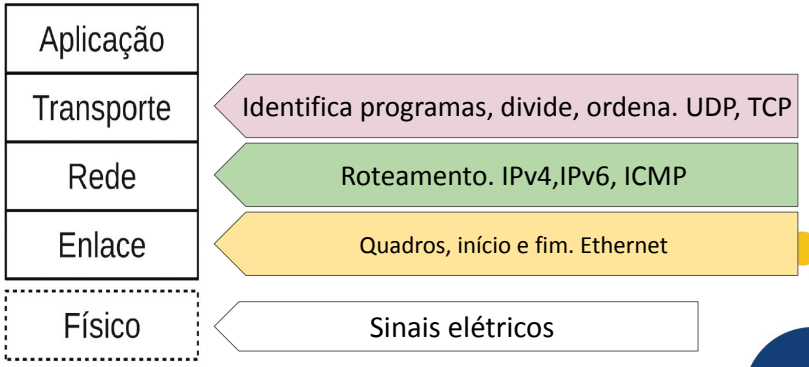
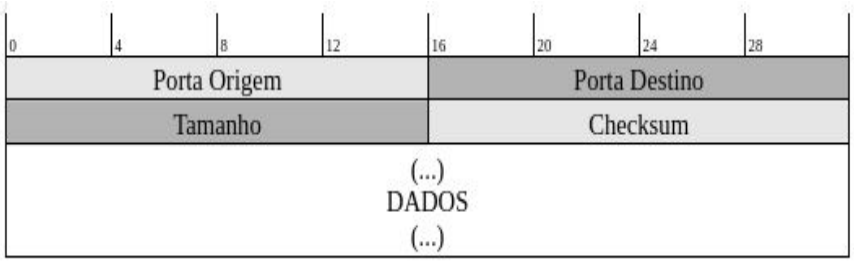
Redes de Computadores
Aula 12

Elgio Schlemer
elgio.schlemer@unilasalle.edu.br

Camada de Transporte



Cabeçalho UDP



Implementar confiabilidade em UDP

- Implementar na parte da aplicação
  - Não pode alterar o UDP
  - Cabeçalho de aplicação (parte dos dados)
- destino: quem receberá os dados
- origem: quem envia
- Duas abordagens:
  - destino controla confiabilidade
    - Exemplo: pede para reenviar
  - origem controla confiabilidade
    - Exemplo: reenvia automaticamente se não teve confirmação

Sugestão de Algoritmo 1

- Pacotes numerados de 1 a N
  - não se conhece antecipadamente o N
  - Último pacote tem "-", indicando ser o último
- destino recebe e imediatamente confirma
- Formato dos pacotes:
  - Envio de dados: "ID,+/-,DADOS"
    - Exemplo: Enviando "Teste 123 CACA"
      "1,+,Teste " "2,+,123 " "3,-,CACA"

Sugestão de Algoritmo 1

- Formato da confirmação: "ID,="
  - Exemplo: "1,=" "2,=" "3,="
- origem só envia próximo pacote ao receber a confirmação do anterior.
  - espera um tempo X pela confirmação
  - reenvia se X estourar
- Ao receber o primeiro pacote de um origem, destino NEGARÁ pacotes de qualquer outro
  - Quando destino NEGAR um origem, irá devolver "0,="
- Tarefa do origem saber que deve tentar mais tarde
- destino usa IP e porta de origem para identificar origem

Avaliação do Algoritmo 1

- pacotes desordenados:
  - não vai ocorrer, pois o origem só enviará um segundo pacote quando receber a confirmação do enviado antes
- pacotes perdidos:
  - serão detectados pelo cliente ao estourar o tempo X sem receber confirmação.
- pacotes repetidos:
  - uma repetição de roteamento
  - porque o origem achou que não chegou e reenviou

Avaliação do Algoritmo 1

- Ao receber um pacote duplicado o destino deverá:
  - (a) CONFIRMAR: pois o origem pode estar esperando a confirmação
  - (b) descartar: porque já havia recebido.
- duplicação da confirmação:
  - sempre que um origem receber a confirmação de um pacote N-1
  - estando esperando pela confirmação N
  - descarta e continuar esperando pela confirmação N.
- mistura de pacotes de clientes distintos:
  - não vai acontecer
  - pois o destino irá descartar pacotes que não vierem do origem com quem ele está conversando

Avaliação do Algoritmo 1

- concorrência:
  - não há. destino atende apenas um origem por vez.
  - Só está apto a atender outro depois que terminar a conversa com o atual
  - pelo recebimento do pacote “-”
- Um novo origem irá receber um 0,= e saberá que precisa esperar.

Problemas do Algoritmo 1

- transferência HALF-DUPLEX
  - envia/confirma: péssimo desempenho
- se origem abandonar a transferência sem enviar último pacote
- se confirmação do último pacote for perdida
  - origem ficará reenviando ele eternamente.
- se tempo de espera pela confirmação for curto demais ou longo demais

Confiabilidade em UDP: Algoritmo 2

- resolver o desempenho do algoritmo 1
- numeração de pacotes, como no algoritmo 1
- origem não espera confirmação
  - destino é quem pede retransmissão
- origem possui um buffer para os N últimos pacotes enviados
  - quando buffer encher, reutiliza a primeira posição em uma lista circular
- origem sabe tratar pedidos de retransmissões do destino

Confiabilidade em UDP: Algoritmo 2

- Formato de pacotes:
  - Envio de dados: "ID, +/-, DADOS"
  - Pedido de retransmissão: "ID, ?"
- destino vai pedir a retransmissão do pacote K ao receber um pacote de ID maior que K
- Mistura de pacotes
  - descarta quem não é do origem esperado
  - como no algoritmo 1

Algoritmo 2

- Lógica do origem:
  - define um buffer para N pacotes
  - Repete k iniciando em 1 ate k ser último
  - envia pacote k
  - armazena pacote k enviado na próxima posição livre do buffer
  - posição do buffer = ((k-1) mod N) + 1
    - sendo N o tamanho do buffer (buffer circular).
- Se destino pediu retransmissão
  - reenvia todos os pacotes da solicitação até o pacote atual (inclusive)
- Exemplo: k é 7, destino pediu 5. Envia 5, 6 e 7
  - Fim da transmissão do pacote k, volta para fazer k+1

Algoritmo 2

- Lógica do destino:
  - aguarda primeiro pacote de algum origem
  - armazena Ip e porta origem
  - Repete (iniciando k=1)
    - Recebe um pacote X, supostamente de número k
      - ele espera o k
    - Se não for do origem esperado, devolve erro (0,=) e descarta.
    - Se pacote X < k: Duplicidade de pacote. Descarta

Algoritmo 2

- Se X > K
  - FORA DE ORDEM (esperava o 1, mas veio o 2)
  - pede a retransmissão do pacote k que era esperado
  - descarta o pacote X que veio errado
- Se X = K
  - Pacote esperado
  - K = K + 1
  - Se K tinha marcação de último: ENCERRA Repete
- Volta no repete para o próximo pacote
- FIM DO LACO REPETE
- Volta para aguardar transmissão de outro origem

Avaliação do Algoritmo 2

- pacotes desordenados:
  - serão considerados como perdidos
    - pois ao receber um pacote X > K vai descartar X e solicitar o reenvio de K
- pacotes perdidos:
  - detectados pelo destino
  - ao receber o pacote K sem ter recebido k-1
- pacotes repetidos:
  - repetição de roteamento
  - porque houve um desordenamento (que o destino tratou com perda).
  - Se destino espera K e receber o um X < K, sabe que é repetido
    - descarta
- mistura de pacotes
- concorrência: não há. destino atende apenas um origem por vez.

Problemas do Algoritmo 2

- desempenho SOMENTE se pacotes forem na ordem e sem perdas
- pacotes desordenados serão considerados como PERDIDOS
  - destino não criou buffer para guardar pacotes recebidos fora de ordem.
- Se houver muito desordenamento, vira o HALF-Duplex novamente
- PROBLEMA Sério:
  - origem envia último pacote e encerra, mas último pacote é perdido.
  - destino recebeu pN e espera Pn+1. Por quanto tempo?
  - O que faz depois que destino julgar estar esperando demais?
  - E se Pn+1 era o último, foi enviado e perdido e o cliente encerrou?

Adicionando Concorrência ao UDP

- Concorrência:
  - habilidade de identificar cada cliente e não misturar
  - Atender vários simultaneamente
  - Nenhum dos algoritmos anteriores tem concorrência
- identifica cada cliente pelo par IP/Porta de Origem
  - 2 clientes no mesmo IP deverão ter porta de origem distintas
  - E DUAS portas iguais de origem deverão vir de Ips distintos

Concorrência em UDP: abordagem A

- Um mesmo servidor lida com clientes diversos
- Para cada pacote, identifica qual o estado da "conversa"
  - uma tabela de clientes para isto e o estado de cada um
- Exemplo: se for uma ferramenta de upload de arquivos cada tabela teria
  - IP/Porta ORIGEM
  - nome do arquivo
  - último pacote recebido

Concorrência em UDP: abordagem B

- Outro servidor lida com cada novo cliente
  - Um novo processo por cliente
  - Servidor original apenas cria novos processos para cada cliente
- Problema:
  - a porta do servidor só pode ser de um mesmo processo
  - outros processos não podem ler dados na mesma porta
- Solução:
  - servidor repassa para cada filho ou
  - cada processo que atende um cliente tem uma nova porta

Concorrência em UDP: abordagem B1

- Lógica do servidor: ESPERA pacote UDP
  - recebeu pacote?
    - obtém IP e Porta de origem
    - Ip/Porta já estão em sua tabela?
      - SIM: envia o pacote para o filho criado.
      - NÃO:
        - cria um novo filho
        - adiciona na tabela filho e cliente IP/PORTA
        - envia pacote para o filho criado
  - Filho notificou que encerrou: desaloca posição da tabela.

Concorrência em UDP: abordagem B1

IP Origem	Porta Origem	PID do Filho
192.168.100.40	5000	2345
192.168.100.40	5001	2346
192.168.102.27	5000	2347

Exemplo de dados desejáveis na Tabela de filhos:

- Se chegar um dado vindo de 192.168.100.4 porta 5000, servidor envia para o PID 2345.
- Se chegar um dado vindo de 192.168.100.5 porta 5001, servidor cria um novo filho pois é um novo cliente.
- Se chegar um dado vindo de 192.168.200.5 porta 5000, servidor cria um novo filho e registra entrada na tabela

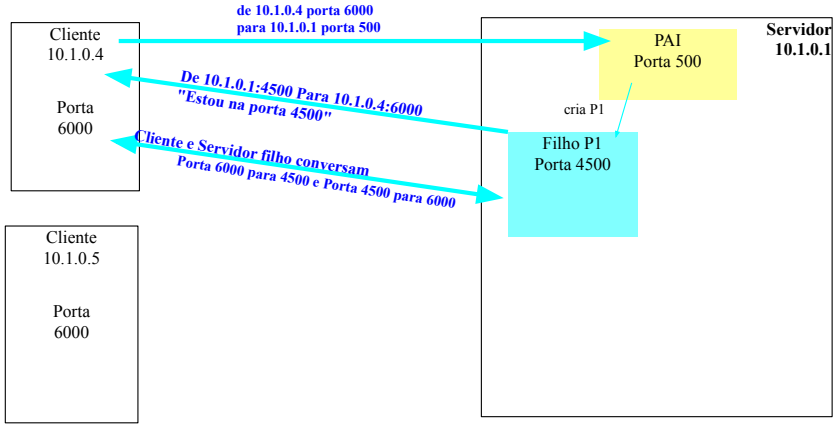
Concorrência em UDP: Problemas B1

- Complexidade no gerenciamento:
  - manter tabela
  - manter filhos
  - desviar para filhos
- forma de comunicação de servidor com filho
  - Em LINUX: Memória compartilhada, pipes, mensagens (sinais para acordar o filho, etc)
- único processo gerencia tudo
  - gargalo?

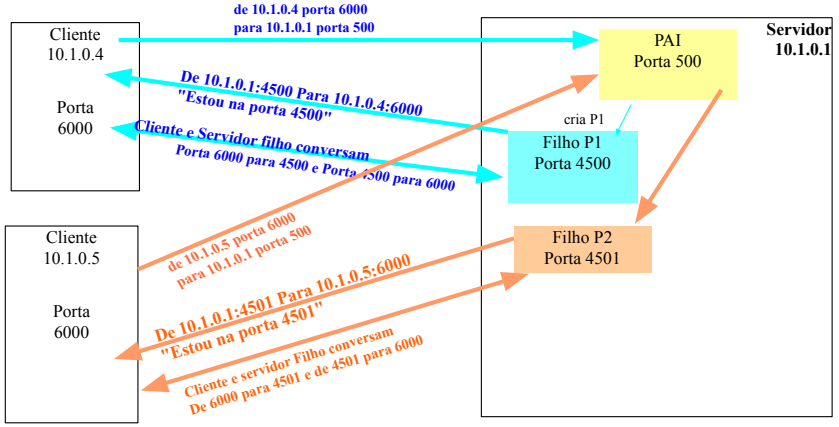
Concorrência em UDP: abordagem B2

- Cada filho tem uma porta nova para si
  - Problema: como o cliente saberá qual porta é?
- Solução:
  - filho conta para o cliente qual porta!
- Servidor cria processo filho para tratar
- processo NOVO pede uma nova porta ao SO
- processo FILHO informa ao cliente que a comunicação agora continua na porta X

B2 Demonstração



B2 Demonstração



Avaliação abordagem B2

- Tempo de reciclagem (em quanto tempo o servidor estará apto a receber outra requisição)
  - O servidor principal precisará apenas:
    - ao receber mensagem, criar filho
    - passar a mensagem para o filho
  - Voltar para esperar nova mensagem
  - Servidor principal recebe APENAS UM PACOTE
    - tempo apenas para criar o filho: aceitável

Concorrência em UDP: abordagem B2

- Lógica do Servidor FILHO
  - pede ao SO uma porta qualquer
  - obtém ip e porta de origem do cliente com quem devo conversar (herdado do pai)
  - Envio um pacote de confirmação ao cliente
    - informando que os dados devem vir na porta X
  - Repete
    - recebe dados do cliente (se não for do meu cliente, descarta)
    - Recebeu último pacote? MORRE
- Poderia implementar apenas confiabilidade
  - mas sem concorrência pois este filho nasceu apenas para este cliente!

Conclusão UDP

- UDP não possui confiabilidade
- UDP não possui estrutura de dados para auxiliar a implementação
- Implementar confiabilidade e concorrência
  - exige mais esforço do programador
- Confiabilidade e concorrência exige mais detalhes de implementação
  - criação de buffers
  - gerenciamento de clientes
  - confirmações ou pedidos de retransmissão
  - tabelas de estados
  - etc

Protocolo TCP

-

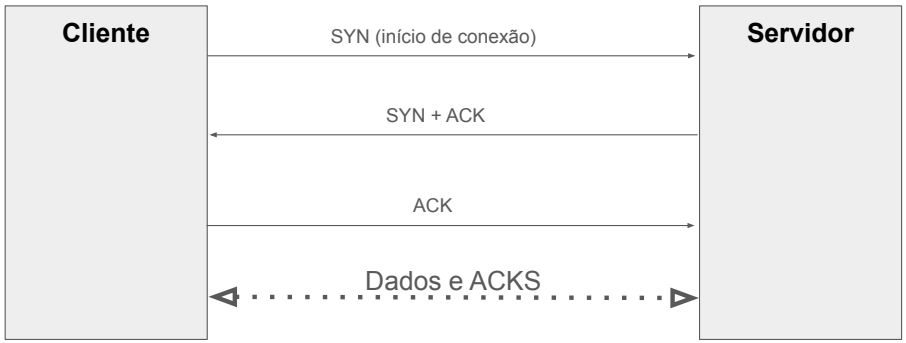
Protocolo TCP

- exemplo de uma ligação telefônica
- Garante recebimento
- garante ordem no recebimento
- Como garantir a ordem?
  - A) receptor confirma que recebeu, na falta desta confirmação, transmissor reenvia por timeout
  - B) ou receptor solicita que seja retransmitido algum que faltou
- O caso do TCP é semelhante ao A:
  - O remetente espera confirmação de recebimento
  - se não receber, torna a reenviar
- Estabelecimento de timeouts

Handshake TCP

- Exemplo de uma ligação telefônica
- nenhum byte de dado é enviado sem handshake
- Demonstração:
  - SYN
  - SYN+ACK
  - ACK

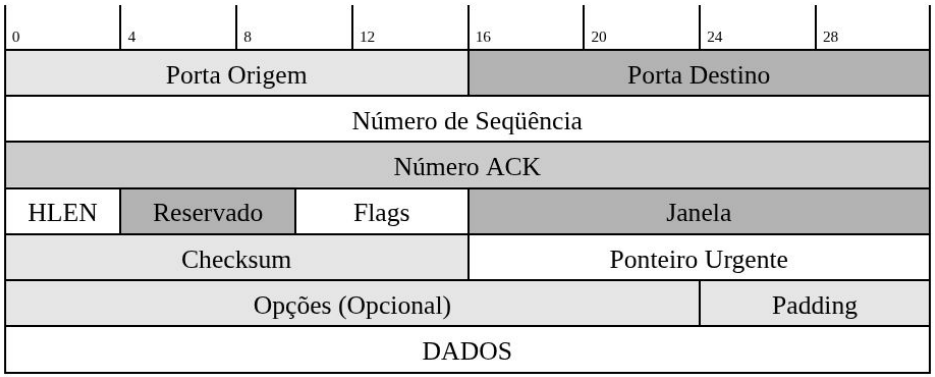
Handshake TCP



Protocolo TCP

- O UDP não tem relação forte de quem é servidor
  - Mas o TCP tem
- Definição do servidor:
  - Vai esperar uma conexão
  - Porta no modo PASSIVO
- Definição do Cliente
  - Vai INICIAR uma conexão
    - fará o handshake
  - Porta no modo ATIVO
    - de ação, tomar a iniciativa

Cabeçalho TCP



Confiabilidade implementada pelo TCP

- baseada em confirmações
  - Confirmação enviada na forma de próximo byte esperado
- uso de Syn e Acks
- Cada máquina escolhe um número sequencial inicial
  - Este número será a primeira posição do seu buffer de envio
  - No primeiro SYN, envia o seu número no campo sequência
- Recebe o número de B no campo sequência e o seqA + 1 no de ACK.
  - Este número vai ser usado para a primeira posição do buffer de recebimento
- Confirmações podem ser cumulativas
- Campo Janela = tamanho do buffer
- Demonstração

Uso de janelas deslizantes

- utiliza uma janela de tamanho conhecido
  - transmite todos da janela sem precisar receber confirmação
  - janela anda a medida que recebe confirmações
- Questões quanto ao tamanho da janela
  - Janela pequena:
    - travado, pouca memória
  - Janela grande:
    - Mais rápido (menos travado)
    - muita memória
    - problema do efeito cascata
    - perda de um pacote causa retransmissões desnecessárias
  - Janela = 0 => PAUSE