Pontifícia Universidade Católica do Rio Grande do Sul
Escola Politécnica – Ciência de Dados e Inteligência Artificial

Fundamentos de Redes de Computadores

Trabalho Final – Rede Local em Anel

Gustavo Nóbrega - 21100654
Thiago Macedo – 21104690
Vitor Pires – 21101680

Porto Alegre, RS
Novembro, 2023

Rede Local em Anel

1. O Problema

O trabalho envolve a criação de uma rede de computadores disposta em uma configuração circular, conhecida como topologia de anel. O objetivo é desenvolver um sistema que habilite os computadores interligados a se comunicarem, enviando mensagens uns aos outros por meio de sockets.

Para ordenar e regular o envio dessas mensagens, é empregado um token, que é um pacote de dados especial que circula pela rede. Esse token é crucial porque assegura que apenas um computador por vez tenha a permissão de enviar suas mensagens, evitando assim qualquer confusão ou sobreposição de comunicação.

Adicionalmente, a rede deve ter a capacidade de lidar com tarefas como enfileirar novas mensagens para envio, detectar e corrigir erros nos pacotes de dados utilizando o método CRC32, retransmitir mensagens que apresentem falhas e monitorar a localização do token ao longo de sua operação.

2. A Solução

A solução proposta é um sistema de comunicação de rede que simula o funcionamento de uma topologia em anel utilizando o método de passagem de token para controle de acesso ao meio. O sistema é composto por uma série de classes em Python que trabalham em conjunto para simular o comportamento de máquinas em uma rede, a criação e o gerenciamento de pacotes de dados, e o controle de logs para monitoramento das atividades da rede.

Classe **Machine**

A classe **Machine** é o coração do sistema, representando uma máquina individual na rede. Cada instância desta classe é capaz de enviar e receber

pacotes, além de manter o controle do token. A máquina é inicializada com parâmetros que definem seu endereço na rede, apelido, tempo de posse do token, e outras configurações relevantes para a simulação, como a probabilidade de erro na transmissão e os tempos de timeout.

Gerenciamento de Conexão e Token

Cada **Machine** mantém uma conexão de socket UDP para enviar e receber pacotes. O token é um tipo especial de pacote que dá à máquina o direito de transmitir. Quando uma máquina recebe o token, ela pode optar por enviar um pacote de sua fila de mensagens ou passar o token adiante se não tiver nada para enviar.

Pacotes: **DataPacket** e **TokenPacket**

O sistema define dois tipos de pacotes: **DataPacket** para mensagens regulares e **TokenPacket** para o controle da passagem do token. A classe **Packet** é a base para esses dois tipos, fornecendo a estrutura comum necessária para a criação, manipulação e verificação de pacotes.

Controle de Erros e Logs

A simulação inclui um mecanismo de controle de erros que pode introduzir falhas nos pacotes para simular condições de rede reais. Cada máquina possui um sistema de logging que registra todas as suas atividades, permitindo o monitoramento e a depuração do processo de comunicação.

Threads e Concorrência

Para simular o comportamento assíncrono de uma rede real, o sistema utiliza threading. Cada máquina executa múltiplas threads para lidar com diferentes aspectos da comunicação, como escutar por pacotes entrantes, verificar o status do token e interagir com o usuário.

Criação de Máquinas a partir de Arquivos de Configuração

Máquinas podem ser configuradas e inicializadas a partir de arquivos de configuração, permitindo a definição de parâmetros de rede e comportamento de forma flexível e reutilizável.

Interatividade com o Usuário

O sistema permite interações em tempo real com o usuário, que pode adicionar pacotes à fila de mensagens, desligar a máquina ou visualizar a fila de mensagens atual.

Conclusão

Em resumo, a solução implementada oferece um modelo detalhado e interativo de uma rede de passagem de token, com uma arquitetura que pode ser estendida ou adaptada para diferentes cenários de simulação de rede. A modularidade das classes e a clareza dos logs facilitam o entendimento e a análise do comportamento da rede, tornando o sistema uma ferramenta valiosa tanto para fins educacionais quanto para testes de conceitos de redes de computadores.

3. O Código

Inicialização de uma Máquina

A inicialização de uma máquina é feita através do método **__init__** na classe **Machine**. Este método configura a máquina com todos os parâmetros necessários para sua operação na rede, incluindo IP, porta, apelido, tempo de posse do token, entre outros. A máquina é configurada para escutar em um socket UDP e registrar suas atividades em um arquivo de log.

```python
def __init__(self, ip: str, nickname: str, time_token: str, has_token: bool = False,
                error_probability: float = 0.2, TIMEOUT_VALUE: int = 100, MINIMUM_TIME: int = 2,
                local_ip: str = "127.0.0.1", local_port: int = 6000) -> None:

        """
        Inicializa uma máquina na rede.

        Args:
            ip (str): "ip:porta" da próxima máquina da rede
            nickname (str): Nome da máquina
            time_token (str): Tempo que a máquina segura o token
            has_token (bool, optional): Se a máquina está com o token.
            error_probability (float, optional): Probabilidade de erro na transmissão.
            TIMEOUT_VALUE (int, optional): Tempo máximo permitido sem ver o token.
            MINIMUM_TIME (int, optional): Tempo mínimo esperado entre as passagens do token.
            local_ip (str, optional): IP da máquina local.
            local_port (int, optional): Porta da máquina local. Por padrão, 6000.
        """

        # IP and Port extraction
        self.ip, self.port = self._extract_ip_and_port(ip)

        # Basic attributes
        self.nickname = nickname
        self.time_token = time_token
        self.error_probability = error_probability
        self.TIMEOUT_VALUE = TIMEOUT_VALUE
        self.MINIMUM_TIME = MINIMUM_TIME

        # Token control attributes
        self.has_token = has_token
        self.last_token_time = None if not has_token else datetime.datetime.now()
        self.controls_token = self.has_token

        # Networking setup
        self.message_queue = []
        self.socket = socket.socket(socket.AF_INET, socket.SOCK_DGRAM)
        self.socket.bind((local_ip, local_port))

        # Token generation if the machine starts with a token
        if self.has_token:
            self.generate_token()

        # Set up logging
        self.logger = logging.getLogger('MachineLogger')
        self.logger.setLevel(logging.DEBUG)
        fh = FlushingFileHandler(f"logs/{self.nickname}_log.log", "a")
        formatter = logging.Formatter('%(asctime)s - %(levelname)s - %(message)s')
        fh.setFormatter(formatter)
        self.logger.addHandler(fh)
```

Envio de Pacotes

A função **send_packet** lida com o envio de pacotes de dados ou tokens para a próxima máquina na rede. Esta função também inclui a lógica para introduzir erros nos pacotes de forma controlada, simulando as condições de uma rede real.

```python
def send_packet(self, packet: Packet, add_error_chance: bool = False):
        """
        Envia um pacote para a próxima máquina da rede.

        Args:
            packet (Packet): Pacote a ser enviad
o.          add_error_chance (bool, optional): Se deve gerar a possibilidade de erro na transmissão.
                - É usado para evitar que o erro seja gerado ao reenviar um pacote com erro ou ao passar um pacote adiante.
        """

        # Log
        if isinstance(packet, DataPacket):
            self.logger.debug("Enviando pacote de dados...")
        elif isinstance(packet, TokenPacket):
            self.logger.debug("Enviando token...")

        # Gera erro na transmissão com chance error_probability
        if isinstance(packet, DataPacket) and random.random() < self.error_probability:
            if add_error_chance == True:
                packet.crc = packet.crc[:-1] +  '0' if packet.crc[-1] == '1' else '1') # altera o crc do pacote
                packet.header = packet.create_header() # cria o header com o crc alterado
                self.logger.debug(f"Erro introduzido no pacote com destino: {packet.destination_name}")

        # envia o pacote com socket
        self.socket.sendto(packet.header.encode(), (self.ip, self.port))
```

Recebimento e Processamento de Pacotes

Quando um pacote é recebido, a função **receive_packet** é chamada, que por sua vez invoca **process_packet**. Este método é crucial, pois determina o que fazer com um pacote recebido, seja ele um token ou um pacote de dados. A função verifica se o pacote é destinado à máquina atual, se deve ser passado adiante, ou se é um pacote de retorno que contém uma confirmação de recebimento.

```python
def receive_packet(self):
        """
        Recebe um pacote através do socket. A partir do pacote recebido, chama o método process_packet.

        Returns:
            func: Função process_packet
        """
        data, _ = self.socket.recvfrom(1024) # recebe o pacote
        packet_type = Packet.get_packet_type(data.decode()) # pega o tipo do pacote
        packet = TokenPacket() if packet_type == "1000" else DataPacket.create_header_from_string(data.decode())
# cria o pacote a partir do header recebido
        self.logger.debug("Pacote recebido. Iniciando processamento...\n")
        return self.process_packet(packet) # processa o pacote
```

```python
def process_packet(self, packet: Packet):
        """
        Processa um pacote recebido. Esse método é chamado sempre que a máquina recebe um pacote. Lida com todos os possíveis cenários de pacotes recebidos.

        Args:
            packet (Packet): Pacote a ser processad
o.      """

        # recebeu um token
        if packet.id == "1000":
            self.last_token_time = datetime.datetime.now() # atualiza o tempo do último token
            self.logger.debug("Token recebido.")
            if not self.has_token:
                self.has_token = True
                self.token = packet
                self.run() # roda o processo de segurar o token e enviar mensagens
            else:
                pass

        # recebeu um pacote de dados
        elif packet.id == "2000":

            if packet.destination_name == self.nickname: # se o pacote é para mim

                self.logger.debug("Pacote para mim!")
                calculated_crc = packet.calculate_crc() # calcula crc
                if calculated_crc == packet.crc:
                    packet.error_control = "ACK" # altera o estado
                    self.logger.debug(f"Mensagem recebida com sucesso! Conteúdo: {packet.message}")
                else:
                    packet.error_control = "NACK" # altera o estado
                    self.logger.debug(f"Erro na mensagem recebida. CRC divergente!")

                self.logger.debug("Enviando pacote de volta...\n")
                self.logger.debug('-'*50+'\n')
                packet.header = packet.create_header() # cria o header
                self.send_packet(packet) # manda de volta

            elif packet.origin_name == self.nickname: # se o pacote foi enviado por mim e está voltando

                self.logger.debug("Pacote de volta!")
                self.logger.debug(f"Mensagem contida no pacote: {packet.message}\n")

                if packet.error_control == "ACK": # se a mensagem foi recebida com sucesso

                    self.logger.debug(f"Mensagem enviada foi recebida pelo destino!")
                    self.message_queue.pop(0) # tira da fila

                    self.logger.debug("pacote removido da fila")
                    self.logger.debug("passando o token...")

                    self.send_packet(self.token) # manda o token
                    self.has_token = False # não tem mais o token

                elif packet.error_control == "NACK": # se a mensagem não foi recebida com sucesso

                    self.logger.debug("Ocorreu um erro na mensagem")
                    self.send_packet(packet) # reenvia o pacote se houver erro

                elif packet.error_control == "maquinanaoexiste": # se a máquina não existe

                    self.logger.debug("Máquina não foi encontrada na rede.")

                    self.message_queue.pop(0) # tira da fila
                    self.logger.debug("Enviando o token...")

                    self.send_packet(self.token) # manda o token
                    self.has_token = False # não tem mais o token

            elif packet.destination_name == "TODOS": # se o pacote é para todos

                self.logger.debug("Pacote para todos!")

                if packet.origin_name == self.nickname: # se o pacote foi enviado por mim para todos e está voltando
                    self.logger.debug("Pacote de volta!")
                    self.logger.debug(f"Mensagem contida no pacote: {packet.message}\n")
                    self.logger.debug("pacote removido da fila")
                    self.logger.debug("passando o token...")
                    self.message_queue.pop(0) # tira da fila
                    self.send_packet(self.token) # manda o token
                    self.has_token = False # não tem mais o token

                else:
                    calculated_crc = packet.calculate_crc() # calcula crc
                    if calculated_crc == packet.crc:
                        self.logger.debug(f"Mensagem recebida com sucesso! Conteúdo: {packet.message}")
                        packet.error_control = "ACK" # altera o estado
                    else:
                        self.logger.debug(f"Erro na mensagem recebida. CRC divergente!")
                        packet.error_control = "NACK" # altera o estado

                self.logger.debug("Enviando pacote de volta...\n")
                self.logger.debug('-'*50+'\n')
                packet.header = packet.create_header() # cria o header
                self.send_packet(packet) # manda de volta

            else:
                self.send_packet(packet) # passa para o próximo
```

Controle de Token e Timeout

A função **check_token_status** é executada em uma thread separada e monitora o estado do token na rede. Se o token não for visto dentro de um intervalo de tempo especificado (**TIMEOUT_VALUE**), a função pode gerar um novo token para manter a rede em funcionamento.

```python
def check_token_status(self):
        """
        Processo que checa se o token está sendo passado corretamente. Esse processo ocorre em uma thread separada, que só é iniciada
se a máquina controla o token.
        """

        while not self.terminate_event.is_set():  # Check for the terminate_event here
            time.sleep(int(self.time_token))

            if self.last_token_time is None:
                continue

            time_since_last_token =  datetime.datetime.now() - self.last_token_time).total_seconds()
                                    (
            if time_since_last_token > self.TIMEOUT_VALUE:  # TIMEOUT_VALUE é o tempo máximo permitido sem ver o token

                self.logger.debug('\n'+'-'*50+'\n')
                self.logger.debug(f"Token não visto por muito tempo. Gerando novo token.")
                self.logger.debug('\n'+'-'*50+'\n')

                self.generate_token()
                self.last_token_time = datetime.datetime.now()
                pass

            elif time_since_last_token < self.MINIMUM_TIME:  # MINIMUM_TIME é o tempo mínimo esperado entre as passagens do token

                self.logger.debug('\n'+'-'*50+'\n')
                self.logger.debug(f"Token visto muito rapidamente. Retirando token da rede.")
                self.logger.debug('\n'+'-'*50+'\n')

                self.has_token = False
                pass
```

Interatividade

A função **user_interaction** permite que o usuário interaja com a máquina em tempo real, adicionando pacotes à fila ou desligando a máquina. Essa função roda em uma thread específica, que fica esperando por uma entrada do usuário a qualquer momento. Dessa forma, um novo pacote pode ser adicionado a fila de mensagens a qualquer momento, além do desligamento da máquina.

```python
def user_interaction(self):
        """
        Processo de interação com o usuário. O usuário pode escolher entre:
        - Adicionar um novo pacote à fila
        - Desligar a máquina
        - Mostrar a fila de mensagens atual

        Esse processo ocorre em uma thread separada. O usuário pode interagir com a máquina enquanto ela está rodando a qualquer momento.
        """

        while not self.terminate_event.is_set():
            print("\nOpções:")
            print("1. Adicionar um novo pacote à fila")
            print("2. Desligar a máquina")
            print("3. Mostrar fila de mensagens atual")
            choice = input("Digite sua escolha: ")

            if choice == "1":
                print("Que tipo de pacote você deseja enviar? Digite token (1000) ou dados (2000).")
                tipo = input("Digite o tipo do pacote: ")
                if tipo == "2000":
                    destination_name = input("Digite o nome do destino: ")
                    message = input("Digite a mensagem: ")
                    new_packet = DataPacket(origin_name=self.nickname, destination_name=destination_name, error_control="maquinanaoexiste", message=message)
                    print(f"Pacote adicionado à fila para {destination_name} com a mensagem: {message}")
                elif tipo == "1000":
                    new_packet = TokenPacket()
                    print(f"Token adicionado à fila.")
                else:
                    print("Tipo de pacote inválido. Por favor, tente novamente.")

                self.add_packet_to_queue(new_packet)

            elif choice == "2":
                print("Desligando a máquina...")
                self.terminate_event.set()
                self.stop_listening()
                print("Desligamento da máquina concluído.")
                sys.exit(0)

            elif choice == "3":
                print("Fila de mensagens atual:")
                for packet in self.message_queue:
                    print(packet.message)

            else:
                print("Escolha inválida. Por favor, tente novamente.")
```

Conclusão do Código

Essas funções são apenas uma parte do código, mas são vitais para o funcionamento do sistema. Elas demonstram a complexidade e a eficiência do sistema de comunicação em rede proposto, onde cada componente trabalha em conjunto para simular uma topologia de rede em anel robusta e realista. A modularidade e a clareza do código facilitam a expansão ou modificação do sistema para atender a diferentes necessidades ou para incorporar novas funcionalidades.

4. Resultados

A seguir, são apresentados os logs de duas máquinas **‘bob’** e **‘ace’**, criadas em uma execução teste simulando uma rede com dois computadores.

Log da máquina ‘bob’:

```
2023-11-06 12:06:02,640 - DEBUG - Máquina bob iniciada.
2023-11-06 12:06:02,640 - DEBUG - ------------------------------------------------
                        -
2023-11-06 12:06:02,640 - DEBUG - Máquina bob possui o token. Dormindo por 5 segundo
                                                                              s...
2023-11-06 12:06:07,645 - DEBUG - Enviando token...
2023-11-06 12:06:07,646 - DEBUG - Token enviado.
2023-11-06 12:06:07,646 - DEBUG - ------------------------------------------------
                      -
2023-11-06 12:06:12,650 - DEBUG - Pacote recebido. Iniciando processamento...

2023-11-06 12:06:12,650 - DEBUG - Pacote para mim!
2023-11-06 12:06:12,650 - DEBUG - Mensagem recebida com sucesso! Conteúdo: oi bob 1
2023-11-06 12:06:12,650 - DEBUG - Enviando pacote de volta...

2023-11-06 12:06:12,650 - DEBUG - ------------------------------------------------
                      -
2023-11-06 12:06:12,651 - DEBUG - Enviando pacote de dados...
2023-11-06 12:06:12,651 - DEBUG - Pacote de dados enviado.
2023-11-06 12:06:12,651 - DEBUG - ------------------------------------------------
                      -
2023-11-06 12:06:12,652 - DEBUG - Pacote recebido. Iniciando processamento...

2023-11-06 12:06:12,652 - DEBUG - Token recebido.
2023-11-06 12:06:12,652 - DEBUG - segurando o token por 5 segundo
2023-11-06 12:06:17,657 - DEBUG - enviando mensagem...   s...
2023-11-06 12:06:17,657 - DEBUG - Enviando pacote de dados...
2023-11-06 12:06:17,657 - DEBUG - Pacote de dados enviado.
2023-11-06 12:06:17,657 - DEBUG - ------------------------------------------------
                      -
2023-11-06 12:06:17,658 - DEBUG - Pacote recebido. Iniciando processamento...

2023-11-06 12:06:17,658 - DEBUG - Pacote de volta!
2023-11-06 12:06:17,658 - DEBUG - Mensagem contida no pacote: oi ace 1

2023-11-06 12:06:17,658 - DEBUG - Mensagem enviada foi recebida pelo destino!
2023-11-06 12:06:17,658 - DEBUG - pacote removido da fila
2023-11-06 12:06:17,658 - DEBUG - passando o token...
2023-11-06 12:06:17,658 - DEBUG - Enviando token...
2023-11-06 12:06:17,658 - DEBUG - Token enviado.
2023-11-06 12:06:17,658 - DEBUG - ------------------------------------------------
                      -
```

Log da máquina ‘bob’: