Camada de Transporte

Luiz Arthur

A Camada de transporte e responsável por fornecer um **meio de transporte lógicos** aos dados transmitidos pela rede. Esse transporte pode ser **orientado** a conexão e **não orientado a conexão**, isto na prática é tratado pelos protocolos TCP (Transmission Control Protocol) e UDP (User Datagram Protocolo).

O transporte dos dados pode na prática ser tratado na Camada de Enlace, porém o modelo TCP/IP trabalha com a pior hipótese possível, que é a de a Camada de Enlace não controlar o transporte dos dados, então o modelo TCP/IP cuida disto de forma lógica na Camada de Transporte, utilizando principalmente o protocolo TCP que é orientado a conexão.

Além, disso na camada de Inter-redes, um endereço **IP** de destino **identifica** apenas um **host** e nenhuma outra distinção é feita com referencia a qual usuário ou qual programa receberá o datagrama.

Então na **Camada de Transporte** os protocolos UDP e o TCP ampliam a pilha de protocolos TCP/IP, **acrescentando** um mecanismo que permite discernir entre diversas direções em um dado host (isto é chamado de **Porta** de Rede), permitindo que vários programas aplicativos executados em determinado computador enviem e recebam datagramas isoladamente.

Isto é necessário pois a maioria dos **Sistemas Operacionais são multitarefa** podendo executar diversos aplicativos ao mesmo tempo. Tais tarefas são o geralmente o destino final da maioria das tarefas.

Camada de Transporte

Cada Porta de protocolo que identifica um processo de rede é identificado por um número inteiro positivo. O Sistema Operacional fornece um mecanismo de interface que os processos usam para especificar uma porta ou acessá-la.

Para comunicar-se com uma porta exterior **um transmissor precisa conhecer tanto o endereço IP da máquina de destino como o número da porta de destino** de protocolo nessa máquina. Identificando assim o host de destino da mensagem bem como o aplicativo de destino.

Cada mensagem de rede deve transportar o número de porta de destino da máquina para qual a mensagem é enviada e também da **porta de origem** da máquina de origem para a qual as respostas deverão ser endereçadas. Assim, é possível que qualquer processo que receba uma mensagem responda ao transmissor.

As portas também servem para identificar uma dada conexão por exemplo, imagine uma conexão da máquina 10.0.0.1 para um servidor de páginas na máquina 10.0.0.254, tal conexão pode ser representada da seguinte forma: **10.0.0.1:33001 → 10.0.0.254:80**. Sendo que 10.0.0.1 é o host de origem 33001 é a porta de origem (que é aleatória) e 10.0.0.254 é o host de destino é 80 é aporta bem conhecida do servidor de páginas. A resposta do servidor de páginas ocorrerá no sentido inverso, exemplo: **10.0.0.254:80 → 10.0.0.1:33001**, isto permite a identificação da conexão, além de permitir várias conexões na mesma máquina de origem e destino.

UDP – (User Datagram Protocol)

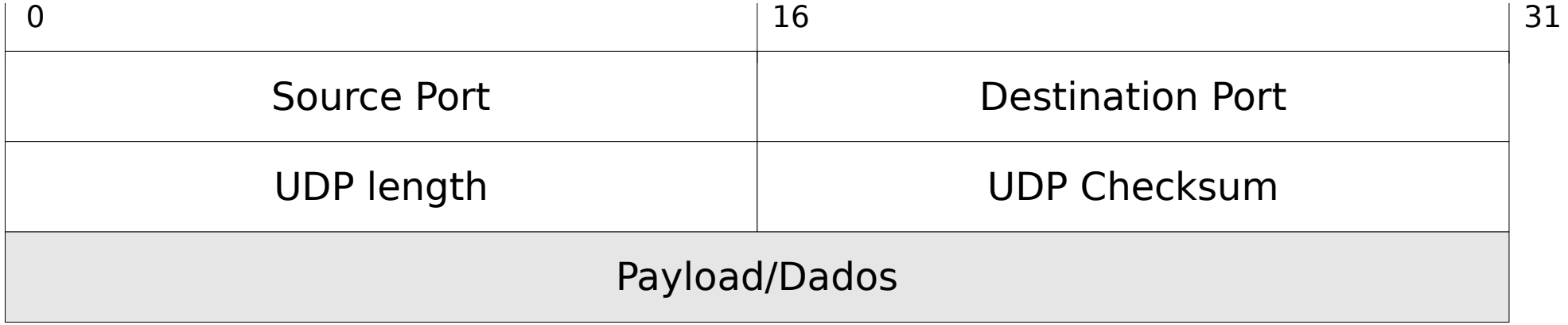
O User Datagram Protocol (UDP) fornece um serviço de entrega que dados sem conexão não confiável, usando IP para transportar mensagens entre as máquinas. O UDP usa o IP para identificar hosts (destino final), mas acrescenta a capacidade de distinguir entre vários destinos (aplicativos) dentro de determinado host, utilizando as Portas.

Então o UDP fornece o mecanismo principal utilizado pelos programas aplicativos para enviar datagramas a outros programas. Fornecendo Portas de protocolo para estabelecer a distinção entre os diversos programas executados em um host.

Cada mensagem UDP leva, os dados a serem enviados, um número de porta de destino e também um número de porta de origem, tornando possível que o software UDP no destino, entregue a mensagem ao destinatário correto e possibilitando, ao destinatário enviar uma resposta.

Um programa aplicativo que usa UDP aceita inteiramente a responsabilidade de lidar com o problema de confiabilidade, inclusive perda de mensagem, duplicação, retardo, transmissão defeituosa e perda de conectividade. Cada mensagem UDP é conhecida como um datagrama de usuário.

Em o UDP consiste em duas partes: cabeçalho UDP; e área de dados UDP.



Cada campo do cabeçalho UDP tem as seguintes funções:

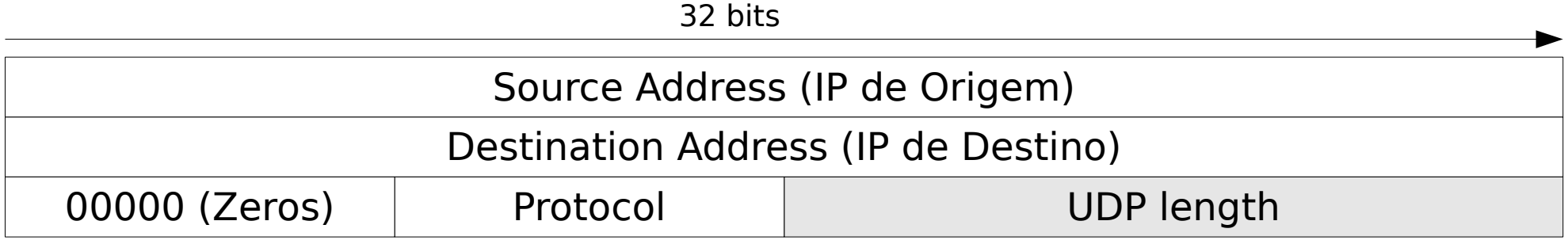
Source Port (Porta de Origem) e **Destination Port** (Porta de Destino): Contêm os números de Porta do protocolo UDP de 16 bits usados para demultiplexar (entregar) os datagramas entre os processos (programas) que esperam para recebê-los. A Porta de origem é opcional e especifica a porta para a qual devem ser enviadas as respostas; se não usada deverá ser zero.

UDP lengh: É o tamanho do datagrama e contém uma contagem de octetos do datagrama UDP, incluindo o cabeçalho UDP e os dados de usuários. Assim o valor mínimo para esse é oito (apenas cabeçalho).

UDP checksum: Este campo recebe a soma de verificação para ver se o Datagrama UDP está correto. Esse campo é opcional (neste caso é preenchida com zeros), isso foi feito para permitir ao UDP operar com o mínimo de overhead possível, e torná-lo mais simples. Mas lembre-se que o IP não faz a verificação de integridade da parte de dados. Portanto, a soma de verificação oferece o único modo de assegurar que os dados tenham chegado intactos e possam ser usados.

A soma de verificação UDP abrange mais informações do que consta do datagrama UDP sozinho. Para calcular a soma de verificação, o UDP inicialmente adiciona um pseudocabeçalho ao datagrama UDP, acrescenta um octeto de zeros para preencher o datagrama e fazer dele um múltiplo exato de 16 bits e calcula a soma de verificação cobrindo todo o objeto.

Porém o octeto usado para o preenchimento e o pseudocabeçalho não são transmitidos com o datagrama UDP, nem são incluídos no comprimento. O uso de um pseudocabeçalho tem como finalidade verificar se o datagrama UDP atingiu o seu destino correto.



O pseudocabeçalho é feito com o endereço IP de origem e destino, o campo protocol (que indica o protocolo usado neste caso o UDP). Note que estes campos são obtidos no datagrama IP e não no datagrama UDP.

O campo UDP length do pseudo cabeçalho é o único campo extraído do datagrama UDP.

A soma de verificação UDP inclui então os endereços IP de origem, IP de destino e protcolo. O que significando que o software UDP deve interagir com o software IP para encontrar endereços antes de enviar datagramas.

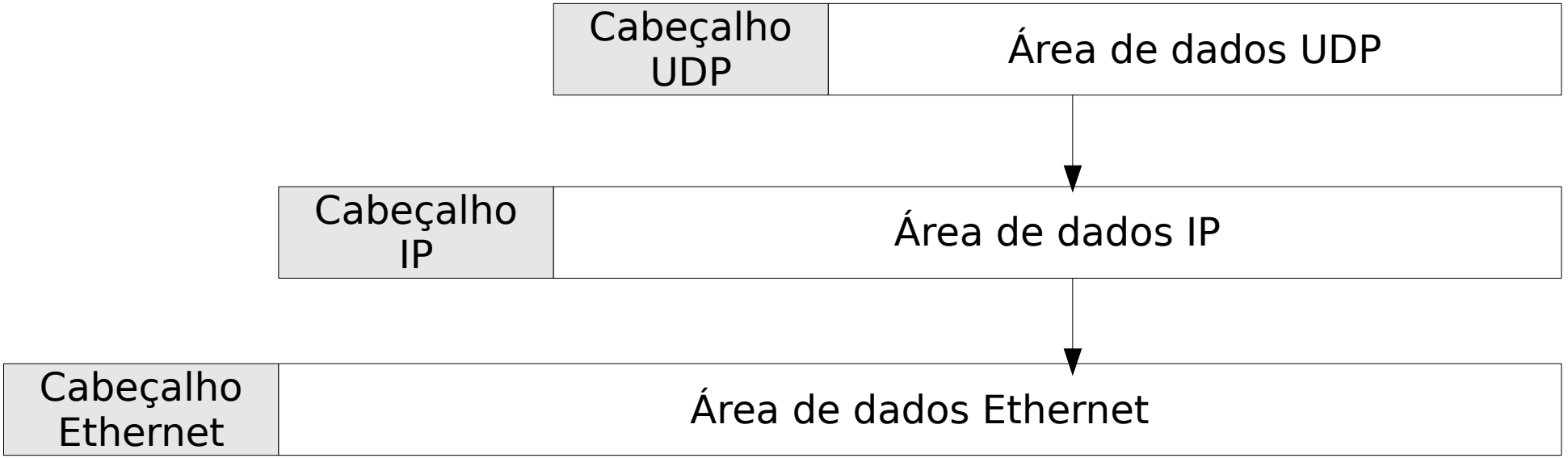
A forte interação ente UDP e IP violam a premissa básica de que a colocação em camadas reflete separação de funcionalidades. Ou seja, cada camada deveria ser independente da outra, porém como é possível ver isto não acontece na prática com o modelo TCP/IP. Já que o UDP da Camada de Transporte está estreitamente integrado com o protocolo IP da Camada de Inter-rede.

O TCP inclui o mesmo pseudo cabeçalho para a soma de verificação.

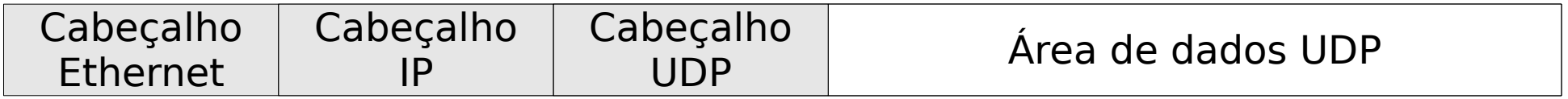
Por fim, depois do cabeçalho vem o playload/Dados, que irá conter as informações uteis do usuário, a ser entregue ao protocolo da Camada de Aplicação.

Encapsulamento do UDP

O Encapsulamento do UDP ocorre da seguinte forma:



Neste caso iniciamos o encapsulamento na Camada de Transporte utilizando o protocolo UDP e o encapsulamos (colocamos dentro) do datagrama IP na Camada de Inter-rede, por fim, usamos um quadro (frame) Ethernet para enviar pela rede a informação na Camada de Enlace. No host de destino é feito o desempacotamento do datagrama UDP. Note que a informação enviada pela rede seria algo como a figura a seguir:



Multiplexação, demultiplexação e portas UDP

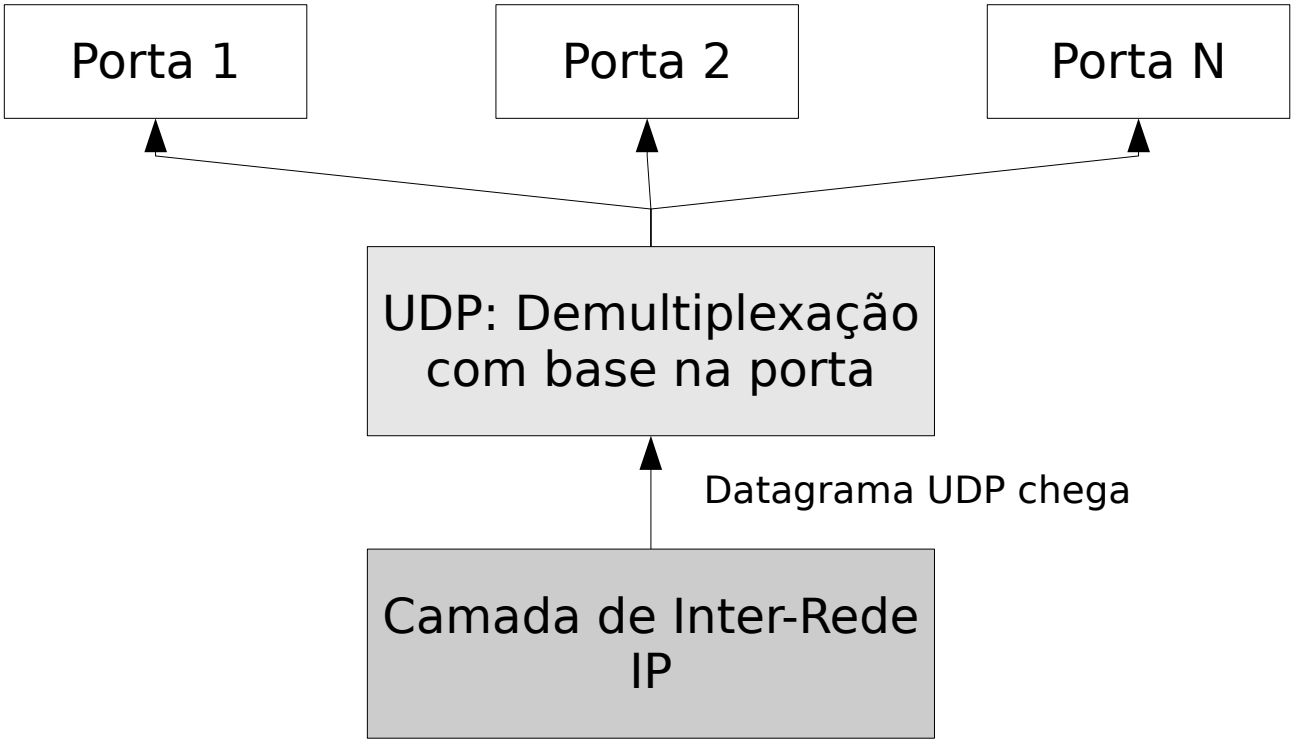
O conceito de multiplexar significa basicamente sair de uma camada e ir para outra, por exemplo, sair da Camada de Aplicação e ir para a Camada de Transporte. Já o conceito de demultiplexação é o processo inverso, ou seja, tirar a informação da Camada de Transporte e ir para a Camada de Aplicação.

Note que este conceito já foi usado na Camada de Enlace, com o campo Tipo do padrão Ethernet, por exemplo para entregar a informação para a Camada de Inter-Rede (para o protocolo IP, por exemplo). E na Camada de Inter-Rede para ir para a Camada de Transporte, usando o campo Protocol do IP para entregar a informação para o UDP ou TCP.

Conceitualmente, toda a multiplexação e demultiplexação entre UDP e os programas aplicativos ocorre por meio do mecanismo de Porta. Assim, na prática cada programa precisa negociar com o sistema operacional para obter uma Porta de protocolo e um número de porta associado antes de poder envia um datagrama UDP. Quando a porta tiver sido atribuída, qualquer datagrama que o programa aplicativo enviar por ela terá esse número de porta em seu campo Source Port.

Ao processar a entrada, o UDP aceita datagramas que chegam do IP e desmultiplexa com base na porta de destino UDP.

Camada de Transporte



Quando o UDP recebe um datagrama, ele verifica se o número da porta de destino combina com uma das portas atualmente em uso. Se não, ele envia uma mensagem de erro ICMP de porta inalcançável (port unreacheable) e descarta o datagrama. Se for achada uma combinação, o UDP enfileira o novo datagrama na porta em que o programa aplicativo pode acessá-lo. Naturalmente pode acontecer desta fila já estar cheia e desta forma pode haver erros na entrega, e o UDP descarta o datagrama que chega.

A multiplexação também se aplica ao protocolo TCP.

Número de porta UDP reservados e disponíveis

Antes de qualquer operação de rede dois computadores precisam combinar a respeito dos números de portas a serem usadas. Existem duas técnicas fundamentais para a atribuição de porta.

A primeira técnica usa uma autoridade central atribua números de porta confirme a necessidade e publique a lista de todas as atribuições. Depois, todo software é criado de acordo com a lista. Essa técnica às vezes é chamada de atribuição universal, e as atribuições de porta especificadas pela autoridade são chamadas atribuições de porta bem conhecidas (well-known port assignments).

A segunda técnica para atribuição de porta utiliza o vínculo dinâmico. Na técnica de vínculo dinâmico, as portas não são conhecidas globalmente. Em vez disso, sempre que um programa precisa de uma porta, o software de rede atribui uma. Para descobrir a atribuição de porta em outro computador, é preciso enviar uma requisição que pergunte a atribuição de porta atual (por exemplo, “que porta o serviço de transferência de arquivos está usando?”). A máquina de destino responde dando o número de porta correto a usar.

Os projetistas do TCP/IP adotaram uma técnica híbrida que atribui alguns números de porta a priori, mas deixa muitos disponíveis para os sites locais ou os programas aplicativos atribuírem dinamicamente. Os números de porta atribuídos começam com valores baixos e se estendem para cima, deixando valores inteiros grandes disponíveis para atribuição dinâmica. O mesmo vale para o TCP.

Vale a pena relembrar explicitamente algumas ações que o UDP não realiza. Ele não realiza controle de fluxo, controle de erros ou retransmissão após a recepção de um segmento incorreto. Ou seja, ele é um protocolo não orientado a conexão assim como IP só fornecendo então uma maneira de entregar os dados na Camada de Aplicação (multiplexação).

O UDP fornece apenas aquilo que é determinado. Uma área na qual o UDP é especialmente útil é a de situações cliente/servidor. Com freqüência, o cliente envia uma pequena solicitação ao servidor e espera uma pequena resposta de volta. Se a solicitação ou a resposta se perder, o cliente simplesmente chegará ao timeout (exceder o tempo) e tentará de novo.

Não só o código é simples, mas é necessário um número menor de mensagens (uma em cada sentido) do que no caso de um protocolo que exige uma configuração inicial, tal como o TCP.

Uma aplicação que utiliza o UDP desse modo é o DNS (Domain Name System), que estudaremos no posteriormente na Camada de Aplicação. Em resumo, um programa que precisa pesquisar o endereço IP de algum nome de host— por exemplo, www.google.com — pode enviar um pacote UDP contendo o nome do host a um servidor DNS. O servidor responde com um pacote UDP que contém o endereço IP. Não é necessária nenhuma configuração antecipada e também nenhum encerramento posterior. Basta enviar duas mensagens pela rede.

O UDP então é um protocolo muito simples de se implementar e de se utilizar. E algumas aplicações iram requerer esta simplicidade, tal como aplicações em tempo real, tal como Voz sobre IP – VoIP, dentre outras. Assim, podemos destacar as vantagens e desvantagens do UDP (de um protocolo não orientado a conexão).

Desvantagens do UDP:

- Não estabelece circuíto entre transmissor e receptor;
- Não confirma a entrega de dados;
- Não ordena os datagramas enviados;
- Todo e qualquer erro deve ser tratado na aplicação.

Vantagens do UDP:

- Sua implementação é mais simples;
- Proporciona menos overhead na rede, principalmente com mensagens de controle;
- Teoricamente, envia as mensagens de forma mais rápida que o TCP;

O UDP e um protocolo não orientado a conexão tipico e deve ser utilizado em aplicações que não requerem segurança na entrega dos dados e sim que requerem velocidade e simplicidade na entrega dos dados. Lembrando que todo e qualquer controle deve ser oferecido pelo programador da aplicação e não pelo protocolo de transporte UDP.

Transmission Control Protocol – TCP

A grande maioria dos protocolos do modelo TCP/IP vistos até aqui (**IP, UDP**, etc) **não são orientados a conexão** não confiável que forma a base para toda a comunicação da Internet e o protocolo IP que a define.

O **TCP** apresentado aqui **implementa** uma funcionalidade substancial aos protocolos já discutidos que é a **orientação a conexão**, mas que sua implementação também é substancialmente mais complexa.

Então em um nível mais baixo (Camadas de Inter-Rede e Enlace), as redes de comunicação fornecem uma entrega de pacotes não confiável.

Em um nível mais alto, os programas aplicativos frequentemente precisam enviar grandes volumes de dados de um computado a outro. A utilização de um sistema de transmissão **sem conexão e não-confiável torna-se tediosa e irritante**, e requer que os programadores criem detecção e recuperação de erros em cada programa aplicativo.

O **TCP** é um protocolo de propósito geral que ajuda a isolar dos detalhes de interligação em redes, de programas aplicativos e **possibilita** definir **uma interface uniforme para o serviço de transferência de streams de pacotes orientados a conexão**.

Camada de Transporte

A interface entre programas aplicativos e o serviço TCP de transmissão confiável pode ser representada por quatro características:

Orientação de stream: Quando dois programas aplicativos transferem grande volume de dados, consideramos os dados como um stream (fluxo) de bits. O serviço de transmissão de streams da máquina de destino passa para o receptor exatamente a mesma seqüência de octetos (bytes) que o transmissor passa para ele na máquina de origem.

Conexão de circuito virtual: Fazer uma transferência de stream de dados equivale a fazer uma chamada telefônica. Em tese, um aplicativo faz uma “chamada” que deve ser aceita pelo outro. A comunicação deve ser autorizada e estar pronta para comunicação dos dois lados (transmissor/receptor). Durante a transferência de dados, os protocolos das duas máquinas continuam a comunicar-se para verificar se os dados são recebidos corretamente. Porem, é utilizado o termo circuito virtual porque embora os programas aplicativos considerem a conexão um circuito de hardware dedicado, a confiabilidade é uma ilusão proporcionada pelo serviço de transmissão de stream de dados.

Transmissão bufferizada: Os programas enviam um stream de dados através do circuito virtual, repetidamente passando octetos de dados ao protocolo. Durante a transferência de dados, cada aplicativo utiliza quaisquer tamanhos que achar adequados, e que podem ser tão pequenos quanto um só octeto. Ao final da recepção, o protocolo entrega os octetos do stream de dados exatamente na mesma ordem em que foram enviados. O protocolo é livre para dividir em pacotes um stream de dados, independente das partes que o programa aplicativo transfere. Para que a transferência seja mais eficaz, e para minimizar o tráfego de rede, as implementações normalmente coletam de um stream os dados suficientes para preencher um datagrama razoavelmente extenso, antes de transmiti-lo por uma interligação em redes. Assim mesmo se o programa aplicativo gerar ao stream um octeto de cada vez, a transferência na interligação em redes poderá ser muito eficaz. Da mesma forma, se o programa aplicativo optar por gerar blocos extremamente extensos, o protocolo talvez prefira dividir cada bloco em pequenas partes para a transmissão.

Conexão full duplex: As conexões fornecidas pelo serviço de stream TCP/IP permitem transferência em ambas as direções (full duplex). Do ponto de vista de um processo aplicativo, uma conexão full duplex consiste em dois streams independentes fluindo em direções opostas.

Fornecendo confiabilidade

O serviço de entrega de fluxo confiável **garante** entregar um fluxo de **dados** enviados de uma máquina para outra **sem duplicação ou perda** de dados.

Porém, é necessário observar que **toda a base do sistema é não confiável**. Então para solucionar este problema utiliza-se uma técnica chamada como ***confirmação positiva com retransmissão*** (Positive **Ack**nowledgement with Retransmition – PAR).

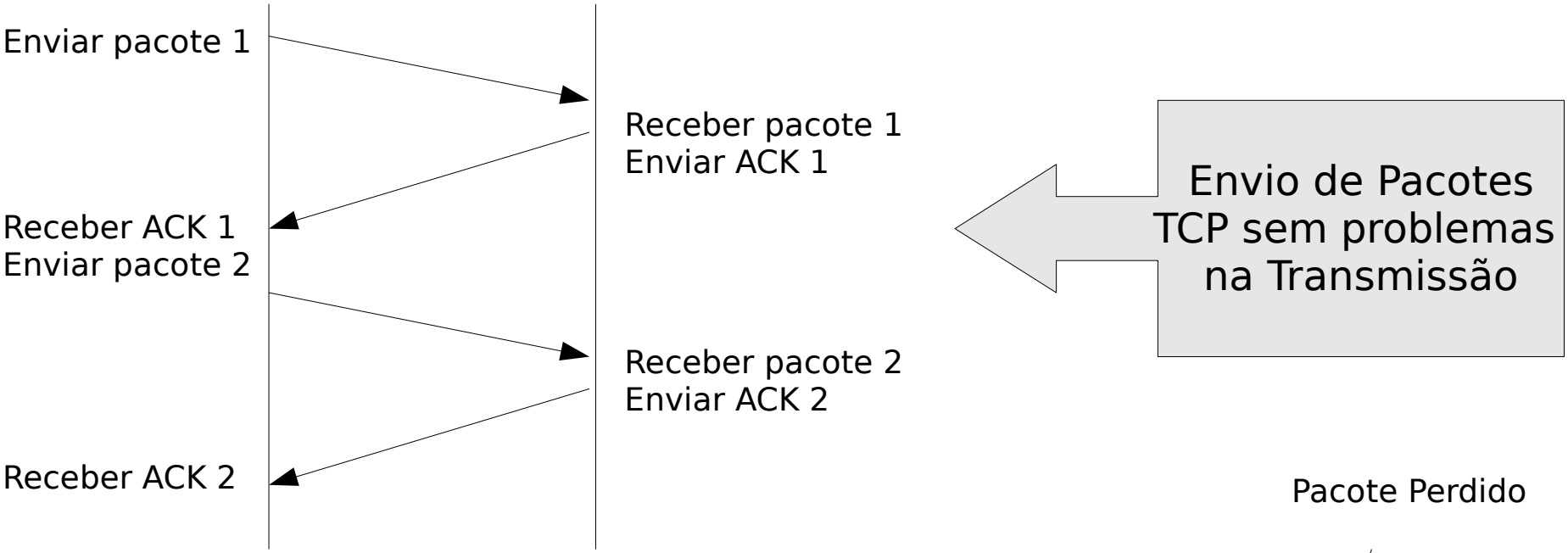
A técnica requer que um destinatário se comunique com a origem, enviando de volta uma mensagem de confirmação (**ACK**) enquanto recebe os dados.

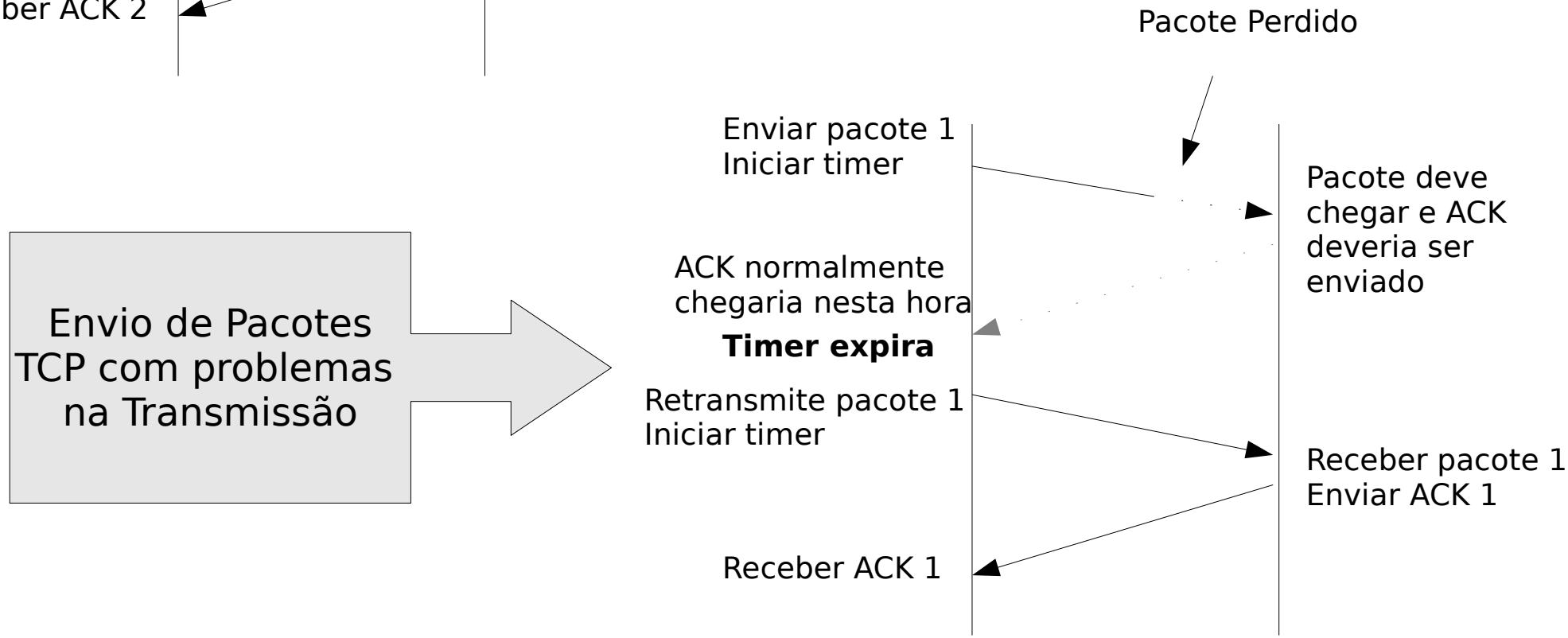
O emissor mantém em registro de cada pacote que envia e espera uma confirmação antes de enviar o próximo pacote.

O emissor mantém um registro de cada pacote que envia e espera uma confirmação antes de enviar a próximo pacote. O emissor também inicia um **timer** quando envia um pacote, e retransmite o pacote se o timer expirar antes que a confirmação chegue.

Então quando o timer expira, o emissor assume que o pacote foi perdido e o retransmite. É possivel notar isto nas figuras a seguir.

Camada de Transporte





Janelas deslizantes

Antes de continuar a os estudos sobre o protocolo TCP, precisamos entender o conceito de Janelas deslizantes, que **torna a transmissão de pacotes mais eficiente**.

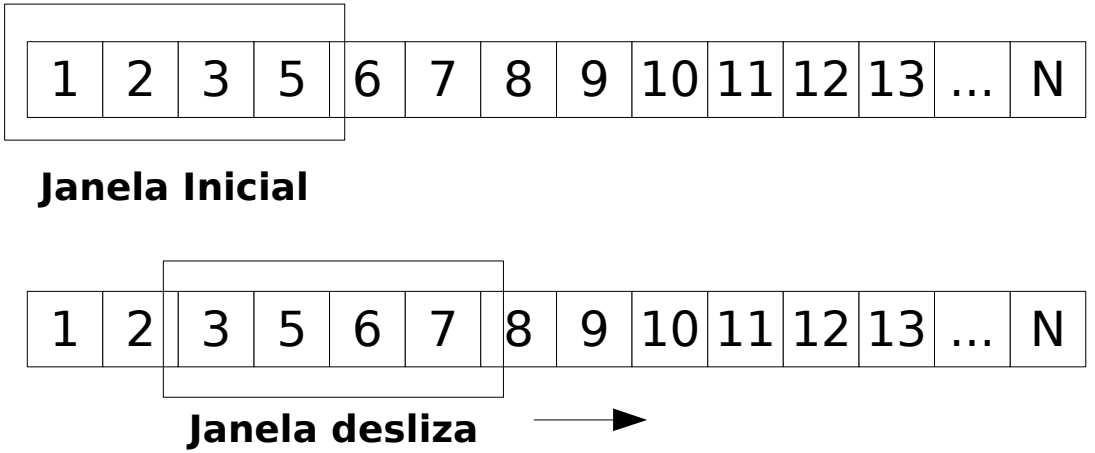
Lembre-se que teoricamente, **para conseguir confiabilidade o emissor transmite um pacote e depois espera por uma confirmação antes de transmitir outro pacote**. Assim, a **rede** permanecerá completamente **ociosa** durante os momentos em que os hosts atrasam as respostas.

Um ***protocolo de confirmação positiva simples*** (tal como o TCP) **desperdiça** uma quantidade substancial de **largura de banda** de rede, pois precisa adiar o envio de um novo pacote até que receba uma confirmação para o pacote anterior.

A **técnica de janela deslizante** é uma forma mais complexa de confirmação positiva e retransmissão do que o método simples discutido anteriormente. Os protocolos de janela deslizante **utilizam melhor a largura de banda** de rede, pois permitem que o emissor transmita vários pacotes antes de esperar uma confirmação.

O protocolo coloca uma janela pequena, de tamanho fixo, na seqüência e transmite todos os pacotes que se encontram dentro da janela.

Camada de Transporte



Um pacote é tido como não-confirmado se tiver sido transmitido e nenhuma confirmação tiver sido recebida. Tecnicamente, o número de pacotes que podem ser confirmados em determinado momento é restringido pelo tamanho da janela, que é limitado a um número pequeno, fixo. Por exemplo, em um protocolo de **janela deslizante com tamanho de janela 5, o emissor tem permissão de transmitir 5 pacotes antes de receber uma confirmação**.

Como mostra a figura, **quando o emissor recebe uma confirmação para o primeiro pacote dentro da janela, ele "desliza" a janela e envia o próximo pacote**. A janela continua a deslizar enquanto as confirmações são recebidas.

Com um protocolo de janela deslizante bem ajustado **mantém**-se **a rede** completamente **saturada** com pacotes, e é possível obter um throughput substanciosamente maior do que um protocolo de confirmação simples.

Serviços oferecidos pelo TCP

O TCP é bem complexo, de modo que não existe uma resposta simples, mas ele específica:

- O formato de dados;
- Confirma que dois computadores trocam dados para conseguir transferência confiável;
- Implementa mecanismos para garantir que os cados cheguem corretamente no destino;
- Permite distinguir entre vários destinos em determinado host;
- Como os hosts devem recuperar pacotes perdidos ou duplicados;
- Como iniciar uma transferência de fluxo TCP e como terminar;

O TCP não específica:

- Não descreve como os programas aplicativos devem utilizar o TCP em termos gerais, a documentação não discute estes detalhes. Isto dá uma flexibilidade maior ao programador, porém o programador geralmente usa a interface oferecida pelo Sistema Operacional;

Portas e conexões

Assim, como UDP, o **TCP** também permite que vários programas aplicativos em determinado host se comuniquem simultaneamente, e **demultiplexa** o tráfego TCP que chega entre os programas aplicativos **com os números de portas** de protocolo.

Porém as portas TCP são mais complexas que as UDP, pois determinado número de porta não corresponde a um único objeto. Em vez disto, o TCP foi **montado sobre a abstração da conexão**, em que os objetos a serem identificados são conexões de circuito virtual, e não portas individuais.

É fundamental entender que o TCP usa a noção de conexões, pois ajuda a explicar o significado e o uso dos números de porta TCP. **Cada conexão são identificadas por um par de extremidades**. Sendo a conexão um par de inteiros (host, porta), isto é feito com as duas extremidades, destino e origem (exemplo: 192.168.0.1:33001 → 192.168.0.254:80).

Como o TCP identifica uma conexão por um par de extremidades, determinado número de porta TCP pode ser compartilhado por várias conexões na mesma máquina. Isto significa que um programador pode criar um programa que **ofereça serviço concorrente a várias conexões simultaneamente**, sem exigir portas locais para cada conexão.

Aberturas passivas e ativas

Diferente do UDP, **o TCP é um protocolo orientado a conexão, que exige que as duas extremidades (host de origem e destino) concordem em participar da transmissão**.

Para isso, o programa aplicativo **em uma extremidade** realiza uma função de **abertura passiva**, contatando seu Sistema Operacional e indicando que ele aceitará uma conexão que chega.

Neste momento, o Sistema Operacional atribui um número de porta TCP para sua extremidade de conexão. O programa aplicativo **na outra extremidade** precisa então contatar seu Sistema Operacional usando uma requisição de **abertura ativa para estabelecer uma conexão**.

Os dois módulos de TCP se comunicam para estabelecer e verificar uma conexão.

Quando uma conexão tiver sido criada, os programas aplicativos podem começão a passar dados; os módulos TCP em cada extremidade trocam mensagens que garantem a entrega confiável. Veremos este assunto melhor posteriormente.

Tamanho de janelas variável e controle de fluxo

Uma **diferença** entre o protocolo de janela deslizante TCP do **protocolo simplificado**, ocorre porque **o TCP permite que o tamanho da janela varie com o tempo**.

Cada confirmação, que especifica quantos octetos foram recebidos, contém um anúncio de janela que específica quantos octetos adicionais de dados o receptor está preparado para aceitar.

Podemos pensar no anúncio de janela como **especificador do tamanho de buffer atual do receptor**. Em resposta a um anuncio de janela maior, o emissor aumenta o tamanho de sua janela deslizante e prossegue para enviar octetos que não foram confirmados. Em resposta a um anúncio de janela diminuída, o emissor diminui o tamanho de sua janela e pára de enviar octetos além do limite.

O TCP não pode contradizer os anúncios anteriores, encurtando a janela além das posições anteriores, e **para** isto **anúncios menores devem receber uma confirmação da outra extremidade**.

A vantagem de usar uma janela de tamanho variável é que ela oferece controle de fluxo e também transferência confiável. Para evitar que os hosts recebam mais dados do que podem armazenar. Ou até mesmo informar um buffer zero para interromper uma transmissão, e outro tamanho para reiniciar a transmissão.

É essencial ter um **mecanismo para controle de fluxo** em um ambiente no qual as máquinas de várias velocidades e tamanhos se comunicam por redes e roteadores de diversas velocidades e capacidades (exemplo, um **computador** e um **celular**).

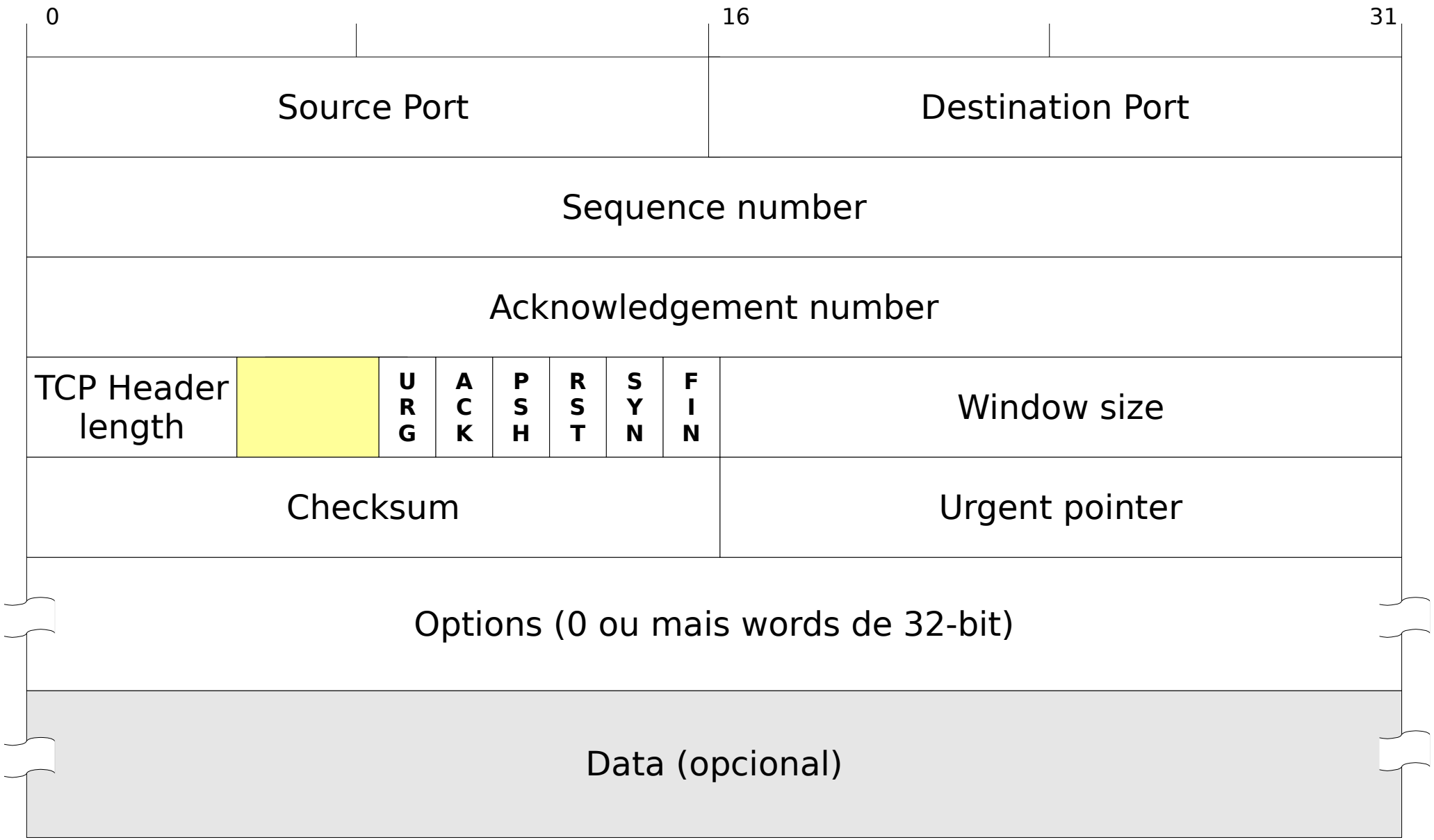
Assim, **existem dois problemas um é solucionar o fluxo de dados fim-a-fim TCP**, que acabamos de ver como o TCP faz, e outro é **controlar o congestionamento** (nos hosts intermediários a conexão, tal como roteadores) este assunto será discutido posteriormente.

Pacotes TCP

A **unidade de transferência** entre o **TCP** nos dois hosts **é chamada de segmento ou pacote**. Os pacotes são trocados para se estabelecer conexões. Sendo ainda que um segmento pode fazer o serviço de **piggybacking** (carona) permitindo que um único segmento envie dados de um host e controle de outro host. Ou seja em uma mensagem podemos enviar os dados do host A para o host B e na mesma mensagem confirmar uma pacote enviado de B para A, o que ameniza o problema de overhead.

Cada segmento ou pacote é dividido em duas partes, um **cabeçalho** e a **área de dados**. O cabeçalho transporta a identificação esperada e informações de controle. No slide a seguir podemos ver o formato do cabeçalho TCP.

Camada de Transporte



Source Port: Porta de origem, indica a aplicação que originou os dados;

Destination Port: Porta de destino, indica a aplicação para a qual os dados serão entregues no receptor;

Sequence number: Número de seqüência que identifica a posição no fluxo de bytes do emissor dos dados no segmento;

Acknowledgement number: Número de confirmação, identifica o número de octeto que a origem espera receber em seguida.

Observe que o número de seqüência refere-se ao fluxo que segue na mesma direção do segmento, enquanto o número de confirmação refere-se ao fluxo que segue na direção oposta do segmento;

TCP Header length: Esse campo especifica o tamanho do cabeçalho do pacote TCP, contando em números de 32 bits.O tamanho mínimo do cabeçalho é de cinco palavras de 32 bits (20 bytes), sendo esse o valor mais comum do campo HLEN, já que normalmente o campo Option não é usado. Quando este campo é usado, o valor do campo HLEN é seis.

Existem depois do campo Header length 6 bits reservados, que não são usados e ficam reservados para uso futuro.

Depois dos bits reservados vem uma seqüência de bits chamados de bits de Controle ou Bits de Código: Esses bits são usados por exemplo, estabelecer conexões TCP, manter e finalizar. O software TCP utiliza tais campos para determinar a finalidade e o conteúdo do segmento. Os seis bits podem ser vistos a seguir e dizem como interpretar outros campos no cabeçalho TCP.

Bit	**Significado (caso esteja ativo - com valor 1)**
URG	O campo Urgent Point é válido.
ACK	O campo Acknowledgement number é válido.
PSH	Força a entrega dos dados (push – empurrar).
RST	Reiniciar a conexão.
SYN	Sincroniza números de seqüência.
FIN	O transmissor chegou ao fim de seus dados.

O valor 1 é atribuído a **URG se Urgent pointer estiver sendo usado**. Urgent pointer é usado para **indicar um deslocamento de bytes no fluxo de dados TCP** a partir do número de seqüência atual em que os dados urgentes devem ser encontrados. Esse recurso substitui as mensagens interrompidas. Esse recuso representa uma forma estruturada de permitir que o transmissor envie um sinal ao receptor sem envolver o serviço TCP no motivo da interrupção.

Ao bit **ACK** é atribuído o valor 1 para **indicar que Acknowledgement number é válido**. Se ACK for igual a zero, isso significa que o segmento não contém uma confirmação e assim o campo Acknowledgement number é ignorado.

O bit **PSH** indica dados com o flag PUSH. Com ele **é solicitado a entregar os dados à aplicação mediante sua chegada**, em vez de armazená-los até que um buffer completo tenha sido recebido (o que ele poderia fazer para manter a eficiência).

O bit **RST é utilizado para reinicializar uma conexão** que tenha ficado confusa devido a uma falha no host ou por qualquer outra razão. Ele também é utilizado para rejeitar um segmento inválido ou para recusar uma tentativa de conexão. Em geral, se receber um segmento com o bit RST ativado, isso significa que a conexão tem um problema.

O bit **SYN é usado para estabelecer conexões**. A solicitação de conexão tem SYN=1 e ACK=0 para indicar que o campo de confirmação de piggyback não está sendo utilizado. A resposta contém uma confirmação e, portanto, tem SYN = 1 e ACK = 1. Basicamente, o bit SYN é usado para denotar pedido de conexão e conexão aceita, enquanto o bit ACK é usado para distinguir entre essas duas possibilidades.

O bit **FIN** é **utilizado para encerrar uma conexão**. Ele indica que o transmissor não tem mais dados a enviar. Entretanto, um processo pode continuar a receber dados indefinidamente, mesmo depois da conexão ter sido encerrada. Tanto o segmento SYN quanto o segmento FIN têm números de seqüência e, portanto, são processados na ordem correta.

Window size: Tamanho da janela, **define o tamanho da janela, em bytes, que será usada na conexão**, ou seja, o TCP anuncia quantos dados ele espera aceitar toda vez que envia um segmento, especificando seu tamanho de buffer neste campo. Este campo normalmente utiliza a técnica de piggybacking, pois acompanha todos os segmentos, especialmente aqueles carregando dados e confirmações.

O controle de fluxo no TCP é administrado por meio de uma janela deslizante de tamanho variável. O campo Window size **indica quantos bytes podem ser enviados a partir do byte confirmado**. Um campo **Window size igual a 0** é válido e **informa** que todos os bytes até Acknowledgement number - 1 inclusive foram recebidos, mas **que o receptor precisa de um descanso no momento** e agradeceria muito se nenhum outro dado fosse enviado.

A permissão para retomar a transmissão de dados pode ser enviada mais tarde com o mesmo Acknowledgement number e com um campo Window size diferente de zero.

Checksum: É calculado de forma similar ao checksum do protocolo UDP, com a criação de um pseudocabeçalho.

Urgent pointer: Embora o TCP seja orientado a fluxo (stream), às vezes é importante que o programa em uma extremidade de uma conexão **envia dados fora da faixa**, sem esperar que o programa na outra extremidade da conexão consuma o buffer.

Por exemplo, se o usuário quiser abortar a conexão antes de ela chegar ao fim, essa informação deve ser processada antes de os dados chegarem ao fim, ou o usuário não terá como abortar o processo. Esta informação, portanto, seguirá em um segmento com o bit URG ativado.

Para acomodar a sinalização fora de faixa, o TCP permite que o emissor especifique dados como urgentes, significando que o programa receptor deve ser notificado sobre sua chegada o mais rápido possível, independente de sua posição no fluxo de dados. O receptor deve entrar no modo urgente e depois que todos os dados urgentes tiverem sido consumidos, o TCP diz ao programa aplicativo para retornar à operação normal.

Detalhes exatos de como o TCP informa ao programa aplicativo sobre dados urgentes dependem do Sistema Operacional.

Options: O campo opções **pode conter zero ou mais opções no cabeçalho**. Cada opção começa com um campo de 1 octeto que específica o tipo da opção, seguida por um campo de 1 octeto que especifica o tamanho do da opção em octetos. Se as opções não ocuparem um múltiplo exato de 32 bits, vai ser feito um enchimento (**padding**) no final do cabeçalho.

Algumas **opções do campo Options** do cabeçalho TCP são:

- Opção de tamanho máximo do segmento

Um emissor pode escolher a quantidade de dados que é colocada em cada segmento. Porém, as **duas extremidades de uma conexão TCP devem combinar sobre um segmento máximo que elas transferirão**.

O TCP usa uma opção de tamanho máximo de segmento (**Maximum Segment Size – MSS**) para permitir que um receptor especifique o segmento de tamanho máximo que ele desejará receber. Assim um host como um PDA pode se comunicar com um Supercomputador, permitindo que máquinas e redes heterogenias se comuniquem.

Então para maximizar o throughput, quando dois hosts se conectam à mesma rede física, o TCP normalmente calcula o **MMS** de modo que o datagrama IP resultantes **combinem com o MTU da rede**.

Se as extremidades não estiverem na mesma rede física, elas poderão tentar **descobrir a MTU mínima ao londo do caminho entre elas**, ou escolher um tamanho de segmento máximo de 536 (que é o tamanho padrão de um datagrama IP, 576, menos o tamanho padrão dos cabeçalhos IP e TCP.

É claro que em redes como a Internet a escolha de um bom MMS pode ser difícil se não impossível. Isto se torna um grande **problema** para a rede, pois a escolha de **pacotes** extremamente **pequenos** podem causar, por exemplo **overhead** na rede, já a escolha de um **MMS** muito **grande** pode obrigar, por exemplo, os roteadores a **fragmentar** o pacote e gerar problemas de performance da rede.

Em teoria, **o tamanho de segmento ideal**, é quando os datagramas IP transportando os segmentos **são os maiores possíveis sem exigir fragmentação** em qualquer lugar ao longo do caminho da origem ao destino. Más na prática isto é muito difícil, pois por exemplo, os roteadores podem mudar os pacotes de rota para redes com MTU menores.

- Opção de janela móvel

Como a janela no cabeçalho TCP tem 16 bits de extensão, **o tamanho máximo da janela é de 64 Kbytes**. E este tamanho pode ser um problema para se obter alto throughput (por exemplos, links via satelite). Então, **para acomodar janelas maiores existe uma opção chamada de escala de janela**, que apresenta alguns campos extras que permite informar o tamanho real da nova janela.

- Opção de estampa de tempo

A opção de estampa de tempo TCP foi inventada para **ajudar o TCP a calcular o atraso na rede subjacente**, e também é usada para lidar com o caso em que os números de seqüência do TCP excedem dois elevado a trinta e dois bits (conhecidos como números Protect Agaist Wrapper Sequence – PAWS).

A opção de estampa inclui um valor de tempo e um valor de estampa de tempo de **resposta de eco**.

Assim, um emissor coloca a hora de seu clock atual no campo estampa de tempo ao enviar um pacote; um receptor copia o campo de estampa de tempo para o campo de resposta de eco antes de retornar uma confirmação para o pacote. **Assim, quando uma confirmação chega, o emissor pode calcular com precisão o tempo gasto total desde que o segmento foi enviado**.

- Retransmissão seletiva

Outra opção é o uso da retransmissão seletiva, **em vez do protocolo go back n**, nesse protocolo um receptor receber um segmento defeituoso seguido de um grande número de segmentos perfeitos, o protocolo TCP normal eventualmente sofrerá um timeout e **retransmitirá todos os segmentos não confirmados**, **incluindo os que chegaram em perfeitas condições** (isto é, o protocolo go back n).

Esta opção (NAKs) para **permitir que o receptor solicite um segmento** (ou segmentos) **específico**(s). Depois de receber esses segmentos, o receptor poderá confirmar todos os dados armazenados no buffer, reduzindo assim o volume de dados a serem retransmitidos.

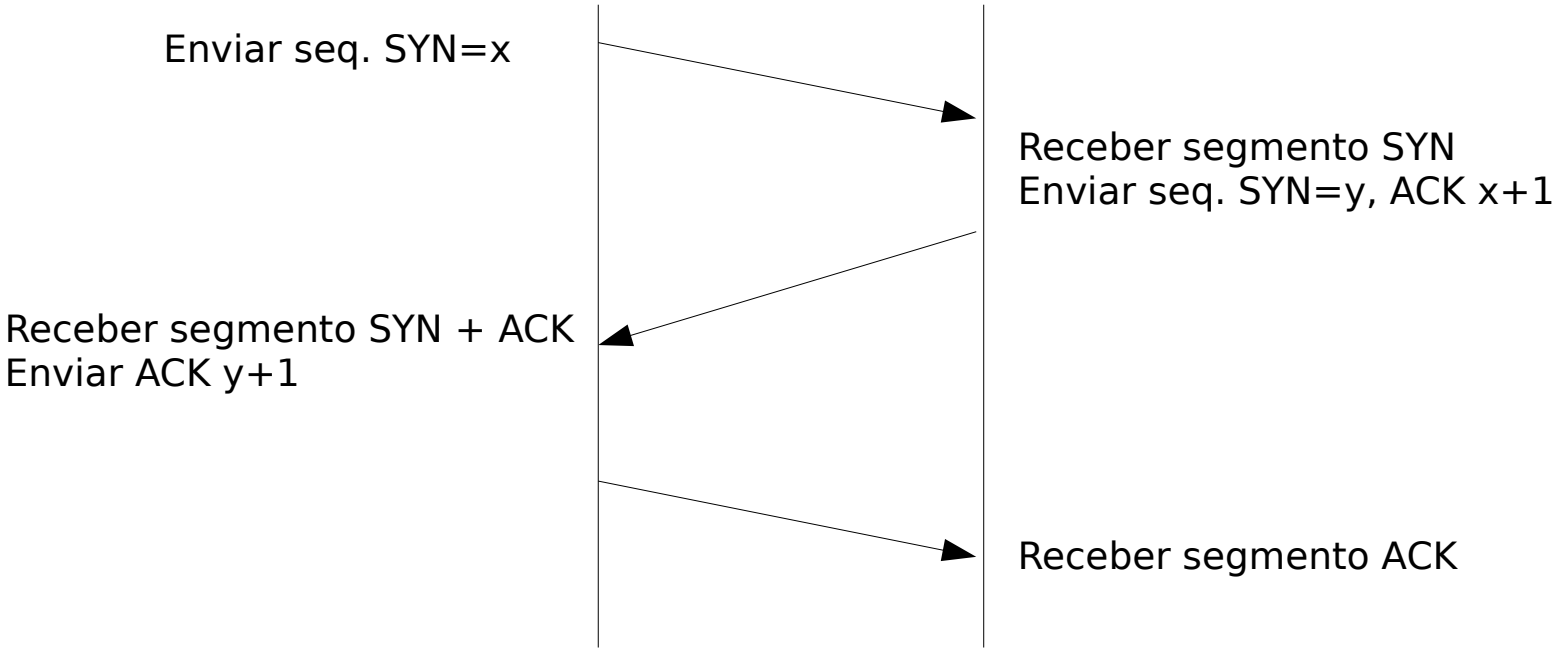
Data: **Á ultima parte do pacote TCP é a área de dados**, que é opcional, pois em muitos casos o TCP pode enviar apenas o cabeçalho, para efetivar uma operação de controle TCP por exemplo. É claro que isto aumenta o overhead na rede (este é o preço que se paga por um protocolo orientado a conexão), mas existem técnicas que tentam amenizar este tipo de problema, usando por exemplo a técnica de piggybacking (carona), que tenta enviar dados de controle junto com pacotes TCP que contenham dados.

Estabelecimento de conexões TCP

Estabelecer uma conexão parece uma tarefa fácil mas, na verdade, trata-se de um procedimento complicado. O problema é que a rede pode perder, armazenar e duplicar pacotes. Esse comportamento causa sérias complicações.

Para estabelecer uma conexão, o TCP usa um handshake de três vias.

No caso simples o processo de handshake funciona como a figura a seguir.



O primeiro segmento de um shandshake pode ser identificado porque possui o bit SYN marcado. A segunda mensagem possui os bits SYN e ACK marcados, indicando que ele confirma o primeiro segmento SYN, além de continuar com o handshake. A mensagem de handshake final é apenas uma confirmação, e é simplesmente usada para informar ao destino de que ambos os lados concordam que uma conexão foi estabelecida.

O problema surge se as requisições retransmitidas e originais chegarem enquanto a conexão estiver sendo estabelecida (lembre-se que o meio não é confiável). Mas o handshake de três vias soluciona estes problemas.

Números de seqüências iniciais

O handshake realiza duas funções importantes. Ele **garante que os dois lados estão prontos para transferir dados e permite que os dois lados concordem sobre números de seqüência iniciais**. Sendo esses números enviados durante o handshake.

Cada máquina precisa escolher um número de **seqüência inicial aleatoriamente**, que usará para identificar os bytes no fluxo que está enviando. Naturalmente, **é importante que os dois lados concordem sobre um número inicial**, de modo que os números de octetos usados nas confirmações combinem com aqueles usados nos segmentos de dados.

Para ver como as máquinas podem combinar sobre números de seqüência para dois fluxos, imagine que: O host A envia seu número de seqüência (x) na primeira mensagem do handshake, e o bit SYN.

Um host B, recebe o SYN, registra o número de seqüência e reponde enviando seu número de seqüência, além da confirmação que especifica que B espera x+1.

Na mensagem final do handshake, A confirma receber B todos os octetos até y. Em todos os casos as confirmações seguem a convenção de usar o número do próximo octeto esperado.

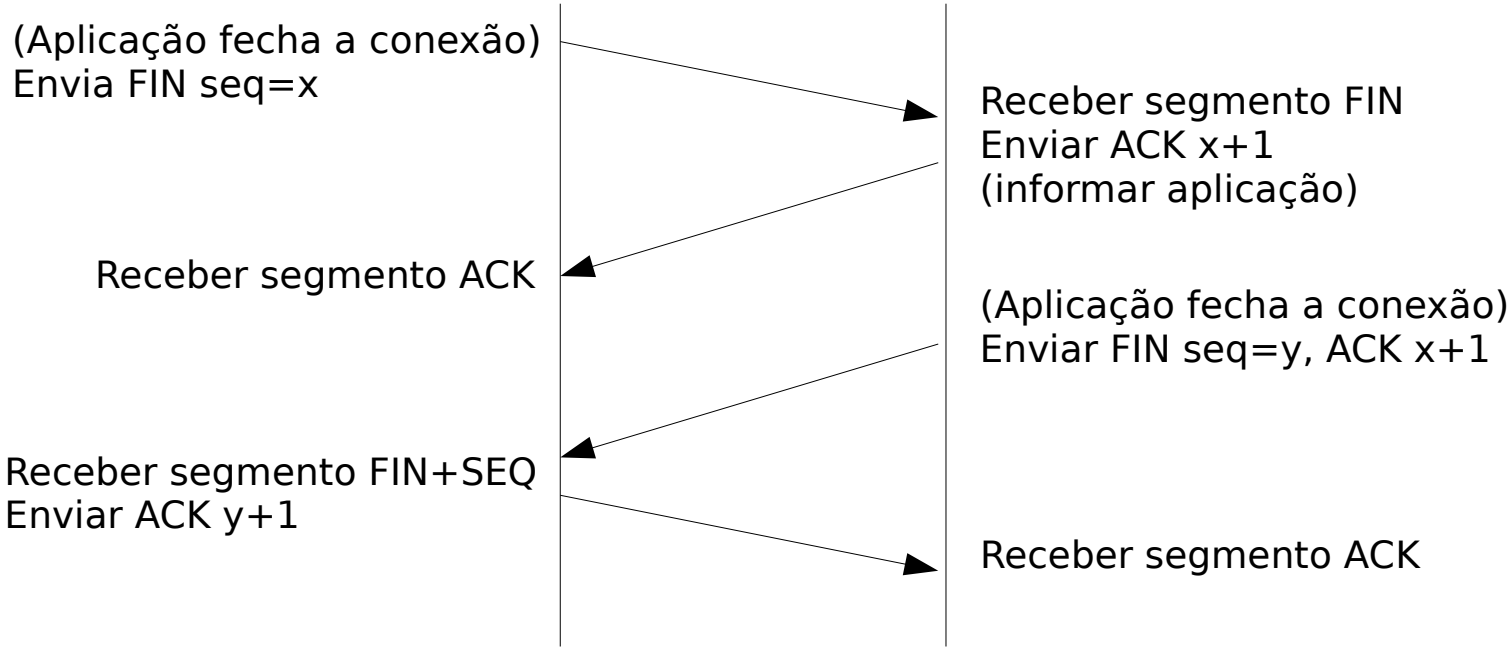
Fechando uma conexão TCP

Dois programas que usam o TCP para se comunicar podem/devem terminar a conversação de modo controlado. Lembre-se de que as conexões do TCP são **full duplex** e que as vemos como contendo duas transferência de fluxos independentes, cada uma em uma direção.

Quando um programa aplicativo TCP que não possui mais dados para enviar, o TCP **fecha a conexão em uma direção**. Para fechar sua metade de uma conexão, o TCP de envio termina e transmitir os dados restantes, espera que o receptor confirme e depois envie um segmento com o bit FIN marcado. O TCP receptor confirma o segmento FIN e informa ao programa aplicativo em sua extremidade que não há mais dados disponíveis.

Quando uma conexão tiver sido fechada em determinada direção, o TCP se recusa a aceitar mais dados para essa direção. Mas neste meio tempo, **os dados podem continuar fluindo na direção oposta** até que o emissor a feche. Naturalmente, as confirmações continuam a fluir de volta ao emissor, mesmo depois que uma conexão tiver sido fechada.

Os detalhes do fechamento de uma conexão é feito usando um handshake de três vias modificado para fechar uma conexão.



A diferença entre um handshake tradicional e o procedimento de finalização é que em vez de gerar um segundo FIN imediatamente, o **TCP enviará uma confirmação e depois informará à aplicação quanto à requisição para encerrar**.

Pode ser necessário muito tempo para informar ao programa aplicativo quanto à requisição e obter uma resposta (isto pode depender, por exemplo da interação humana). A confirmação impede a retransmissão do segmento FIN inicial durante a espera.

Finalmente, quando o programa aplicativo instruir o TCP a encerrar a conexão completamente, o TCP envia o segundo segmento FIN, e o site original responde com a terceira mensagem, um ACK.

Quando os dois sentidos da conexão estiverem desativados, a conexão será encerrada. De modo geral, são necessários quatro segmentos TCP para encerrar uma conexão, isto é, um FIN e um ACK para cada sentido. Porém, é possível que o primeiro ACK e o segundo FIN ocupem o mesmo segmento, o que baixa o número total para três.

É claro que podem ocorrer problemas e um host não terminar corretamente uma conexão (falha no link, por exemplo) assim se uma resposta para um FIN não chegar no período equivalente a duas vezes o tempo máximo de duração de um pacote, o transmissor do FIN encerrará a conexão. Eventualmente, o outro lado perceberá que não há mais ninguém na escuta e também sofrerá um **timeout**. Mesmo que essa solução não seja perfeita, pois a solução perfeita é teoricamente impossível, ela terá de bastar. Na prática, os problemas são raros.

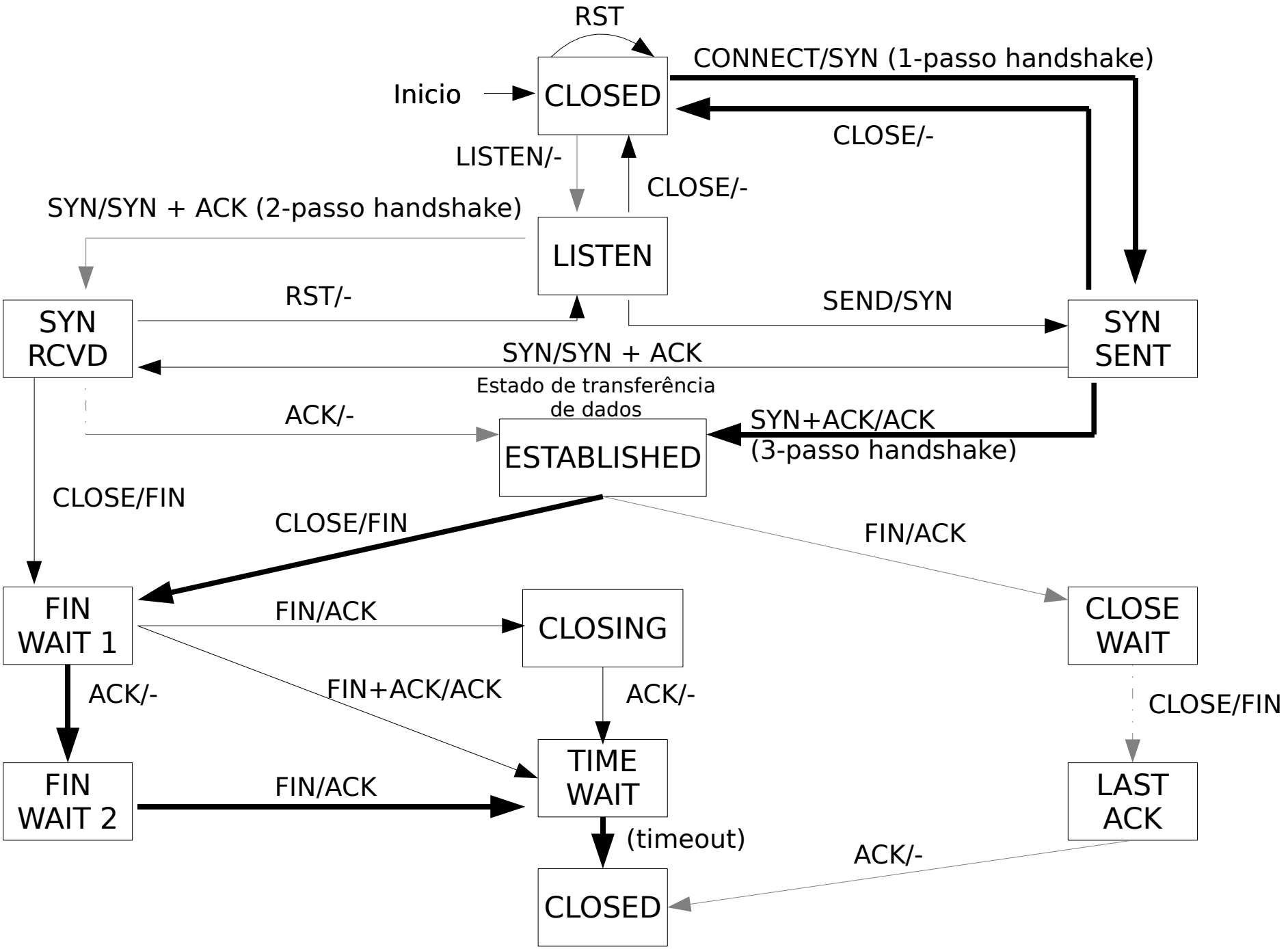
Reinício da conexão TCP

Às vezes surgem condições anormais que forçam um programa aplicativo ou software de rede a interromper uma conexão. O TCP oferece uma facilidade de reset para essas desconexões. Para reiniciar uma conexão, um lado inicia o termino enviando um segmento bom o bit RST e o outro lado responde a um segmento de reset imediatamente. Abortando a conexão. O TCP também informa ao programa aplicativo que houve um reset. **Um reset é um aborto instantâneo**, que significa que a transferência nas duas direções termina imediatamente e que recursos como buffers são liberados.

Máquina de estados TCP

As etapas necessárias para o estabelecimento e o encerramento de conexões podem ser representadas em uma máquina de estados finitos com os 11 estados mostrados. Em cada estado, determinados eventos são válidos. Quando ocorre um evento válido, é possível executar uma ação. Se ocorrer algum outro evento, será reportado um erro. Lembre-se **o TCP possuí estados e o UDP não**.

Estado	Descrição
CLOSED	Nenhuma conexão está ativa ou pendente
LISTEN	O servidor está esperando pela chegada de uma chamada
SYN RCVD	Uma solicitação de conexão chegou; espera por ACK
SYN SENT	A aplicação começou a abrir uma conexão
ESTABLISHED	O estado normal para a transferência de dados
FIN WAIT 1	A aplicação informou que acabou de transmitir
FIN WAIT 2	O outro lado concordou em encerrar
TIMED WAIT	Aguarda a entrega de todos os pacotes
CLOSING	Ambos os lados tentaram encerrar a transmissão simultaneamente
CLOSE WAIT	O outro lado deu início a uma encerramento
LAST ACK	Aguarda a entrega de todos os pacotes


RST
CONNECT/SYN (1-passo handshake)
Inicio
CLOSED
CLOSE/-
LISTEN/-
CLOSE/-
SYN/SYN + ACK (2-passo handshake)
LISTEN
RST/-
SEND/SYN
SYN RCVD
SYN SENT
SYN/SYN + ACK
Estado de transferência de dados
ACK/-
SYN+ACK/ACK
(3-passo handshake)
ESTABLISHED
CLOSE/FIN
CLOSE/FIN
FIN/ACK
FIN WAIT 1
FIN/ACK
CLOSING
CLOSE WAIT
ACK/-
FIN+ACK/ACK
ACK/-
CLOSE/FIN
TIME WAIT
FIN WAIT 2
FIN/ACK
LAST ACK
(timeout)
ACK/-
CLOSED

Cada **conexão começa** no estado CLOSED. Ela sai desse estado ao executar uma abertura passiva (LISTEN) ou ativa (CONNECT). Se o outro lado executar a primitiva oposta, a conexão será estabelecida e o estado passará a ser ESTABLISHED. O encerramento da conexão pode ser iniciado por qualquer um dos lados. Quando ele se completa, o estado volta a ser CLOSED.

Uma situação comum em que um cliente se conecta ativamente a um servidor passivo é representado pelas linhas mais escuras — a **linha contínua para o cliente** e a **linha cinza para o servidor**. As linhas mais claras representam seqüências de eventos incomuns.

Cada linha é marcada por um par evento/ação. O evento pode ser uma chamada de sistema iniciada pelo usuário (CONNECT, LISTEN, SEND ou CLOSE), uma chegada de segmento (SYN, FIN, ACK ou RST) ou, em um único caso, um período de timeout igual a duas vezes a duração máxima dos pacotes. A ação é o envio de um segmento de controle (SYN, FIN ou RST) ou nada, indicado por um travessão (—). Os comentários são mostrados entre parênteses.

Política de transmissão do TCP

Como mencionamos antes, **o gerenciamento de janelas no TCP não está diretamente vinculado às confirmações**, como acontece na maioria dos protocolos de enlace de dados.

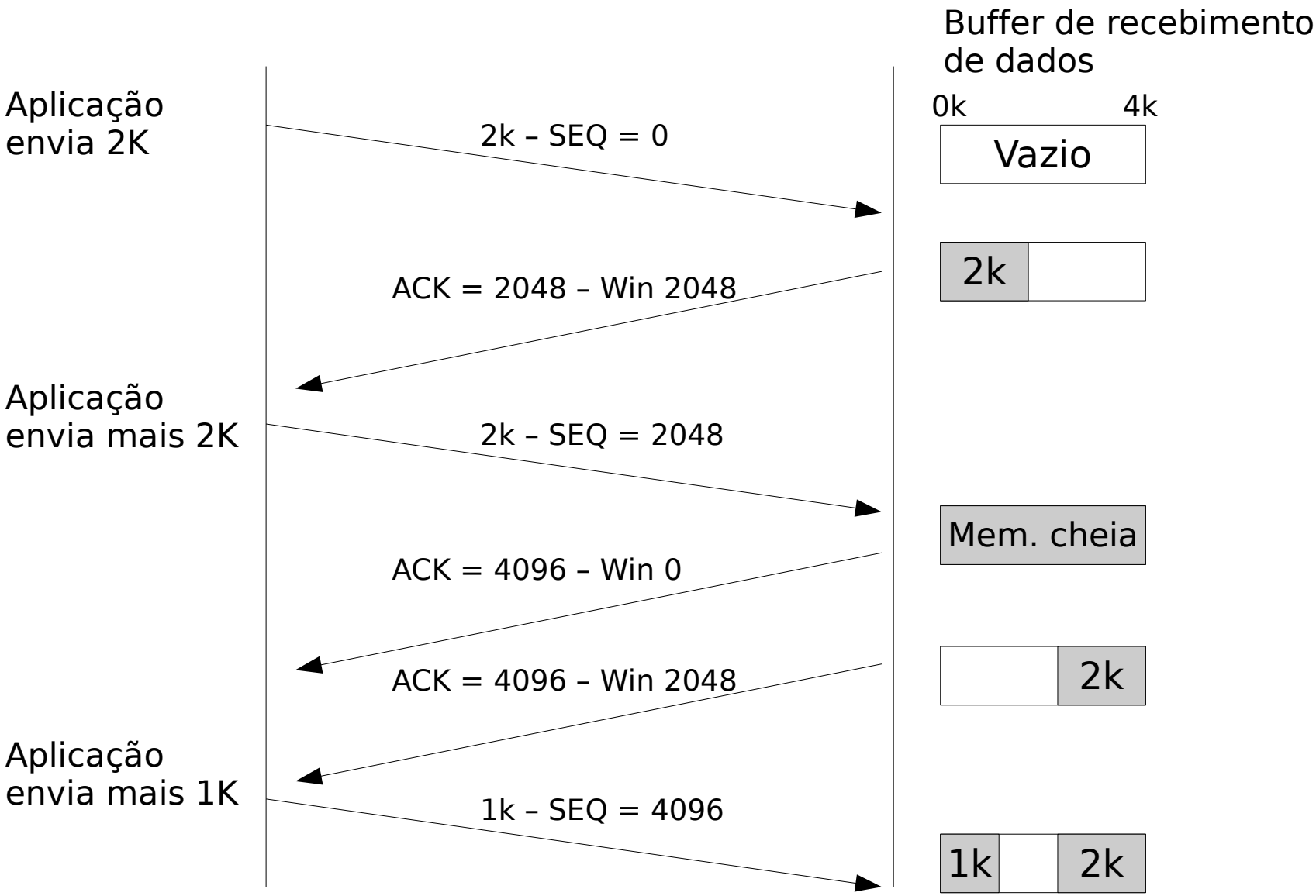
Por exemplo, suponha que o receptor tenha um buffer de 4096 bytes. **Se o transmissor enviar um segmento de 2048 bytes e este for recebido de forma correta, o receptor confirmará o segmento**. Porém, como agora ele só tem 2048 bytes de espaço disponível em seu buffer (até que alguma aplicação retire alguns dados do buffer), **o receptor anunciará uma janela de 2048 bytes começando no próximo byte esperado**.

Agora o transmissor envia outros 2048 bytes, que são confirmados, mas a janela anunciada é 0. O **transmissor deve parar** até o processo de aplicação no host receptor remover alguns dados do buffer, quando então o TCP poderá anunciar uma janela maior. Quando a janela é 0, o transmissor não pode enviar segmentos da forma como faria sob condições normais, mas há **duas exceções**. Na primeira, os **dados urgentes** podem ser enviados para, por exemplo, permitir que o usuário elimine o processo executado na máquina remota. Na segunda, **o transmissor pode enviar um segmento de 1 byte para fazer com que o receptor anuncie novamente o próximo byte esperado e o tamanho da janela**. O padrão TCP oferece essa opção de forma explícita para evitar um impasse no caso de um anúncio de janela se perder.

Os transmissores não são obrigados a enviar os dados assim que os recebem das aplicações. **Nem os receptores têm a obrigação de enviar as confirmações imediatamente.**

Este exemplo pode ser visto na figura a seguir.

Camada de Transporte



O problema da ineficiência na confirmação de segmentos pequenos

Considere uma conexão telnet com um editor interativo que reage a cada tecla pressionada. Na pior das hipóteses, quando um caractere chegar à entidade TCP receptora, o TCP cria um segmento TCP de 21 bytes, que será repassado ao IP para ser enviado como um datagrama IP de 41 bytes. No lado receptor, o TCP envia imediatamente uma confirmação de 40 bytes (20 bytes de cabeçalho TCP e 20 bytes de cabeçalho IP). Mais tarde, quando o editor tiver lido o byte, o TCP enviará uma atualização de janela, movendo a janela um byte para a direita. Esse pacote também tem 40 bytes. Por último, quando o editor tiver processado o caractere, ele o ecoará como um pacote de 41 bytes. No total, 162 bytes de largura de banda são utilizados e quatro segmentos são enviados para cada caractere digitado. **Quando a largura de banda é escassa, esse método não se mostra uma boa opção**.

Uma abordagem usada por muitas implementações do TCP para otimizar essa situação é retardar as confirmações e atualizações de janelas durante 500 ms, na esperança de encontrar algum dado que dê "uma carona (piggybacking)". Supondo que o editor ecoe dentro de 500 ms, apenas um pacote de 41 bytes precisará ser retornado ao usuário remoto, reduzindo à metade a contagem de pacotes e o uso da largura de banda.

Embora essa regra reduza a carga imposta à rede pelo receptor, o transmissor ainda estará operando de modo ineficiente, enviando pacotes de 41 bytes que contêm apenas um byte de dados. Uma forma de reduzir esse uso é conhecida como algoritmo de Nagle (Nagle, 1984).

A sugestão de Nagle é simples: quando os dados chegarem ao transmissor um byte por vez, basta enviar o primeiro byte e armazenar no buffer todos os outros, até que o byte pendente tenha sido confirmado. Em seguida, envie todos os caracteres armazenados no buffer em um único segmento TCP e comece a armazenar no buffer novamente, até todos os caracteres terem sido confirmados.

Se o usuário estiver digitando com rapidez e a rede for lenta, cada segmento poderá conter um número significativo de caracteres, reduzindo bastante a largura de banda utilizada. O algoritmo permite ainda que um novo pacote seja enviado se houver dados suficientes para preencher metade da janela ou um segmento máximo. **O algoritmo de Nagle é amplamente utilizado** por implementações do TCP, **mas há ocasiões em que é melhor desativá-lo**. Em particular, quando uma aplicação X Windows está sendo executada na Internet, os **movimentos do mouse** devem ser enviados ao computador remoto. (O sistema X Windows é o sistema de janelas utilizado na maioria dos sistemas UNIX.) Agrupá-los para depois enviá-los em rajadas faz com que o cursor do mouse se movimente de forma errática, o que deixa os usuários insatisfeitos.

Síndrome da Janela boba

Outro problema que pode arruinar o desempenho do TCP é a síndrome da janela boba. **Esse problema ocorre quando os dados são repassados para a entidade TCP transmissora em grandes blocos, mas uma aplicação interativa no lado receptor lê os dados um byte por vez**.

Inicialmente, o buffer TCP no lado receptor está cheio e o transmissor sabe disso (ou seja, ele tem uma janela de tamanho 0). Em seguida, **uma aplicação interativa lê um caractere do fluxo TCP**. **Essa ação faz com que o TCP receptor fique satisfeito e envie uma atualização de janela ao transmissor, informando que ele pode enviar 1 byte**. O transmissor agradece e envia 1 byte. Agora, o buffer se enche outra vez; portanto, o receptor confirma o segmento de 1 byte e atribui o valor 0 ao tamanho da janela. **Esse comportamento pode durar para sempre**.

A solução apresentada por Clark é evitar que o receptor envie uma atualização de janela para 1 byte. Em vez disso, ele é forçado a aguardar até que haja um espaço considerável na janela para então anunciar o fato. Para ser mais específico, o receptor **não deve enviar uma atualização de janela até que possa lidar com o tamanho máximo de segmento que anunciou quando a conexão foi estabelecida**, ou até que seu buffer esteja com metade de sua capacidade livre (o que for menor).

Além disso, o transmissor também pode ajudar não enviando segmentos muito pequenos. Ele deve tentar aguardar até ter acumulado espaço suficiente na janela para **enviar um segmento inteiro ou pelo menos um segmento que contenha dados equivalentes à metade da capacidade do buffer no lado receptor** (o que ele deve estimar a partir do padrão de atualizações de janelas que já recebeu).

Isso reduz o número de chamadas ao TCP e, conseqüentemente, o overhead. É óbvio que isso também **aumenta o tempo de resposta**; no entanto, para aplicações não interativas como a transferência de arquivos, a eficiência prevalece sobre o tempo de resposta a solicitações individuais.

Controle de congestionamento do TCP

Quando a carga oferecida a qualquer rede é maior que sua capacidade, acontece um congestionamento. A Internet não é exceção a essa regra. Embora a camada de Inter-rede também tente gerenciar o congestionamento, o trabalho mais pesado é feito pelo TCP, pois **a verdadeira solução para o congestionamento é diminuir a taxa de transmissão de dados.**

Na teoria, é possível lidar com o congestionamento aplicando-se um princípio emprestado da física: a **lei da conservação de pacotes. A idéia é não injetar um novo pacote na rede até que um pacote antigo sai da rede** (ou seja, até que o pacote antigo tenha sido entregue). O **TCP tenta alcançar esse objetivo manipulando dinamicamente o tamanho da janela.**

O primeiro passo para gerenciar o congestionamento é detectá-lo. Antigamente, detectar um congestionamento era difícil. Um timeout causado por um pacote perdido podia ter sido provocado por (1) ruído em uma linha de transmissão ou (2) pelo fato de o pacote ter sido descartado em um roteador congestionado. Era difícil saber a diferença entre os dois casos. **Hoje em dia, a perda de pacotes devido a erros de transmissão é relativamente rara**, porque a maioria dos troncos de longa distância é de fibra (embora também existam as redes sem fios). **Como conseqüência, a maioria dos timeouts de transmissão na Internet se deve a congestionamentos.**

Antes de discutirmos como o TCP reage aos congestionamentos, vamos descrever o que ele faz para tentar evitar sua ocorrência. Quando uma conexão é estabelecida, deve-se escolher um tamanho de janela adequado. O receptor pode especificar uma janela a partir do tamanho de seu buffer. Se o transmissor se mantiver dentro do tamanho da janela, **não haverá problemas causados pela sobrecarga dos buffers na extremidade receptora, mas eles ainda poderão ocorrer devido a congestionamentos internos na rede.**

A solução da Internet é entender que existem dois problemas potenciais — a capacidade da rede e a capacidade do receptor — e lidar com cada um deles separadamente. Para isso, **cada transmissor mantém duas janelas**: a janela fornecida pelo receptor e uma segunda janela, a janela de congestionamento. Cada uma reflete o número de bytes que o transmissor pode enviar. O número de bytes que podem ser transmitidos é o valor mínimo entre as duas janelas. Desse modo, a janela efetiva é a janela mínima que o transmissor imagina ser correta, e que o receptor também imagina ser correta. Se o receptor pedir "Envie 8 KB", mas o transmissor souber que qualquer rajada com mais de 4 KB irá congestionar a rede, ele enviará apenas 4 KB. Por outro lado, se o receptor pedir "Envie 8 KB", e o transmissor souber que rajadas de até 32 KB passam pela rede sem problemas, ele enviará os 8 KB solicitados.

Quando uma conexão é estabelecida, o transmissor ajusta a janela de congestionamento ao tamanho do segmento máximo em uso na conexão. Em seguida, ele envia um segmento máximo. Se esse segmento for confirmado antes de ocorrer um timeout, o transmissor incluirá o número de bytes de um segmento na janela de congestionamento, de modo que ela tenha capacidade equivalente a dois segmentos máximos, e em seguida enviará dois segmentos. À medida que cada um desses segmentos for confirmado, a janela de congestionamento será aumentada em um tamanho de segmento máximo.

Quando a janela de congestionamento chegar a n segmentos, se todos os n segmentos forem confirmados a tempo, a janela de congestionamento será aumentada no número de bytes correspondente a n segmentos. Na prática, cada rajada confirmada duplica a janela de congestionamento. **A janela de congestionamento mantém seu crescimento exponencial até que ocorra um timeout ou que a janela do receptor seja alcançada**. Se rajadas de tamanho igual a 1024, 2048 e 4096 bytes funcionarem bem, mas uma rajada de 8192 bytes causar um timeout, a janela de congestionamento deverá ser mantida em 4096 bytes para evitar congestionamentos.

Esse algoritmo é conhecido como inicialização lenta, mas ele não é lento de modo algum. Esse algoritmo é exponencial. Todas as implementações do TCP devem ser compatíveis com ele.

Agora, vamos ver o algoritmo de controle de congestionamento da Internet. Ele utiliza um terceiro parâmetro, o limiar, inicialmente igual a 64 KB, além das janelas do receptor e de congestionamento. **Quando há um timeout, o limiar é definido como a metade da janela de congestionamento atual, e a janela de congestionamento é redefinida como um segmento máximo.**

Em seguida, a inicialização lenta é usada para determinar o que a rede é capaz de gerenciar, exceto pelo fato de o crescimento exponencial ser interrompido quando o limiar é alcançado. A partir daí, as transmissões bem-sucedidas proporcionam um crescimento linear à janela de congestionamento (o aumento é de um segmento máximo para cada rajada) em vez de um para cada segmento.

Na prática, esse algoritmo parte do princípio de que deve ser aceitável reduzir à metade a janela de congestionamento, e depois retomar seu crescimento a partir daí.

Gerenciamento de timers do TCP

O TCP utiliza **vários timers** (pelo menos conceitualmente) para realizar seu trabalho. **O mais importante deles é o timer de retransmissão**. **Quando um segmento é enviado, um timer de retransmissão é ativado. Se o segmento for confirmado antes do timer expirar, ele será interrompido. Por outro lado, se o timer expirar antes da confirmação chegar, o segmento será retransmitido** (e o timer disparado mais uma vez). Com isso, surge a seguinte pergunta: **Qual deve ser o intervalo de timeout?**

Esse problema é mais difícil na camada de transporte da Internet do que nos protocolos da camada de Enlace.

Determinar o tempo de ida e volta para o destino não é uma tarefa fácil. **Se o timeout for curto demais, ocorrerão retransmissões desnecessárias, sobrecarregando a Internet com pacotes inúteis. Por outro lado, se ele for longo demais, o desempenho será prejudicado devido ao longo retardo de retransmissão sempre que um pacote se perder.** Além disso, a média e a variância na distribuição da chegada das confirmações podem mudar com muita rapidez em um intervalo de poucos segundos, devido à ocorrência ou à resolução de um congestionamento.

A solução é utilizar um algoritmo altamente dinâmico que ajuste constantemente os intervalos de timeout, com base na contínua avaliação do desempenho da rede. O algoritmo quase sempre utilizado pelo TCP foi criado por **Jacobson** (1988) e funciona da seguinte forma: para cada conexão, o TCP mantém uma variável, **RTT, que é a melhor estimativa no momento para o tempo de percurso de ida e volta até o destino em questão**. Quando um segmento é enviado, um timer é disparado, não só para verificar quanto tempo a confirmação leva para chegar, como também para acionar uma retransmissão, caso a confirmação demore demais. Se a confirmação voltar antes do timer expirar, o TCP medirá o tempo necessário, que será M. Em seguida, o TCP atualiza a variável RTT de acordo com a fórmula:

$$\mathbf{RTT = \alpha RTT + (1 - \alpha)M}$$

onde é um fator de suavização que determina o peso dado ao antigo valor. Normalmente = 7/8. Jacobson alterou posteriormente esta formulá para melhor desempenho.

Um problema com a estimativa dinâmica de RTT é o que fazer quando um segmento sofre um timeout e é retransmitido. Quando uma confirmação chega, não fica claro se ela se refere à primeira transmissão ou a uma transmissão posterior. **Uma suposição errada pode comprometer seriamente a estimativa de RTT. Phil Karn descobriu isto na prática** (já que ele é um entusiasta do radioamadorismo, um meio notoriamente não confiável). **Ele fez uma proposta simples: não atualizar RTT em qualquer segmento que tenha sido retransmitido. Em vez disso, o intervalo de timeout seria duplicado a cada falha, até os segmentos chegarem ao destino da primeira vez. Essa correção é conhecida como algoritmo de Karn.** A maioria das implementações do TCP utiliza esse algoritmo.

O timer de retransmissão não é o único timer que o TCP utiliza. Há outro timer, chamado **timer de persistência**. Ele foi **projetado para evitar o impasse descrito a seguir. O receptor envia uma confirmação com um tamanho de janela 0, pedindo ao transmissor para esperar. Mais tarde, o receptor atualiza a janela, mas o pacote com a atualização se perde. Agora, tanto o transmissor quanto o receptor estão aguardando que o outro faça alguma coisa. Quando o timer de persistência expirar, o transmissor enviará um teste ao receptor. A resposta ao teste fornece o tamanho da janela.**

Se ela ainda for zero, o timer de persistência será ativado novamente e o ciclo se repetirá. Se for diferente de zero, os dados poderão ser transmitidos. Um terceiro timer utilizado por algumas implementações é o timer keepalive (manter vivo).

Quando uma conexão permanece inativa por muito tempo, o timer keepalive pode expirar para fazer um lado verificar se o outro lado ainda está ativo. **Se esse outro lado não responder, a conexão será encerrada.** Esse recurso é polêmico porque, além de aumentar o overhead, pode encerrar uma boa conexão devido a uma partição transitória na rede.

O último timer usado em cada conexão TCP é o **timer empregado no estado TIMED WAIT durante o encerramento. Ele é executado por um tempo igual a duas vezes a duração máxima dos pacotes para garantir que, quando uma conexão for fechada, todos os pacotes criados por ela serão extintos**.

Resumindo o TCP

O principal protocolo de transporte da Internet é o TCP, que fornece um fluxo de bytes bidirecional confiável. Ele utiliza um cabeçalho de 20 bytes em todos os segmentos. Os segmentos podem ser divididos pelos roteadores da Internet; portanto, os hosts devem estar preparados para reorganizá-los. Sendo função também do TCP otimizar as transmissões de dados, melhorando o desempenho e controlando problemas adversos, como congestionamento.

Este material é retirado dos seguintes livros:

TANENBAUM, Andrew S. **Redes de Computadores**. Editora Campus, 4 Edição. 2003.

COMER, Douglas E. **Interligação de Redes com TCP/IP, volume 1**. Editora Campus, 5 Edição. 2006.

TORRES, Gabriel. **Redes de Computadores: Curso Completo**. Editora Axcel Books. 1 Edição. 2001.

Todos os slides são apenas uma base para a disciplina e não dispensa a leitura dos próprios livros para compreensão do assunto como um todo.

fim