Desenvolvimento de uma aplicação para realizar testes de desempenho de rede usando programação de *sockets* UDP e TCP.

A aplicação deve ter 3 componentes:

- **Cliente**: depois de "acionado" entra num loop, enviando uma mensagem com um tamanho configurado, aguardando a resposta, calculando o tempo de RTT. Ao iniciar, o cliente fica aguardando uma mensagem de multicast do orquestrador. A mensagem deve conter o tamanho da mensagem a ser enviada e o número de repetições. O cliente retorna uma mensagem de "ack", também por multicast. Aguada a mensagem do servidor, também avisando o "ack". Assim que a mensagem chegar, o cliente entra no loop. Ao terminar, envia uma mensagem de "terminei" também por multicast. E, por último, aguarda a mensagem de multicast do orquestrador, avisando que a mensagem "terminei" do cliente e do servidor forma recebidas. Depois de receber a mensagem, volta para o primeiro estágio.

- **Servidor**: depois de "acionado" se prepara para receber mensagens do cliente, com o tamanho configurado, e responde com uma mensagem do mesmo tamanho (um eco). Não deve perder tempo com mais nada, não faz cálculos. Mas, para depuração, pode mostrar mensagens. Ao iniciar, o servidor fica aguardando uma mensagem de multicast do orquestrador. A mensagem deve conter o tamanho da mensagem e o número de repetições. Com isso o servidor consegue se preparar para receber o tamanho de mensagem correto e o número de repetições também em sincronização com o cliente. Assim que receber a mensagem, envia uma mensagem de "ack" por multicast e aguarda o mesmo tipo de mensagem do cliente. Em seguida, entra no loop, fazendo o eco. Ao terminar, envia uma mensagem de "terminei" por multicast. Por último aguarda uma mensagem de "terminei" do orquestador. Depois de receber a mensagem, volta para o primeiro estágio.

- **Orquestrador**: recebe como parâmetro o tamanho da mensagem, e o número de repetições. Em seguida, dispara uma mensagem de multicast para o cliente e para o servidor e fica aguardando uma mensagem de "ack" do cliente e do servidor. Em seguida aguarda uma mensagem de "terminei" de ambos. Em seguida envia uma nova mensagem de "ack", avisando ter recebido a mensagem de "terminei" de ambos. Tem que verificar e se certificar de que as duas mensagens, do cliente e do servidor, chegaram. Só depois se coloca pronto para receber novo comando, para iniciar uma nova rodada de testes.
    - Uma mensagem especial do orquestrador, tipo "shutdown", faz com que cliente e servidor efetivamente terminem. Ou seja, cliente e servidor vão ter que interpretar esta mensagem e saber que não vai entrar em novo loop, mas sim dar um exit.

Cliente e servidor serão desenvolvidos em linguagens de programação diferentes, escolhido na planilha da turma. O orquestrador pode ser sua escolha.

Temos um cliente, de onde partem as mensagens para um servidor, que retorna a mensagem. Queremos fazer medidas de desempenho. Os cálculos e *logs* são feitos no cliente.

Detalhando o loop do cliente e servidor. O cliente envia uma mensagem para ao servidor, aguarda a resposta e faz os cálculos (ou só armazena os dados crus) e logs necessários. As mensagens, envio e resposta, devem ter o tamanho 1 (se for possível), 4, 8, 16, 32, 64, 128, 256, 512, 1024, 2048 ... 32768, 65536 bytes.

Cada rodada será disparada pelo orquestrador. Cliente e servidor não devem terminar (exit) entre uma rodada e outra. Devem se "reciclar". O orquestrador também não precisa parar. A cada disparo, ele aguarda o "terminei" dos dois e volta o "prompt" para o usuário.

A aplicação só termina, quando o usuário pedir para "shutdown". Neste caso, o orquestrador envia uma mensagem de "shutdown" para cliente e servidor, que devolvem um "ack" para o orquestrador e em seguida terminam (exit). O orquestrador, também aqui aguarda o "ack" dos dois elementos, cliente e servidor.

Atenção: temos três (3) máquinas que podem ser acessadas remotamente:
152.92.236.11 – orquestrador
152.92.236.16 – cliente
152.92.236.17 – servidor

Mesmo *login* nas 3 máquinas. Se você mudou a sua senha na 11 até 4/maio as 12:30 a sua senha é a mesma. Se mudou depois disso, a 16 e a 17 estão com a última senha (foram clonadas da 11).
IP multicast: 230.0.0.XX porta 99XX
IP para o teste 88XX

Para facilitar o teste em máquinas diferentes, o número IP e a porta do "servidor" devem poder ser passados como parâmetro. Para o cliente, servidor e orquestrador.

As medidas de RTT (tempo de ida e volta) devem ser salvas em um arquivo. No processo, cuidado para não introduzir mais gasto de tempo nas medidas. Ou seja, recomendamos que os valores sejam armazenados em memória (num *array*, por exemplo) e depois persistidos.

Você pode armazenar os valores "crus", e fazer o cálculo da média e desvio padrão depois, ou ir fazendo isso durante a execução e só salvar a média e o desvio padrão.

Ao coletar as informações, dados crus, ou o "kit" de média-desvio padrão, consolidar estes dados na forma sugerida abaixo, ou algo neste sentido.

UDP/10	2	4	8	16	32	64	128	...
MED	X	X	X	X	X	X	X	...
DVP	Y	Y	Y	Y	Y	Y	Y	...

Plotar a curva com a média e o desvio padrão ou o intervalo de confiança.

Estrutura das Mensagens

Seria importante para o trabalho, que vocês pensassem e propusessem a estrutura das mensagens. Mas vamos fazer isso, aqui, com um objetivo: neutralidade e interoperabilidade. No final, ousadamente ... rs ..., vamos tentar intercambiar os módulos dos grupos.

As mensagens têm que ser "neutras". Ou seja, um array de bytes. No máximo, podemos interpretar como *caracter*.

Byte 1. Tipo da mensagem:
0: orquestrador disparando um teste
1: cliente e servidor respondendo "ack" ao disparo do teste
2: orquestrador terminando tudo, shutdown
3: cliente e servidor respondendo "ack" ao shutdown
4: cliente ou servidor concluindo ("terminei")
5: orquestrador respondendo "ack" à mensagem de "terminei"
6: cliente enviando mensagem de teste
7: servidor enviando eco de teste

Byte 2: Origem
0: orquestrador
1: cliente
2: servidor

Byte 3: Tamanho
Potência de 2 expressando o tamanho da mensagem, expresso em um byte. O valor numérico. Exemplo: 2 = tamanho 2**2; 9 = tamanho 2**9. O valor numérico vai ser usado diretamente no cálculo do tamanho da mensagem. Valores úteis: 0 .. 16. Com isso conseguimos expressar todos os tamanhos desejados.

Byte 4: Repetições
Número de repetições expressa em um byte, em potência de 2. Valor numérico. Com isso conseguimos repetições de 128, 256, 512, etc. 128 já seria um bom número de repetições. Números maiores geralmente resultam em menor desvio padrão.

O resto da estrutura mensagem é variável, e está relacionada ao conteúdo "liquido", ligado ao tamanho da mensagem do teste ... 2, 4, 8, 16 ... 65534.

Testes

A turma vai fazer uma combinação de linguagens, testes (UDP, TCP-com uma conexão, abrindo e fechando conexão). Cada grupo vai ter duas linguagens e um tipo de teste. Vamos atribuir isso numa planilha compartilhada no Google Docs.

Viabilizamos interfaces de rede configuradas para 10Mbps e 100Mbps (além das interfaces "normais" de 1000Mbps. Mas, vamos usar a interface de 10Mbps apenas.

As interfaces de 10 são respectivamente: 10.0.10.11, 10.0.10.16 e 10.0.10.17 para as máquinas 11, 16 e 17.

Entrega do trabalho

Deve ser feita da seguinte forma:

- Deixar os fontes e executáveis nas VMs, prontas para serem inspecionadas e os testes reproduzidos pelo professor e numa rápida apresentação.
- Deixar na VM 11 a sua planilha com os dados. No futuro poderemos (o professor) montar um único gráfico comparando todos os testes ...
- Desenvolver um relatório. Em qualquer editor
  - O relatório de deve ter:
    - uma breve explicação da aplicação, as linguagens usadas e o protocolo testado;
    - os gráficos dos testes, com a média e desvio padrão para cada tamanho. Podem usar o Excel, GnuPlot ou outra aplicação para plotar os gráficos;
    - uma explicação para os resultados.
    - Evidências da transmissão do tamanho de mensagem correta (por exemplo um print do Wireshark ou do TCPdump) – para 128 ou 256 bytes