SERVIÇOS DA CAMADA DE TRANSPORTE

Protocolos de Transporte da Internet

Protocolo UDP

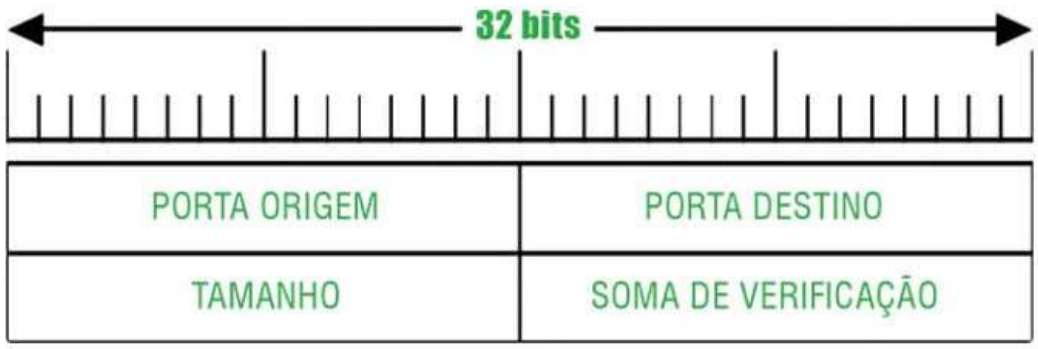
O protocolo de transporte mais simples que poderia existir seria aquele que, no envio, fosse limitado a receber mensagens da camada de aplicação e as entregasse diretamente na de rede. Na recepção, ele, por outro lado, receberia os pacotes da camada de rede e os entregaria na de aplicação. Esse **tipo de protocolo**, em suma, **não efetua** nenhum trabalho para **garantir** a **entrega das mensagens**. Felizmente, o UDP não se limita a isso.

Atenção!

O UDP é um protocolo rápido na entrega das mensagens no destino, realizando a multiplexação e demultiplexação; além disso, ele oferece um mecanismo de verificação de erros nessa entrega.

Um processo pode ter **um ou mais endereços de transporte** (conhecidos como portas na arquitetura TCP/IP) pelos quais dados passam da rede para o processo e vice-versa. Desse modo, a camada de transporte do hospedeiro destino os entrega diretamente a uma porta.

A imagem a seguir ilustra os campos do cabeçalho de um segmento UDP:



Os campos **porta origem e porta destino** têm a função de identificar os processos nas máquinas origem e destino. Quando uma **aplicação A** deseja

enviar dados para uma **B**, o UDP coloca o número da porta da aplicação origem (**A**) no campo “porta origem” e o de porta da aplicação (**B**) em “porta destino”.

Comentário

Quando a mensagem chega ao destino, o UDP pode entregá-la para a aplicação correta por meio do campo “porta destino”. Já a “porta origem” é importante para a aplicação que recebe a mensagem, pois ela torna possível saber o número da porta para a qual a resposta deve ser enviada. É por meio desses campos que o UDP realiza a multiplexação e a demultiplexação.

O campo **tamanho** especifica o tamanho do segmento, incluindo o cabeçalho. Como se trata de um campo de 16 bits, isso significa que o maior segmento UDP será de: $\mathbf{2^{16} = 65.536}$ **bytes (64 KBytes)**.

O campo **soma de verificação** tem a função de garantir que a mensagem chegue ao destino livre de erros. Para tanto, o UDP calcula o CRC dela e o envia nesse campo. No destino, o CRC é novamente calculado e comparado. Se ambos forem iguais, a mensagem é considerada livre de erros e entregue na aplicação destino.

Saiba mais

Para saber mais sobre CRC e hash, consulte, respectivamente, os capítulos 5.2.3 e 8.3.1 da obra de Kurose e Ross (2013).

Caso haja alguma divergência no valor do CRC, o segmento normalmente é descartado, porém algumas implementações permitem – acompanhadas de uma mensagem de aviso – a entrega dele com erro.

O UDP é um protocolo sem estado e não orientado à conexão. Descrito pela **RFC 768**, ele é projetado tanto para pequenas transmissões de dados quanto para aqueles que não requerem um mecanismo de transporte confiável.

Apesar de o UDP não oferecer uma confiabilidade nas transmissões, isso não significa que aplicações que o utilizam não possam ter uma garantia de entrega.

Atenção!

Cabe ao **programador cuidar** dessa garantia no código da própria aplicação.

São protocolos de aplicações que **utilizam o UDP**:

1. DNS
2. SNMP
3. TFTP
4. RPC

Protocolo TCP

Enquanto o UDP é um protocolo de transporte simples, voltado para aplicações que não necessitam de confiabilidade na transmissão, o TCP é um **orientado à conexão**, sendo indicado para aplicações que precisam trocar uma grande quantidade de dados por meio de uma rede com múltiplos roteadores.

O TCP oferece um fluxo de bytes fim a fim confiável, podendo ser utilizado, inclusive, em redes de baixa confiabilidade.

Na transmissão, ele aceita **fluxos de dados** da **aplicação**: dividindo-os em partes de, no máximo, 64 KBytes, ele envia cada uma em um datagrama IP distinto.

Quando os datagramas IP com dados TCP chegam ao hospedeiro destino, eles são, em seguida, enviados à entidade TCP, que restaura o fluxo de dados original.

A camada de rede (protocolo IP) não oferece nenhuma garantia de que os datagramas serão entregues corretamente. Portanto, cabe ao TCP administrar os temporizadores e retransmitir os datagramas sempre que for necessário.

Os datagramas também podem chegar fora de ordem, cabendo a ele reorganizá-los em mensagens na sequência correta. O **TCP** deve **fornecer a confiabilidade** que o **IP não oferece**.

O TCP é definido pelas seguintes RFCs:

1. 793
2. 2018
3. 1122
4. 2581
5. 1323

Para concluirmos nossa última etapa de estudos, precisamos entender **três aspectos** fundamentais da **TCP**:

1. **Modelo de serviço TCP**
2. **Cabeçalho de segmento TCP**
3. **Gerenciamento de conexão TCP**

Modelo de serviço TCP

O serviço TCP é obtido quando tanto o transmissor quanto o receptor criam pontos terminais; denominados **portas**, eles são identificados por um número de 16 bits. É necessário que uma conexão seja explicitamente estabelecida entre um hospedeiro transmissor e um receptor.

Todas as **conexões TCP** são:

Full-duplex

Dados podem ser enviados e recebidos por ela simultaneamente. Uma linha telefônica é um exemplo de sistema full-duplex, pois permite que dois interlocutores falem de forma simultânea. O walkie-talkie, no entanto, é diferente: como tal equipamento está no modo de transmissão ou no de recepção, ele nunca consegue transmitir e receber ao mesmo tempo.

Ponto a ponto

Interliga diretamente dois hospedeiros, não permitindo a participação de um terceiro na conversação. Exemplo: quando ligamos um smartphone a um computador por meio de seu cabo de dados, ocorre uma ligação ponto a ponto, pois somente eles podem utilizar esse meio (cabo de dados) para a troca de informações.

Do ponto de vista da aplicação, uma conexão consiste em dois fluxos independentes de direções opostas. Uma **conexão TCP** é um **fluxo de dados**, e **não de mensagens**. Isso significa que as fronteiras das mensagens não são preservadas de uma extremidade à outra. Quando uma aplicação passa dados para a entidade TCP, ela pode enviá-los imediatamente ou armazená-los em um buffer de acordo com suas necessidades.

As entidades TCP transmissoras e receptoras trocam dados na forma de segmentos. Um segmento consiste em um cabeçalho fixo de 20 bytes mais uma parte opcional, seguido de zero ou mais bytes de dados. O software TCP decide qual deve ser o tamanho dos segmentos, podendo:

1. **Acumular dados de várias escritas em um único segmento.**
2. **Dividir os dados de uma única escrita em vários segmentos.**

Cada segmento **não** pode **ser superior** à **quantidade máxima** de **dados** que um **datagrama** do protocolo IP é capaz de carregar.

O protocolo básico utilizado pelas entidades TCP é o de **janela deslizante**. Quando envia um segmento, o transmissor dispara um temporizador. Assim que ele chega ao destino, a entidade TCP receptora retorna um segmento (com ou sem dados segundo as circunstâncias) com um número de confirmação igual ao próximo número de sequência que ela espera receber. Se o temporizador do transmissor expirar antes de a confirmação ser recebida, o segmento será retransmitido.

Vamos entender melhor o conceito de **janela deslizante** a seguir:

Janela deslizante

Para aumentar a eficiência da transmissão, foram elaborados protocolos que permitem o envio de vários segmentos de dados mesmo sem a confirmação daqueles enviados anteriormente. O número máximo de dados que podem ser enviados sem que tenha chegado uma confirmação define o tamanho da janela de transmissão.

Conforme eles vão sendo confirmados, o ponto inicial da janela de transmissão desloca-se no sentido do fluxo de dados; por isso, ela é conhecida como “janela

deslizante”. Protocolos que utilizam a janela de transmissão para o envio de dados são conhecidos como protocolos de janela deslizante.

Para saber mais sobre janela deslizante, leia o capítulo 3.4.3 da obra de Kurose e Ross (2013).

Caso um segmento chegue ao destino e apresente erro em sua soma de verificação, o TCP simplesmente o descarta. Como não haverá confirmação de recebimento nesse caso, a entidade TCP do transmissor entenderá que o segmento não chegou ao destino e providenciará sua **retransmissão**.

Cabeçalho TCP

Cabeçalho de Segmento no TCP

Cada segmento TCP começa com um cabeçalho de formato fixo – podendo ser seguido por opções de cabeçalho – de 20 bytes. Depois das opções, é possível haver 65.515 bytes de dados.

Válidos, os segmentos sem dados, em geral, são utilizados para confirmações e mensagens de controle. A tabela seguir mostra um exemplo de **segmento TCP**:

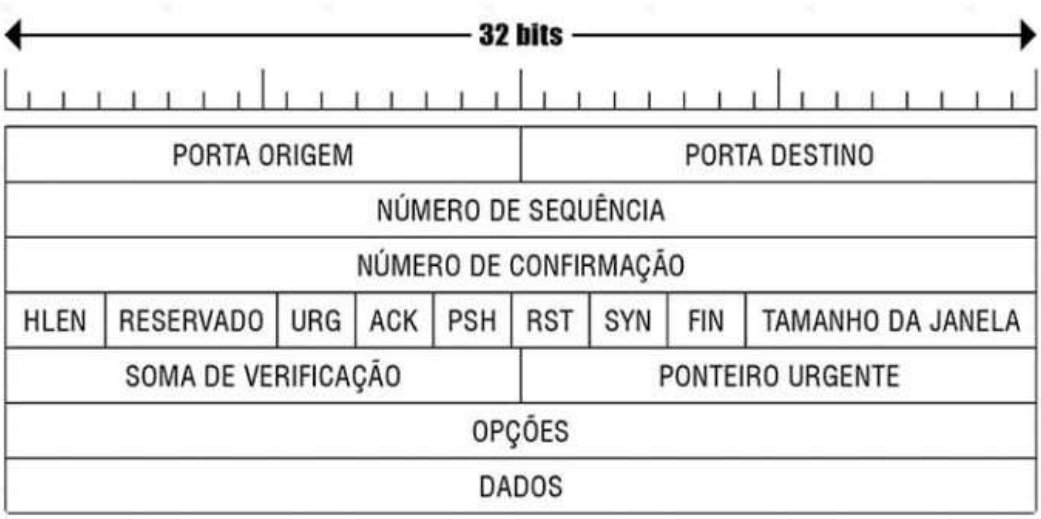


Tabela: 32 Bits.

A seguir, apresentaremos o destrinchamento de cada elemento mostrado na tabela acima.

1. **Porta origem e porta destino**

Identificam, respectivamente, os processos origem e destino nos hospedeiros. Da mesma forma que os campos porta origem e porta destino do UDP, eles permitem que aplicações compartilhem o uso do protocolo de transporte (multiplexação e demultiplexação).

2. **Número de sequência**

   Indica o número de sequência do segmento, ou seja, em que posição (byte) dos dados ele deve ser colocado.

3. **Número de confirmação**

   Especifica o próximo byte aguardado no fluxo contrário. Quando é enviado do processo A para o B, indica o próximo byte que A espera receber no fluxo de B para A. Quando informa que espera receber o byte N, fica implícito que todos os bytes até N-1 foram recebidos corretamente.

4. **HLEN**

   Abreviação de *header length* (tamanho do cabeçalho), o HLEN informa o tamanho do cabeçalho em palavras de 32 bits.

5. **Reservado**

   Campo não utilizado. Qualquer valor preenchido nele será desconsiderado.

6. **Campo de *flags* de 1 bit cada**

   Um flag é um campo de 1 bit que pode possuir apenas os valores 0 e 1. Normalmente, 0 representa “desligado” e 1, “ligado”.

   Exemplo: Quando o *flag* ACK do TCP está com o valor 1 (ligado), isso significa que o segmento TCP carrega um número de confirmação. Se ele estiver com 0 (desligado), é a indicação de que TCP não conta com tal número.

7. **URG**

   O *urgent pointer* (ou ponteiro urgente) é usado para apontar que o segmento carrega dados urgentes.

8. **ACK**

O *acknowledgement* serve para indicar que o campo “número de confirmação” contém uma confirmação. Se estiver com valor 0, este campo deve ser desconsiderado.

9. **PSH**

   O *push* se trata de uma solicitação ao receptor para entregar os dados à aplicação assim que eles chegarem em vez de armazená-los em um buffer.

10. **RST**

    O *reset* é utilizado para reinicializar uma conexão que tenha ficado confusa devido a uma falha. Também serve para rejeitar um segmento inválido ou recusar uma tentativa de conexão.

11. **SYN**

    Aplicado para estabelecer conexões.

12. **FIN**

    Usado para encerrar uma conexão.

13. **Tamanho da janela**

    O controle de fluxo no TCP é gerenciado por meio de uma janela deslizante de tamanho variável. O campo “tamanho da janela” indica quantos bytes podem ser enviados a partir do byte confirmado.

14. **Soma de verificação**

    Utilizada para verificar se existem erros nos dados recebidos. Ela confere a validade tanto do cabeçalho quanto dos dados.

15. **Ponteiro urgente**

    Válido somente se o *flag* URG estiver ativado, ele indica a porção de dados do campo de dados que contém os que são urgentes.

16. **Opções**

    Projetadas como uma forma de oferecer recursos extras, ou seja, aqueles que não foram previstos pelo cabeçalho comum.

17. **Dados**

São os dados enviados pela camada superior. Neste campo, estão os que serão entregues na camada superior do hospedeiro destino.

Conexão TCP

Gerenciamento de conexão TCP

As conexões estabelecidas no TCP utilizam um esquema conhecido como *three way handshake*. Para estabelecer uma conexão, um lado aguarda passivamente, enquanto o outro solicita uma especificando o endereço de rede (endereço IP) e a porta com a qual deseja se conectar.

É enviado então um segmento TCP com o bit SYN ativado e o bit ACK desativado. Quando ele chega ao destino, a entidade TCP do destino verifica se existe um processo aguardando na porta destino. Se não existir, ela enviará uma resposta com o bit RST ativado para rejeitar a conexão.

No entanto, se algum processo estiver na escuta dessa porta, a ele será entregue o segmento TCP recebido. Em seguida, tal processo poderá aceitar ou rejeitar a conexão. Se ele aceitar, um segmento de confirmação será retornado.

Tipos de portas

Portas conhecidas

Para que uma aplicação possa acessar outra remota, é necessário conhecer o endereço do hospedeiro no qual ela se encontra. Ele serve, portanto, para que se consiga chegar ao hospedeiro remoto.

Como o **protocolo de transporte do destino** consegue saber para qual de suas aplicações deve **entregar a mensagem**? A resposta é o conceito de porta, que é responsável por identificar a aplicação no destino.

Como podemos identificar a porta utilizada pela aplicação?

Existem **duas saídas** que respondem a essa questão:

1

Empregar um sistema no qual a aplicação é registrada toda vez que inicializa para o cliente poder consultar sua porta.

2

Usar sempre o mesmo endereço de forma que as aplicações a iniciarem a conversação saibam de antemão com qual endereço trocar mensagens.

Uma **porta TCP** ou **UDP** é identificada por um **número inteiro** de **16 bits**.

Para que um pacote chegue à aplicação de destino, é necessário que o transmissor saiba, de alguma forma, em que porta a aplicação está esperando a chegada do pacote. Para facilitar o trabalho dele, algumas aplicações esperam seus pacotes sempre na mesma porta: a “porta conhecida” da aplicação.

A **RFC 3232** define um repositório on-line no qual podem ser consultadas as portas conhecidas. No momento da criação deste documento, o repositório on-line estava definido como *service name and transport protocol port number registry*.

A tabela a seguir mostra as portas reservadas para algumas aplicações:

PORTA	APLICAÇÃO
7	echo
20	ftp-data
21	ftp
22	ssh
23	telnet
25	smtp
53	domain
69	tftp
80	http
110	pop-3
119	nntp
161	snmp
162	snmp-trap
443	https

Tabela: Portas reservadas para algumas aplicações.
Fabio Contarini Carneiro