Universidade Federal de Minas Gerais
Departamento de Ciência da Computação

# Trabalho Prático 3
# Protocolo de Roteamento por Vetor de Distância

**Nome:** Paula Jeniffer dos Santos Viriato
**Matrícula:** 2015114240
**Data:** 08 de novembro de 2018, quinta-feira

## 1 Introdução

Este trabalho prático trata-se da simulação de um roteador em uma topologia virtual. Cada roteador é associado a um endereço IP local,além de utilizar um soquete UDP associado ao seu respectivo endereço para trocar mensagens de roteamento e dados com outros roteadores. o trabalho desenvolvido tem uma interface de linha de comando e suporte a doid comandos de gerência da topologia de enlace virtual:

**add ip weight** — Este comando adiciona um enlace virtual entre o roteador corrente e o roteador associado ao endereço ip. Ao calcular rotas com menor distância, o peso de transmitir no enlace virtual (do roteador corrente para o roteador associado ao endereço ip) é dado por weight.

**del ip** — Este comando remove o enlace virtual entre o roteador corrente e o roteador associado ao endereço ip.

Enlaces virtuais são utilizados para transmitir dados. Os pesos são utilizados no cálculo das rotas mais curtas. Os comandos acima servem para configurar a topologia virtual.

Neste trabalho iremos executar programas (roteadores) que se associam (bind) a diferentes endereços IP e trocam mensagens por soquetes de rede. Este modo de operação é equivalente ao modo de operação entre programas executando em computadores distintos.

Neste trabalho, todas as mensagens são codificados utilizando JSON, um formato para codificação de dados amplamente utilizado em aplicações Web. O formato JSON utiliza apenas texto e pode ser facilmente inspecionado por humanos. Todas as mensagens trocadas neste trabalho são dicionários (JSON objects) com pelo menos três campos:

**type** — Especifica o tipo da mensagem, sua semântica, e quaisquer campos adicionais. Neste trabalho foi implementado três tipos de mensagem: data, update e trace.

**source** — Especifica o endereço IP do programa que originou a mensagem.

**destination** — Especifica o endereço IP do programa destinatário da mensagem.

Mensagens de dados possuem o campo type preenchido com o string "data". Mensagens de dados contêm os três campos acima e um campo payload, cujo valor é um string qualquer. O programa imprime na tela o payload de todas as mensagens data que receber.

O roteador gerado envia uma mensagem de atualização de rotas, chamada também de update, periodicamente para cada um de seus vizinhos. Um vizinho é um outro roteador que está diretamente conectado. Vizinhos são adicionados pelo comando add e removidos pelo comando del da interface iterativa de teclado.

Mensagens de update possuem o campo type preenchido com string "update". Os campos source e destination são preenchidos com o endereço IP do roteador corrente e do roteador vizinho destinatário, respectivamente. Mensagens de update também possuem um quarto campo distances contendo um dicionário informando ao vizinho as melhores rotas conhecidas pelo roteador corrente. Mais especificamente, o campo distances contêm um dicionário contendo pares chave-valor mapeando um endereço IP de destino à menor distância conhecida para alcançar aquele destino.

Além das mensagens de data e update, o roteador também processa mensagens de trace. Mensagens de trace são utilizadas para medir a rota utilizada entre dois roteadores na rede. Além dos campos type, source e destination, mensagens de trace possuem um campo hops, que armazena a lista de roteadores por onde a mensagem de trace já passou. Ao receber uma mensagem de trace, o roteador adiciona seu endereço IP ao final da lista no campo hops da mensagem. Após isso, o roteador verifica se é o destino do trace. Se o roteador não for o destino do trace, ele encaminhar a mensagem através de um dos caminhos mais curtos que conhece para o destino.

Se o roteador for o destino do trace, ele envia uma mensagem data para o roteador que originou o trace; o payload da mensagem data é uma string contendo o JSON correspondente à mensagem de trace. Em outras palavras, o resultado do trace é enviado no payload de uma mensagem data até o solicitante. O roteador cria uma mensagem de rastreamento para o roteador com endereço ip quando receber um comando de teclado com o formato abaixo:

**trace ip**

O roteador DCCRIP deve ser inicializado como segue:

**./router.py ADDR PERIOD [STARTUP]**

Onde **ADDR** indica a qual endereço IP o roteador deve se associar e **PERIOD** representa o período entre envio de mensagens de update. O parâmetro **STARTUP** é opcional e deve ser o nome de um arquivo contendo comandos de teclado para adicionar links. O roteador também recebe os parâmetros **ADDR**, **PERIOD** e **STARTUP** através de opções de linha de comando chamadas –**addr**, –**update-period** e –**startup-commands**, respectivamente. Todos os roteadores normalmente usam a porta 55151 para comunicação neste trabalho, mas esta implementação permite que a porta seja modificada pelo comando –**update-port**.

# 2 Desenvolvimento

## 2.1 Programa

O programa foi implementado em Python 3.6.7 e possui duas classes, correspondente aos dispositivos do enlace (*Device*) e ao roteador (*Route*), além de possuir três threads, correspondente a um designador de tipos dos pacotes recebidos pela conexão (os tipos podem ser "data", "update" e "trace"), ao receptor de pacotes e ao atualizador de tempo.

O *Device* possui os campos *source* (referente ao ip do roteador de onde o dispositivo veio), *destiny* (referente ao ip do dispositivo), e *weight* (referente ao "peso" do enlace). Já *Route* possui uma lista destes objetos *Device*. A classe *Route* consegue determinar as menores rotas até um dispositivo através das classes *getLessRoute* e *getRoutes*

À seguir as ideias implementadas para resolver os principais problemas dados:

## 2.2 Atualizações Periódicas

A classe *Device* possui um campo *timer*, que indica o momento da última adição (add) do dispositivo. Este campo é importante no atualizador de tempo (*updateTimer*), que a cada *pi* segundos verifica se algum dispositivo excedeu o tempo de atualização, e se for o caso, elimina o dispositivo.

Quando o roteador recebe uma mensagem de "update", ele chama a função responsável pela adição para cada um dos dispositivos informados, inclusive os já existentes. Caso um dispositivo já exista e seja dado um novo *add* para o mesmo dispositivo, o roteador atualiza seu tempo, reiniciando-o. Isto permite que a tabela de dispositivos permaneça atualizada.

## 2.3 Split Horizon

Para satisfazer esta restrição, basta que a função responsável pela atualização veja se o campo *source* do dispositivo é o mesmo que a do dispositivo que receberá o "update", o que representa que neste caso o dispositivo que receberá o "update" quem informou o dispositivo citado. Isto pode ser visto na função *updateRoutes*.

## 2.4 Balanceamento de Carga

Para balancer as cargas, a alternativa utilizada foi probabilistica, ou seja, utilizando uma função randômica de probabilidade uniforme. Quando um pacote é enviado, o roteador procura todas as menores rotas, e caso o roteador tenha mais de uma menor rota para o dispositivo de destino, ele efetua um "sorteio" entre os dispositivos possíveis. Este método pode ser visto na função *packageSending* responsável pelo envio dos pacotes referentes ao roteador.

## 2.5 Reroteamento Imediato

O reroteamento imediato é automático e intuitivo, sem necessidade de uma manutenção específica. Isto ocorre devido à implementação das classes, que aproveitam todas as in-

formações dadas por "updates", não descartando adições de mesmos dispositivos que são obtidos de enlaces diferentes. Assim, se um dispositivo é retirado do enlace, automaticamente o sistema o substitui caso haja um substituto disponível. Isto ocorre já que a cada novo envio é procurada a menor rota conhecida pelo roteador.

### 2.6 Remoção de Rotas Desatualizadas

Como dito no item "Atualizações Periódicas", uma rota é removida caso não seja atualizada (ou seja, receba um "update" referente ao dispositivo) em pi*4 segundos, na função *updateTimer*.

## 3 Análise Experimental

Foram feitas análises utilizando o Python 3.6.7 e Python 2.7.15, porém o programa só funciona corretamente no Python 3.6.7, ocorrendo vários erros no Python 2.7.15. Foram feitos testes em cada uma das funções, com e sem conexão.

Os testes primeiro ocorreram no programa sem que as conexões fossem realizadas, afim de garantir que internamente as funções funcionavam corretamente. Os resultados foram muito satisfatórios, e com a geração de logs incluída. Depois houve a conexão, e por fim a conclusão dos resultados.

Foi visto que o programa funciona corretamente com as conexões, mas devido à problemas com o sistema operacional, não foi possível simular muitos roteadores. Supõe-se que funcione corretamente, uma vez que testes unitários foram feitos e bem sucedidos.

### 3.1 Conclusão

Concluí-se que o software desenvolvido funcionou como esperado, dentro dos padrões de testes disponíveis. Foi bastante interessante no sentido de entender melhor sobre o funcionamento de roteadores e do protocolo TCP.

Houveram dificuldades na criação deste documento, devido a falta de informação e de habilidade da aluna em expôr suas ideias em forma de texto sobre os programas por ela criados.

## 4 Referências

1 - Enunciado do Trabalho Prático 3 - DCCRIP: Protocolo de Roteamento por Vetor de Distância. 27 de setembro de 2018.