Pontifícia Universidade Católica de Minas Gerais
Instituto de Ciências Exatas e Informática

Laboratório de Redes e SO

Programação de socket com TCP

Objetivos: Conhecer e verificar o funcionamento do protocolo de comunicação TCP e do uso de sockets.

O Protocolo TCP

O protocolo TCP (Transmission Control Protocol – RFC 793) como o próprio nome já diz é um protocolo que faz o controle da transmissão, assim, os dados sempre são entregues.

Principais características:

1. Orientado à conexão.
    - É estabelecida uma conexão entre o cliente e o servidor para transferir dados.
2. É confiável, garante entrega dos pacotes.
    - O protocolo TCP usa técnicas para garantir uma entrega confiável dos pacotes de dados. O TCP permite a recuperação de pacotes perdidos, a eliminação de pacotes duplicados, a recuperação de dados corrompidos. Se necessários, dados são retransmitidos.
3. Entrega Ordenada
    - O TCP garante a reconstrução do stream de dados no destinatário mediante os números de sequência, assim, mesmo que os pacotes não cheguem na ordem eles podem ser entregues ordenados.
4. Controle de fluxo
    - O protocolo TCP usa o campo janela (ou window) para controlar o fluxo. Á medida que os dados são entregues e é confirmada a recepção é informada a quantidade máxima de bytes aceita pelo receptor. Isso serve para evitar uma saturação da rede.

A imagem a seguir apresenta um esquema do cabeçalho do pacote TCP.

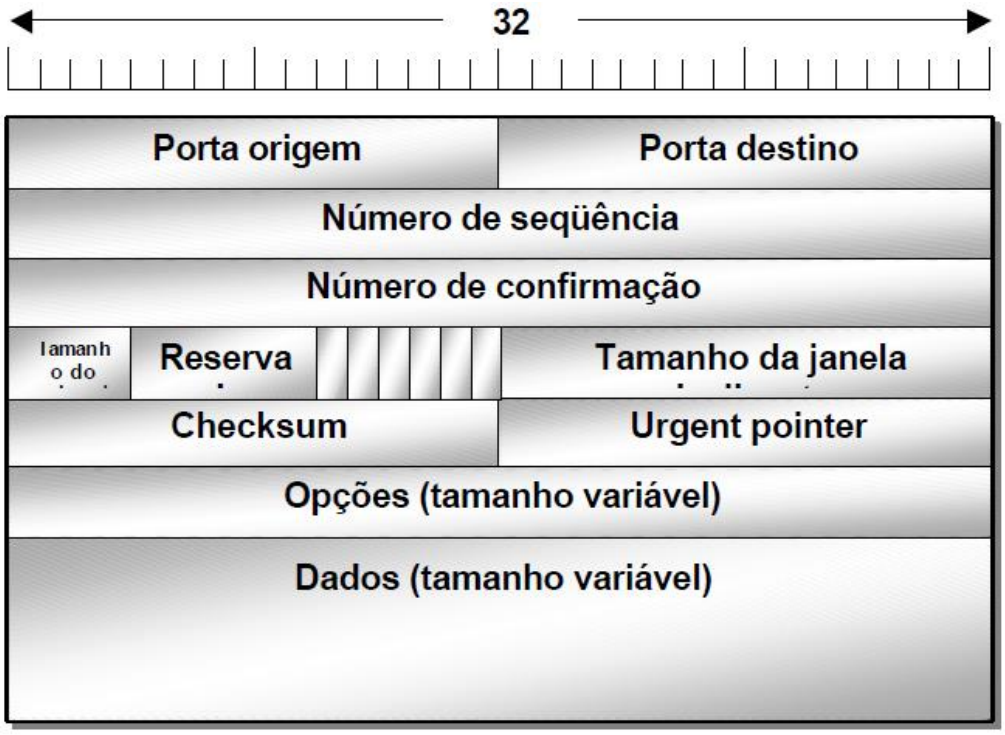


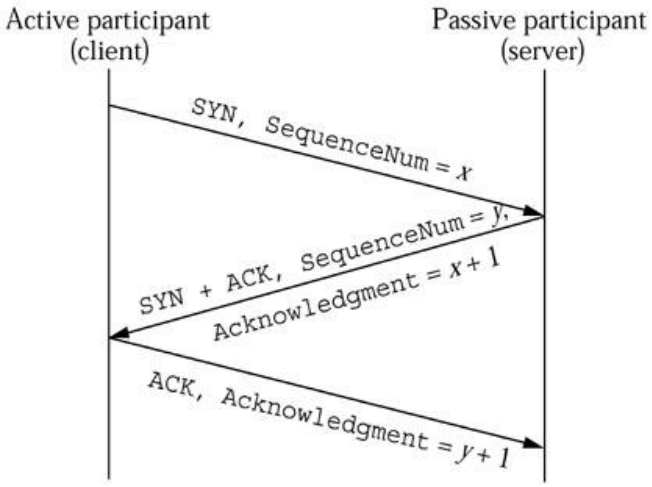
Figura 1 – Cabeçalho TCP

Os campos do TCP são:

- Porta Fonte (16 bits): Porta relativa à aplicação corrente na máquina fonte
- Porta Destino (16 bits): Porta relativa à aplicação corrente na máquina de destino
- Número de sequência (32 bits): identifica a posição no fluxo de bytes do segmento enviado pelo transmissor. O número de sequência refere-se ao fluxo de dados que vai na mesma direção do segmento.
- Número de confirmação (32 bits): ste campo identifica a posição do byte mais alto (ou último byte) que o fonte recebeu. O número de reconhecimento refere-se ao fluxo de dados na direção contrária ao segmento. Os reconhecimentos sempre especificam o número do próximo byte que o receptor espera receber..
- Offset (4 bits) : permite localizar o início dos dados no pacote. contém um inteiro que especifica o início da porção de dados do segmento. Este campo é necessário já que o campo Options varia em comprimento dependendo de quais opções tenham sido incluídas. De modo que o tamanho do cabeçalho TCP varia dependendo das opções selecionadas.
- Reservado(6 bits): previsto para o futuro
- CODE (flags) (6x1 bit): representam informações suplementares:
    1. URG: deve ser tratado de maneira urgente.
    2. ACK: é um aviso de recepção.
    3. PSH (PUSH): o pacote funciona de acordo com o método PUSH.
    4. RST: a conexão é reiniciada.
    5. SYN: pedido de estabelecimento de conexão.
    6. FIN: a conexão é interrompida.

- Tamanho Janela (16 bits): através deste campo o software TCP indica quantos dados ele tem capacidade de receber em seu buffer.
- Checksum: (ou CRC) : é usado para verificar a integridade tanto do cabeçalho como dos dados do segmento TCP.
- Ponteiro de emergência (ou Urgent Pointer - 6 bits): através deste campo permite que o transmissor especifique que alguns dados são urgentes, isto significa que os dados serão expedidos tão rápido quanto seja possível.
- Opções (Dimensão variável): Opções diversas
- Dados : dados trafegados.

1. Para que ocorra a comunicação, as duas máquinas devem sincronizar as suas sequências, através do mecanismo chamado three ways handshake (aperto de mãos em três tempos). Esse mecanismo também ocorre no encerramento de sessão.



2.

Figura 2 - 3-way handshake

3. O Servidor deve estar ativo, ou seja “escutando” em uma porta. O cliente transmite uma mensagem de SYN com um número (X), o número de ordem inicial do cliente.

4. O servidor recebe a mensagem inicial do cliente, e envia uma mensagem acusando a recepção, um ACK. Esta mensagem contem um número do servidor (Y) e o número de ordem inicial do cliente mais um (X+1).

Por último, o cliente envia uma mensagem acusando a recepção, do tipo ACK, que contem o número do servidor mais um (Y+1)

Sockets

Para que uma aplicação possa trocar dados remotamente com outra aplicação elas devem criar um “canal” de comunicação. O socket provê uma interface de comunicação entre aplicações fazendo o mapeamento de um par: IP:porta entre dois computadores.

Os sockets podem ser usados para comunicação via qualquer um dos protocolos UDP ou TCP. Existem vários tipos de sockets, em geral um para cada tipo de protocolo. Nesta e na próxima prática vamos trabalhar com dois tipos o "Stream Sockets" (ou SOCK_STREAM) e o "Datagram Sockets" (ou SOCK_DGRAM).

Nesta prática vamos usar o "Stream Sockets" (ou SOCK_STREAM) para o TCP.

Uso de Sockets para envio de dados por TCP em Java

Quando temos uma conexão entre dois computadores um deve assumir o papel de Servidor e o outro de Cliente. O servidor fica aguardo uma solicitação do cliente. O cliente inicia a conversação enviando alguma mensagem ao servidor.

A figura a seguir ilustra uma comunicação entre duas máquinas.

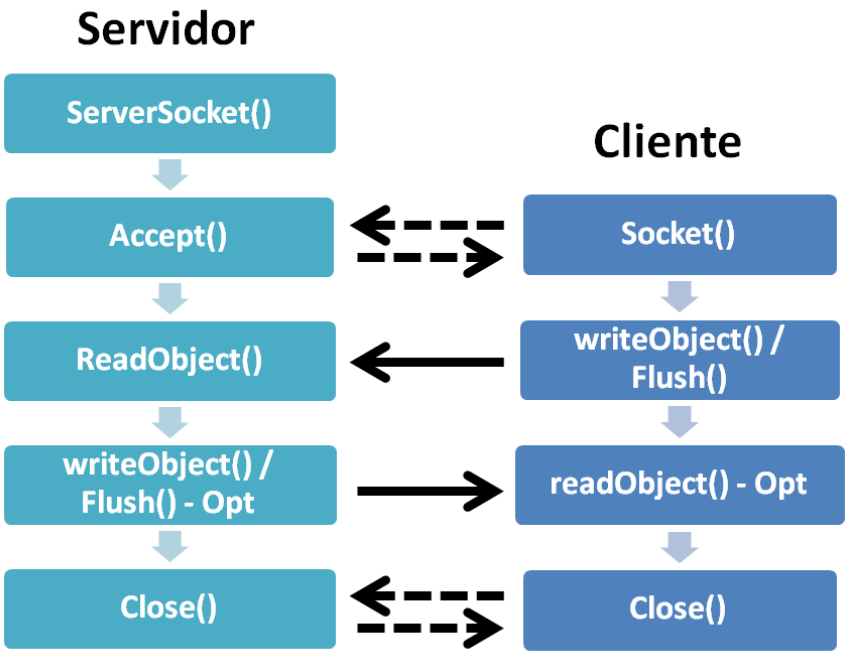


Figura 3 - Comunicação TCP

O código a seguir mostra a como o socket é criado e a mensagem é enviada pelo **cliente**:

```
// Cria o socket na porta definida, abre uma conexão
Socket socktCli = new Socket (IPServidor,PortaServidor);
// Cria uma “fluxo de saída” para enviar os dados
ObjectOutputStream sCliOut = new
                        bjectOutputStream(socktCli.getOutputStream());
// Cria a mensagem a ser enviada
sCliOut.writeObject("MENSAGEM TESTE");
// Envia a mensagem para o servidor
sCliOut.flush();
```

O código a seguir mostra a como o socket é criado e a mensagem é recebida pelo **servidor**:

```
// Cria o socket na porta definida, abre uma conexão
ServerSocket socktServ = new ServerSocket(PortaServidor);
Socket conSer = socktServ.accept();
// Cria uma “fluxo de entrada” para receber os dados
ObjectInputStream sServIn = new
                            ObjectInputStream(conSer.getInputStream());
// Aguarda a mensagem do cliente
Object msgIn = sServIn.readObject();
// Imprime mensagem
System.out.println(" -S- Recebido: " + msgIn.toString());
```

Os arquivos ServidorTCP.java e ClienteTCP.java contem os fontes de uma aplicação que envia uma mensagem TXT e recebe a mesma mensagem de volta acrescida da String “Retorno”.

Para executar o código do Servidor/Cliente basta compilar os arquivos.,.java e gerar os arquivos .class. Isso pode ser feito em uma IDE (como Eclipse) ou na linha de comando do Windows.

Para compilar execute:

```
C:> javac ServidorTCP.java
```

Para executar o Sevidor/Cliente:

```
C:> java ServidorTCP
```

Ao executar o comando acima o servidor/cliente serão exibidas as saídas do programa e as mensagens serão trocadas.

Atividades

Use a mesma proposta do exercício de UDP, com um segundo computador ou com uma VM, se não for possível capture no Wireshark a interface loopback.

O servidor deve ser executado e em seguida o cliente deve ser executado para as atividades de 1 a 4.

1 – Enviar uma mensagem do cliente em uma máquina para o servidor na mesma máquina e observar as mensagens de retorno (Mensagem da maq. A para maq. A).

2 – Enviar uma mensagem do cliente em uma máquina para o servidor na outra máquina e observar os pacotes trafegados. (Mensagem da maq. A para maq. B).

3 – Enviar uma mensagem de duas máquinas cliente (ao mesmo tempo) para apenas um dos servidores e observar os pacotes trafegados (Mensagens das maqs. A e B para maq. A).

4 – Parar o servidor. Enviar uma mensagem do cliente para o servidor na mesma máquina e observar os pacotes trafegados (Mensagem da maq. A para maq. A).

5 – Executar o cliente e depois executar o servidor. Observar os pacotes trafegados e o comportamento da aplicação.

6 – Alterar o código do servidor de forma que ele possa receber 3 mensagem de máquinas diferentes antes de ser finalizado. Executar o servidor, enviar 3 mensagens de clientes diferentes e observar o resultado.