Programação de socket com UDP

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

Protocolos de Rede

"*Um protocolo é uma convenção ou padrão que controla e possibilita uma conexão, comunicação, transferência de dados entre dois sistemas computacionais. De maneira simples, um protocolo pode ser definido como 'as regras que governam' a sintaxe, semântica e sincronização da comunicação. Os protocolos podem ser implementados pelo hardware, software ou por uma combinação dos dois*." *Fonte: http://pt.wikipedia.org/wiki/Protocolo_de_rede*

Existem vários tipos de protocolos em cada camada de rede. A tabela a seguir lista alguns dos principais:

Camada	**Protocolos**
5. Aplicação	HTTP, SMTP, FTP, SSH, Telnet, SIP, RDP, IRC, SNMP
4. Transporte	TCP, UDP, RTP, SCTP, DCCP
3. Rede	IP (IPv4, IPv6) , ARP, RARP, ICMP
2. Enlace	Ethernet, 802.11 WiFi, IEEE 802.1Q, 802.11g
1. Física	Modem, RDIS, RS-232,EIA-422, RS-449, Bluetooth

O Protocolo UDP

O protocolo UDP (User Datagram Protocol - RFC 768) é um protocolo da camada de transporte que fornece um meio para os usuários enviarem datagramas IP encapsulados sem que seja necessário estabelecer uma conexão.

Principais características:

1. Não-orientado à conexão, não confiável.
    - O protocolo não fornece nenhum mecanismo de controle de entrega de pacotes, ou seja, se o pacote não chegar ao destino ele não será retransmitido.
    - Não existe necessidade de manter uma conexão de sockets entre o servidor e o cliente, cada envio de pacote é independente e a entrega dos pacotes pode ser de forma não ordenada.
2. Usado em aplicações tipo cliente-servidor que usam mecanismo request-reply.
    - O protocolo é feito para transmissão de dados de tempo real ou que não sejam muito sensíveis, como streaming de áudio ou vídeo.

3. Usado onde velocidade é mais importante que confiabilidade
    - Seu cabeçalho é bem reduzido e não tem nenhuma mensagem adicional de controle, assim, é mais eficiente na transmissão.
    - Pacotes recebidos fora de ordem ou corrompidos são descartados.

O UDP transmite segmentos que consistem em um cabeçalho de apenas 8 bytes mais os dados do usuário. Basicamente o UDP adiciona ao IP bruto as informações de portas

A imagem a seguir apresenta um esquema do dado em um pacote UDP e um pacote IP.

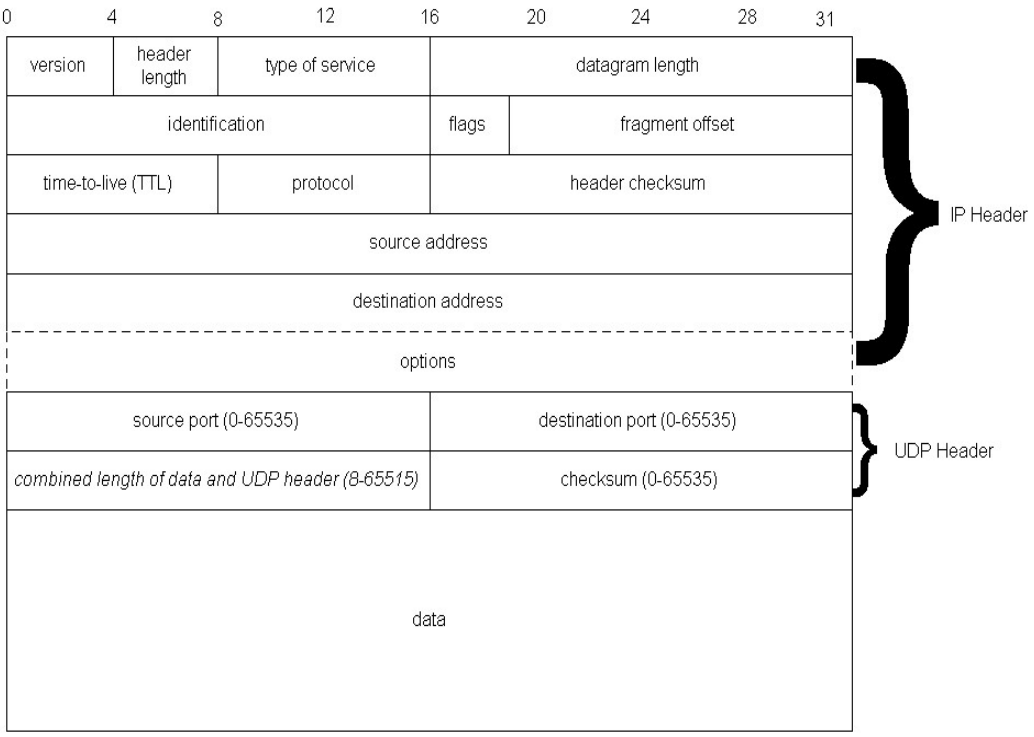

Figura 1 - Cabeçalhos - IP/UDP

Os campos do UDP são:

- Porta Fonte (source port): contem o número de porta que corresponde à aplicação emissora do segmento UDP. Este campo representa um endereço de resposta para o destinatário. É opcional para mensagem que não terá resposta. São usados 16 Bits.
- Porta Destino (destination port) : contém o número da porta que corresponde à aplicação da máquina destinatário da mensagem. Campo obrigatório. São usados 16 Bits.
- Comprimento (combined lenght of data): contém o comprimento total do segmento, incluindo o cabeçalho.
- Validador (checksum) : contem um valor de controle, utilizado para verificar a integridade do segmento.

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).

O "Datagram Socket", que é chamado de socket sem conexão, é usado pelo protocolo UDP. Este tipo de socket implementa uma conexão não confiável, ou seja, se você enviar um datagrama, ele pode chegar, pode chegar fora de ordem, ou pode não chegar.

Basicamente o funcionamento deste socket é feito da seguinte maneira: um pacote é criado, é anexado um cabeçalho IP com informações de destino, e é feito o envio.

A figura a seguir ilustra o envio de pacote a um servidor:

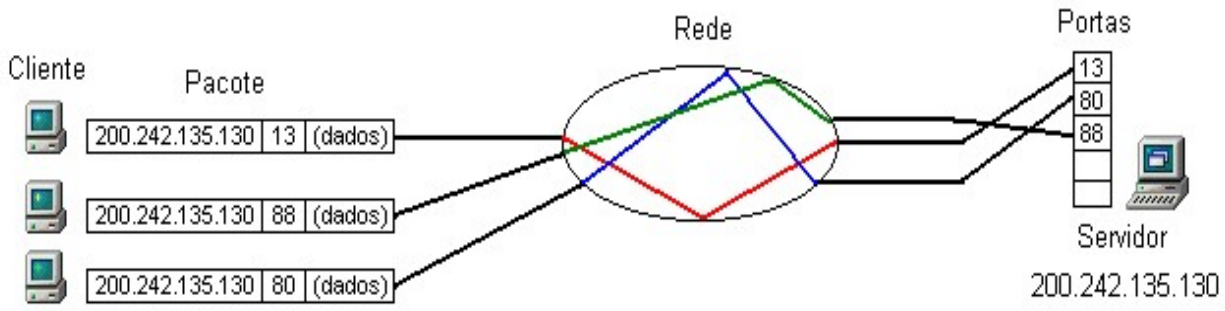

Figura 2 - Envio de mensagens

Uso de Sockets para envio de dados por UDP 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. O servidor cria um socket e fica aguardando o recebimento de alguma mensagem. O cliente também cria um socket e envia uma mensagem para o servidor. O servidor pode responder a mensagem (ou, não, isso é opcional) que será recebida pelo cliente. Ao final desta comunicação, ambos, finalizam seus sockets.

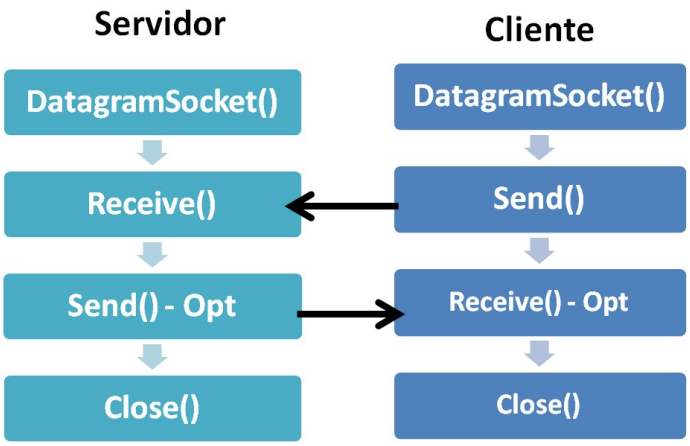

Figura 3 - Comunicação UDP

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

```
// Cria o socket na porta definida para o servidor
DatagramSocket ds = new DatagramSocket(PortaServidor);
// Cria uma mensagem e converte para byte
String strEnvio = "MENSAGEM";
byte[] bytEnvio = strEnvio.getBytes();
// Cria o pacote com os dados + IP:Porta do servidor
DatagramPacket pktEnvio = new DatagramPacket(bytEnvio,
   bytEnvio.length, InetAddress.getByName(IPServidor), PortaServidor);
// Envia o pacote com os dados
ds.send(pktEnvio);
```

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

```
// Cria o socket na porta definida
DatagramSocket ds = new DatagramSocket(ServerPort);
// Cria variável para receber os dados
byte[] bytRec = new byte[100];
DatagramPacket pktRec = new DatagramPacket(bytRec, bytRec.length);
// Recebe o pacote com os dados
ds.receive(pktRec);
// Recupera os dados do pacote
bytRec = pktRec.getData();
String strMsg = new String(bytRec, 0,bytRec.length);
```

Os arquivos Servidor.java e Cliente.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 Sevidor/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 Servidor.java
```

Para executar o Sevidor/Cliente:

```
C:> java Servidor
```

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

Atividades

As atividades deverão sem em duplas (mas cada um em computador diferente). Os resultados devem ser observados com o Wireshark.

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 os pacotes trafegados (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 das 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.