Redes de Computadores I

Aula 14 - Protocolo UDP

Apresentação

Na aula passada, você estudou o protocolo de transporte TCP e viu que ele é bastante completo, oferecendo uma série de facilidades para o desenvolvedor das aplicações. Acontece que nem sempre todas aquelas características do TCP são necessárias, ou mesmo desejáveis, para todas as aplicações. Assim sendo, nesta aula, estudaremos o segundo protocolo de transporte mais utilizado na internet, que é um protocolo bastante simples e se chama ***User Datagram Protocol*** (**UDP** – Protocolo de Datagrama do Usuário). Após o estudo do UDP faremos uma discussão sobre quando se deve usar TCP e quando usar UDP, e, por fim, mostraremos como os programas utilizam esses dois protocolos.

Vídeo 1 - Apresentação

Objetivos

Após o final desta aula, você será capaz de:

- Identificar as funcionalidades oferecidas pelo protocolo UDP aos desenvolvedores de aplicações.
- Decidir qual protocolo de transporte utilizar quando for escrever um programa que transmita informações pela rede.
- Identificar a estrutura de um programa cliente e de um programa servidor que utilizam UDP para se comunicarem.
- Identificar a estrutura de um programa cliente e de um programa servidor que utilizam TCP para se comunicarem.

Protocolo UDP

O protocolo UDP é um protocolo de transporte bastante simples que procura oferecer para as aplicações um serviço de entrega de pacotes básico. Este serviço consiste apenas em colocar a mensagem recebida da camada de aplicação dentro de um segmento (o chamaremos de pacote UDP), utilizando os números de portas para identificar as aplicações. Portanto, as características oferecidas pelo UDP são praticamente as mesmas que o próprio IP oferece, com a diferença que o pacote UDP utiliza números de porta para identificar as aplicações.

Uma boa forma de você entender o UDP é comparando ele com o TCP. A seguir listamos as principais diferenças entre os dois protocolos.

- O UDP não é baseado em conexão. Desse modo, quando uma máquina deseja transmitir uma informação para outra, ela apenas prepara o pacote UDP e o envia, sem saber nem mesmo se a máquina destino está ligada ou se a aplicação que deverá receber a informação está executando.

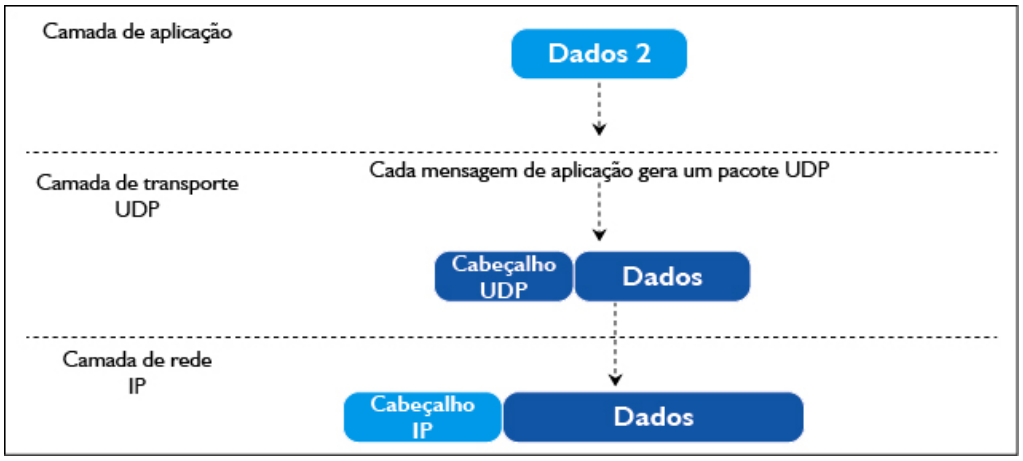
- O UDP não garante que os pacotes serão entregues na ordem em que foram transmitidos. Ou seja, os pacotes são passados para a camada de aplicação na máquina destino na ordem em que são recebidos por ela – e essa ordem pode ser diferente da que foram transmitidos.

- O UDP não garante a entrega de pacotes, pois ele não retransmite pacotes perdidos ou com erro. Na verdade, a origem nem sabe se os pacotes foram perdidos. Como você verá mais adiante, existe um campo de checksum no pacote UDP que permite a detecção de pacotes que chegam com erro. Mas os pacotes recebidos com erro são descartados sem que esse fato seja avisado à máquina que transmitiu o pacote.

- Não existe nenhum mecanismo para controlar a taxa com que os pacotes são enviados para a máquina de destino. Eles são enviados na taxa que a aplicação os gerar.

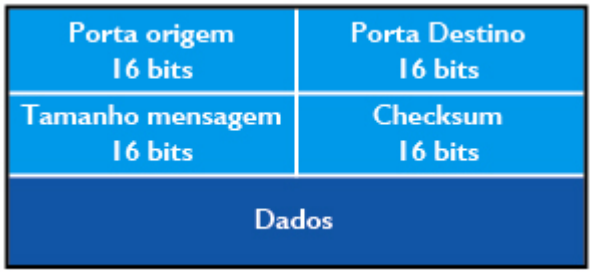
- O UDP não realiza segmentação nem blocagem, ou seja, cada mensagem da camada de aplicação que é passada para o UDP gera exatamente um pacote UDP. A **Figura 1** mostra esse procedimento. Observe que Dados' é formado pelo conteúdo do cabeçalho UDP mais Dados.

Figura 01 - Cada mensagem de aplicação gera exatamente um pacote UDP



Normalmente, quanto mais funcionalidades um protocolo tem, mais campos ele precisa ter no seu cabeçalho. Como o UDP não oferece muitas funcionalidades, isso se reflete no formato do pacote, fazendo com que ele seja bastante simples. A **Figura 2** mostra o formato de um pacote UDP. O campo *Tamanho da Mensagem* é o tamanho total do pacote UDP em bytes (incluindo o cabeçalho e os dados). Observe que **Dados** não é um campo do cabeçalho! Ele apenas indica que após o cabeçalho UDP, o pacote contém a mensagem recebida da camada de aplicação.

Figura 02 - Formato do pacote UDP



Veja aqui a explicação, em vídeo, sobre o protocolo UDP.

Vídeo 2 - UDP

Atividade 01

1. O protocolo UDP retransmite pacotes perdidos?
2. Quando uma máquina recebe um pacote UDP com erro ela avisa a máquina que o enviou?
3. Se o tamanho de uma mensagem passada para o protocolo UDP for muito grande, o UDP a divide para ser transmitida em dois ou mais pacotes UDP?

Escolhendo qual protocolo utilizar: TCP ou UDP

O fato de o protocolo UDP ser muito mais simples do que o TCP não significa que ele é pior. É verdade que se você utilizar UDP e precisar de alguma característica que ele não fornece, como numeração dos pacotes, por exemplo, seu programa é que terá que implementar essa função. Mas isso lhe permite ter uma solução (protocolo de transporte + sua aplicação) que tenha apenas as características necessárias. Além disso, o UDP é muito mais leve e rápido do que o TCP, pois seu cabeçalho é menor e ele precisa realizar menos operações. Podemos citar alguns pontos importantes a serem considerados no UDP:

- Perda de pacotes – Algumas aplicações suportam a perda de pacotes, como é o caso de aplicações que transmitem voz sobre o IP. Quando usamos TCP, se algum pacote é perdido os demais pacotes recebidos não podem ser passados para a aplicação, até que o pacote perdido seja retransmitido e chegue com sucesso. Isso implica que se você está transmitindo um áudio pela rede e reproduzindo na máquina de destino e um pacote se perde, o destino irá para a reprodução do áudio até receber o pacote perdido, gerando aquele conhecido efeito de “corte” quando

ouvimos músicas pela internet. Talvez fosse melhor ignorar o pacote perdido e continuar a reprodução do áudio com os pacotes que já chegaram, pois o ouvido humano provavelmente não perceberia a perda.

- Multicast e Broadcast – O UDP suporta multicast e broadcast que são formas de transmissão para várias máquinas simultaneamente. O TCP não suporta esse recurso, exigindo uma conexão individual para cada máquina. Naturalmente, isso pode acarretar um maior tráfego na rede, para enviar as mesmas informações.

- Número de pacotes transmitidos – Para aplicações que usam TCP e transmitem um grande número de pacotes, os pacotes trocados para abertura e encerramento da conexão não farão muita diferença. Entretanto, existem aplicações que transmitem apenas uma pequena quantidade de informação, por exemplo, um pacote com a solicitação e recebem a resposta em outro pacote. Nesses casos, gastar três pacotes para abrir a conexão e quatro para o encerramento da conexão significa uma grande sobrecarga. Ou seja, ao invés serem transmitidos apenas dois pacotes (1 + 1) entre as duas máquinas, são necessários aproximadamente nove (3 + 2 + 4).

Podemos dizer que não existe um protocolo melhor ou pior, existe o mais adequado às necessidades de cada aplicação. Se sua aplicação, por exemplo, transmite pequenas quantidades de informação de cada vez ou pode lidar com a perda de pacotes, considere a possibilidade de usar UDP. Caso sua aplicação precise de confiabilidade na entrega dos pacotes, ou seja, que eles cheguem à ordem correta e que os pacotes perdidos sejam retransmitidos, use TCP. Além disso, o mecanismo de controle de fluxo do TCP é muito importante quando as aplicações se comunicando estão em máquinas situadas em diferentes redes na internet. A verdade é que a maioria das aplicações precisa de confiabilidade e, portanto, utilizam TCP.

Como exemplos de protocolos da camada de aplicação que utilizam UDP, podemos citar: DNS (tradução de nomes para endereços IP), DHCP (configuração automática do TCP/IP) e SNMP (gerência). Como exemplos de protocolos que utilizam TCP, podemos citar: SMTP (e-mail), POP3 (e-mail), IMAP (e-mail), HTTP (web), FTP (transferência de arquivo), LDAP (autenticação, entre outros), SMB (compartilhamento de arquivos) e SSH (terminal remoto).

Vídeo 3 - TCP x UDP

Como uma aplicação utiliza a camada de transporte

Você já sabe que os programas utilizam um protocolo de transporte para transmitirem suas informações. A forma exata que o seu programa terá depende da linguagem de programação que você estiver utilizando e da API de rede que você vai utilizar. Você pode usar sockets, RPC, RMI, entre outras abordagens. Não se preocupe com o que esses nomes significam nesse momento. O que importa é que existe várias formas de fazer um programa transmitir informações pela rede.

Nesta seção, não pretendemos de modo algum lhe ensinar a programar em rede, queremos apenas lhe dar uma noção de como a camada de transporte é utilizada pelos programas. Para isso, vamos utilizar uma linguagem bem simples, com nomes de funções fictícios, para exemplificar as funções que um programa utilizaria. São essas funções que fazem a comunicação do programa com a camada de transporte.

Programa TCP cliente

Imagine um programa cliente que utiliza TCP para transmitir dados para um servidor, conforme mostrado na **Figura 3**. Naturalmente, o servidor também utiliza TCP. Inicialmente, o cliente estabelece uma conexão com o servidor utilizando a função *Conectar*. Depois o programa transmite dados para o servidor usando a função *Enviar_dados*. Essa função recebe como parâmetros a conexão por onde enviar os dados, e os dados a serem transmitidos.

Normalmente, após transmitir Dados o cliente deve receber algum dado enviado como resposta aos dados transmitidos. Isso é feito com a função *Receber_dados*. Essa função recebe como parâmetro a conexão de onde ler os dados e um buffer

para onde os dados recebidos devem ser copiados. Apesar de só aparecer uma chamada para cada uma das funções *Enviar_dados* e *Receber_dados*, tipicamente um programa chama essas funções diversas vezes.

A função *Enviar_dados* passa os dados que a aplicação deseja transmitir para a camada de transporte TCP, mas lembre-se que nada garante que para cada chamada a essa função será gerado exatamente um pacote TCP (veja segmentação e blocagem).

A função *Receber_dados* copia o conteúdo da parte de dados do pacote de transporte recebido (que são os dados transmitidos pela aplicação remota) para o *buffer* informado como parâmetro. Depois a aplicação iria analisar o *buffer* para interpretar a informação que foi recebida.

Quando o programa não desejar mais transmitir (nem receber) dados, ele solicita o fechamento da conexão.

```
1 Conexão = Conectar (IP, Porta)
2 Enviar_dados (conexão, Dados)
3 Receber_dados (conexão, buffer)
4 Fechar_conexão (conexão)
```

Figura 3 - Exemplo de programa cliente usando TCP

Para listar as conexões estabelecidas e qual programa está usando cada conexão na sua máquina Linux, digite:

```
1 # netstat -tnp
```

Cada letra no comando é chamada um *flag* e tem o seguinte significado:

- t: listar apenas informações a respeito de conexões TCP (as informações sobre comunicações usando UDP, por exemplo, não seriam mostradas).
- n: mostrar todas as informações, como números mesmo, sem tentar traduzir para o nome equivalente (exemplo: endereços IP e números de porta).
- p: mostrar o nome do processo (programa) que está usando a conexão.

Programa TCP servidor

O código do programa servidor é ligeiramente diferente do cliente, pois ele precisa realizar algumas tarefas a mais que o cliente. O código do servidor é mostrado na **Figura 4**.

Inicialmente, é necessário solicitar a porta desejada ao sistema operacional. Depois disso, o programa servidor tipicamente entra em loop infinito, uma vez que os servidores devem executar indefinidamente. Dentro desse loop o programa deve esperar que algum cliente conecte e aceitar essa nova conexão. Quando isso ocorre, deve ler os dados enviados pelo cliente, que tipicamente contêm alguma solicitação, e processar essa solicitação, que poderia ser, por exemplo, o pedido de uma página web. A seguir, o servidor tipicamente envia dados para o cliente, por exemplo, a página web solicitada. De modo semelhante ao cliente, podem existir várias chamadas a *Enviar_dados* e *Receber_dados* dentro do loop do servidor.

Finalmente, o servidor encerra a conexão e volta para o início do loop para esperar por novas conexões. Quando algum evento causar o encerramento do programa servidor a porta que havia sido solicitado ao sistema operacional é liberada.

```
1 Solicitar_Porta (porta)
2 Inicio de Loop
3 Conexão = Esperar_e_Aceitar_Conexão (porta)
4 Receber_dados (conexão, buffer)
5   Processar mensagem recebida
6 Enviar_dados (conexão, Dados)
7 Fechar_conexão (conexão)
8 Fim loop
```

Figura 4 - Exemplo de programa servidor usando TCP

Para listar as os programas que estão esperando conexões TCP, e em que portas, em uma máquina Linux, digite:

```
1 # netstat -tlnp
```

Cada letra no comando é chamada um *flag* e tem o significado descrito a seguir (observe que a única diferença do comando utilizado no cliente é o flagl):

- t: listar apenas informações a respeito de conexões TCP (as informações sobre comunicações usando UDP, por exemplo, não seriam mostradas).

- n: mostrar todas as informações, como números, sem tentar traduzir para o nome equivalente (exemplo: endereços IP e números de porta).

- l: mostrar as portas em que as aplicações estão escutando (esperando conexão).

- p: mostrar o nome do processo (programa) que está usando a conexão.

Programa UDP cliente

Imagine agora um programa cliente que utiliza UDP para transmitir dados para um servidor, conforme mostrado na Figura 5. Naturalmente, o servidor também utiliza UDP.

Observe que como o UDP não utiliza o conceito de conexão o cliente já envia os dados, sem estabelecer nenhuma comunicação prévia com o servidor. Ou seja, o cliente não sabe nem mesmo se a máquina para quem ele deseja transmitir está ligada, nem se o programa servidor está sendo executado!

Diferente da função *Enviar_Dados* do TCP, que identificava a máquina de destino pelo parâmetro (variável) conexão, a função para transmitir dados usando UDP recebe como parâmetros o IP e a porta de destino. O mesmo se aplica a função *Receber_Dados*.

Portanto, um programa UDP consiste, basicamente, de chamadas as funções para enviar os dados e para ler os pacotes que chegam pela rede.

A função *Enviar_dados* passa os dados que a aplicação deseja transmitir para a camada de transporte UDP, gerando exatamente um pacote UDP para cada chamada dessa função.

Receber_dados no UDP copia o conteúdo da parte de dados do pacote de transporte recebido (que são os dados transmitidos pela aplicação remota) para o buffer informado como parâmetro. Depois a aplicação iria analisar o *buffer* para interpretar a informação que foi recebida.

```
1 Enviar_dados (IP_Porta, Dados)
2 Receber_dados (IP_Porta, buffer)
```

Figura 5 - Exemplo de programa cliente usando UDP

Por utilizarem conexão, os pacotes transmitidos por clientes TCP usam sempre a mesma porta de origem. Isso é fundamental para que o servidor possa identificar a aplicação cliente com quem está se comunicando. Se o UDP não usa conexão, será que cada pacote UDP transmitido pelo cliente sai com uma porta de origem diferente?

Felizmente não! O sistema operacional garante que todos os pacotes transmitidos por sua aplicação cliente para uma dada aplicação remota (em outra máquina) saiam sempre com a mesma porta de origem. Isso permite que o servidor identifique o cliente, mesmo sem utilizar conexão! Para isso, na primeira vez que seu programa transmite um pacote para um dado IP e porta, o sistema operacional aloca uma porta para você. Todos os demais pacotes que sua aplicação transmitir para esse dado IP e porta sairão usando como porta de origem a porta que foi alocada.

Programa UDP Servidor

Veja na **Figura 6** que a estrutura de um programa servidor UDP é semelhante ao de um servidor TCP. A principal diferença é que o servidor UDP não precisa esperar por conexões e, consequentemente, não precisa fechar conexões. Ao ser executado esse programa iria ficar esperando na função *Receber_dados* até que algum pacote UDP chegasse. Quando isso acontecesse o programa continuaria, após copiar o conteúdo do pacote recebido para o *buffer*.

Veja que as funções *Enviar_dados* e *Receber_dados* são idênticas as utilizadas no programa cliente UDP.

```
Solicitar_Porta (porta)
Inicio de Loop

Receber_dados (IP_Porta, buffer)
    Processar mensagem recebida
Enviar_dados (IP_Porta, Dados)

Fim loop
```

Figura 6 - Exemplo de programa servidor usando UDP

Para listar os programas que estão esperando pacotes UDP, e em que portas, em uma máquina Linux, digite:

```
netstat -ulnp
```

Cada letra no comando é chamada um *flag* e tem o seguinte significado:

- u: listar apenas informações a respeito de comunicações usando UDP.
- n: mostrar todas as informações, como números, sem tentar traduzir para o nome equivalente (exemplo: endereços IP e números de porta).
- l: mostrar as portas em que as aplicações estão escutando.
- p: mostrar o nome do processo (programa) que está usando a conexão.

Veja aqui a explicação, em vídeo, sobre como os programas utilizam a rede

Vídeo 4 - Analise de Captura

Vídeo 5 - Programas

Resumo

Nesta aula, você estudou o protocolo UDP e com isso concluiu o estudo sobre os dois principais protocolos de transporte utilizados na internet, que são o TCP e o UDP. Aprendeu também a comparar os dois protocolos, de modo a saber quais critérios considerar para escolher qual deles utilizar quando for escrever uma aplicação. Nessa comparação, viu que apesar do UDP ser um protocolo bem mais simples que o TCP, algumas vezes ele pode ser a melhor escolha. Finalmente, analisou a forma como os programas interagem com a camada de transporte, por meio de exemplos de programas cliente-servidor usando esses dois protocolos.

Autoavaliação

1. Quais as principais características do protocolo UDP?
2. Quais as diferenças entre o UDP e o TCP?
3. Quais campos são comuns nos cabeçalhos TCP e UDP?
4. Quais aplicações são mais indicadas para usarem o protocolo UDP?

Referências

FOROUZAN, B. **Comunicação de Dados e Redes de Computadores**. 4. ed. São Paulo: MCGRAW-HILL, 2008.

KUROSE, J.; ROSS, K. **Redes de computadores e a internet**. 5. ed. São Paulo:Addison Wesley, 2010.

STEVENS, R. **Programação de Rede Unix**: API para Sockets de Rede. 3. ed. São Paulo: Editora Bookman, 2005.

TANENBAUM, Andrew S. **Redes de computadores**. 4. ed. Rio de Janeiro: Editora Elsevier, 2003.