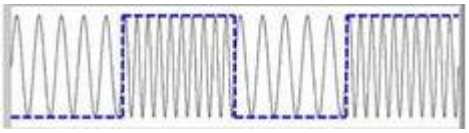


CAMADA FÍSICA DA COMPUTAÇÃO

ENGENHARIA DA COMPUTAÇÃO - Rodrigo Carareto

PROJETO 3 – Fragmentação

Normalmente, a camada responsável pela transmissão dos dados gerados em camadas superiores, realiza a transmissão de duas formas: dados fragmentados em pacotes (datagramas) ou streaming. As principais diferenças entre uma transmissão em streaming e uma transmissão baseada em pacotes (datagramas) estão relacionadas à forma como os dados são enviados, recebidos e processados. Vamos analisar cada uma:

Transmissão em Streaming

- Definição: É uma forma de transmissão contínua de dados, geralmente usada para conteúdos multimídia (áudio e vídeo).
- Protocolo comum: Utiliza protocolos como RTP (Real-time Transport Protocol) sobre UDP ou HLS/DASH sobre HTTP.
- Modo de entrega: Os dados são entregues e processados continuamente, permitindo a reprodução do conteúdo quase em tempo real, sem necessidade de download completo.
- Latência: Baixa a moderada, dependendo da técnica utilizada (por exemplo, buffering para evitar interrupções).
- Controle de erro: Pode tolerar pequenas perdas sem comprometer significativamente a experiência do usuário.
- Exemplos de uso: Plataformas como Netflix, YouTube, Spotify e transmissões ao vivo.

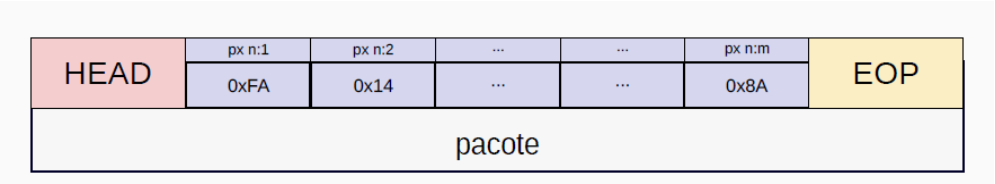
Transmissão em Pacotes (Datagramas)

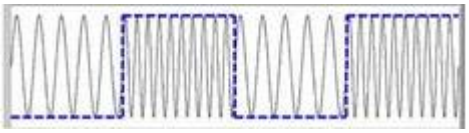
- Definição: Baseia-se no envio de pacotes de dados independentes, onde cada datagrama pode seguir um caminho diferente na rede.
- Exemplos: Protocolos como UDP (User Datagram Protocol), que não garante a entrega ou a ordem dos pacotes. E protocolos como o TCP (Transmission Control Protocol), que provê garantia de entrega.
- Modo de entrega: Os pacotes são enviados de forma discreta, sem garantir que chegarão na mesma ordem ou mesmo que serão entregues (dependendo do protocolo).
- Latência: Aumenta quando há necessidade de confirmação de recebimento de pacotes, como no TCP.
- Controle de erro: Garantia de retransmissão no caso do TCP.
- Exemplos de uso: Aplicações em tempo real como VoIP (ligações pela internet), jogos online, DNS, transmissões multicast.

Como são os pacotes (datagramas)

Um datagrama é tipicamente dividido em 3 partes:

- Um cabeçalho (header)
- Payload (dados)
- EOP (end of package)]





Nos bytes reservados ao cabeçalho, são colocados os bytes de comunicação entre as partes. O metadado. Com os bytes posicionados nas posições corretas, as partes podem conversar. Confirmar envio, informar quem está enviando, qual o número do pacote, qual o tamanho do payload, confirmar recebimento... Enfim, esses bytes são utilizados para uma conversa entre as partes.

No payload são colocados os bytes de dados que foram acomodados no pacote.

O EOP é uma espécie de fim de pacotes, é uma sequência combinada que marca o fim do pacote.

Por que enviar os dados segmentados em pacotes?

Os principais motivos para o uso de transmissão em pacotes são:

1. Eficiência na utilização da rede

- Dividir dados em pacotes permite que várias transmissões ocorram simultaneamente na rede.
- Diferentes pacotes podem seguir **rotas distintas** para evitar congestionamentos.
- Se um pacote for perdido, **somente ele será retransmitido**, e não toda a mensagem.

2. Melhor controle de erro

- Cada pacote contém um **checksum** para verificar a integridade dos dados.
- Se houver erro na transmissão, **apenas o pacote corrompido será reenviado**, reduzindo desperdício de banda.

3. Gerenciamento de tráfego e controle de congestionamento

- Protocolos como **TCP** ajustam o tamanho dos pacotes e a taxa de envio conforme a capacidade da rede.
- Se a rede estiver congestionada, pacotes podem ser **redirecionados** ou **retransmitidos**.

4. Fragmentação e reassemblagem

- Pacotes menores são mais fáceis de transportar e evitam **bloqueios** em redes de diferentes capacidades.
- Dispositivos intermediários, como roteadores, podem **fragmentar ou juntar pacotes** conforme necessário.

5. Comunicação mais robusta e confiável

- Se uma conexão for interrompida, pacotes já transmitidos **não são perdidos**.
- A comunicação pode ser mantida mesmo com falhas parciais na rede.

Exemplo de datagrama: Cabeçalho TCP (20 bytes fixos + opções variáveis)

Campo	Tamanho (bits)	Descrição
Porta de origem	16	Número da porta do remetente
Porta de destino	16	Número da porta do destinatário

Campo	Tamanho (bits)	Descrição
Número de sequência	32	Indica a posição do primeiro byte deste segmento dentro do fluxo de dados
Número de confirmação (ACK)	32	Confirma o recebimento do último segmento válido do emissor
Tamanho do cabeçalho	4	Indica o tamanho do cabeçalho TCP (mínimo 20 bytes)
Reservado	3	Reservado para uso futuro
Flags de controle	9	Indica o estado da conexão (SYN, ACK, FIN, etc.)
Tamanho da janela	16	Quantidade de bytes que o receptor pode aceitar sem receber confirmação
Checksum	16	Verifica a integridade do segmento
Ponteiro de urgência	16	Indica se há dados urgentes (caso a flag URG esteja ativa)
Opções TCP	Variável	Usado para configurações extras, como escala de janela, timestamps, etc.

Além do cabeçalho, o protocolo tem um payload variável, com tamanho máximo de 1469 bytes. O tamanho do payload de cada pacote é normalmente informado no cabeçalho.

Explicação dos principais campos do caceçacho TCP

1. **Número de sequência (Sequence Number)**
    - Indica qual é o primeiro byte do segmento dentro do fluxo de dados.
    - Importante para remontar os dados na ordem correta.
2. **Número de confirmação (Acknowledgment Number)**
    - Usado pelo destinatário para informar qual o próximo byte esperado.
    - Se um segmento for perdido, ele não será confirmado, e o remetente o reenviará.
3. **Flags de controle (9 bits)**
    - **URG** (Urgent) → Indica dados urgentes.
    - **ACK** (Acknowledgment) → Confirma recebimento de dados.
    - **PSH** (Push) → Solicita entrega imediata ao aplicativo.
    - **RST** (Reset) → Reinicia a conexão abruptamente.

- **SYN** (Synchronize) → Inicia uma conexão.
- **FIN** (Finish) → Finaliza uma conexão.

4. **Tamanho da Janela (Window Size)**
   - Define quantos bytes o receptor pode armazenar antes de precisar enviar uma confirmação.
   - Essencial para o **controle de congestionamento** e **fluxo de dados**.
5. **Checksum**
   - Validado pelo receptor para garantir que os dados não foram corrompidos durante a transmissão.

Enunciado do projeto 03

Nesse projeto, sua aplicação que exerce o papel de client deverá enviar um arquivo para a aplicação server. Esse arquivo deverá ser fragmentado e enviado através de "pacotes" (datagramas). Deverá existir handshake e confirmação de recebimento de cada pacote!

Datagrama.

De agora em diante você está PROIBIDO de trocar mensagens entre server e client que não sejam um datagrama completo (um pacote). Isso significa que mesmo que queira enviar um único byte, mesmo que não faça parte dos dados a serem enviados, deverá enviar um pacote compondo um datagrama. Para isso vamos considerar o seguinte datagrama:

- HEAD – 12 BYTES - fixo
- PAYLOAD – variável entre 0 e 70 BYTES (pode variar de pacote para pacote)
- EOP – 3 BYTES – fixo (valores de sua livre escolha)

Não importa o tipo de mensagem, sempre será enviada através de um datagrama como definido acima. Tanto de client para server como no sentido oposto.

Handshake

Antes do início do envio da mensagem, o client deve enviar uma mensagem para verificar se o server está "vivo", pronto para receber o arquivo a ser enviado. O server então deve responder como uma mensagem informando que ele está pronto para receber. Enquanto a mensagem não for recebida pelo cliente, este não começa o envio. Caso o cliente não receba a resposta do servidor dentro de 3 segundos, informando que está pronto para receber o arquivo, o usuário recebe uma mensagem: "Servidor inativo. Tentar novamente? S/N". Se o usuário escolher "S", outra mensagem de verificação é enviada ao server. Caso escolha não. Tudo se encerra.

Caso o servidor responda ao cliente em menos de 3 segundos, o cliente deve iniciar a transmissão do arquivo com o envio do primeiro pacote.

Fragmentação

Como seu payload é menor que o arquivo a ser enviado, você deverá enviá-lo em partes (pacotes)! Lembre-se que cada pacote tem que seguir obrigatoriamente a estrutura de seu datagrama! A razão para se dividir uma mensagem em pacotes pode ser devido a uma limitação de hardware (pouco espaço no buffer, por exemplo), ou ainda gestão do tempo de ocupação do canal de comunicação, evitando-se manter a linha ocupada por muito

tempo. Outra razão é a de se evitar ter que retransmitir uma mensagem inteira quando acontecer um erro, podendo-se reenviar apenas o fragmento com problema.

Acknowledge / Not acknowlwdge

Durante a transmissão de dados é muito comum a troca de mensagens como confirmação de recebimento de um pacote ou mesmo informando um problema na recepção do pacote. Esse tipo de comunicação gera uma robustez para a transmissão, embora possa afetar a velocidade de transmissão. Existe então um compromisso entre a velocidade de transmissão e a segurança da transmissão no que diz respeito à integridade dos dados.

FUNCIONALIDADES:

- Nesse projeto, seu datagrama não pode ultrapassar 85 bytes. Consequentemente, como já dito antes, seu payload será menor que isso, exigindo uma fragmentação da imagem a ser feita pelo cliente.
- Quando o client enviar um pacote, deve informar obrigatoriamente (em algum espaço do head reservado a isso), o número do pacote e o número total de pacotes que serão transmitidos.
- Ao receber um pacote, o server deve fazer duas verificações: verificar que o número do pacote é 1 a mais que o anterior, ou seja, a ordem está correta. Deve também verificar se o EOP está no local correto (vieram todos os bytes).
- Se tudo estiver ok, o server deve enviar uma mensagem ao cliente para que este envie o próximo pacote.
- Após o envio de um pacote o cliente não envia o próximo enquanto não receber a confirmação do server de recebimento do pacote enviado.
- Se algo estiver errado, o server deve enviar uma mensagem para o cliente solicitando o reenvio do pacote, seja por não ter o payload esperado, ou por não ser o pacote correto.
- Ao receber o último pacote o Server deve ser capaz de reagrupá-los e salvar o arquivo em seu estado original. Após o reagrupamento, o server faz uma última resposta ao cliente, informando que a transmissão foi feita com sucesso.
- Quando houver algum erro de transmissão, o client não precisa corrigir o erro e continuar a transferência de dados. A comunicação deve se encerrar.

ENTREGA:

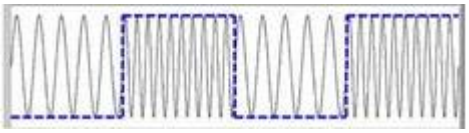
Você deverá mostrar até 06/03/2025

Conceito C

Uma transmissão de sucesso com seu server recebendo os pacotes de um arquivo, conferindo e respondendo ao cliente.

Conceito B-

- Uma situação em que o server não estava pronto para receber o arquivo por mais de 3 segundos, forçando o cliente a enviar uma outra mensagem de teste (tentativa de handshake). Para simular essa situação, você pode colocar um time.sleep de vários segundos no lado do server após o byte de sacrifício, seguido de um clearBuffer(). Simule uma situação em que o cliente tenta por 2 vezes iniciar sem sucesso a transmissão, e na terceira vez o server finalmente responde e a transmissão então se inicia.
- Uma simulação onde o client erre o número do pacote. Mostre a resposta do servidor perante o envio fora de ordem.
- Faça uma simulação onde o tamanho real do payload de um pacote não corresponde ao informado no head. Mostre a resposta do servidor.



CAMADA FÍSICA DA COMPUTAÇÃO

ENGENHARIA DA COMPUTAÇÃO - Rodrigo Carareto

Conceito (A+)

Uma simulação em que se os fios entre os Arduinos (qualquer um deles ou ambos) forem desconectados e conectados novamente, a transmissão retorna e termina com sucesso!

Exemplo de protocolo em pacotes com garantia total de entrega (esse exemplo é mais complexo que o projeto 3)!

h0 – tipo de mensagem

h1 – livre

h2 – livre

h3 – número total de pacotes do arquivo

h4 – número do pacote sendo enviado

h5 – se tipo for handshake: id do arquivo (crie um)
h5 – se tipo for dados: tamanho do payload

h6 – pacote solicitado para recomeço quando a erro no envio.

h7 – último pacote recebido com sucesso.

h8 – h9 – CRC (Por ora deixe em branco. Fará parte do projeto 5)

PAYLOAD – variável entre 0 e 114 bytes. Reservado à transmissão dos arquivos.

EOP – 4 bytes: 0xAA 0xBB 0xCC 0xDD

IMPORTANTE: A MÉTRICA PARA SEU SUCESSO SÃO A INTEGRIDADE DOS DADOS RECEBIDOS E O THROUGHPUT !

Documento de definição do protocolo de comunicação

Padrao:
UART, baudrate 115200, sem bit de paridade

Datagrama
Cada envio deve ser feito como um datagrama completo, contendo head, payload e eop, ou seja, não é permitido envios que não contenham head, payload(ocasionalmente nulo) e eop. O tamanho do payload não pode ultrapassar 114 bytes e o tamanho do datagrama não deve ser maior que 128 bytes

Tipos de mensagens
***TIPO** 1 – Esta mensagem representa um chamado do cliente enviado ao servidor convidando-o para a transmissão. Nesse caso, o head deve conter o byte h0 com o número 1, indicando mensagem tipo 1, e outro byte com um identificador. O identificador é o número do servidor, sendo que quando este receber uma mensagem tipo 1, verifica se é para ele mesmo o envio. A mensagem tipo 1 já deve conter o número total de pacotes que se pretende enviar!*

***TIPO 2** – Essa mensagem é enviada pelo servidor ao cliente, após o primeiro receber uma mensagem tipo 1 com o número identificador correto. Deve conter no head o número 2 no byte reservado ao tipo de mensagem. O significado de uma mensagem tipo 2 é que o servidor está ocioso e, portanto, pronto para receber o envio dos pacotes.*

***TIPO 3** – A mensagem tipo 3 é a mensagem de dados. Este tipo de mensagem contém de fato um bloco do dado a ser enviado (payload). Deve conter o número 3 no byte reservado ao tipo de mensagem. Essa mensagem deve conter também **o número do pacote que envia (começando do 1) e o total de pacotes a serem enviados**.*

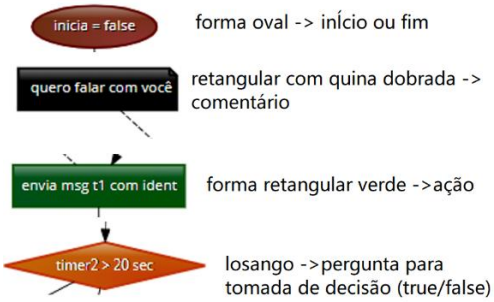
TIPO 4 *– Essa mensagem é enviada do servidor para o cliente toda vez que uma mensagem tipo 3 é recebida pelo servidor e averiguada. Quando verificado que a mensagem é realmente o pacote que o servidor estava esperando e que tal mensagem chegou em perfeitas condições (eop no local correto), o servidor envia então a mensagem tipo 4, afirmando que recebeu o pacote. Essa mensagem deve ter o número 4 no byte reservado ao tipo de mensagem. Além disso, deve conter o número do último pacote recebido e já aferido.*

TIPO 5 *– É uma mensagem de time out. Toda vez que o limite de espera exceder o timer dedicado a isso, em qualquer um dos lados, deve-se enviar essa mensagem e finalizar a conexão. Essa mensagem deve ter o número 5 no byte reservado ao tipo de mensagem.*

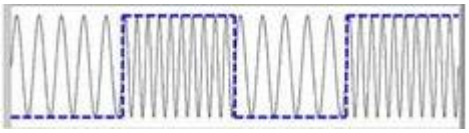
TIPO 6 *– É uma mensagem de erro. O servidor deve enviar esta mensagem ao cliente toda vez que receber uma mensagem tipo 3 inválida, seja por estar com bytes faltando, fora do formato correto ou por não ser o pacote esperado pelo servidor (pacote repetido ou fora da ordem). Essa mensagem deve ter o número 6 no byte reservado ao tipo de mensagem. Além disso, deve conter o número correto do pacote esperado pelo servidor na posição h6, independentemente do problema que invalidou a mensagem. Isso orienta sempre o cliente para o reenvio.*

A comunicação deve ocorrer de acordo com os diagramas a seguir (atente para a legenda):

Legenda:

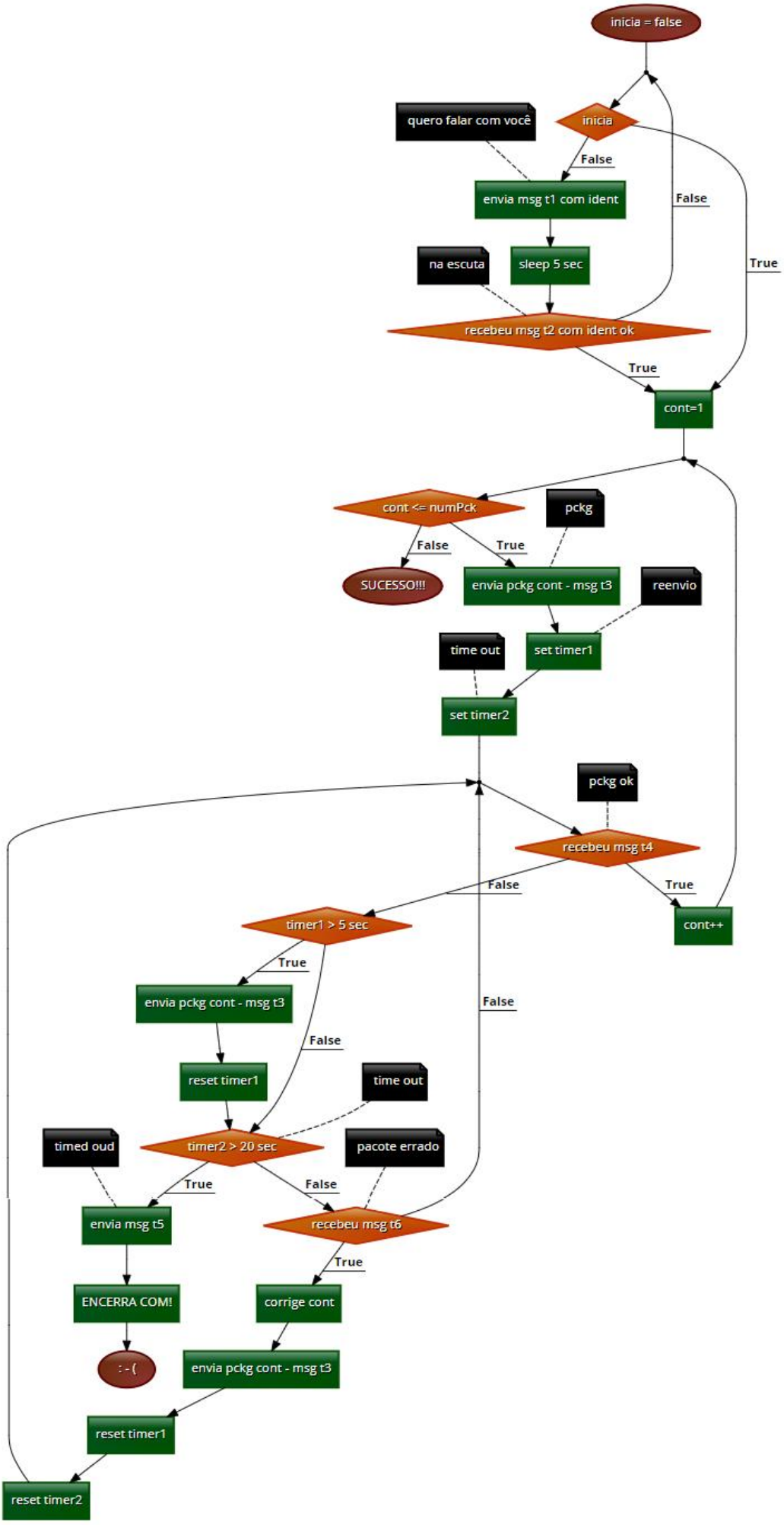


1) PROTOCOLO CLIENTE



CAMADA FÍSICA DA COMPUTAÇÃO

ENGENHARIA DA COMPUTAÇÃO - Rodrigo Carareto



2) PROTOCOLO SERVER