedeiro (ou até mesmo um congestionamento da interface virtual) podem ter influenciado nessa falta de um padrão claro.

Nos gráficos das Figuras 2 e 3 estão representadas, respectivamente, a quantidade de pacotes (em números absolutos) enviados e recebidos.

Como explicado na Seção 1.4, não foi possível determinar a causa exata do comportamento que ocorreu com o número de pacotes enviados em cada teste. Porém, se for considerado a razão entre o número de pacotes enviados e recebidos em cada um dos testes, temos os respectivos resultados de perdas de pacotes:

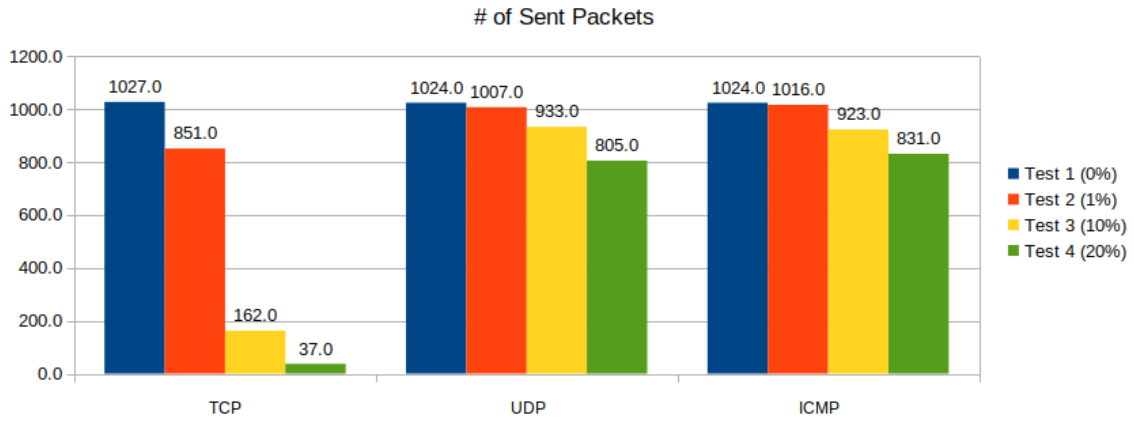

Figura 2: Gráfico apresentando o número de pacotes enviados em cada um dos 4 testes e protocolos

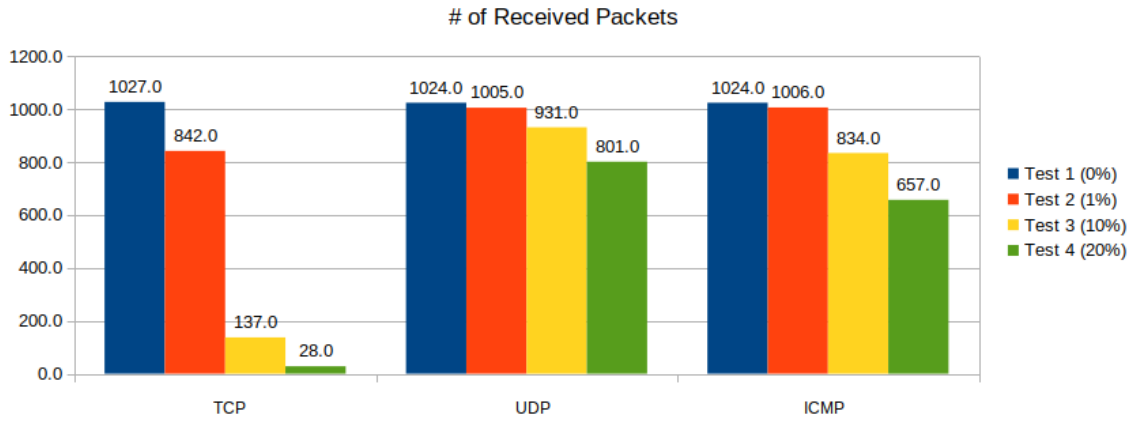

Figura 3: Gráfico apresentando o número de pacotes recebidos em cada um dos 4 testes e protocolos

- TCP: 0.00%, 1.06%, 15.43% e 24.32%
- UDP: 0.00%, 0.20%, 0.21% e 0.50%
- ICMP: 0.00%, 0.98%, 9.64% e 20.94%

Tendo em conta estes resultados, é tranquilo visualizar que os protocolos TCP e ICMP, considerando-se uma margem de erro, seguiram próximos ao padrão de perda esperado de acordo com o laboratório de teste, entretanto o protocolo UDP não se alterou em praticamente nada. Eventualmente, esse comportamento pode ter ocorrido devido à maneira com a qual a contagem dos grupos de pacotes foi realizada (por meio de filtros no programa Wireshark[2]).

3 Principais conclusões

Como principais conclusões tem-se que: é possível construir um *script* que automatize testes de geração de tráfego em rede local; executar os testes utilizando a interface virtual de *loopback*; fazer a captura do tráfego gerado; e, após análise, determinar padrões (ou a falta deles) à partir das condições do ambiente de testes. Foi possível observar que o protocolo TCP foi o mais conservador considerando os 3 indicadores analisados e que o UDP foi o protocolo mais estável mesmo ao ser testado com os diferentes parâmetros.

Referências

[1] Wikipedia. *Shell script*. Acesso em 25 de Setembro de 2022. 2022. URL: https://en.wikipedia.org/wiki/Shell_script.

[2] *Wireshark. Go deep*. Acesso em 25 de Setembro de 2022. 2022. URL: https://www.wireshark.org/.