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 da Solução

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
class Machine:

    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:

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

        # 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 = socke .socket(socket.AF_INET, socke .SOCK_DGRAM)
        self.socket.bind((local_ip, local_port))   t

        # 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.

```
def send_packet(self, packet: Packe , add_error_chance: bool = False):
        # Gera erro na transmissão com chance error_probability
        if isinstance(packet, DataPacket) and rando .random() < self.error_probability:
            if add_error_chance == True:     m
                bit_to_invert = rando .randint(0, 31) # escolhe um bit aleatório para inverter
                mask = 1 << bitmto_invert # cria uma máscara para inverter o bit
                packet.crc ^= mask # inverte o bit usando xor
                packet.header = packe .create_header() # cria o header com o crc alterado
                                t
        # 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.

```
def receive_packet(self):
        data, _ = self.socket.recvfrom(1024) # recebe o pacote
        packet_type = Packe .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
        if isinstance(packet, DataPacket):
            packet.crc = int(data.decode().split(":")[3]) # pega o crc do pacote
        return self.process_packet(packet) # processa o pacote
```

```
def process_packet(self, packet: Packe ):
        # recebeu um token      t
        if packe .id == "1000":
          t 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 = packe
                self.run() #troda o processo de segurar o token e enviar mensagens
            else:
                self.send_packet(self.token)

        # recebeu um pacote de dados
        elif packe .id == "2000":
            t
            if packe .destination_name == self.nickname: # se o pacote é para mim
              t
                calculated_crc = packe .calculate_crc() # calcula crc
                if calculated_crt == packe .crc:
                    packet.error_control = "ACK" # altera o estado
                else:
                    packet.error_control = "NACK" # altera o estado

                packet.header = packe .create_header() # cria o header
                self.send_packet(packet) # manda de volta

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

                if packe .error_control == "ACK": # se a mensagem foi recebida com sucesso
                  t
                    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
                    self.last_token_time = datetime.datetime.now() # atualiza o tempo do último token

                elif packe .error_control == "NACK": # se a mensagem não foi recebida com sucesso
                    t
                    self.message_queue[0].crc = self.message_queue[0].calculate_crc()
                    self.message_queue[0].header = self.message_queue[0].create_header()
                    self.send_packet(self.token)
                    self.has_token = False
                    self.last_token_time = datetime.datetime.now()

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

                    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
                    self.last_token_time = datetime.datetime.now() # atualiza o tempo do último token

            elif packe .destination_name == "TODOS": # se o pacote é para todos
                t
                self.logger.debug("Pacote para todos!")

                if packe .origin_name == self.nickname:
# se o pacote foi t nviado por mim para todos e está voltando
                    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
                    self.last_token_time = datetime.datetime.now() # atualiza o tempo do último token

                else:
                    calculated_crc = packe .calculate_crc() # calcula crc
                    if calculated_crt == packe .crc:
                        packet.error_control = "ACK" # altera o estado
                    else:
                        packet.error_control = "NACK" # altera o estado

                packet.header = packe .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.

```
def check_token_status(self):
        time_waiting = 0

        while not self.terminate_event.is_set():

            if self.has_token == False:

                while time_waiting < self.TIMEOUT_VALUE and self.has_token == False:
                    time_waiting = (datetime.datetime.now() - self.last_token_time).total_seconds()
                    time.sleep(0.1)

                if time_waiting >= self.TIMEOUT_VALUE:
                    self.generate_token()
                    self.send_packet(self.token)
                    self.token = None
                    self.has_token = False
                    self.last_token_time = datetime.datetime.now()
                    time_waiting = 0

                elif time_waiting < self.MINIMUM_TIME:
                    self.has_token = False
                    self.token = None
```

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):
        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")
            print("4. Remover token da rede")
            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}")
                    self.add_packet_to_queue(new_packet)
                elif tipo == "1000":
                    token = TokenPacket()
                    self.send_packet(token)
                    self.last_token_time = datetime.datetime.now()
                    print(f"Novo token adicionado à rede.")
                    self.logger.debug(f"Novo token adicionado à rede.")
                else:
                    print("Tipo de pacote inválido. Por favor, tente novamente.")

            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)

            elif choice == "4":
                if self.has_token:
                    print("Removendo token da rede...")
                    self.has_token = False
                    self.token = None
                    print("Token removido da rede.")
                else:
                    while self.has_token == False:
                        pass
                    self.has_token = False
                    self.token = None
                    print("Token removido da rede.")

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

Processo de recebimento do token e envio de mensagem

Outra parte importante do código é processo realizado quando a máquina recebe um token. Nesse caso, existem duas opções: quando a máquina tem uma mensagem na fila, ela envia essa mensagem. Se não tiver, manda o token adiante. Essa função faz exatamente isso:

```
def run(self):
        """
        Roda o processo da máquina de segurar o token e enviar mensagens. Esse processo é executado sempre que a máquin
a recebe o token.
        """
        if self.has_token == True:
            if len(self.message_queue) > 0:
                self.logger.debug(f"segurando o token por {self.time_token} segundos...")
                time.sleep(int(self.time_token))
                self.logger.debug("enviando mensagem...")
                packet = self.message_queue[0] # pega o primeiro da fila
                self.send_packet(packet, add_error_chance=True) # envia o pacote
            else:
                self.logger.debug(f"Nenhuma mensagem para enviar, segurando o token por {self.time_token} segundos...")
                self.logger.debug('-'*50+'\n\n')
                time.sleep(int(self.time_token))
                self.send_packet(self.token) # manda o token
                self.has_token = False
                self.last_token_time = datetime.datetime.now()
        else:
            pass
```

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’:

```
2023-11-08 15:34:52,645 - DEBUG - ------------------------------------------------
2023-11-08 15:34:52,651 - DEBUG - Máquina bob iniciada.
2023-11-08 15:34:52,658 - DEBUG - ------------------------------------------------


2023-11-08 15:34:52,663 - DEBUG - ------------------------------------------------
2023-11-08 15:34:52,669 - DEBUG - Máquina bob possui o token. Dormindo por 3 segundo
2023-11-08 15:34:52,675 - DEBUG - ----------------------------------------s...----


2023-11-08 15:34:55,693 - DEBUG - ------------------------------------------------
2023-11-08 15:34:55,700 - DEBUG - Enviando toke
2023-11-08 15:34:55,711 - DEBUGn..Token enviado.
2023-11-08 15:34:55,717 - DEBUG - ------------------------------------------------


2023-11-08 15:34:58,712 - DEBUG - ------------------------------------------------
2023-11-08 15:34:58,712 - DEBUG - Pacote recebido. Iniciando processamento...
2023-11-08 15:34:58,712 - DEBUG - Token recebido.
2023-11-08 15:34:58,712 - DEBUG - Nenhuma mensagem para enviar, segurando o token por 3 segundo
2023-11-08 15:34:58,713 - DEBUG - ------------------------------------------------   s...


2023-11-08 15:35:01,724 - DEBUG - ------------------------------------------------
2023-11-08 15:35:01,730 - DEBUG - Enviando toke
2023-11-08 15:35:01,742 - DEBUGn..Token enviado.
2023-11-08 15:35:01,749 - DEBUG - ------------------------------------------------


2023-11-08 15:35:04,743 - DEBUG - ------------------------------------------------
2023-11-08 15:35:04,743 - DEBUG - Pacote recebido. Iniciando processamento...
2023-11-08 15:35:04,743 - DEBUG - Token recebido.
2023-11-08 15:35:04,743 - DEBUG - segurando o token por 3 segundo
2023-11-08 15:35:07,754 - DEBUG - enviando mensagem...   s...
2023-11-08 15:35:07,760 - DEBUG - ------------------------------------------------
2023-11-08 15:35:07,766 - DEBUG - Enviando pacote de dados...
2023-11-08 15:35:07,778 - DEBUG - Pacote de dados enviado.
2023-11-08 15:35:07,784 - DEBUG - ------------------------------------------------


2023-11-08 15:35:07,784 - DEBUG - ------------------------------------------------
2023-11-08 15:35:07,790 - DEBUG - Pacote recebido. Iniciando processamento...
2023-11-08 15:35:07,797 - DEBUG - Pacote de volta! Foi enviado por mim para ace
2023-11-08 15:35:07,803 - DEBUG - Mensagem contida no pacote: oi ace
2023-11-08 15:35:07,808 - DEBUG - Mensagem enviada foi recebida pelo destino!
2023-11-08 15:35:07,815 - DEBUG - pacote removido da fila
2023-11-08 15:35:07,821 - DEBUG - passando o token...
2023-11-08 15:35:07,826 - DEBUG - ------------------------------------------------
```

Log da máquina ‘ace’:

```
2023-11-08 15:34:49,383 - DEBUG - ----------------------------------------------------
2023-11-08 15:34:49,383 - DEBUG - Máquina ace iniciada.
2023-11-08 15:34:49,383 - DEBUG - ----------------------------------------------------


2023-11-08 15:34:55,705 - DEBUG - ----------------------------------------------------
2023-11-08 15:34:55,705 - DEBUG - Pacote recebido. Iniciando processamento...
2023-11-08 15:34:55,705 - DEBUG - Token recebido.
2023-11-08 15:34:55,705 - DEBUG - Nenhuma mensagem para enviar, segurando o token por 3 segundo
2023-11-08 15:34:55,705 - DEBUG - ----------------------------------------------------   s...


2023-11-08 15:34:58,711 - DEBUG - ----------------------------------------------------
2023-11-08 15:34:58,711 - DEBUG - Enviando toke
2023-11-08 15:34:58,711 - DEBUGn←.Token enviado.
2023-11-08 15:34:58,712 - DEBUG - ----------------------------------------------------


2023-11-08 15:35:01,737 - DEBUG - ----------------------------------------------------
2023-11-08 15:35:01,737 - DEBUG - Pacote recebido. Iniciando processamento...
2023-11-08 15:35:01,737 - DEBUG - Token recebido.
2023-11-08 15:35:01,737 - DEBUG - Nenhuma mensagem para enviar, segurando o token por 3 segundo
2023-11-08 15:35:01,737 - DEBUG - ----------------------------------------------------   s...


2023-11-08 15:35:04,742 - DEBUG - ----------------------------------------------------
2023-11-08 15:35:04,742 - DEBUG - Enviando toke
2023-11-08 15:35:04,742 - DEBUGn←.Token enviado.
2023-11-08 15:35:04,743 - DEBUG - ----------------------------------------------------


2023-11-08 15:35:07,771 - DEBUG - ----------------------------------------------------
2023-11-08 15:35:07,771 - DEBUG - Pacote recebido. Iniciando processamento...
2023-11-08 15:35:07,772 - DEBUG - Pacote para mim! Recebido de bob
2023-11-08 15:35:07,772 - DEBUG - Mensagem recebida com sucesso! Conteúdo: oi ace
2023-11-08 15:35:07,772 - DEBUG - Enviando pacote de volta...
2023-11-08 15:35:07,772 - DEBUG - ----------------------------------------------------

2023-11-08 15:35:07,772 - DEBUG - ----------------------------------------------------
2023-11-08 15:35:07,772 - DEBUG - Enviando pacote de dados...
2023-11-08 15:35:07,772 - DEBUG - Pacote de dados enviado.
2023-11-08 15:35:07,772 - DEBUG - ----------------------------------------------------
```