CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
CAMPUS VII - UNIDADE TIMÓTEO - Engenharia da Computação

Sistemas Distribuídos
André Marcelino de Souza Neves e Leonam Teixeira de Vasconcelos

Trabalho 04:
Comunicação Peer-to-Peer

1 Introdução

Nesse relatório, descreveremos como implementamos um *chat* que opera no modelo *Peer-to-peer*.

2 Metodologia

O trabalho tem como objetivo implementar uma comunicação ponto a ponto entre todos os dispositivos de uma rede. O requisito básico é que todos os dispositivos são, ao mesmo tempo, cliente e servidor.

A implementação da comunicação *P2P* foi feita com uma metodologia de descoberta colaborativa, onde ao estabelecer conexão, os *peers* compartilham entre si a lista de *peers* que já conhecem, o que possibilita a conexão mútua entre todos os pontos da rede.

3 Desenvolvimento

3.1 Sobre sockets

A comunicação entre os dois jogadores é feita utilizando *sockets*. O protocolo de transporte utilizado foi o *TCP*, implementado no *Node.js* pela biblioteca *net* [1].

3.2 Execução

O software é executado com múltiplos terminais simultâneos, cada um com uma instância de conexão P2P. O comando para execução é:

```
$ PORT=<PORT> npm run start [...hosts]
```

Onde **<PORT>** é a porta de escuta do *peer* a ser inicializado, e **hosts** é uma lista de *peers* à qual a nova instância deverá conectar-se. Exemplo:

```
$ PORT=3005 npm run start:watch localhost:3001
```

3.3 Implementação da comunicação

O programa tem início no arquivo *index.js*, onde o *Peer* é inicializado como servidor na porta informada, e conecta-se aos *hosts* informado.

Nesse arquivo, definimos os *handlers* de alto nível (com abstração do controle *P2P*) para quando uma nova conexão é feita ou quando uma nova mensagem é recebida:

```
1  function main() {
2    port = getPort();
3
4    const hosts = getHosts();
5
6    // Inicializar Peer a partir da porta informada
7    peer = new Peer(port);
8
9    // Conectar-se à cada host informado na inicialização
10   hosts.forEach((peerAddress) => peer.connectTo(peerAddress));
11
12   // Definir funções para manipular novas conexões e dados vindos para esse Peer
13   peer.onConnection = onConnection;
14   peer.onData = onData;
15 }
```

O arquivo *Peer.js* implementa toda a lógica de inicialização do servidor, conexão com outros *Peers* e funcionalidade de descoberta colaborativa de serviço.

O processo de comunicação é inicializado da serguinte maneira:

- O *Peer* que inicializa a conexão como cliente envia uma mensagem de boas vindas, informando para o outro *peer* qual é sua porta de escuta.
- O outro *peer* também inicializa a conexão, agora sendo ele cliente, e o outro *peer*, servidor. Ao inicializar a conexão, ele também envia ao primeiro *Peer* uma mensagem de boas vindas, junto com a lista de *peers* que ele já conhece.
- O primeiro *peer* recebe a mensagem de boas vindas com a lista de *hosts* enviada pelo outro ponto, e então abre uma conexão como cliente de cada um deles.

Obviamente, na primeira conexão, a lista de *hosts* conhecidos é vazia. Nas próximas conexões, os novos *hosts* conectados são capazes de conectar-se diretamente aos *hosts* que já estavam na rede anteriormente.

Todo o código comentado será anexado no final do relatório e também está disponibilizado no **github**.

3.4 Resultados e análise

A demonstração de execução do programa foi demonstrada nesse vídeo: **link para o youtube**.

O teste foi feito com utilização de 6 *peers*, de forma a testar a estabilidade da rede mesmo quando os primeiros *hosts* se desligam, tanto a capacidade de descoberta para novos *hosts* que entram depois que a rede foi inicializada.

Inicialmente foi feito testes entre dois *peers*, onde o primeiro inicializa seu servidor, e o segundo conecta-se a ele.

Após isso, um terceiro *peer* se conecta o primeiro, e uma comunicação ponto a ponto é mantida entre todos os três.

Logo em seguida, uma rede paralela foi inicializada pelos terminais mostrados na parte inferior da tela. Dois nós conectam-se isoladamente aos três primeiros, e então o último *peer* estabelece conexão com um *peer* da primeira subrede e um *peer* da segunda subrede.

Com isso, foi possível que todos os nós tivesse conexão entre si. Também foram feitos testes com desligamento dos primeiros *peers* inicializados, para mostrar que a rede continuava com seu funcionamento normal, o que mostra que todos os *peers* estavam em conexão ponto a ponto.

4 Conclusão

Nesse trabalho, experimentamos a prática de implementação de uma arquitetura *Peer-to-Peer*, onde a comunicação não deve estar centralizada em um servidor, mas todos os *hosts* devem ser capazes de auto gerenciar a manutenção da comunicação direta entre todos os pontos.

Apesar de já existirem muitas soluções prontas para realização de comunicação P2P, a realização dessa prática nos permitiu ter contato com os desafios enfrentados para gerenciamento de uma rede descentralizada, onde são necessárias muitos controles para que a comunicação seja fluida, consistente entre todos os *hosts*.

5 Anexos

5.1 index.js

```
1  const Peer = require("./Peer");
2
3  let port;
4  let peer = null;
5
6  (function () {
7    console.log("\n\n");
8    main();
9
10   // Escutar entrada do teclado
11   process.stdin.on("data", (bufferData) => {
12     const data = bufferData.toString();
13     onNewInputMessage(data);
14   });
15 })();
16
17 function main() {
18   port = getPort();
19
20   const hosts = getHosts();
21
22   // Inicializar Peer a partir da porta informada
23   peer = new Peer(port);
24
25   // Conectar-se à cada host informado na inicialização
26   hosts.forEach((peerAddress) => peer.connectTo(peerAddress));
27
28   // Definir funções para manipular novas conexões e dados vindos para esse Peer
29   peer.onConnection = onConnection;
30   peer.onData = onData;
31 }
32
33 function getPort() {
34   const port = process.env.PORT;
35   if (!port) throw Error("PORT not found");
36
37   return port;
38 }
39
40 function getHosts() {
41   return process.argv.slice(2);
42 }
43
44 function onNewInputMessage(data) {
45   if (data === "\n") {
46     return;
47   }
48
49   if (!peer) {
50     console.log("There's no active peer");
51     return;
52   }
```

```
53
54    // Enviar mensagem digitada no terminal à todos os peers conectados a esse
55    peer.broadcastMessage({ type: "message", message: data, myPort: port });
56  }
57
58  function onConnection(socket) {}
59
60  // Receber e exibir mensagem recebida
61  function onData(socket, data) {
62    const { remoteAddress } = socket;
63    const { type, message, myPort } = data;
64
65    if (type === "message") {
66      console.log(`\n[Message from ${remoteAddress}:${myPort}]: ${message}`);
67    }
68  }
```

5.2 Peer.js

```
1   const net = require("net");
2
3   module.exports = class Peer {
4     constructor(port) {
5       this.port = port;
6       this.connections = [];
7       this.knownHosts = [];
8
9       // Inicializa o servidor com a porta informada
10      const server = net.createServer((socket) =>
11        this.handleClientConnection(socket)
12      );
13
14      server.listen(port, () => console.log("Listening on port: ", port));
15    }
16
17    // Estabelece conexão com outro Peer que atua como servidor
18
19    // Uma conexão loopback é feita quando o outro peer é servidor de uma nova conexão
20    // O outro peer realiza outra conexão, mas agora como cliente.
21    connectTo(address, sendKnownHosts = true, loopback = false) {
22      const splittedAddress = address.split(":");
23
24      if (splittedAddress.length < 2) {
25        throw Error("Invalid host address. Expected host:port ");
26      }
27
28      const port = splittedAddress.splice(-1, 1)[0];
29      const host = splittedAddress.join(":");
30
31      const socket = net.createConnection({ port, host }, () => {
32        // Adiciona Peer na lista de conexões ativas
33        this.addConnection(socket);
34        // Ativa a estuda de dados enviados pelo cliente
35        this.listenClientData(socket);
36
37        // Envia ao servidor a mensagem de boas vindas
```

```
38        this.sendWelcomeMessage(socket, this.port, loopback, sendKnownHosts);
39
40        console.log("Connected to", address);
41        console.log("\n\n");
42      });
43    }
44
45    getHostObj(host, port) {
46      return { host, port };
47    }
48
49    // Função que inclui host na lista de hosts conhecidos
50    addKnownHost(hostObj) {
51      console.log("\n[Added known host ", hostObj);
52      this.knownHosts.push(hostObj);
53    }
54
55    // Função invocada quando o Peer atual recebe lista de hosts conhecidos de outro peer
56    connectToReceivedKnownHosts(knownHosts) {
57      knownHosts.forEach((hostObj) => {
58        this.connectToNewKnownHost(hostObj);
59      });
60    }
61
62    // A função chamada para realizar conexão com algum peer descoberto
63    connectToNewKnownHost(hostObj) {
64      if (this.isKnownHost(hostObj)) {
65        return;
66      }
67
68      this.connectTo(`${hostObj.host}:${hostObj.port}`, false);
69    }
70
71    // Função que verifica se o host já é conhecido
72    isKnownHost(hostObj) {
73      if (hostObj.port === this.port) {
74        return true;
75      }
76
77      const alreadyKnownHostObj = this.knownHosts.find(
78        (knownHost) =>
79          knownHost.host === hostObj.host && knownHost.port === hostObj.port
80      );
81
82      return alreadyKnownHostObj != null;
83    }
84
85    handleClientConnection(socket) {
86      this.listenClientData(socket);
87    }
88
89    // Função que adicona novos sockets à lista de conexões ativas
90    addConnection(socket) {
91      this.connections.push(socket);
92    }
93
94    // Função que implementa a escuta de novas mensagens
95    listenClientData(socket) {
96      this.onConnection(socket);
97
98      socket.on("data", (bufferData) => {
```

```
99        const jsonData = bufferData.toString();
100       const data = JSON.parse(jsonData);
101
102       this.onData(socket, data);
103
104       this.handleWelcomeMessage(socket, data);
105     });
106   }
107
108   // Função que recebe e trata a mensagem de boas vindas
109   handleWelcomeMessage(socket, data) {
110     if (data.type !== "welcome") {
111       return;
112     }
113
114     const { remoteAddress } = socket;
115     const { myPort } = data;
116
117     // Estabelecer conexão com novos peers descobertos
118     this.connectToReceivedKnownHosts(data.knownHosts);
119
120     const hostObj = this.getHostObj(remoteAddress, myPort);
121
122     // Adiciona peer que enviou a mensagem de boas vindas à lista de hosts conhecidos
123     this.addKnownHost(hostObj);
124
125     // Se a mensagem enviada não for de uma conexão loopback, realiza a conexão loopback
126     // O valor `myPort` é a porta enviada pelo outro Peer, que indica sua porta de escuta
127     if (!data.loopback) {
128       this.connectTo(`${remoteAddress}:${myPort}`, true, true);
129     }
130   }
131
132   // Essa função desse ser sobrescrita pelo serviço que utiliza o P2P. Nesse caso, o arquivo index.js
133   // Função que manipula dados recebidos
134   onData(socket, data) {
135     throw Error("onData handler not implemented");
136   }
137
138   // Essa função desse ser sobrescrita pelo serviço que utiliza o P2P. Nesse caso, o arquivo index.js
139   // Função que manipula o evento de nova conexão
140   onConnection(socket) {
141     throw Error("onConnection handler not implemented");
142   }
143
144   // Função que envia a mensagem de boas vindas, com a porta de escuta.
145   // A porta é enviada para possibilitar que o peer do outro lado possa realizar uma nova
146   // conexão como cliente
147   sendWelcomeMessage(socket, myPort, loopback = false, sendKnownHosts) {
148     const obj = {
149       type: "welcome",
150       myPort,
151       loopback,
152       knownHosts: this.knownHosts,
153     };
154
155     this.sendMessage(socket, obj);
156   }
157
158   // Envia uma mensagem a todos os peers conectados
159   broadcastMessage(jsonData) {
```

```
160     this.connections.forEach((socket) => this.sendMessage(socket, jsonData));
161   }
162
163   // Envia mensagem a um peer individual
164   sendMessage(socket, jsonData) {
165     const data = JSON.stringify(jsonData);
166
167     try {
168       if (!socket._writableState.ended) {
169         socket.write(data);
170       }
171     } catch (e) {}
172   }
173 };
```

Referências

[1] Node.js v15.11.0 documentation - net. URL `https://nodejs.org/api/net.html`.