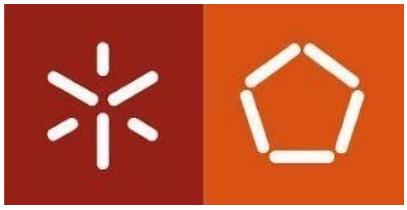
Universidade do Minho

Mestrado em Engenharia Informática



Engenharia de Serviços em Redes
1.º Ano, 1.º Semestre
Ano letivo 2021/2022

Trabalho Prático 3
Dezembro 2021

Grupo PL68

Pedro Oliveira – PG47570

Luis Pereira – PG47424

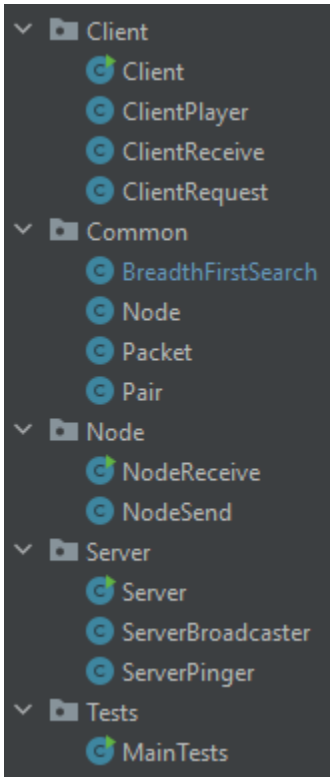
Guilherme Martins - A70782

Introdução

Com o exponencial aumento do consumo de conteúdo durante as últimas décadas, um paradigma de comunicação extremo a extremo, ou seja, o cliente receber conteúdo diretamente do servidor, mostrou-se ser extremamente ineficiente, especialmente conteúdos contínuos e em tempo real. Uma possível solução muito utilizada nos dias de hoje por serviços como Netflix e Twitch, passa pela utilização de redes sofisticadas de entrega de conteúdos (CDNs) e com serviços específicos, desenhados sobre a camada aplicacional, e por isso ditos "Over the top"(OTT).

Este projeto assenta na criação de um serviço multimédia OTT utilizando uma camada overlay configurada e gerida para contornar os problemas de congestão e limitação de recursos da rede, entregando em tempo real e sem perdas ao cliente final.

Arquitetura da solução



Para o desenvolvimento da aplicação o grupo decidiu utilizar a linguagem de programação *'Java'* utilizando o *OpenJDK 17* para a compilação do projeto.

O projeto está estruturado por *packages*, uma para o Cliente e todas as classes relacionadas com o Cliente, uma para o Servidor, e uma para os Nodos. Foram também criadas *packages* para classes comuns a vários ficheiros, ou que não pertencem necessariamente a uma das outras *packages*, como, por exemplo, classes de utilidades ou de algoritmos. Por fim, foi também utilizada uma *package* de testes com uma classe principal de testes, onde foram realizados vários testes durante e no fim do desenvolvimento do projeto para verificar se os resultados obtidos eram os esperados e se estavam corretos.

As classes *Client*, *NodeReceive* e *Server* são classes executáveis que, através de threads, executam outras classes auxiliares com uma estratégia semelhante à programação por módulos, ou seja, cada uma das classes auxiliares tem um funcionamento independente, esta estratégia permite uma melhor organização do código.

A aplicação foi desenvolvida exclusivamente fazendo recurso a protocolo TCP. Este protocolo é utilizado para as classes de Nodo, Cliente e Servidor comunicarem e interagirem dentro do sistema criado.

Especificação do protocolo TCP

Para o protocolo o grupo utilizou a seguinte implementação de *Packet*:

```
public class Packet implements Serializable {
    private String srcIp; // Where it comes from
    private String destIp; // Where it's going to
    private List<List<String>> paths; // Paths it will take
    private String content; // Message to be delivered, in this case an animation, could be an image
    private int packetId; // What is the id of this packet
```

Esta classe tem as seguintes propriedades:

- srcIp: String - Contém o IP de onde é enviado o *Packet*;
- destIp: String - Contém o IP para onde será enviado o *Packet*;
- paths: List<List<String>> - Caminhos por onde o *Packet* irá passar;
- content: String - Conteúdo do *Packet*, este conteúdo pode incluir certos comandos como ‘ping’, ‘add’, e ‘remove’, para facilitar a comunicação entre entidades do sistema;
- packetId: Integer - Identificador do *Packet* para saber a ordem pretendida de chegada;

Implementação

O protocolo segue o funcionamento seguinte:

- **Cliente**:

O cliente faz inicialmente um pedido ao Servidor a informar que pretende receber a stream (neste caso uma animação port texto) que está a ser feita, informa o servidor do seu IP e a que nodo está conectado. O cliente fica à espera que o pedido termine antes de continuar para a próxima fase.

De seguida o cliente abre a conexão com o nodo que está adjacente e inicializa o módulo do Player que irá reproduzir a animação por texto. Este Player utiliza o packetId para ordenar os packets recebidos e reproduz a uma taxa configurável as animações recebidas. Quando não tem packets, ou animações, este fica em estado “buffering…” a aguardar mais packets com animações.

Por fim, o cliente fica a aguardar conexões do nodo que está adjacente para receber packets do servidor para enviar para o Player.

- **Servidor**:

O servidor aguarda por conexões de clientes para adicioná-los à sua lista de clientes destino para os quais vai enviar as animações por texto, estes envios são realizados pelo módulo “ServerBroadcaster”.

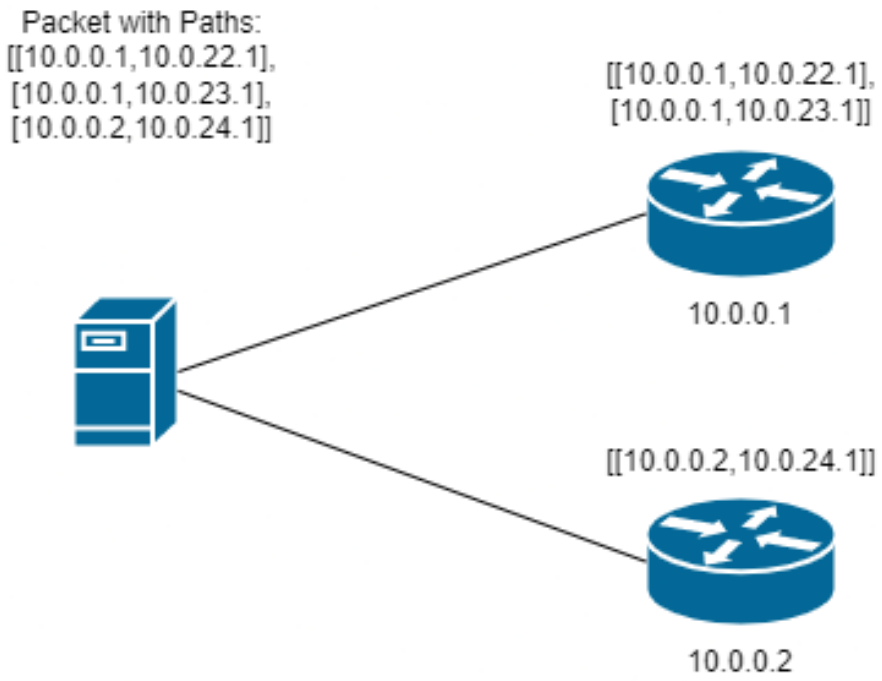
Assim, esta classe tem uma lista de clientes para o qual pretende enviar a sua stream, utilizando um algoritmo de procura “Breadth First Search” ou BFS gera os caminhos ótimos para cada um destes. Estes caminhos são enviados no Packet para o nodo para serem depois processados para os destinos respetivos.

Por fim, o servidor também dispõe de um módulo que verifica se os clientes ainda estão conectados para evitar enviar packets para clientes que estão offline.

- **Nodo**:

O nodo segue um protocolo simples para o envio de dados, este recebe conexões, e remove o seu IP de todos os paths que o packet contém. De seguida, agrupa os paths por próximo nodo a ser visitado e reenvia o packet para cada um desses grupos, ou seja, se um packet tiver 2 paths com o próximo nodo a visitar diferente, o packet é dividido por paths com o próximo nodo em comum, e depois enviado. Esta estratégia permite que sejam enviados o mínimo de packets possíveis evitando assim congestionar o sistema.

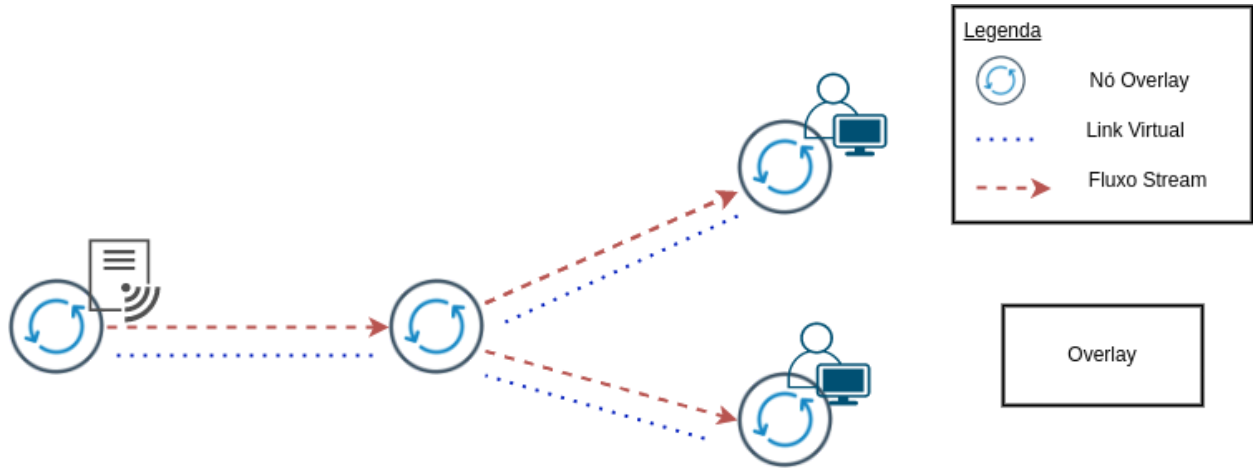
A imagem seguinte exemplifica a estratégia:

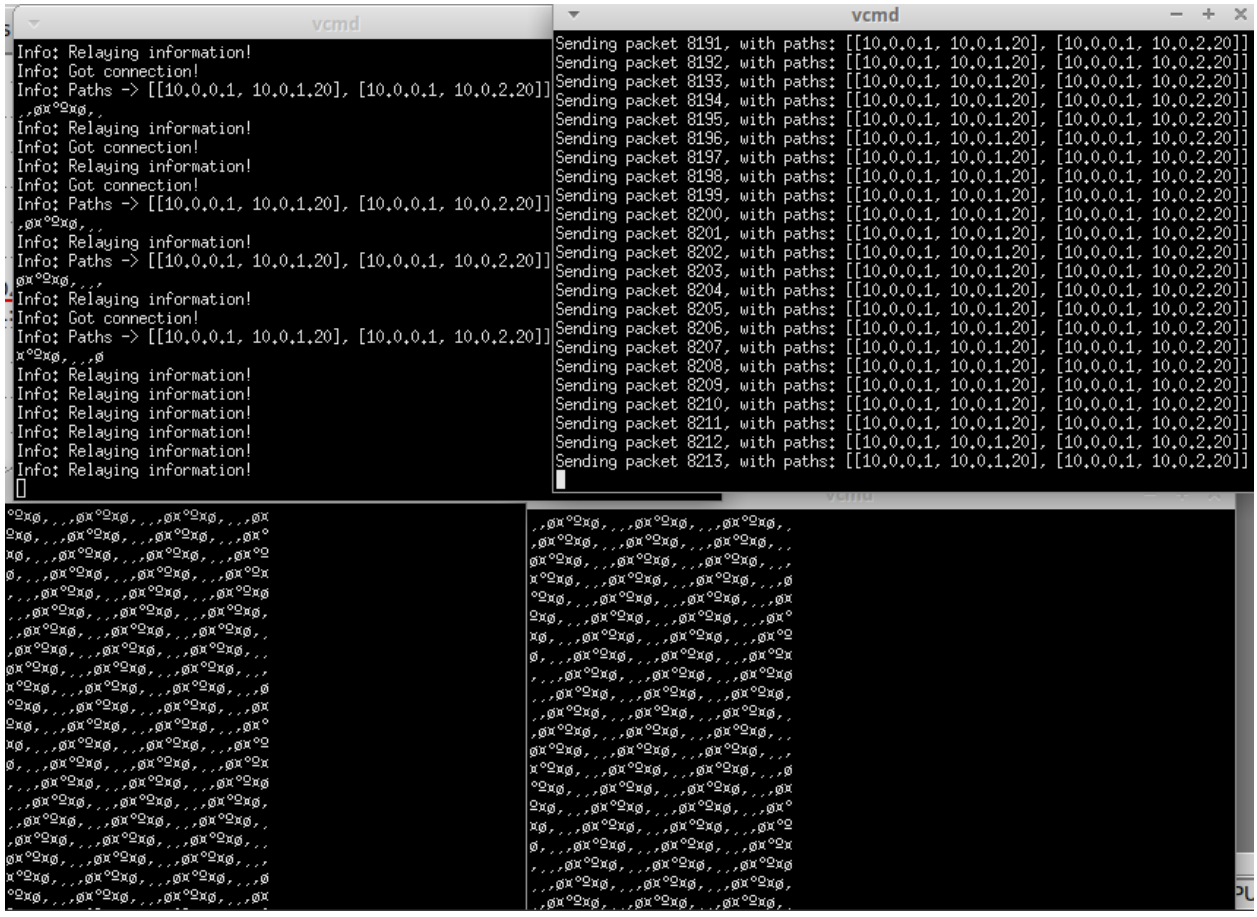


Testes e resultados

Cenário 1

Overlay com 4 nós (um servidor, 2 clientes, 1 nó intermédio)

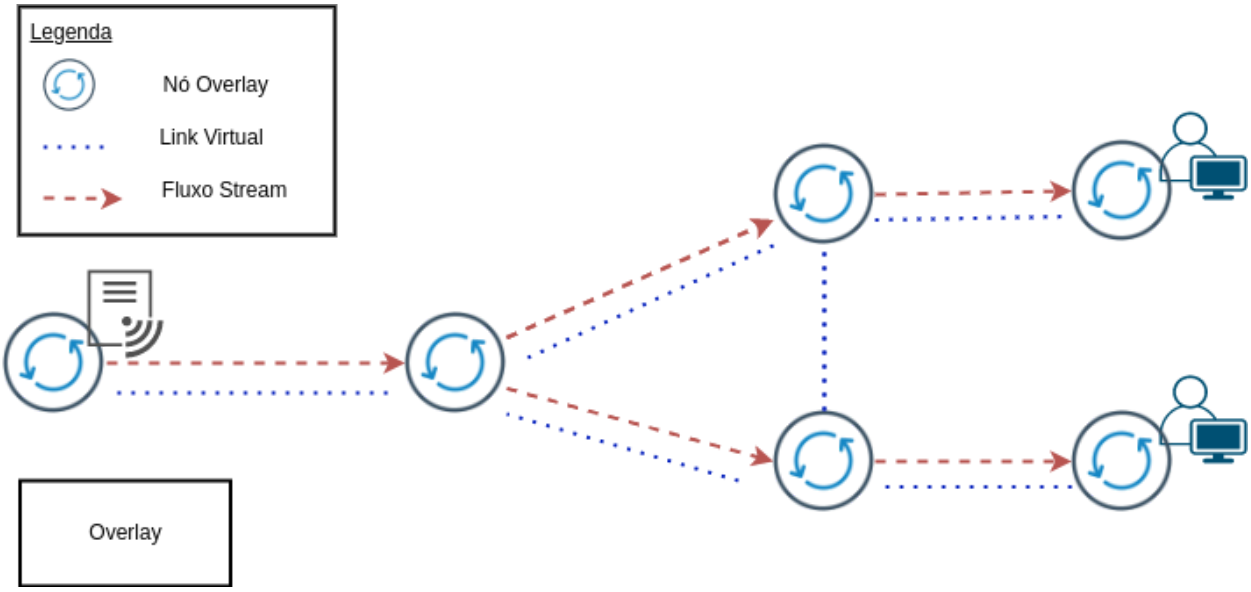


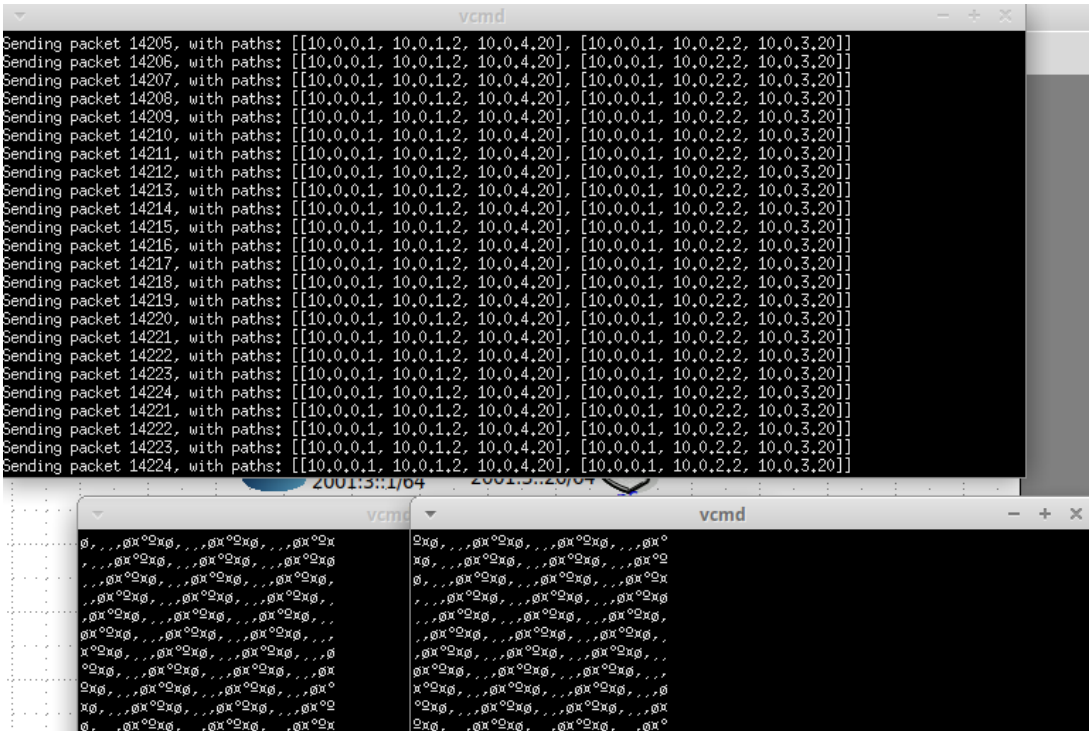


Como podemos observar nas figuras acima, o server cria somente um fluxo e o nodo faz relay do packet para os dois clientes.

Cenário 2

Overlay com 6 nós (um servidor, 2 clientes, 3 nós intermédio)

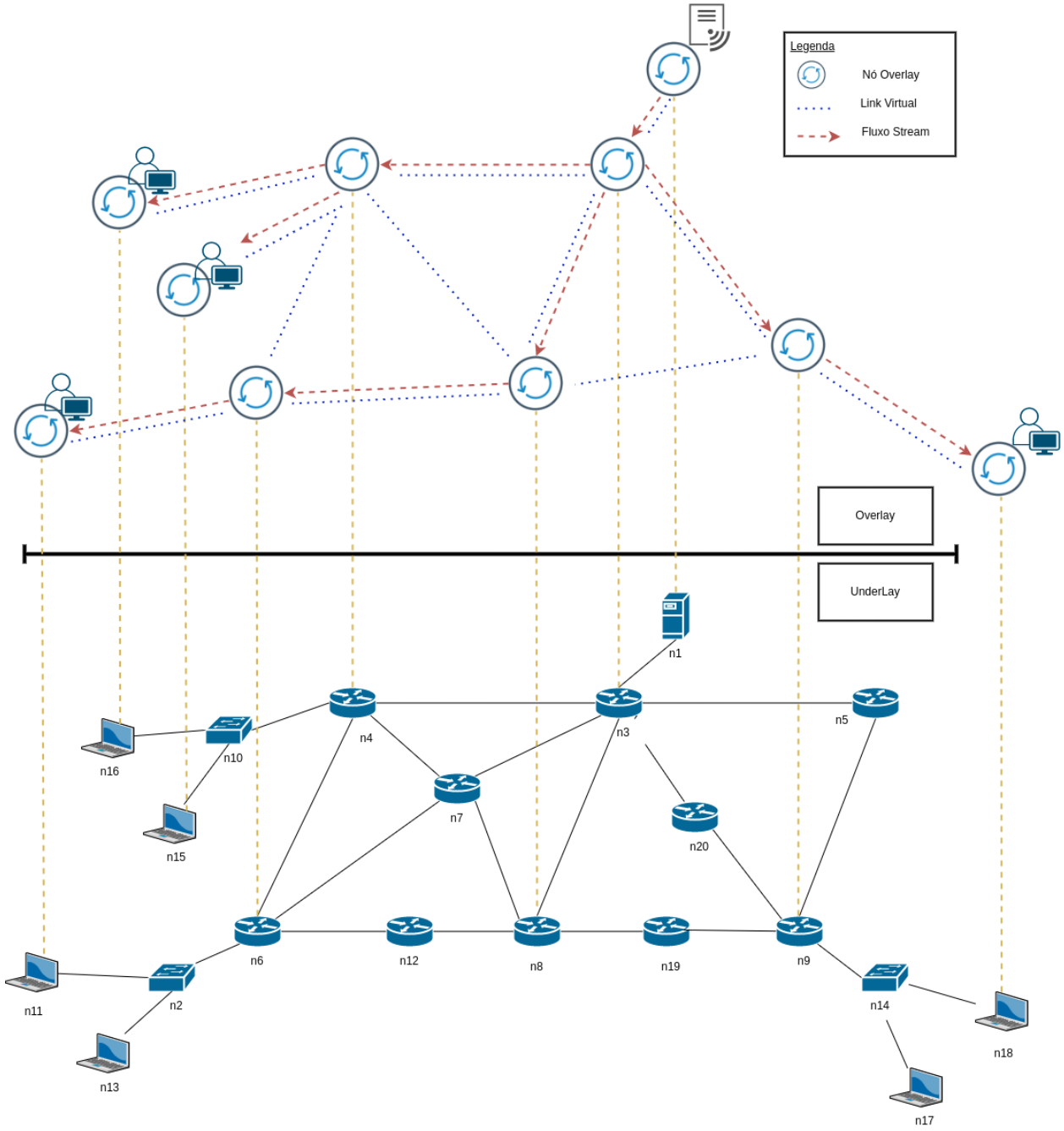


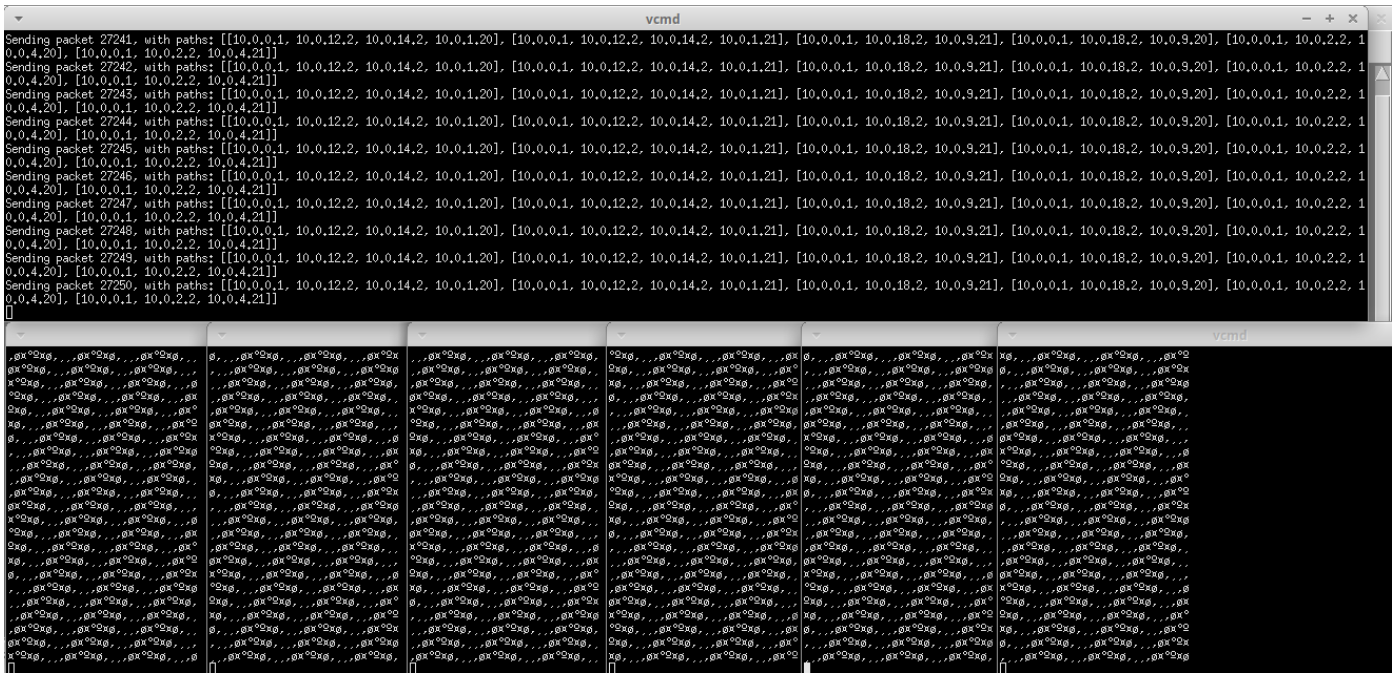


Como podemos observar nas figuras acima, o server cria somente um fluxo, o node 1 faz relay do packet para o node 2 e 3 pelo qual chegam aos 2 clientes.

Cenário 3

Overlay com 10 nós (um servidor, 4 clientes, 5 nós intermédio)





Como podemos observar nas figuras acima, o server cria somente um fluxo e paths utilizando o algoritmo breadth first. Os nodos vão manter somente 1 fluxo até encontrar um IP que difere, pelo qual vai fazer relay, criando fluxos a partir desses nodos até chegarem ao cliente de maneira ordenada.

Conclusões e trabalho futuro

Neste trabalho o grupo percebeu e começou a apreciar as estratégias utilizadas em serviços como Netflix, Hulu, Twitch, que já eram utilizados anteriormente pelos elementos do grupo mas sem a compreensão de como os pacotes deste conteúdo chegam às nossas máquinas.

Durante a discussão, planeamento e implementação deste projeto, adquirimos maior conhecimento sobre a camada aplicacional, em particular **Over-The-Top** application, assim como estratégias que permitem contornar os problemas de congestão e limitação de recursos de rede.

Certamente o grupo está mentalmente melhor preparado para adaptar trabalhos futuros com bases e estratégias de **Over-The-Top**.