Trabalho 3 - Rumor Routing | Marcelo Paulon / 2112515

Trabalho executado com Love2D 11.3b, em um macOS Big Sur (11.4) - shell: zsh - resolução de vídeo: 1920x1080; broker MQTT - Mosquitto 2.0.5, executado localmente.

Cada nó possui um número de linha e coluna, e um id sequencial. O script run.sh recebe como parâmetros obrigatórios a quantidade de nós (indexada inicialmente com 0) e quantidade máxima de nós por linha. Por exemplo `./run.sh 15 4` iria executar 16 nós, em uma grid 4x4.

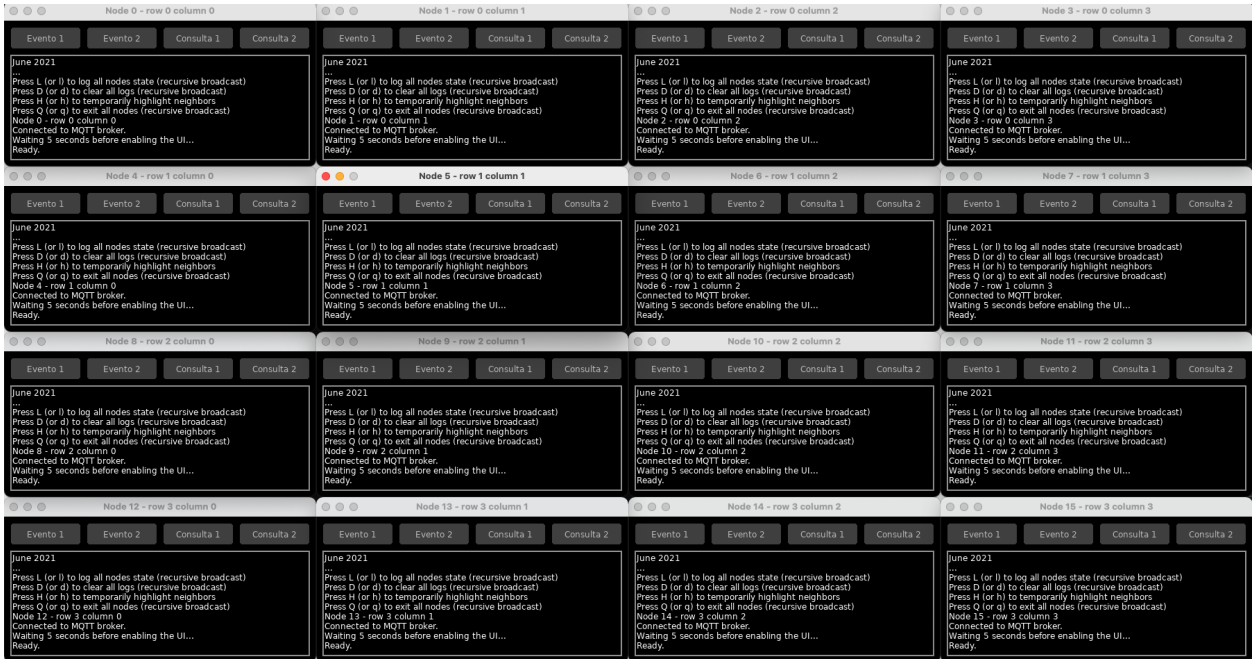

Imagem 1: interfaces em grid 4x4

A interface gráfica de cada nó espera 5 segundos para que alguns heartbeats de outros nós cheguem e então apresenta os botões "Evento 1", "Evento 2", "Consulta 1", "Consulta 2", responsáveis por notificar a percepção dos eventos (1 e 2) e solicitar a consulta por nós que tenham percebido tais eventos.

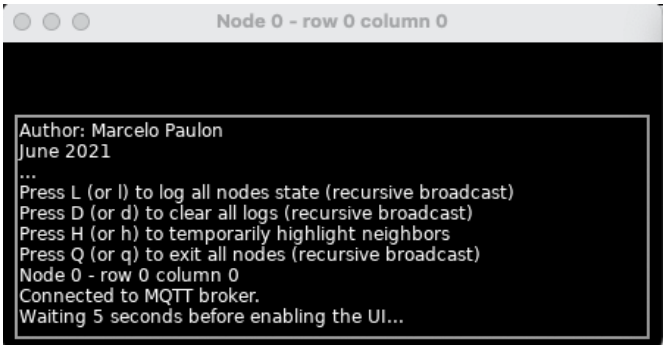

Imagem 2: nó em etapa de inicialização

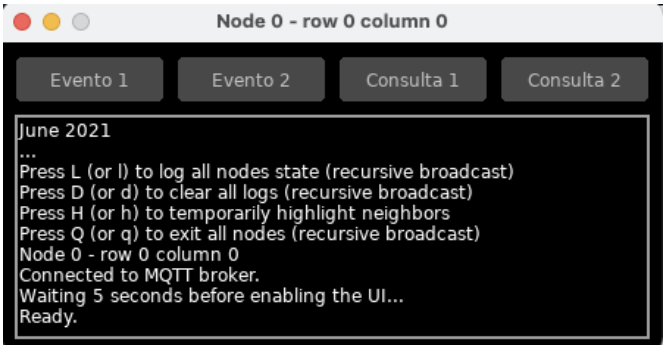

Imagem 3: nó inicializado, com botões

Adicionalmente, o usuário pode pressionar as teclas:

- L (ou l) para que todos os nós exibam no log uma representação em formato json do estado dos eventos armazenados (coleção "events" no artigo sobre Rumor Routing);
- D (ou d) para limpar os logs de todos os nós (na interface gráfica apenas; ao final da simulação os logs são salvos na pasta "results" do projeto, sem a quebra de linha em uma mesma mensagem e com o prefixo da data, linha e coluna do nó, em um arquivo de log por nó);
- H (ou h) para temporariamente mudar a cor dos vizinhos, colocá-los em destaque, o que é útil para verificar se o alinhamento das janelas está correto com relação a vizinhança dos nós;
- Q (ou q) para fechar todos os nós.

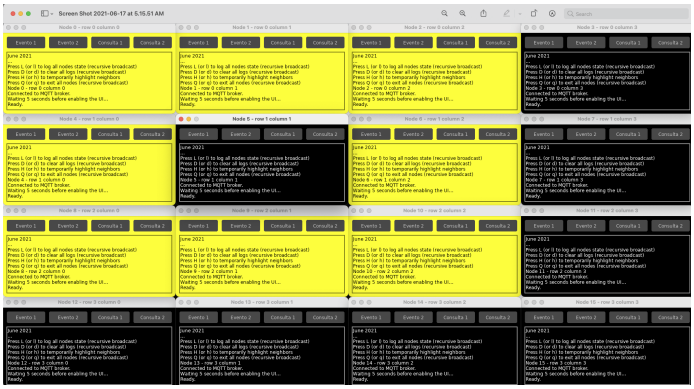

Imagem 4: highlight no centro de grid 4x4

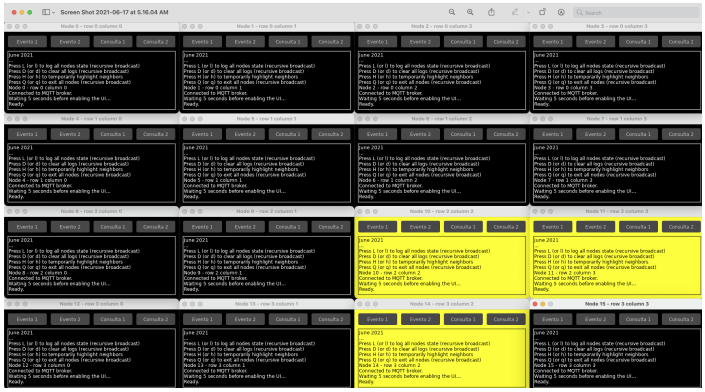

Imagem 5: highlight na borda de grid 4x4

A comunicação dos nós utiliza o protocolo MQTT, com dados serializados usando um formato textual (no caso, foi escolhido o formato JSON) para facilitar a depuração do simulador. Cada nó escreve em um tópico contendo sua linha e coluna. No modelo adotado, cada nó escreve apenas em seu próprio tópico. Todos os nós subscrevem aos tópicos de seus potenciais vizinhos, e passam a escutar mensagens desses nós, inclusive heartbeats. Os heartbeats são usados internamente, apenas para confirmar que um determinado vizinho existe, mas poderiam ser usados para outros propósitos, como por exemplo detectar nós que falharam.

Comandos de eventos percebidos são enviados como eventos MQTT "event", contendo como conteúdo o nome do evento ('1' ou '2' no caso das simulações testadas neste trabalho), e analogamente comandos de query são eventos MQTT "query", também contendo como conteúdo o nome do evento.

Comandos disparados ao pressionar uma tecla são enviados como eventos MQTT "controle", contendo a tecla que foi pressionada. Tais comandos, quando surtem efeito em todos os nós, são propagados via broadcast recursivo. Essa foi a primeira funcionalidade a ser implementada no simulador, e trouxe desafios importantes com relação a loops na comunicação, resolvidos através da adoção de um TTL enviado em cada nova mensagem (o ttl é decrementado ao longo de cada nó que a mensagem passa), assim como da adoção de um cache LRU para

mensagens processadas recentemente (conforme, inclusive, é sugerido pelo artigo sobre Rumor Routing, usando um id único por mensagem - composto neste caso da concatenação do id sequencial do nó com um separador '-' e um id sequencial das mensagens do nó).

Todo o simulador foi implementado no arquivo main.lua, que utiliza funções utilitárias open-source nos arquivos json.lua (serialização/deserialização json), lru.lua (o cache LRU), mqtt_library.lua (comunicação MQTT), e o arquivo utility.lua fornecido no enunciado do trabalho.

No arquivo main.lua, há uma seção de configuração (CONFIGS), em que existem duas variáveis booleanas - 'debug', 'fast', 'AGENT_TTL', 'QUERY_TTL'. A variável 'debug', quando configurada como true, habilita um log mais detalhado acerca dos eventos que ocorrem na simulação. Já a variável 'fast', quando configurada como `true`, faz com que as mensagens sejam publicadas assim que possível, enquanto quando configurada como `false` faz com que as mensagens sejam publicadas apenas 1-2 segundos após a solicitação, o que facilita a depuração em alguns casos. Para implementar o modo fast=false, foi utilizada uma simples fila de mensagens a serem publicadas e um timeout, verificado na função `love.update(dt)`. As configurações 'AGENT_TTL' e 'QUERY_TTL' definem os valores de TTL (time-to-live) para agentes e queries, respectivamente.

Ao longo do desenvolvimento do simulador e da implementação do Rumor Routing, algumas dificuldades foram encontradas, e observou-se que ter uma forma de pausar todos os nós poderia ter sido útil, assim como a possibilidade de controlar a simulação passo-a-passo, o que poderia ser feito por exemplo, com um botão de "continue" que poderia atuar em todos os nós ou apenas um nó específico. A natureza distribuída do simulador torna esse tipo de controle mais difícil, mas ainda assim possível de ser implementada como trabalho futuro.

Com relação ao artigo do Rumor Routing, a implementação foi similar. A forma de escolher nós para encaminhar mensagens foi feita também de forma aleatória, mas tentando evitar nós visitados recentemente pelo agente ou query, usando uma lista de nós visitados (armazenada no conteúdo da mensagem trafegada).

Ao apertar o botão de Evento, um agente sempre é gerado, enviado para um nó escolhido conforme a política acima, com TTL=8 (escolhido após alguns testes realizados com TTLs maiores e menores em que a mensagem ou sempre era recebida ou nem sempre recebida, respectivamente). O "overhearing" de agentes foi implementado conforme descrito no artigo.

Ao apertar o botão de Consulta, o comportamento da query ocorre de forma similar ao descrito pelo artigo, no entanto, ao alcançar um nó que corresponda à consulta, ocorre a etapa de volta, em que o nó que solicitou a consulta recebe o resultado da consulta (o id do nó que percebeu o evento consultado). Esse retorno foi implementado através da lista de nós visitados presente na mensagem "query" e de um índice, que começa com o valor como sendo o tamanho da lista (último índice da lista de nós visitados), começando no nó onde o evento foi encontrado. A mensagem de retorno ("query-response") é enviada de volta em cada etapa para um dos vizinhos do nó que a transporta, de forma a priorizar índices menores na lista (mais próximos

ao nó que originou a consulta). Essa priorização pode encurtar o caminho da volta, dependendo do caso.

Foram realizados diferentes testes manuais, que foram extremamente úteis para identificar bugs e avançar na implementação. Por exemplo, em um caso de teste, foi verificado que algumas das mensagens entravam em loop, por conta de uma falha na implementação do TTL.

A execução dos testes manuais foi feita inicialmente usando várias janelas abertas manualmente, até que a inicialização dos nós e seu correto posicionamento foram automatizados. Após automatizar a execução, tornou-se mais fácil de executar testes com número de nós diferentes.

Alguns dos logs avaliados estão disponíveis na pasta results:

- Pasta test1-simple-slow: simulação com 16 nós, com debug=false e fast=false, query e agent ttl = 8;
- Pasta test2-verbose-fast: simulação com 16 nós, com debug=true e fast=true, query e agent ttl = 8.

Cada pasta se refere a uma simulação executada.

Em ambas as pastas fornecidas os logs apresentam diferentes comportamentos testados:

- consultas realizadas sem que nenhum evento existisse
- evento 1 enviado uma vez
- consultas ao evento 1 em diferentes nós do grid
- evento 1 enviado mais vezes (e queries passando a ter uma distância menor, verificada pelo debug da query exibido no log e através comando L que exibe o estado de cada nó)
- a resposta da query alcançando o nó que solicitou a query
- ... e os mesmos testes para o evento 2.

Além disso, foi testada a percepção de um mesmo evento em múltiplos nós em partes diferentes do grid, e foi verificado o comportamento das queries (algumas alcançando um nó, e outras alcançando o outro).

Essas verificações foram feitas em sua maioria durante a execução da simulação, observando os logs na interface gráfica de cada nó.

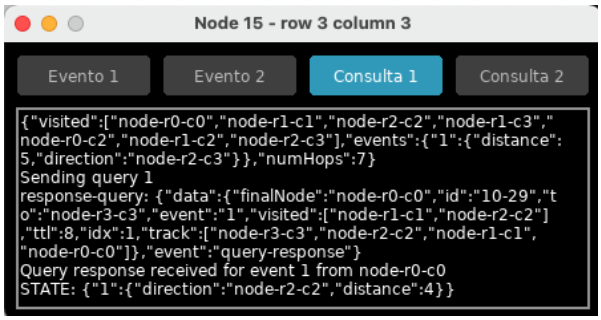

Imagem 6: nó na linha 3 coluna 3 (indexado a partir de 0), logo após receber uma resposta de consulta ao evento 1 (resultado=o nó na linha 0 coluna 0 percebeu o evento 1), e com seu estado impresso no log (distância até o evento 1 = 4, com direção a seu vizinho na linha 2 coluna 2).