Implementação do Protocolo de Parada e Espera usando UDP

Breno Esteves, Gabriel de Paula, Guilherme Francis

Universidade Federal de São João Del Rei
São João del Rei / MG - Brasil

{brenoesteves243,gabriel.meira.2004,guilherme2036}@aluno.ufsj.edu.br

Resumo. *Este trabalho aborda a implementação do protocolo Stop-and-Wait sobre UDP em C para transmissão confiável de arquivos. O cliente lê o arquivo, divide-o em pacotes e os envia ao servidor, aguardando um sinal antes de prosseguir. Se o sinal não chegar no tempo esperado, o pacote é retransmitido. O servidor recebe os pacotes, verifica a integridade e reconstitui o arquivo.*

1. Introdução

Hoje, a troca de dados confiável é essencial para praticamente tudo na internet, desde o envio de mensagens em aplicativos até a transmissão de vídeos e arquivos. No entanto, nem todas as redes são perfeitas: interferências, congestionamento e perdas de pacotes são comuns, especialmente em conexões sem fio ou instáveis. Para lidar com isso, protocolos confiáveis como o TCP garantem que os dados cheguem corretamente, mas há casos em que precisamos de soluções mais simples e eficientes, como no uso de UDP com mecanismos próprios de controle.

Esse trabalho implementa um protocolo Stop-and-Wait sobre UDP, simulando situações reais onde pacotes podem se perder ou ter a integridade comprometida no caminho e precisem ser reenviados. Isso reflete desafios enfrentados em aplicações práticas, como transmissões via satélite, chamadas de voz sobre IP (VoIP) e comunicação em tempo real, onde o controle de erros precisa ser eficiente, mas sem comprometer a velocidade. Além disso, sistemas que transferem dados em redes congestionadas, como servidores de jogos online, podem se beneficiar de abordagens semelhantes para garantir que informações críticas cheguem ao destino, mesmo em condições adversas.

Ao testar o protocolo com simulação de perdas, esse trabalho avalia na prática como a retransmissão e a confirmação de pacotes podem melhorar a confiabilidade da comunicação, garantindo que o destinatário receba todos os dados corretamente. Esse tipo de solução tem impacto direto na estabilidade de serviços conectados e na experiência do usuário em um mundo cada vez mais dependente de redes digitais.

2. Desenvolvimento

O protocolo UDP (User Datagram Protocol), utilizado no trabalho, é um protocolo de transmissão leve e sem conexão, ou seja, ele não garante a entrega dos pacotes e não controla sua ordem de chegada. Diferente do TCP, que possui mecanismos internos para retransmissão e ordenação dos pacotes, o UDP é mais rápido, mas requer soluções na camada de aplicação para garantir a integridade dos dados.

2.1. Contextualização

Para contornar essa limitação e garantir que os pacotes sejam entregues corretamente, é possível implementar um mecanismo de ACKs (Acknowledgements), que pode simplificar com uma contextualização do mundo real:

Imagine que você precisa enviar um grande livro para alguém, mas em vez de enviá-lo inteiro de uma vez, você decide dividi-lo em várias páginas e enviá-las separadamente pelo correio. Cada página recebe um número único para garantir que o destinatário saiba a ordem correta. Esse é o princípio da divisão do arquivo em pacotes. Agora, ao enviar cada página pelo correio (envio do pacote), há sempre o risco de que alguma se perca no caminho. Para simular essa possibilidade, utilizamos um mecanismo que, de tempos em tempos, impede o envio de um pacote, como se fosse um carteiro que perdeu uma página no trajeto.

Quando o destinatário recebe uma página (recebimento e validação), ele verifica se ela chegou corretamente e sem rasuras. Se a página estiver ilegível ou fora da sequência correta, ele simplesmente a descarta e espera outra tentativa. Se a página recebida estiver correta, o destinatário envia um aviso de confirmação (envio de um ACK), dizendo ao remetente: “Recebi essa página, pode mandar a próxima!”. Para evitar confusão, ele usa um método alternado, dizendo algo como “Recebi a página par”ou “Recebi a página ímpar”, garantindo que o remetente saiba exatamente qual foi confirmada. Caso o remetente perceba que enviou uma página e não recebeu nenhuma confirmação após um tempo de espera (reenvio em caso de falha), ele simplesmente envia a mesma página novamente, repetindo o processo até ter certeza de que todas chegaram corretamente.

Esse mecanismo, aplicado à transmissão de arquivos por UDP, garante que os dados sejam entregues de forma confiável, mesmo usando um protocolo que, por si só, não garante a entrega ou a ordem correta das mensagens.

3. Descrição dos Algoritmos e Estruturas de Dados

As estruturas de dados utilizadas no código são essenciais para garantir a correta transmissão e recepção dos pacotes. Cada pacote contém informações fundamentais para que o servidor possa identificar sua posição na sequência de envio, verificar sua integridade e determinar quando a transmissão foi finalizada.

O pacote é composto por um identificador único, que determina sua posição, um total de pacotes para identificar quando a transmissão deve ser finalizada e um checksum, que assegura que os dados não foram corrompidos durante o transporte. Além disso, há um espaço destinado ao conteúdo do arquivo, contendo um fragmento dos dados a serem enviados. A construção do pacote pode ser exemplificada no seguinte pseudocódigo:

```
funcao criar_pacote(dados, numero_pacote, total_pacotes):
    pacote.identificador = numero_pacote
    pacote.total = total_pacotes
    pacote.dados = dados
    pacote.checksum = calcular_checksum(pacote)
    retornar pacote
```

A transmissão dos pacotes começa com o cliente lendo partes do arquivo, as estruturando em pacotes e enviando uma de cada vez ao servidor. Após o envio, o cliente aguarda um ACK do servidor antes de continuar com o próximo pacote. Caso não receba um ACK dentro do tempo limite, ou receba um ACK incorreto, o pacote é retransmitido.

```
para cada parte_do_arquivo em arquivo:
    pacote = criar_pacote(parte_do_arquivo, numero_pacote,
    total_pacotes)
    enquanto nao receber ACK valido:
        enviar pacote para o servidor
        aguardar resposta por tempo limite
        se tempo expirou ou ACK incorreto:
            retransmitir pacote
    avancar para proximo pacote
```

No lado do servidor, cada pacote recebido precisa ser validado antes de ser armazenado. O servidor escuta continuamente a chegada de pacotes e, ao receber um, verifica sua integridade comparando o checksum calculado com o checksum recebido. Se o pacote estiver corrompido ou fora da ordem esperada, ele é descartado. Caso contrário, ele é salvo no arquivo de saída e um ACK é enviado ao cliente, permitindo que ele prossiga com o próximo pacote. Esse processo no servidor pode ser descrito da seguinte forma:

```
enquanto recebendo pacotes:
    pacote = receber_pacote()
    se verificar_checksum(pacote) for falso:
        descartar pacote
        continuar

    se pacote na ordem correta:
        salvar dados no arquivo
        enviar ACK(pacote.identificador)
    senao:
        ignorar e aguardar novo pacote
```

O checksum é um elemento crucial dessa implementação, pois é responsável por identificar se os dados foram corrompidos durante a transmissão. Ele é gerado no momento da criação do pacote e recalculado pelo receptor para validar sua integridade.

```
funcao calcular_checksum(dados):
    soma = 0
    para cada byte em dados:
        soma += byte
    retornar complemento_de_um(soma)
```

Com isso, sempre que um pacote chega ao servidor, ele recalcula o checksum e compara com o valor armazenado no pacote original. Se houver divergência, significa que os dados foram corrompidos no caminho, e o servidor descarta o pacote para que o cliente o envie novamente.

3.1. Controle de Sequência e Retransmissão

Como dito anteriormente, o cliente mantém um identificador único para cada pacote, que indica a posição do pacote no arquivo. Esse identificador é incluído no cabeçalho do pacote antes do envio. Após o envio, o cliente aguarda um ACK do servidor para garantir que ele foi recebido corretamente com o trecho de código abaixo.

```
while (1) {
    if (sendto(sock, buffer, bytes_read + BUFFER_HEADER_SIZE, 0,
    (struct sockaddr *)server_addr, addr_len) < 0) {
        error("Failed to send package to server");
    }

    struct timeval tv;
    tv.tv_sec = TIMEOUT_SEC;
    tv.tv_usec = TIMEOUT_USEC;
    setsockopt(sock, SOL_SOCKET, SO_RCVTIMEO, &tv, sizeof(tv));

    ssize_t ack_bytes = recvfrom(sock, &ack_buffer, 1, 0,
    (struct sockaddr *)&server_addr, &addr_len);

    if (ack_bytes < 0) {
        warn("Timeout");
        continue;
    }
    if (actual_package % 2 != ack_buffer) {
        warn("Wrong ACK");
        continue;
    }

    actual_package++;
    break;
}
```

É importante analisar o contexto da aplicação para definir o tempo limite de espera. A depender dos usuários da aplicação, a transmissão naturalmente pode levar alguns segundos para ser concluída. Algumas abordagens alteram dinamicamente esse tempo, visando não esperar menos que a média e também não esperar demais em vão.

3.2. Fluxograma da implementação

A execução inicia com a leitura do arquivo e sua divisão em pacotes. Cada pacote recebe um identificador único e um checksum antes de ser enviado ao servidor. Após o envio, o cliente aguarda um ACK confirmando o recebimento. Esse ciclo se repete até que todos os pacotes sejam enviados corretamente.

O diagrama de atividades da Figura 1 representa essa sequência lógica detalhadamente. Importante evidenciar a região sujeita a interrupções demarcada pela linha tracejada; ela envolve as duas raias (Cliente e Servidor).

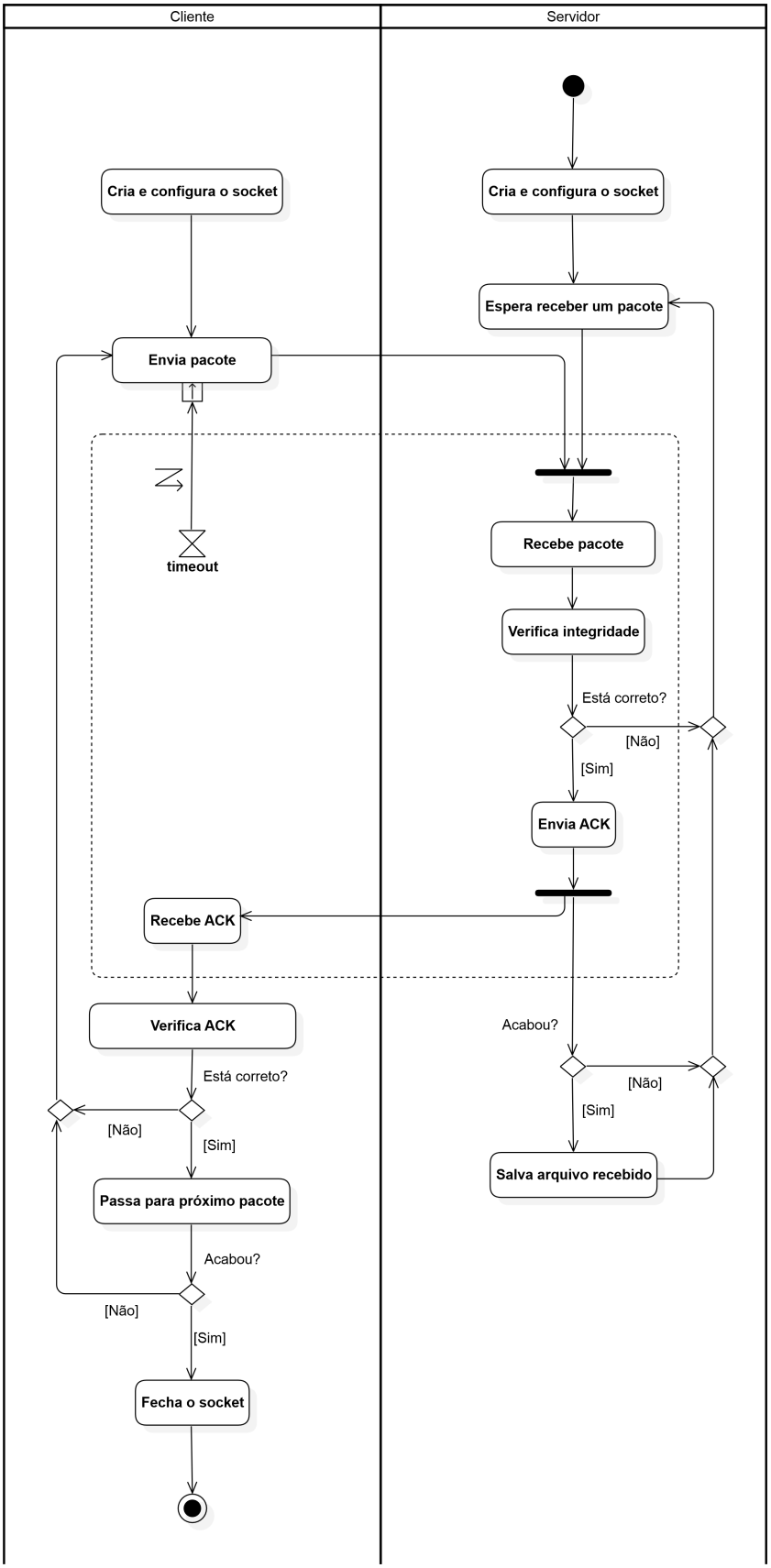

Figura 1. Fluxograma do protocolo de parada e espera

4. Análises e Resultados

Em uma rede local estável, mesmo usando um protocolo que não oferece a transferência confiável de dados como o UDP, a quantidade de erros é ínfima. Sendo assim, para obter dados concretos acerca da eficiência do protocolo, foi definida uma probabilidade de 10% de forçar a ocorrência de erros.

Há três possibilidades de erro nessa comunicação entre cliente e servidor:

1. Servidor não receber pacote do cliente;
2. Verificação do pacote no servidor indicar dados corrompidos;
3. Cliente não receber o ACK do servidor.

A probabilidade P_r de um pacote ser retransmitido pode ser calculada com base na fórmula a seguir, onde P_e é a probabilidade individual de cada erro possível e E é a quantidade de erros possíveis; qualquer ocorrência força uma retransmissão do pacote.

$$P_r = 1 - (1 - P_e)^E$$

Cada pacote, ao sofrer um erro na transmissão e ser retransmitido, está sujeito novamente à mesma probabilidade de erros até que o envio seja concluído. É necessário, portanto, encontrar o número médio de retransmissões por pacote, que é definido pela série geométrica abaixo.

$$R_{medio} = \sum_{i=1}^{\infty} {P_r}^i = \frac{1}{1 - P_r} - 1$$

Sabendo disso, definindo os parâmetros $P_e = 0,1$ e $E = 3$, é gerada $P_r = 0,271$, que, por sua vez, gera um $R_{medio} \approx 0,37$. Considerando que o tempo de espera pela resposta do servidor seja de 2 segundos, a penalidade média do envio de cada pacote é de $0,74$ segundos. É possível analisar esse resultado em um gráfico para se ter ideia do quanto um número alto de pacotes prejudica o desempenho do protocolo.

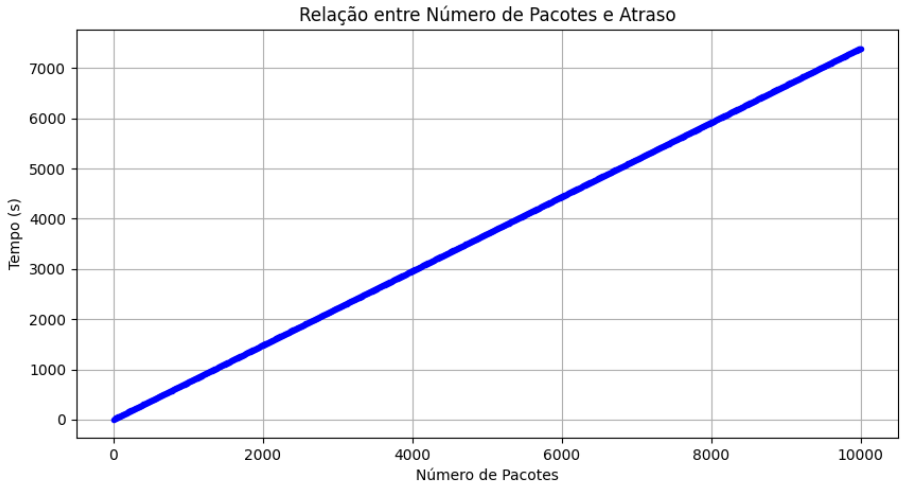

Figura 2. Gráfico do atraso seguindo parâmetros definidos

5. Conclusão

A implementação do protocolo Stop-and-Wait sobre UDP permitiu demonstrar, na prática, como mecanismos adicionais podem transformar um protocolo não confiável em um método seguro para transmissão de arquivos. Ao longo do desenvolvimento, verificamos que a simples adição de ACKs e um esquema de retransmissão de pacotes perdidos foi suficiente para garantir a entrega completa dos dados, mesmo quando simulações de falha foram introduzidas.

A abordagem baseada no parada e espera mostrou-se eficiente para garantir a entrega ordenada dos pacotes e a integridade das informações, apesar de seu custo em desempenho devido ao tempo de espera entre envios. Esse protocolo é especialmente útil para ambientes onde a confiabilidade é mais importante do que a velocidade, como transmissões via satélite, comunicação de sensores IoT e aplicações críticas que exigem controle total sobre a entrega de pacotes.

Durante a implementação, foi observado que o uso de um checksum para verificar a integridade foi crucial para detectar pacotes corrompidos antes que fossem processados, evitando inconsistências nos arquivos recebidos. O fluxo de envio e recepção de pacotes, documentado no fluxograma, demonstrou o funcionamento eficiente do sistema de retransmissão e validação, garantindo que nenhum pacote fosse perdido ou armazenado incorretamente.

Embora o método Stop-and-Wait seja simples e funcional, ele pode se tornar um gargalo em redes de alta latência ou alto tráfego, pois cada pacote precisa ser confirmado antes do envio do próximo. Para otimizar esse processo, protocolos mais avançados, como Go-Back-N e Selective Repeat, podem ser explorados, permitindo o envio contínuo de múltiplos pacotes antes da confirmação individual.

Dessa forma, este trabalho não apenas reforça a compreensão dos desafios enfrentados na transmissão de dados via UDP, mas também destaca como soluções simples podem melhorar significativamente a confiabilidade das comunicações. As técnicas aplicadas aqui podem ser expandidas e aprimoradas para uso em sistemas de transmissão em tempo real, jogos online e redes sem fio, garantindo um equilíbrio entre desempenho e confiabilidade na troca de informações.

Referências

[1] Tanenbaum, Andrew S., and Wetherall, David J. (2010). *Redes de Computadores*. 5ª Edição. Pearson.

[2] Kurose, James F., and Ross, Keith W. (2010). *Redes de Computadores e a Internet: Uma Abordagem Top-Down*. 5ª Edição. Pearson.