Redes de Computadores

Introdução à tolerância a falhas em UDP

Sumário

- Introdução à tolerância a falhas em UDP
  - Protocolo *Stop&Wait*
    - *Timeout* fixo
    - *Timeout* adaptativo
- Sumário
  - Tratamento de de falhas num protocolo Cliente/Servidor simples
  - Exemplo : Programação em Java
  - Exercícios propostos

Objectivo

- Compreender a problemática de tolerância a perca de mensagens na rede em protocolos Cliente-Servidor
- Aproximação à implementação prática (em Java) de um protocolo que tolere falhas que podem ocorrer:
  - Por **perca de mensagens**
  - Por **duplicação de mensagens**
  - Por **desordenação do fluxo de mensagens**
- Implementação de um protocolo simples para transferência fiável de ficheiros, suportado em comunicação em modo datagrama
  - Implementação com *sockets datagrama* (sobre protocolo de transporte UDP)
    - Notar que na comunicação suportada em UDP/IP é susceptível de acontecerem o tipo de "falhas" acima enunciadas (contrariamente à utilização de TCP/IP)

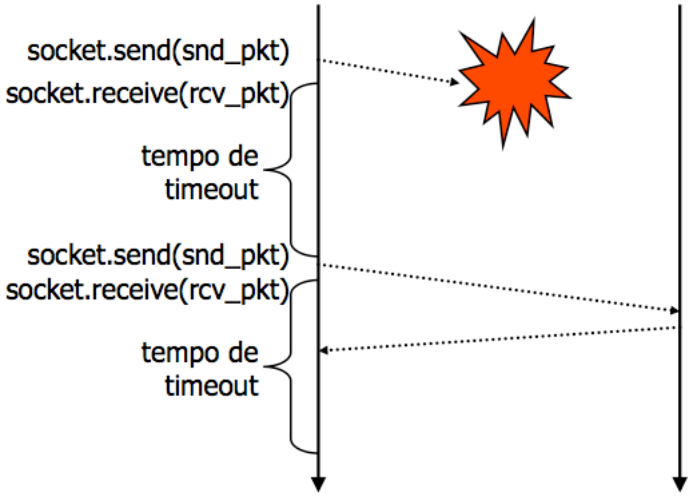
Tolerância a falhas

- Para tolerar as falhas transitórias na comunicação por UDP, o protocolo ("fiável") que se pretende implementar deve:
  - **Detectar pacotes perdidos**: nesse caso deve proceder à retransmissão dos mesmos
    - Detecção da perca (com incorpração de mecanismo de reconhecimento de recepção – ACKs)
    - Controlo da temporização e número de retransmissões
    - Incidência no lado do emissor
  - **Detectar duplicações**: e neste caso descartar pacotes duplicados
    - Incidência no lado do receptor
  - **Detectar desordenações**: controlando a sequência de chegada de pacotes para reordenação dos pacotes recebidos
    - Controlo de sequenciamento de mensagens no emsissor
    - Controlo da sequência esperada e obtenção da ordem correcta no lado do receptor

Comunicação baseada em UDP

- O protocolo UDP não garante a fiabilidade no envio das mensagens
  - Mensagens podem-se perder
  - Duas mensagens enviadas consecutivamente para um mesmo destino podem chegar por qualquer ordem
  - Uma mensagem pode ser recebida mais do que uma vez
- Para tratar estas falhas é necessário recorrer:
  - Ao reenvio das mensagens
  - À verificação da ordem das mensagens na recepção
  - À detecção de duplicados

Reenvio de uma mensagem



Reenvio de uma mensagem

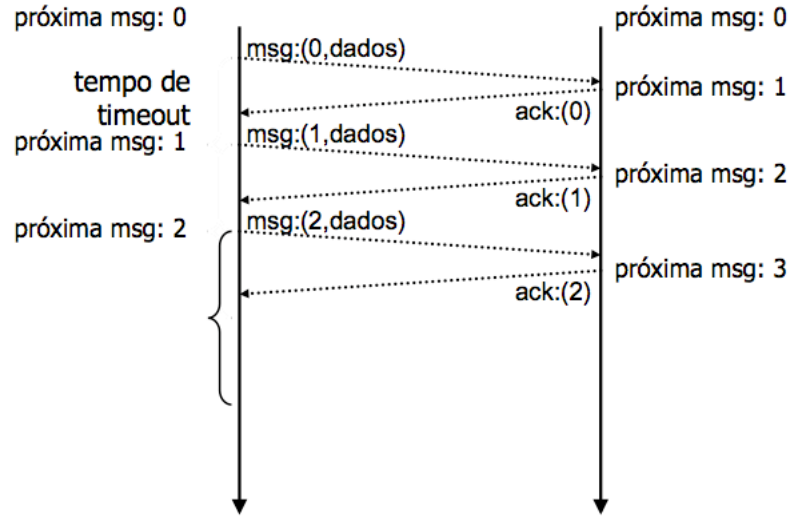
```
DatagramSocket socket = ...
DatagramPacket send_pkt = ...
DatagramPacket rcv_pkt = ...

socket.setSoTimeout( 5000);  //em milisegundos
for( int ntries = 0; ntries < 3; ntries++) {
   try {
         socket.send( send_pkt)
         socket.receive( rcv_packet);
         ... //mensagem recebida
         break;
   } catch( SocketTimeoutException e) {
         // reenviar
   }
}
```

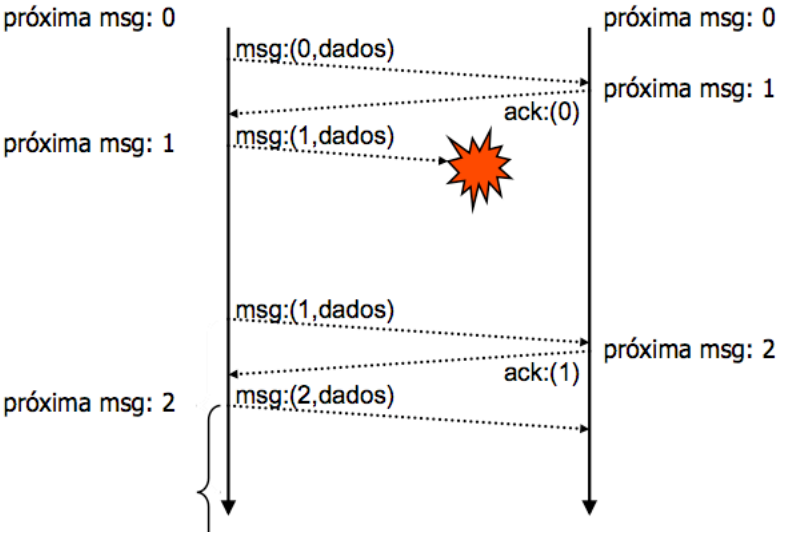
Tratamento das falhas

- O protocolo UDP pode ser usado para enviar uma sequência de mensagens
  - E.g.: transferência de ficheiro
- Neste tipo de protocolos, o tratamento de falhas exige que o cliente e o servidor identifiquem as mensagens que trocam
  - E.g.: usando número de sequência

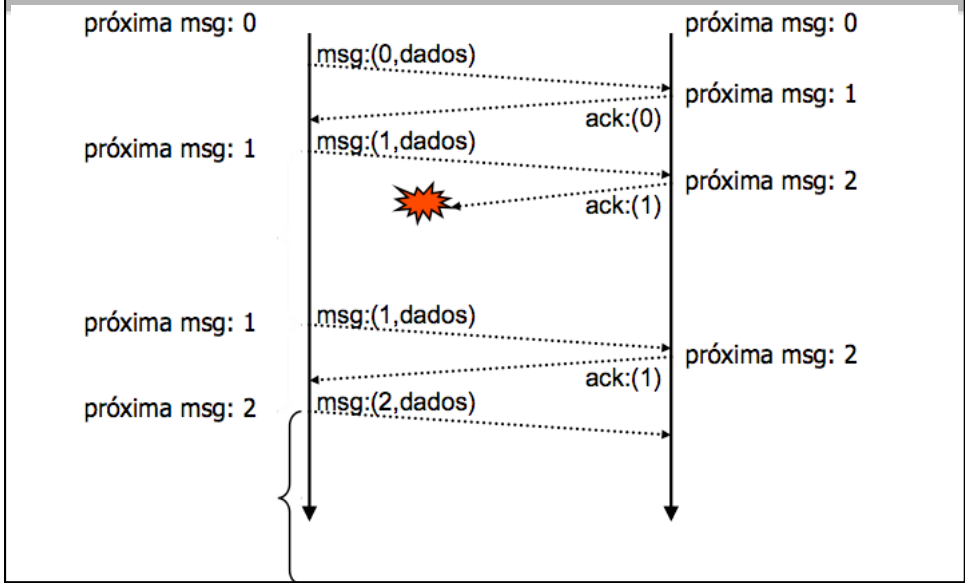
Transferência de ficheiro



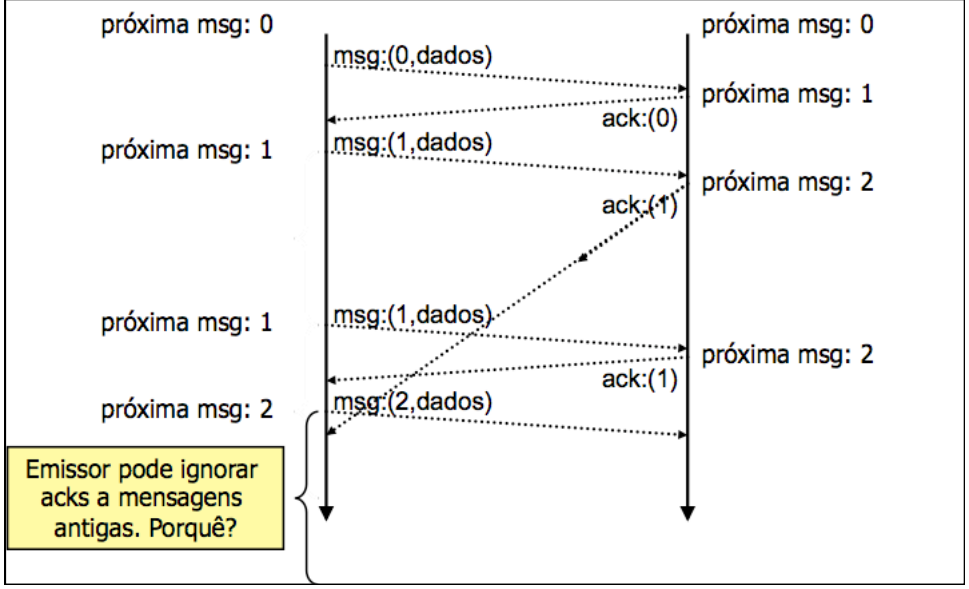
Transferência de ficheiro: tratamento de erros



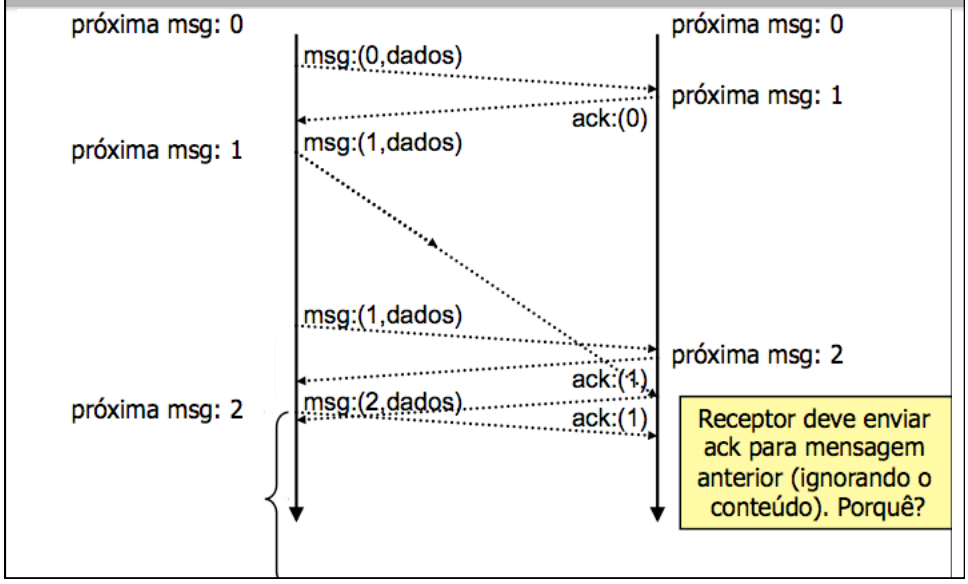
Transferência de ficheiro: tratamento de erros



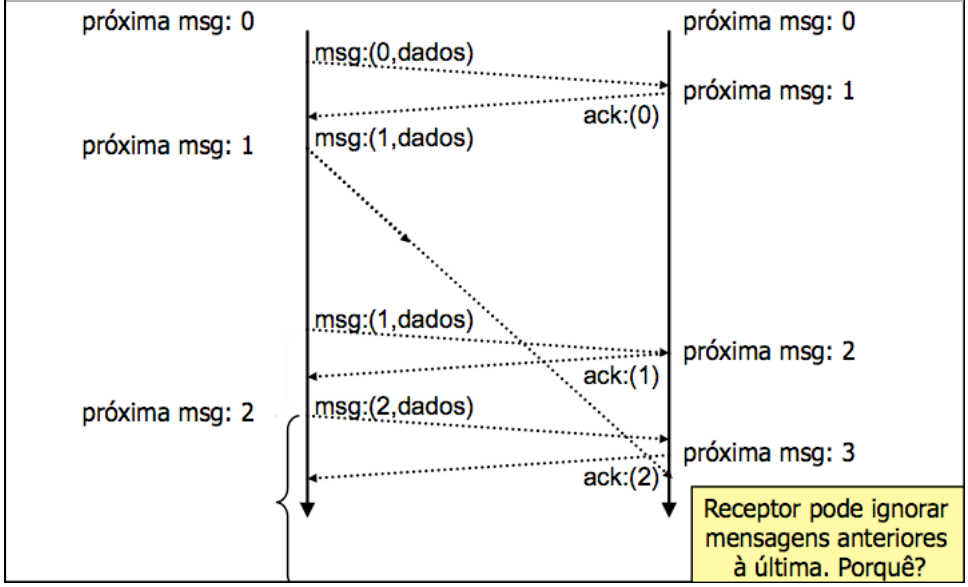
Transferência de ficheiro: tratamento de erros



Transferência de ficheiro: tratamento de erros



Transferência de ficheiro: tratamento de erros



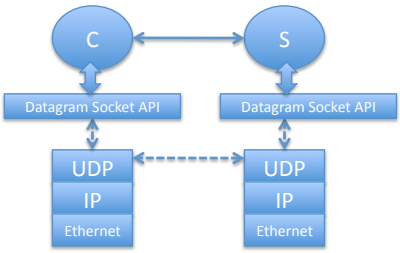
Exercício

- Implementar um sistema para transferência fiável de ficheiros (protocolo cliente/servidor baseado em protocolo ***stop & wait***)
  - Cliente envia o ficheiro (enviando os blocos em diversos pacotes)
  - Servidor recebe os pacotes (pode ir imprimindo ou escrevendo para disco)
    - Pode terminar no final da transferência

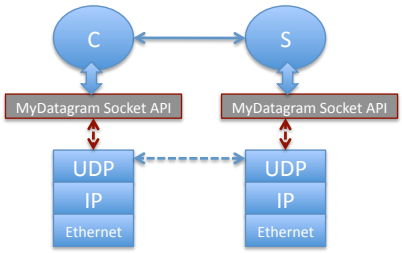
Abordagem para implementação

- 1ª abordagem (mais simples):
  - ACKs e retransmissão do emissor com controlo de temporização fixa (*timeout* constante)
  - Ordenação controlada no protocolo stop&wait
  - Receptor controla e descarta possíveis duplicados
- 2ª abordagem (ainda simples, mas melhor):
  - ACKs e retransmissão do emissor com controlo dinâmico de temporização adaptável (*timeout* variável) com base num estimador para o timeout de envio de cada pacote
  - Adaptação à variação das "condições da rede"
    - Obtido a partir da estimação do RTT observado com controlo de folga
    - **VER AULAS TEÓRICAS: CAP 4, Grupo 2 – Slide 45**
  - Receptor controla e descarta possíveis duplicados que podem ocorrer

Ensaio experimental da implementação



Localhost e ambiente de Rede Local: na prática não se observarão falhas



MyDatagramSocket:
Simula as falhas para ensaiar a fiabilidade do protocolo a implementar

(simulação do comportamento do UDP numa rede de grande escala)

Passos para fazer o exercício

- Estudar, compilar e executar o código do exemplo fornecido e compreender o comportamento observado.
- Compreender a utilização/programação de timeouts/retransmissões
- Verificar suportes adicionais fornecidos:
  - udp_filetransfer
  - myDatagramSocket:
    - #include net.*;
    - myDatagramSocket.init(NNNN,0)
    - socket = new myDatagramSocket("port")
  - File transfer utilities:
    - AckPacket.java, FilePacket.java
- Implementar o protocolo para transferência de ficheiros por UDP, de modo a concretizar uma transferência fiável, baseada no protocolo STOP&WAIT
  - Inicialmente com *timeouts* fixos
  - Depois com *timeout* dinâmico
- Ensaiar o protocolo usando a implementação "myDatagramSockets", observar e analisar:
  - A fiabilidade conseguida
  - O custo (overhead) do protocolo fiável no tempo de transferência

Para depois (próxima aula)

- Fazer e concluir o Stop&Wait com timeout ajustável
- Numa próxima aula (dedicada a comunicação por Multicast) iremos ver como poderíamos usar Multicast para se encontrar automaticamente o servidor, de modo a não ser necessário saber antecipadamente o HOST e o PORTO onde o mesmo está disponível.