Framework for network emulation

1.º Alexandre Mourão
Engenharia Eletrotécnica e de Computadores
Faculdade de Engenharia da Universidade do Porto
Porto, Portugal
up201905967@edu.fe.up.pt

2.º Pedro Duarte
Engenharia Eletrotécnica e de Computadores
Faculdade de Engenharia da Universidade do Porto
Porto, Portugal
up201905050@edu.fe.up.pt

3.º Ruben Santos
Engenharia Eletrotécnica e de Computadores
Faculdade de Engenharia da Universidade do Porto
Porto, Portugal
up201905773@edu.fe.up.pt

***Resumo*—O presente documento descreve o trabalho desenvolvido no âmbito da unidade curricular Sistemas Distribuídos, cujo o principal objetivo compreendia construir uma framework para a emolução de uma rede, tendo como protocolo de comunicação UDP multicast. O desenvolvimento deste projeto passou por implementar uma API para introduzir um layer entre as camadas de aplicação e de transporte, de modo a introduzir falhas e atrasos que se encontram presentes em redes da vida real, e desenvolver uma rede ad-hoc de modo a testar e verificar o comportamento dessa mesma API. Os resultados obtidos mostram que a variação de diferentes parâmetros da rede, nomeadamente o número de nós e a distância, têm como resultado oscilações na percentagem de mensagens perdidas face às enviadas, bem como no atraso calculado desde o envio até à receção.**

***Index Terms*—framework, ad-hoc, UDP, multicast, colisões, ruído, atraso, API, topologia.**

I. Introdução

O desenvolvimento e teste de aplicações distribuídas ocorrem normalmente no mesmo computador, em computadores na mesma LAN ou em máquinas virtuais no mesmo computador, ou seja, sob condições idílicas, que apresentam uma pequena quantidade de erros e atrasos desprezáveis, que permitem obter uma performance ótima. Contudo esta performance poderá não ser verificável, quando se expõe estas aplicações a redes reais, como a internet.

Este projeto teve como objetivo lidar com esta situação, procurando criar uma API que simula possíveis perdas de pacotes, que podem resultar de colisões, de ruído eletromagnético e da distância, bem como atrasos, devidos ao tempo de processamento nos nós e ao tempo de transmissão. Esta API serviu de base para uma rede ad-hoc que apresenta uma topologia em estrela e que usa o protocolo UDP multicast como protocolo de comunicação.

Assim, o projeto pode ser dividido em três fases distintas. Numa primeira fase, desenvolveu-se a respetiva API, *RealMulticastSocket*, de introdução de erros e atrasos, recorrendo à classe *MulticastSocket* da linguagem de programação Java. Posteriormente, implementou-se a rede ad-hoc, que consistiu na criação da classe *AdHocNetworkSimulation*, cuja instância funciona como controlador da rede, e a classe *Node*, que representa os nós presentes na rede. A última secção do trabalho aborda os resultados obtidos aquando da variação do número de nós e da distância, mostrando como a rede se comporta em diferentes condições.

Em resumo, o projeto oferece não apenas uma solução prática para simulações mais realistas de aplicações distribuídas, como também fornece uma base sólida para a análise e compreensão do desempenho dessas aplicações em ambientes desafiantes.

II. Solução implementada

A. Rede ad-hoc

O desenvolvimento da rede ad-hoc para teste da API RealMulticastSocket teve duas etapas distintas: a criação de um controlador da rede, a classe *AdHocNetworkSimulation*, e a criação dos nós que estão presentes nessa rede, a classe *Node*. Foi necessário começar por definir a topologia da rede e que tipo de parâmetros seriam interessantes poder modificar por questões de estudo.

Desta forma, a rede criada apresenta uma topologia em estrela, possuindo um nó central e vários outros nós que se encontram todos à mesma distância do nó central. O protocolo de comunicação utilizado pela rede é UDP multicast. Para além disso, o utilizador consegue variar o número de nós, a distância (em metros) entre o nó central e os restantes, fatores de perda, probabilidade de erro devido ao ruído, número de threads de envio e número de mensagens enviadas por cada nó.

Durante a criação da rede, cada nó, no momento em que é gerado, junta-se ao grupo multicast e cria uma thread de receção de mensagens. Por sua vez, a instância do controlador AdHocNetworkSimulation cria um conjunto de threads de envio. Com isto, pretende-se que, ao invés de todos os nós enviarem as mensagens sequencialmente, seja possível definir um conjunto de nós para enviarem as mensagens em simultâneo, passando depois para o conjunto seguinte.

B. RealMulticastSocket

O desenvolvimento da classe para a introdução de erros e atrasos na rede, *RealMulticastSocket*, teve como base a classe *MulticastSocket* [1], que se encontra no pacote *java.net*.

Esta classe herda da classe MulticastSocket e altera os métodos send(Datagram packet) e receive(Datagram packet). Os erros introduzidos no método send estão relacionados com

o envio do pacote para o nó central, o que implica que a perda desse pacote devido algum erro afeta todos os nós do sistema, dado que o pacote nunca vai chegar a ser enviado para todos os subscritores do grupo multicast. O método send() também introduz os atrasos de processamento do nó de envio e do nó central e os atrasos que ocorrem durante a transmissão do pacote. Por sua vez, o método receive() lida com erros presentes na receção, ou seja, erros que ocorrem no canal durante a transmissão do nó central para o nó de receção e que não têm qualquer influência nos restantes nós do sistema.

Os erros e atrasos introduzidos nos métodos são:

- Erros devido ao ruído: isto é um tipo de erro que representa a perda de pacotes devido ao impacto do ruído eletromagnético. A probabilidade de ocorrência deste tipo de erro é determinística, ou seja, não varia com nenhum fator.
- Erros devido à distância: isto é um tipo de erro que representa a perda de pacotes na transmissão devido à distância entre nós. A probabilidade de ocorrência deste erro, aumenta com a distância entre dois nós, ou seja, quanto mais longe estiverem os nós, maior é a probabilidade do pacote ser perdido. A probabilidade de ocorrência deste erro em ordem à distância é calculada recorrendo à seguinte fórmula:

$$1 - e^{(-\lambda * d^2)},$$

onde:
d - distância (em metros);
λ - fator de perda.
- Erros devido a colisões: isto é um tipo de erro que ocorre devido ao envio de um pacote num canal quando esse canal já se encontra ocupado. Este erro resulta apenas na perda do último pacote enviado e não do pacote que se encontra no canal. A probabilidade deste erro acontecer depende do tráfego da rede, quanto mais tráfego a rede tiver maior é a probabilidade deste erro ocorrer.
- Atrasos de processamento do nó de transmissão: este atraso representa o processamento do nó de transmissão do pacote. Este atraso varia de forma aleatória com base numa distribuição gaussiana, de forma a modelar o momento em que o nó se encontre mais ou menos ocupado.
- Atrasos de processamento do nó de central: este atraso representa o processamento do nó central. Tal como nos restantes nós, o atraso varia de forma aleatória com base numa distribuição gaussiana, de forma a modelar o momento em que o nó se encontre mais ou menos ocupado. Contudo, para representar o processamento intensivo do nó central, o tempo de atraso será, tipicamente, maior do que o atraso associado aos nós de transmissão.
- Atrasos de transmissão: este tipo de atraso representa o atraso que ocorre na transmissão de um pacote devido à distância entre dois nó. Quanto maior for a distância entre o nó recetor e transmissor maior é o tempo de atraso, sendo que o atraso varia em ordem à distância da seguinte forma:

$$\lambda * d,$$

onde: d = distância (em metros);
λ = fator de perda.
foram implementados dois atrasos deste tipo: desde o nó de transmissão até ao nó central e desde este até ao nó receção.

III. Resultados

A seguinte figura 1 mostra a percentagem de mensagens perdidas na rede multicast desenvolvida à medida que o número de nós aumenta. No gráfico consegue-se verificar que a taxa de perda de mensagens permanece relativamente constante, apesar do aumento do número de nós. Isto pode indicar que a rede lida bem com o escalonamento em termos de manutenção da integridade das mensagens num conjunto maior de nós.

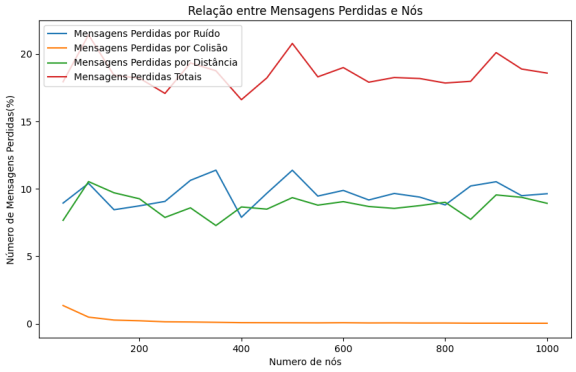

Figura 1. Estabilidade das taxas de perda de mensagens na rede.

O mesmo não acontece quando se fala no atraso de receção das mensagens. A seguinte figura 2 mostra o atraso médio na receção de mensagens à medida que o número de nós na rede aumenta. Ao contrário do gráfico anteriormente discutido sobre as taxas de perda de mensagens, o gráfico do atraso indica uma tendência clara: à medida que o número de nós aumenta, o atraso médio na receção de mensagens também aumenta. Isto sugere que o escalonamento da rede afeta a latência do sistema e, à medida que o número de nós aumenta, os requisitos de processamento da rede intensificam-se, possivelmente excedendo a capacidade da CPU para gerir eficientemente a carga acrescida, conduzindo a atrasos mais longos. Essa correlação sugere que a infraestrutura de rede e os recursos computacionais devem ser dimensionados adequadamente para manter o desempenho em configurações de rede maiores.

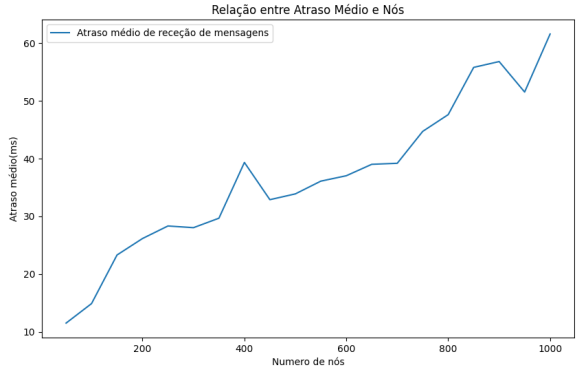

Figura 2. Atraso das mensagens em relação ao aumento dos nós.

A figura 3 demonstra um aumento linear do atraso na receção de mensagens numa rede constituída por 1000 nós, em que as mensagens são enviadas sequencialmente. É evidente que, à medida que a sequência avança, o atraso aumenta consistentemente, provando que existe um declínio no desempenho da CPU à medida que esta lida com a carga cumulativa. Esta tendência reflete a capacidade decrescente da CPU para processar as mensagens recebidas com a mesma eficiência ao longo do tempo, realçando a necessidade de recursos de computação escaláveis para gerir eficazmente redes de grande escala.

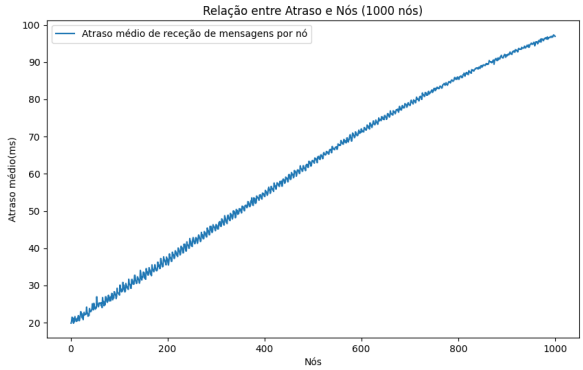

Figura 3. Aumento cumulativo do atraso numa rede de 1000 nós.

Ao contrário do gráfico anterior para 1000 nós, que mostrava um aumento linear no atraso, o gráfico da figura 4 com apenas 50 nós apresenta mais variabilidade, sugerindo que redes menores podem apresentar mais flutuação nos tempos de atraso. No entanto, a escala de atraso é significativamente menor do que a observada com 1000 nós, indicando que a CPU pode gerenciar melhor as demandas de processamento com menos nós.

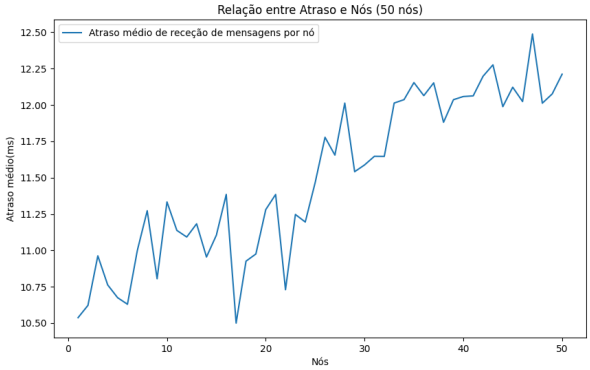

Figura 4. Aumento cumulativo do atraso numa rede de 50 nós.

Para concluir esta secção de resultados, a atenção virou-se da quantidade de nós para as distâncias variáveis, explorando a sua influência na perda de mensagens na nossa simulação de rede. Os resultados apresentados no gráfico 5 revelam um aumento acentuado da perda de mensagens à medida que a distância aumenta, destacando particularmente o crescimento logarítmico à medida que nos aproximamos do máximo teórico em que se prevê uma perda de 100%. Esta tendência reforça o efeito considerável que a distância física tem na fiabilidade da rede e a importância de

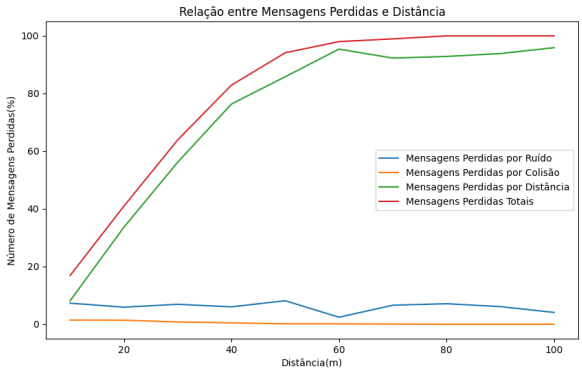

Figura 5. Impacto da distância na percentagem de perda de mensagens na simulação de rede.

Com base na análise da perda de mensagens, o gráfico da Figura 6 alarga o estudo ao correlacionar o atraso das mensagens com o aumento das distâncias, mostrando claramente um aumento evidente do atraso à medida que a distância entre os nós aumenta, o que corresponde às expectativas. É crucial notar que os valores de atraso não são realistas e foram exagerados na simulação para visualizar distintamente as mudanças no atraso. Além disso, na distância máxima teórica de 100 metros, o atraso cai para zero, não devido a uma diminuição real no tempo de transmissão, mas porque nenhuma mensagem foi enviada com sucesso a essa distância, validando assim a precisão da simulação.

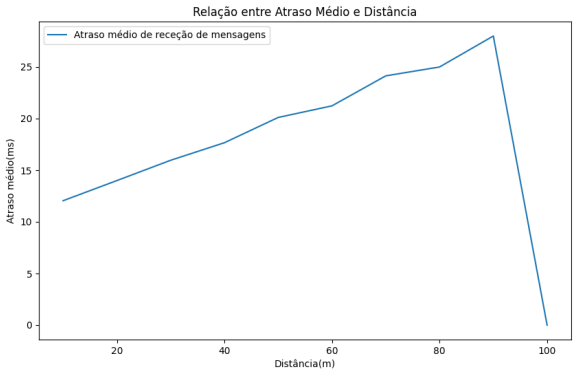

Figura 6. Variação do atraso simulado com o aumento da distância entre os nós.

IV. Conclusão

Para concluir este relatório, a nossa simulação ofereceu uma visão crítica da dinâmica dos sistemas de rede multicast. Destacou a resiliência da integridade das mensagens face à expansão da quantidade de nós, ao mesmo tempo que sublinhou o profundo impacto da distância no atraso e na perda de mensagens. Estas descobertas sublinham a necessidade de estratégias adaptativas que possam manter o desempenho da rede no meio de topologias e cargas de trabalho variáveis. O trabalho futuro deve centrar-se no aumento da eficiência computacional e no desenvolvimento de protocolos avançados para atenuar os problemas de latência evidentes em redes de nós extensos. Esta investigação contribui para uma compreensão mais ampla da conceção de redes escaláveis, fornecendo uma base para estruturas de comunicação mais robustas e fiáveis.

Contribuições: Alexandre Mourão (33.3(3)%): Construção de rede, Testes e Simulações. Pedro Duarte (33.3(3)%): Construção de rede e API *RealMulticastSocket*. Ruben Santos (33.3(3)%): Construção de rede e API *RealMulticastSocket*.

Referências

[1] “MulticastSocket (Java Platform SE 8),” Oct. 04, 2023. https://docs.oracle.com/javase/8/docs/api/java/net/MulticastSocket.html

[2] Rodrigues, Eduardo; Fonseca, Diogo; Teixeira, Rui. (2018). ”Emulação de uma Topologia Real para Teste de Algoritmos Distribuídos”. [Relatório].