TCP: Transmissão Confiável, Controle de Fluxo e Gerência de Conexão

Diego Passos

1 Escolhendo Valores para o Temporizador

Para cada segmento transmitido — porém, não retransmitido — o TCP obtém uma amostra do RTT do caminho entre transmissor e receptor subtraindo o instante de recepção do *ack* do instante de transmissão do segmento. Este valor, denotado por `SampleRTT` neste material, é então utilizado para uma estimativa do RTT médio utilizando uma Média Móvel Exponencialmente Ponderada denotada pela seguinte equação de recorrência:

$$\text{EstimatedRTT} = (1 - \alpha) \times \text{EstimatedRTT} + \alpha \times \text{SampleRTT},$$

onde `EstimatedRTT` é a estimativa do RTT médio da conexão que é atualizada a cada nova amostra de RTT calculada pelo TCP. O valor típico de α é $0,125$.

O TCP também utiliza estas amostras para estimar uma aproximação do desvio padrão dos RTTs amostrados, denotada a seguir por `DevRTT`. A cada nova amostra obtida, seu valor é subtraído — em valor absoluto — do valor atual de `EstimatedRTT` e esta grandeza é passada por uma Média Móvel Exponencialmente Ponderada segundo a seguinte equação:

$$\text{DevRTT} = (1 - \beta) \times \text{DevRTT} + \beta \times \|\text{SampleRTT} - \text{DevRTT}\|,$$

onde β é tipicamente $0,25$.

Uma vez computados estes dois valores, o TCP define o valor inicial do temporizador de um segmento, ou `TimeoutInterval`, como:

$$\text{TimeoutInterval} = \text{SampleRTT} + 4 \times \text{DevRTT}.$$

A ideia desta última equação é que o valor do temporizador seja o RTT médio somado a uma margem de segurança dada por quatro vezes o desvio médio, evitando assim que pequenas variações do RTT em relação à média causem retransmissões precipitadas. Note que `SampleRTT`, `EstimatedRTT`, `DevRTT` e `TimeoutInterval` são recomputados à medida que novos *acks* são recebidos pelo transmissor. Logo, todos estes valores variam no tempo, idealmente permitindo ao TCP acompanhar — e se adaptar — a mudanças no desempenho da rede.

2 Transmissão Confiável de Dados no TCP

Há três eventos básicos pertinentes ao serviço de transmissão confiável de dados do TCP, a saber:

1. **Recebimento de dados da aplicação**. Se houver números de sequência suficientes na janela do transmissor, dados são encapsulados em um novo segmento com o próximo número de sequência disponível. O segmento recém criado é transmitido pela rede e o temporizador é disparado — se já não estava em execução antes.

2. **Estouro de temporizador**. O segmento na base da janela é retransmitido e o temporizador é reiniciado.

3. **Recepção de um *ack***. Se o *ack* é novo (*i.e.*, reconhece segmentos até então considerados pendentes pelo transmissor), os números de sequência correspondentes são marcados como reconhecidos, a janela é deslocada até o número de sequência informado no *ack* e o temporizador atualmente em execução é interrompido. Caso ainda haja segmentos em trânsito, o temporizador é iniciado novamente.

2.1 Acks Atrasados

Ao receber um segmento de dados, um receptor TCP não é obrigado a gerar um *ack* imediatamente. Ao contrário, ele pode **atrasar a geração do *ack***, deixando este pendente por um período de até 500 ms.

Para que este mecanismo seja permitido, as seguintes restrições precisam ser atendidas **simultaneamente**:

1. **O segmento recebido está em ordem**. O segmento tem exatamente o número de sequência esperado e não foram recebidos segmentos de número de sequência superior anteriormente.
2. **Não há *ack* pendente**. O receptor não está com o *ack* de algum segmento anterior sendo atualmente atrasado.

Acks atrasados podem melhorar a eficiência do TCP, reduzindo o número gerado de segmentos transmitidos para uma mesma quantidade de dados enviados. Isso é possível tanto pela capacidade de suprimir dois *acks* com um único *ack* cumulativo, quanto pela possibilidade de aglomerar *ack* e dados em um único segmento.

2.2 Fast Retransmit

O *Fast Retransmit* é uma otimização comumente utilizada no TCP para acelerar a detecção de perdas de segmento. Ao invés de sempre aguardar pelo estouro do temporizador para inferir perdas e realizar retransmissões, no *Fast Retransmit* o TCP também infere perdas com base na recepção de ***acks* duplicados**.

Depois de receber três *acks* repetidos para um mesmo número de sequência (**totalizando quatro *acks* para o mesmo número: o original e mais três duplicatas**), o *Fast Retransmit* assume que o segmento repetidamente requisitado pelo receptor através dos *acks* foi perdido e dispara uma retransmissão do mesmo, ainda que o temporizador associado a ele não tenha expirado.

3 Controle de Fluxo

Além da transmissão confiável de dados, o TCP fornece um serviço de *controle de fluxo*. O controle de fluxo tem como objetivo impedir que um transmissor TCP sobrecarregue o receptor, transmitindo dados mais rapidamente que a aplicação no receptor consegue lê-los do *socket*.

Isso é importante porque, depois que um segmento de dados é recebido pelo TCP, este envia um *ack* ao transmissor confirmando o recebimento é armazena estes dados temporariamente em um *buffer*, esperando pelo momento em que a aplicação requisitará a leitura do *socket*. Logo, se a aplicação demorar para executar esta leitura, os dados recebidos se acumularão no *buffer* de recepção que pode, eventualmente, ficar completamente cheio. Nesta situação, ao receber mais dados novos pela rede, o receptor teria que descartar alguma coisa, seja o segmento recém recebido — gerando retransmissões desnecessárias no transmissor — ou algum segmento recebido anteriormente — causando corrupção no fluxo de dados, já que estes segmentos já foram reconhecidos e, portanto, nunca mais serão retransmitidos pelo transmissor.

O mecanismo de controle de fluxo funciona de maneira relativamente simples no TCP. Basicamente, de tempos em tempos, o receptor anuncia para o transmissor o espaço disponível no seu *buffer*. Este anúncio é feito através de um campo específico do cabeçalho TCP — geralmente denominado `rwnd` ou *receive window*. Portanto, a cada segmento transmitido pelo receptor — seja um *ack* ou um segmento de dados — o receptor inclui a informação mais atualizada sobre o espaço livre na sua janela. Um receptor pode, inclusive, gerar segmentos vazios com o único objetivo de informar uma mudança na sua janela, caso ele não tenha dados ou *acks* a transmitir.

De posse da informação sobre o espaço atualmente livre no *buffer* do receptor, o transmissor evita transmitir dados em excesso simplesmente limitando o tamanho da sua janela de transmissão ao valor reportado pelo receptor. Como o tamanho da janela determina a quantidade máxima de dados em trânsito, este processo de limitação garante que nunca haverá mais dados trafegando pela rede — e, portanto, potencialmente chegando ao receptor e necessitando de espaço no *buffer* — que o receptor é capaz de armazenar.

É importante destacar que **o controle de fluxo se preocupa com a capacidade do receptor**, não tendo, portanto, relação com a capacidade da rede. Esta diferenciação é relevante para que o controle de fluxo não seja confundido com o controle de congestionamento, tópico de aulas posteriores.

4 TCP: Gerenciamento de Conexão

Ao longo de sua vida, uma conexão TCP — e, por extensão, um *socket* TCP — passa por vários estados diferentes. Os estados de uma conexão permitem que o TCP responda de maneiras diferentes a um evento, a depender do contexto atual. Por exemplo, um segmento requisitando a abertura de uma conexão TCP é respondido com um erro, caso recebido durante uma conexão já aberta.

Dá-se o nome de *gerenciamento da conexão* o mecanismo pelo qual o TCP alterna entre estes vários estados.

4.1 Abertura de uma Conexão

A abertura de uma conexão TCP utiliza um mecanismo chamado de *3-way handshake*. Este nome faz alusão ao fato de que o processo de estabelecimento da conexão demanda o troca de três mensagens:

1. **Segmento SYN**. Gerado pelo cliente (*i.e.*, quem tem a iniciativa da conexão), este segmento **não carrega dados** e possui o bit *syn* ativo no cabeçalho TCP. Este segmento pode ser entendido como uma requisição de estabelecimento de conexão.

2. **Segmento SYNACK**. Gerado pelo servidor (*i.e.*, quem recebe a requisição de estabelecimento de conexão) em resposta a um segmento SYN. Este segmento **também não carrega dados** e é caracterizado por possuir ambos os bits *syn* e *ack* ativos no cabeçalho TCP. Este segmento pode ser entendido como uma confirmação, por parte do servidor, de que a conexão pode ser estabelecida.

3. **Segmento ACk**. Gerado pelo cliente, este é o último segmento envolvido no *3-way handshake*. Ele é caracterizado por possuir o bit *ack* ativo, porém o bit *syn* inativo. Ao contrário dos segmentos anteriores, este segmento **já pode conter dados** do cliente para o servidor.

O bit *syn* só é ativado para estes dois segmentos iniciais do *3-way handshake*. Uma vez estabelecida a conexão, ele é sempre 0 para todos os segmentos. Ao contrário, o bit *ack* é igual a 1 para todos os segmentos após o segmento SYN, até que o processo de fechamento de conexão seja iniciado.

Durante a troca de SYN e SYNACK, cliente e servidor negociam alguns aspectos da conexão. Talvez a informação mais básica estabelecida neste processo sejam os números de sequência. Tanto cliente quanto servidor podem escolher arbitrariamente seus números de sequência iniciais. Como esta informação é necessária para os mecanismos de garantia de confiabilidade, ela informada durante o *3-way handshake*. Em particular, o cliente informa seu número de sequência inicial utilizando-o como número de sequência do segmento SYN. O servidor faz o mesmo utilizando o segmento SYNACK.

4.2 Fechamento de uma Conexão

O fechamento de uma conexão é realizado através do envio de um segmento FIN (*i.e.*, um segmento com o bit *fin* igual a 1 no cabeçalho TCP. Ao receber um segmento deste tipo, o TCP responde com um ACK (*i.e.*, segmento com o bit *ack* ativo). Qualquer um dos lados pode iniciar o processo de fechamento de uma conexão enviando o segmento FIN. Neste caso, diz-se que o lado que iniciou este processo esta executando um **encerramento ativo** da conexão, enquanto o outro lado realiza um **encerramento passivo**. O protocolo também lida corretamente com casos em que ambos os lados geram segmentos FIN simultaneamente — *i.e.*, ambos os lados realizam um encerramento ativo.

Quando um dos lados da conexão envia um segmento FIN ele se compromete a não enviar mais dados. Por outro lado, ao responder um segmento FIN com um ACK, o outro lado ainda está autorizado a enviar dados que porventura estejam em seu *buffer* de transmissão.

Por este motivo, ao iniciar o fechamento da conexão enviando um segmento FIN, o TCP não pode simplesmente abandonar a conexão. Mais que isso, mesmo após receber um ACK relativo a seu segmento FIN, o TCP ainda deve permanecer na conexão até que o outro lado também envie um segmento FIN — mostrando, assim, que não há mais dados a serem enviados de sua parte. O lado que recebe um segmento FIN pode enviar também um FINACK — isto é, um segmento representando, simultaneamente, um reconhecimento do segmento FIN recebido e o segmento FIN enviado pelo nó receptor.

4.3 Estados de uma *Socket* TCP

Dados os procedimentos de abertura e fechamento de uma conexão, um *socket* TCP pode transicionar por uma série de estados. Alguns dos principais estados são:

1. **CLOSED**. Estado inicial do *socket*, anterior à tentativa de estabelecimento da conexão. Também denota o estado final, após o procedimento de encerramento de uma conexão.

2. **LISTEN**. Estado pertinente apenas ao servidor. Denota a situação na qual o servidor aguarda o recebimento de uma nova conexão.

3. **SYN_SENT**. Estado pertinente apenas ao cliente. Denota um momento durante a tentativa de estabelecimento de conexão, logo após o envio de um segmento SYN.

4. **SYN_RCVD**. Estado pertinente apenas ao servidor. Denota o momento durante a tentativa de abertura de conexão, logo após o recebimento de um segmento SYN.

5. **ESTAB**. Estado pertinente a ambos cliente e servidor. Denota a conexão estabelecida. Do lado do cliente, este estado é alcançado a partir do SYN_SENT, uma vez recebido um SYNACK. Do lado do servidor, este estado é alcançado a partir do SYN_RCVD, uma vez recebido o *ack*.

6. **FIN_WAIT_1, FIN_WAIT_2, TIME_WAIT e CLOSING**. Vários estados relacionados ao processo de encerramento **ativo** de uma conexão. O encerramento ativo ocorre quando este é iniciado pelo nó local.

7. **CLOSE_WAIT e LAST_ACK**. Vários estados utilizados durante o processo de encerramento **passivo** de uma conexão TCP. O encerramento passivo ocorre quando este é iniciado pelo nó remoto.