Resumo P2 TR2

Capítulo 3

Camada de transporte: prover a comunicação lógica entre processos rodando em hosts, abstraindo as camadas inferiores. Ou seja, os processos enxergam como se estivessem comunicando diretamente.

Pacote na camada de transporte: segmento

Protocolos da internet: TCP e UDP

UDP: não confiável, não orientado à conexão. TCP: confiável, orentado à conexão.

A extensão do ato de entrega host-to-host para processo-a-processo é chamado de multiplexação e demultiplexação.

O UDP inclui campos de detecção de erro e multiplexação. São os 2 serviços da camada de transporte que o UDP provê. Ele não garante que o dado chegará intacto, nem garante que o dado chegará.

TCP é confiável. Possui controle de fluxo, números de sequência, acknowledgments, contadores. Ele garante que o dado é entregue corretamente e na ordem. O TCP também provê controle de congestionamento.

Um processo pode ter um ou mais sockets, que funcionam como portões para trafegar o dado da rede para o processo, e do processo para a rede. Quando um dado é recebido, o dado não é entregue para o processo, mas sim para um socket intermediário. Como podem existir mais de um socket, cada um possui um identificador único.

O receptor examina os campos do segmento para identificar o socket que deve receber a informação. Então, ele direciona o segmento para o socket. Isso é a demultiplexação.

O emissor encapsula partes de um dado em diversos segmentos, com informações de cabeçalho para cada um deles. Os segmentos são passados para a rede. Isso é a multiplexação.

Um socket TCP é identificado por uma tupla contendo 4 elementos: porta de origem, IP de origem, porta de destino, IP de destino. Socket UDP é identificado por tupla de 2 elementos: IP de destino e porta de destino.

Se 2 segmentos UDP possuírem o mesmo endereço IP de destino e a mesma porta de destino, serão direcionados para o mesmo processo por meio do mesmo socket.

Se 2 segmentos TCP tiverem IPs de origem diferentes mas mesma porta de origem, não é um problema, porque a tupla inteira será diferente, justamente porque o IP de origem é diferente.

Protocolos

O **UDP** realiza as duas funções essenciais da camada de transporte: multiplexação e demultiplexação, com uma leve detecção de erro. Como não há handshake, ou seja, não há um setup da conexão, o UDP é não orientado à conexão.

Vantagens do UDP:

- Controle sobre quais dados são enviados e quando são enviados. Não existe o controle de congestionamento do TCP nem a retransmissão dos dados. Aplicações em tempo real requerem uma taxa mínimo de envio e podem tolerar um pouco de erro.
- Não há estabelecimento da conexão. Não há um delay para estabelecer a conexão, não existe o overhead de estabelecimento da conexão.
- Não mantém o estado da conexão. Ou seja, não tem que guardar informações sobre números de sequência, buffers, parâmetros para controle de congestionamento. Com isso, o UDP ocupa menos espaço, memória e processamento.
- O cabeçalho do UDP é menor (8 bytes) contra 20 bytes do cabeçalho do TCP.

UDP possui checksum. Não é garantido que detecção de erro está implementada na camada de enlace nem na memória, então o UDP precisa implementar detecção de erro. Além disso, funções colocadas em camadas inferiores podem ser redundantes ou de menor valor quando comparadas a essas mesmas funções implementadas em camadas superiores.

O UDP provê detecção de erro, mas não faz nada para corrigir desse erro. Algumas implementações descartam o segmento, outras passam para a camada de aplicação com uma flag de "warning" setada.

Protocolos **RDT**. Comentário meu: "acho que aqui é importante saber as características que influenciaram o TCP, eu não tô nem aí pra essas máquinas de estado."

RDT 2.0: Detecção de erro, feedback do receptor (ack ou nak) retransmissão em caso de erro. Esses protocolos 2.0 são conhecidos como stop-and-wait.

RDT 2.1 e 2.2: número de sequência do pacote e do ack/nak

RDT 3.0: timers para indicar timeout.

RDT com pipeline: transmitir vários pacotes em uma janela, aguardar o ack na ordem para mover a janela. Não precisa esperar o ack de um pacote para transmitir o próximo, é só transferir todos na janela.

Go-Back-N: transmitir os pacotes na janela. Em caso de timeout, retransmitir todos os pacotes anteriormente enviados e que não receberam ack.

Selective Repeat: em caso de timeout, retransmitir apenas os pacotes que não receberam ack, e não todos os não confirmados a partir de um certo ponto.

TCP

É orientado à conexão porque antes que os processos enviem dados, eles devem fazer o handshake, ou seja, devem enviar segmentos para que estabeleçam e concordem com os parâmetros da conexão.

Uma conexão TCP é full-duplex, ou seja, dados podem ser trafegados de A a B e de B a A ao mesmo tempo. Além disso, ela é ponto-a-ponto, ou seja, suporta apenas um emissor e um receptor.

Quando a conexão é estabelecida, dados podem ser enviados. O TCP direciona os dados para o buffer de envio da conexão, que é uma das variáveis setadas na conexão. De tempos em tempos, o TCP irá pegar os dados no buffer de envio e colocar na rede.

O tamanho máximo que pode ser encapsulado em um segmento é o MSS, que depende do MTU da camada de enlace. Os dados + o cabeçalho, que é de 40 bytes aproximadamente, devem caber em um quadro da camada de enlace. O valor típico do MSS é de 1460 bytes, porque o MTU típico é de 1500 bytes.

Segmento TCP: porta de origem e destino, número de sequência, número do ack, variáveis de controle (usadas principalmente no handshake), tamanho da janela do receptor, checksum, options e dados.

Número de sequência: o número de sequência é o número do primeiro byte de um segmento, considerando que todos os segmentos fossem um único stream.

ACK: o número de ack é o número de sequência do próximo byte esperado. O TCP confirma os bytes apenas até o primeiro byte ainda não recebido. Os acks são cumulativos. Se o segmento for recebido fora de ordem, não há uma regra definida, e isso irá depender da implementação do TCP. Mas geralmente o pacote recebido fora de ordem é descartado, ou eles são bufferizados, até que a parte que está faltando seja recebida.

O número de sequência é escolhido aleatoriamente durante o handshake da conexão.

Estimar o RTT

Pegar um novo SampleRTT a cada RTT. Os valores irão flutuar. Estimar um RTT com base em um parâmetro alfa, que é tomado empiricamente como 0.125. O EstimatedRTT é uma média ponderada exponencial móvel do SampleRTT. Ele reflete as condições atuais de congestionamento. Um desvio dos valores também é calculado.

O timeout é, portanto, 4*EstimatedRTT.

Transmissão confiável TCP

Analisar cenários de perda no pacote, no ack, timeout, ack duplicado.

Eventos - Ação no Receptor

Segmento na ordem com número de sequência esperado. Todos os dados até ele foram ack. - Esperar 500 mseg pela chegada de outro segmento na ordem. Se o próximo segmento esperado não chegar, enviar ACK para o último recebido.

Segmento na ordem com número de sequência esperado. Um outro segmento na ordem aguardando o envio do ACK. - Enviar ACK cumulativo, para confirmar o recebimento de ambos os segmentos de uma só vez.

Segmento fora de ordem com um número de sequência maior do que o esperado. Ou seja: gap detectado. - Enviar ack duplicado, indicando o número de sequência do próximo byte esperado.

Segmento que preenche o gap parcialmente ou totalmente. - Enviar ACK, já que o segmento começa na parte inicial do gap.

Fast retransmit: retransmitir o segmento antes que seu timer expire.

Controle de Fluxo

Serve para eliminar a possibilidade de que o emissor preencha completamente o buffer do receptor. Ele tenta "casar" a velocidade de envio com a velocidade que o receptor está lendo.

O emissor possui uma variável "receive window". Cada lado da conexão tem uma receive window diferente.

O último byte recebido- último byte lido do buffer deve ser menor ou igual ao tamanho do buffer.

Essa diferença é igual a quantidade de dados que não possuem confirmação de recebimento, mas foram enviados para a conexão.

rwnd = Buffer - [Último Recebido - Último lido] (ou seja, é o que sobra)

Se a quantidade de dados sem confirmação de recebimento for menor do que o valor de rwnd, não haverá controle de congestionamento.

Ou seja, Último Byte Enviado - Último Byte Confirmado <= rwnd

Se rwnd chegar a zero, o TCP envia segmentos com um byte de dados, para que sejam confirmados pelo receptor e, dessa forma, o emissor sabe que pode enviar mais dados.

Gerenciamento de conexão

3-way handshake

Passo 1 - O cliente envia um segmento TCP especial, com o SYN bit setado para 1. Ele escolhe aleatoriamente um número de sequência.

Passo 2 - O servidor recebe esse segmento e extrai o segmento. Aloca as variáveis e buffers para a conexão, e envia um segmento de "conexão aceita" para o cliente. Nesse segmento, o SYN bit é 1, o ack é o número de sequência recebido+1 e o servidor escolhe seu próprio número de sequência.

Passo 3 - O cliente recebe o segmento e aloca as variáveis e buffers para a conexão. O cliente envia para o servidor um último segmento. Ele serve para confirmar o recebimento do segmento de "conexão aceita". O valor do ack nesse segmento é o número de sequência do servidor+1. O bit SYN é setado para zero, já que a conexão foi estabelecida.

Encerramento de conexão:

O cliente envia para o servidor um segmento com o FIN bit setado para 1. Então, quando o servidor recebe esse segmento, envia um ACK. Então, o servidor envia seu próprio segmento com FIN bit setado para 1. O cliente envia um ACK desse segmento. Então, os recursos em ambos os hosts são desalocados.

Controle de Congestionamento

Causas de congestionamento

Cenário 1: 2 emissores, roteador com buffers infinitos

Cada emissor tem uma taxa máxima de R/2. Se um emissor mandar a essa taxa, a saída é R/2. Se um emissor mandar a uma taxa maior, a saída continua sendo R/2, porque é o máximo que o enlace permite para o emissor. Então, quando um emissor manda a uma taxa maior do que R/2, o atraso entre a origem e o destino tenderão ao infinito, considerando que as conexões vão operar à taxa de R/2 e que os buffers são infinitos.

Cenário 2: 2 Emissores e um roteador com buffers finitos

Congestionamento ocorre porque acontecem muitas retransmissões para compensar pacotes perdidos por causa de overflow de buffer. Uma parte considerável da capacidade do enlace é utilizada para retransmissões.

Cenário 3: 4 emissores, roteadores com buffers finitos e caminhos com vários saltos

Se a "carga" de um um emissor ficar cada vez maior, a do outro fica cada vez menor, e a saída de uma conexão fica em 0. Quando um pacote é descartado, a capacidade de

transmissão que foi usada em cada enlace para encaminhar o pacote até o ponto em que foi descartado foi completamente desperdiçada.
Abordagens para o controle de congestionamento:

Ponto-a-ponto - A camada de rede não provê suporte explícito para indicar congestionamento. É o que acontece no TCP. Nesse caso, o protocolo da camada de transporte deve descobrir e tratar o congestionamento.

Assistência da rede: A camada de rede provê suporte explícito para indicar congestionamento. Pode ser por meio de um bit que ela seta ao enviar o pacote para a camada superior, por exemplo. O ATM ABR utiliza essa abordagem.

Controle de congestionamento TCP

Congestion window impõe um limite na taxa em que um emissor deve enviar tráfego. A quantidade de dados em confirmação de recebimento no emissor não deve exceder o mínimo entre a janela de congestionamento e a janela do receptor.

A cada RTT, o emissor envia cwnd bytes na conexão.

Indicações de congestionamento: timeout e 3 acks duplicados.

Segmento perdido implica em congestionamento. Se acks estão chegando, significa que a rede tem capacidade para entregar alguma coisa.

Segmento com confirmação de recebimento significa que tudo está ok.

O TCP fica testando aumentar a cwnd até que ocorra um evento que indique que a rede está congestionada.

3 fases:

Slow Start, Congestion Avoidance, Fast Recovery

Slow start: o valor da cwnd é inicializado com 1MSS. Ou seja, em 1 RTT, apenas 1 segmento é enviado. Então, o tamanho da cwnd aumenta em 1MSS para cada ack recebido. Ou seja, a cada RTT, o valor da cwnd é dobrado.

Congestion Avoidance: quando o valor de cwnd é igual ou maior do que o valor do ssthresh, que é metade do valor da cwnd na última vez que um congestionamento foi detectado. O valor da cwnd aumenta em 1MSS a cada RTT.

Fast Recovery: o valor de cwnd aumenta em 1MSS para cada ACK duplicado recebido para o segmento perdido, que causou a entrada do TCP nesse estado. Então, o TCP volta ao estado de Congestion Avoidance, aumentando a cwnd em 1MSS a cada RTT.

TCP Tahoe: sempre volta para o slow start. TCP Reno: timeout volta pra 1, 3 acks duplicados reduz a cwnd pela metade+3.

Additive Increase-Multiplicative Decrease: aumenta a cwnd em 1 MSS a cada RTT. Quando detecta congestionamento, reduz pela metade a cwnd.

TCP é justo?

2 conexões compartilhando um único enlace, com taxa de transmissão R. Mesmo MSS e RTT. Ambos os TCPs irão aumentar suas janelas de congestionamento até que a taxa de cada uma fique maior do que R/2. Então, perdas ocorrerão, e a cwnd será reduzida. A cwnd aumentará até que perdas ocorram. Então, ela será reduzida. A taxa de cada conexão flutuará em torno da taxa R/2. Ambas as conexões tenderão a transmitir a uma mesma taxa. A taxa total tenderá a R e a taxa de cada uma tenderá a R/2. As conexões irão eventualmente convergir para esse cenário. Se uma transmitir mais a ponto de ultrapassar a taxa R, a cwnd será cortada e elas encontrarão o equilíbrio.

Na prática, isso não acontece porque as conexões são muito diferentes e as condições não são ideais.

UDP não é justo por não possuir controle de congestionamento, então ele sempre irá procurar transmitir o dado.

Capítulo 2

Arquiteturas de aplicações de rede:

Cliente-Servidor: existe um host que está sempre ligado, que é o servidor. Ele serve e processa requisições de vários outros hosts, chamados de clientes. Um cliente não comunica diretamente com outro, e o servidor possui um endereço IP fixo e conhecido. Um cliente pode sempre contatar um servidor enviando um pacote para esse endereço IP. É sempre o cliente que inicia a comunicação.

Se um único servidor não conseguir lidar com as requisições dos clientes, um data-center pode ser utilizado, ou seja, vários servidores que são vistos como um só, cada um irá processar várias requisições.

P2P: Existe a comunicação entre hosts, chamados de peers. Os peers comunicam entre si, sem a necessidade de um servidor dedicado. Uma característica importante dessa arquitetura é que ela é auto-escalável. Ou seja, um novo peer aumenta a demanda, já que irá requisitar mais coisas, mas também adiciona capacidade de serviço, já que também servirá muitas coisas.

Algumas aplicações podem ter uma arquitetura híbrida.

Desafios do P2P: não colocar carga muito grande nos provedores, segurança e incentivos, já que as pessoas voluntariam sua banda.

Um processo é a abstração de um programa em execução. Os processos se comunicam através da troca de mensagens pela rede. Um processo pode enviar e receber mensagens.

O processo que inicia a comunicação é o cliente. O processo que espera ser contatado para iniciar uma sessão é o servidor.

Um processo envia e recebe mensagens da/para a rede utilizando uma interface chamada de socket. Um socket é a interface entre a camada de aplicação e de rede.

Para que uma mensagem seja enviada, é preciso conhecer o endereço IP do destino e uma informação que indique o processo que deve receber a informação no host. No caso, essa informação é o número da porta. Algumas aplicações possuem portas definidas e fixas. As portas abaixo de 1024 servem para isso.

Serviços que a camada de rede oferece para a camada de aplicação: transferência confiável, throughput (ou seja, a garantia do "fluxo de saída" de dados, timing e segurança)

Transferência confiável: algumas aplicações não precisam de garantia de chegaga, outras precisam.

Throughput: garantia de throughput a uma taxa x de bits por segundo. Algumas aplicações precisam dessa garantia.

Timing: algumas aplicações podem tolerar um atraso maior, outras não.

Segurança: a mesma coisa.

Comentário meu: o livro fala um pouco aqui de TCP vs. UDP, o TCP é confiável e orientado à conexão, o UDP não provê nada, ou seja, é mais "leve", mas ele não tem nenhuma garantia. Nenhum dos 2 protocolos garante throughput e atraso.

Apesar disso, a internet provê um serviço satisfatório para aplicações que precisam de garantia de atraso e às vezes até de throughput.

Protocolos

Um protocolo da camada de aplicação define os tipos de mensagens que devem ser trocadas entre os processos, define a sintaxe dessas mensagens, como por exemplo os campos das mensagens e os tamanhos deles. Define também a semântica dos campos, ou seja, o que cada campo representa. E define também regras para determinar quando e como um processo envia e responde a mensagens.

HTTP

O HTTP é um protocolo utilizado com base na arquitetura cliente-servidor. Os processos cliente e servidor trocam mensagens HTTP. O HTTP usa o TCP como protocolo da camada de transporte.

O HTTP inicia uma conexão TCP. Uma vez que ela é estabelecida, o cliente e o servidor acessam o TCP por meio da interface provida pelo socket. O HTTP envia mensagens através do socket e recebe respostas também através do socket.

Eventualmente, cada mensagem enviada chegará intacta ao servidor, e vice-versa. HTTP é stateless, ou seja, ele não mantém o estado (informação) sobre os clientes. Se um mesmo cliente requisitar uma informação 2 vezes, o servidor não irá saber que acabou de mandar para esse cliente.

Conexões não persistentes: para cada requisição é criada uma conexão TCP.

Conexões persistentes: várias requests são feitas em uma mesma conexão TCP.

O HTTP utiliza conexões persistentes por padrão.

HTTP não persistente:
1 - inicia uma conexão.
2 - envia uma mensagem requisitando o index.
3 - o servidor recebe a mensagem e envia a resposta.
4 - o cliente fecha a conexão TCP (ela só é encerrada se o servidor tiver a certeza de que o cliente recebeu a informação)
5 - O cliente recebe a mensagem e encontra referência para 10 objetos.
6 - Para cada objeto, repetir os passos de 1 a 4.

Uma conexão deve ser estabelecida e mantida para cada objeto, variáveis de conexão e buffers precisam ser alocados e desalocados, tanto no servidor como no cliente. Ou seja, isso adiciona um overhead muito grande.

HTTP persistente:
Depois que o servidor envia uma resposta, ele deixa a conexão aberta, e o cliente não encerra a conexão. Então, vários objetos podem ser enviados através dessa mesma conexão. Pode existir o pipelining para enviar vários objetos de uma vez, sem esperar pela reply de cada um deles.

Formato da mensagem HTTP:
Request \r\n
Header \r\n
\r\n
Corpo da mensagem

GET vs POST: GET é para obter informações, POST é para que o servidor inclua alguma coisa em seu banco de dados.
Códigos de resposta

Cookies

Cookies são uma forma de manter o estado no HTTP, é uma forma de o servidor identificar um cliente.

Um cookie possui 4 componentes: uma linha de cookie na parte do Header na mensagem de resposta do HTTP, uma linha de cookie na parte do Header na mensagem de request do HTTP, um cookie mantido no cliente e gerenciado pelo navegador e um banco de dados no servidor.

Quando um cliente recebe uma resposta, ele recebe um cookie, que tem um identificador. Então, cada mensagem enviada irá enviar também a informação sobre o cookie. Se o servidor detectar em seu banco de dados que possui esse cookie, ele irá identificar o usuário e poderá mostrar informações personalizadas, e poderá rastrear a atividade do usuário.

Um cookie é usado para identificar um usuário. O usuário pode prover informações de identificação, então sempre que um usuário acessar um site com um determinado cookie, ele será identificado pelas informações que ele forneceu.

Web caching:
Também é chamado de servidor proxy. Ele satisfaz requisições ao invés de um servidor original Web. Ele mantém cópias de objetos e páginas. Um navegador pode ser configurado para que todas as requests sejam direcionadas para o web cache.

O cache verifica se possui uma cópia do objeto requisitado. Se não possuir, ele abre uma conexão com o servidor e obtém o objeto. Então o web cache retorna o objeto para o cliente que o requisitou. O cache é um cliente e um servidor ao mesmo. Geralmente, um provedor de internet possui um cache.

O cache reduz substancialmente o tempo de resposta para uma requisição, e ele também reduz a quantidade de tráfego para um servidor.

GET condicional:
Envia a mensagem e requisita o objeto apenas se ele não foi modificado.
Se não foi modificado, a mensagem de resposta vem com o body vazio.

Ou seja, o tamanho das respostas diminui bastante, e isso economiza banda e o tempo de resposta não precisa aumentar muito, já que a resposta não foi modificada e pode ser retornada prontamente. Tem um código (304) para indicar que a página não foi modificada.

FTP
FTP utiliza o TCP como protocolo da camada de transporte, e sua arquitetura é cliente-servidor.

O FTP também é utilizado para transferência de arquivos, mas ele utiliza 2 conexões TCP em paralelo: a de controle e a de dados. A conexão de controle é utilizada para o envio de informações de controle de conexão, tais como dados de identificação, senha, comandos para mudar o diretório, e comandos para obter e enviar arquivos. A conexão de dados é usada apenas para o envio de um arquivo em si.

Esse tipo de comunicação é chamada de out-of-band, porque a conexão de controle é separada da conexão de dados.

Para iniciar uma sessão FTP, primeiro uma conexão TCP é feita, na porta 21. O cliente envia informações de identificação, na conexão de controle. Quando um cliente manda um comando para obter um arquivo, ele é enviado pela conexão de dados. Apenas um arquivo é enviado, e então a conexão é encerrada. A conexão de controle permanece aberta, mas uma conexão de dados é criada para cada arquivo que deve ser transferido. Ou seja, a conexão de dados não é persistente.

O FTP mantém o estado da conexão. Ele associa uma conexão de controle a um usuário, e ele mantém informações sobre o diretório atual do usuário. Isso reduz significativamente o número de conexões que o FTP pode manter simultaneamente.

Email:

SMTP: protocolo para conectar um servidor de email a outro servidor de email.

O SMTP é utilizado na arquitetura cliente-servidor. Quando um servidor de email envia email para outros servidores de email, ele atua como o cliente SMTP. Quando ele recebe email de outros servidores de email, ele atua como um servidor SMTP.

Alice quer enviar um email para Bob. Ela envia a mensagem para seu servidor de email. Ele é colocado em uma fila. O SMTP no servidor de email de Alice vê a mensagem e abre uma conexão TCP com o servidor de email de Bob. Depois de um handshake SMTP, a mensagem é enviada. O servidor de Bob recebe a mensagem, e ele coloca a mensagem na caixa de entrada de Bob.

O SMTP geralmente não coloca a mensagem em um servidor intermediário, ele estabelece uma conexão TCP direta.

Um cliente SMTP pode enviar várias mensagens para um mesmo servidor de emails utilizando uma mesma conexão. Ou seja, as conexões TCP do SMTP são persistentes.

O HTTP é um protocolo pull, ou seja, alguém requisita informações de um servidor quando deseja requisitar. As informações já existem no servidor.

O SMTP é um protocolo push. O servidor que envia o email "dá um push" para o servidor que recebe. O servidor que recebe não requisitou nenhuma informação.

O SMTP requer que as mensagens estejam em ASCII 7-bit, o HTTP não. Além disso, o HTTP encapsula objetos diferentes em mensagens diferentes. O SMTP coloca todos os objetos em uma única mensagem.

Protocolos de acesso à caixa de entrada

POP3
O user agent abre uma conexão TCP com o servidor de emails. O usuário é autenticado (autorização), o usuário obtém mensagens (transação) e a sessão é encerrada e o servidor deleta as mensagens que foram marcadas para serem deletadas (update).

Um usuário envia comandos, e o servidor responde cada comando com um reply. O usuário pode configurar o protocolo para "download and delete" ou "download and keep".

Se um usuário desejar acessar as mensagens de várias máquinas, ele não pode reler a mensagem com o "download and delete", porque a mensagem será deletada do servidor.

O POP3 não armazena informações de estado entre sessões POP3.

IMAP

O POP não provê um mecanismo para que o usuário crie pastas no servidor e possa visualizar essa configuração de qualquer máquina.

Um servidor IMAP mantém o estado entre sessões. Ou seja, ele mantém informações sobre as pastas criadas e sobre quais mensagens estão em quais pastas. O usuário pode obter apenas componentes de uma mensagem.

O IMAP permite que o usuário crie pastas, mova mensagens entre pastas e pesquise pastas e mensagens que atendam a certo critério de busca.

DNS

O DNS é um protocolo para resolver um hostname, que é um nome mnemônico para um servidor, em um endereço IP.

O DNS é uma base de dados hierárquica e é um protocolo que permite que os hosts consultem esse banco de dados. O DNS utiliza o UDP como protocolo da camada de transporte.

Como o DNS adiciona um atraso, porque primeiro um nome precisa ser resolvido para que uma conexão com o servidor ao qual se deseja conectar seja iniciada. Então, as resoluções de nome podem ser cacheadas em um servidor DNS mais próximo, o que reduz o tráfego para servidores DNS e reduz o atraso.

O DNS provê outros serviços além da tradução de hostnames para endereços IP:

1 - alias do host. Um host pode ter vários apelidos.

2 - alias de servidores de email.

3 - distribuição de carga: vários endereços IP podem estar associados a um nome. O DNS retorna um endereço IP diferente para cada consulta, para que as mensagens sejam direcionadas para IPs diferentes.

O DNS é distribuído porque se existisse apenas um servidor DNS, ele seria um único ponto de falha, o volume de tráfego para ele seria enorme, ele poderia ser distante de muitos pontos e a manutenção seria custosa e grande.

Um banco de dados hierárquicos e distribuídos por vários lugares resolve esses problemas.

3 classes de DNS: raiz, top-level domail e DNS autoritativo.

Raiz: existem 13 servidores, mas são replicados em vários lugares do mundo. Eles resolvem os nomes para os servidores top-level.

Top-level domain: responsáveis pelos domínios "top-level", como .com, .org, .edu, .net, .gov, .uk, .jp, .fr.

Autoritativo: responsáveis pelo DNS de uma organização. Uma organização pode implementar os próprios servidores DNS como também pode pagar por eles.

Existe um outro tipo importante de servidor DNS que é o DNS local. Ele é o DNS mais perto de um host, e ele age como um proxy. Ele é quem encaminha a consulta DNS pelo resto da hierarquia. Além disso, ele pode fazer um cache, armazenando as informações obtidas, para que outras consultas não precisem ir até todos os servidores da hierarquia.

Ou seja, se ele resolver o hostname para uma consulta, o DNS local armazena esse resultado e pode enviar o resultado para outros hosts que precisem resolver esse nome. Eles descartam essa informação após um período de tempo, mas assim que uma nova consulta é feita, a informação é armazenada novamente.

2 tipos de consulta: Iterativa e Recursiva

Iterativa

1 - Host envia consulta para servidor DNS local.
2 - Servidor DNS local envia consulta sobre o TLD do hostname requisitado para o servidor raiz.
3 - DNS local obtém a resposta e envia a consulta para o servidor TLD obtido da consulta anterior.
4 - DNS local obtém a resposta sobre o IP do servidor autoritativo da consulta. Então, ele envia para esse servidor a consulta sobre o hostname desejado.
5 - Obtém a resposta do servidor autoritativo e retorna a resposta para o host que efetuou a consulta.

Recursiva
1 - Host envia consulta para servidor DNS local.
2 - DNS local envia consulta para o servidor raiz.
3 - Servidor raiz envia consulta para o TLD correspondente da consulta.
4 - TLD envia consulta para o servidor autoritativo correspondente da consulta.
5 - Servidor autoritativo resolve o IP e retorna para servidor TLD, que retorna para servidor raiz, que retorna para servidor local, que retorna para o host.

Mensagens DNS:

As mensagens DNS são chamadas de RRs (resource records). Elas possuem 4 tipos: A, NS, MX e CNAME. São formadas por uma tupla com 4 itens: nome, valor, tipo e TTL

Se o tipo é A, então o nome é um hostname e o valor é o endereço IP desse hostname
Se o tipo é NS, o nome é um domínio e o valor é o nome do DNS autoritativo que sabe resolver nomes nesse domínio.
Se o tipo é CNAME, o nome é um hostname alias e o value é um hostname canônico para esse alias.
Se o tipo é MX, o valor é um hostname canônico para um servidor de email que possua o campo nome como alias.

Mensagem DNS

Cabeçalho, question, answer, authority, additional.

Para inserir RRs no banco de dados DNS, você deve registrar um nome de domínio em uma entidade que possa fazer esse registro. Ela irá verificar se o domínio é único e vai registrar o domínio no banco de dados DNS.

Aplicações P2P

Em arquiteturas P2P, não se pode confiar em um servidor que está sempre ligado porque ele não existe.

Pares de hosts, chamados de peers, comunicam diretamente um com o outro.

Os peers não pertencem a um provedor de internet, mas são os dispositivos controlados pelos próprios usuários.

Em uma arquitetura cliente-servidor, o tempo de distribuição, ou seja, o tempo para que todos os clientes obtenham um arquivo de um servidor, aumenta linearmente de acordo com o número de clientes que desejam o arquivo.

Se uma arquitetura P2P for utilizada, quando um peer recebe algum dado do arquivo, ele pode utilizar sua capacidade para redistribuir o arquivo para outros peers. Se um peer puder redistribuir um bit assim que ele recebe, o tempo de distribuição reduz consideravelmente.

O tempo de distribuição mínimo no P2P, em um dado momento, praticamente não aumenta, conforme mais peers forem adicionados. Ou seja, uma arquitetura P2P é escalável porque quanto mais usuários forem adicionados, o tempo mínimo de distribuição de um arquivo não irá aumentar, e o tempo para que cada usuário receba um arquivo pode ser cada vez menor.

Bittorrent

Protocolo que utiliza a arquitetura P2P para transferência de arquivos. Um arquivo torrent é um descritor de um arquivo de fato, que indica o peer que possui esse arquivo.

Os peers fazem o download de "chunks" de mesmo tamanho. Quando um peer acumula chunks, ele realiza o upload desses chunks para outro peer.

Um peer pode sair e retornar a qualquer momento, com parte do arquivo ou com todo o arquivo.

Cada torrent possui um tracker. Quando um peer se junta a um torrent, ele se registra a um tracker e periodicamente informa o tracker que está no torrent. O tracker mantém informações sobre os peers que estão participando de um torrent.

Quando um peer se junta a um torrent, o tracker envia informações de alguns peers para esse peer. Então, o peer estabelece conexões TCP com cada um dos peers. Esses peers são chamados de peers vizinhos.

Um peer irá solicitar uma lista dos chunks do arquivo que cada peer possui, e irá requisitar os chunks mais raros, ou seja, os que possuem menos cópias entre os vizinhos.

Um peer dá prioridade aos vizinhos que estão enviando data na maior taxa. Ele determina os 4 peers com a maior taxa, e envia chunks para esses peers. A cada 30 segundos, um peer adicional é selecionado e chunks são enviados a ele. Se um peer se tornar um dos 4 top peers de outro, eles começarão a trocar dados. Esse mecanismo é o tit-for-tat.

DHT

É uma tabela hash que armazena pares chave-valor de peers. Uma chave pode ser, por exemplo, o nome de um conteúdo, e o valor pode ser o endereço IP de um peer que possui esse conteúdo.

Esse banco de dados é distribuído entre os peers, ou seja, um peer não possui todas as informações sobre os outros peers, mas ele possui informações sobre alguns peers.

Implementação de DHT:

Cada peer possui um identificador. A chave de um arquivo via ser um inteiro X. O peer que tiver o identificador que é o sucessor mais próximo dessa key vai possuir a entrada em sua tabela.

Se a chave é maior do que o maior identificador, armazenar no peer com o menor identificador.

DHT Circular:

Cada peer possui informação apenas de seu sucessor e de seu antecessor. É uma "overlay network", ou seja, abstrai toda a rede de verdade e cria uma rede lógica abstrata.

Quando um peer quer saber quem deve armazenar uma determinada chave, ele pergunta para seu sucessor até que a mensagem chegue no peer adequado.

É uma solução elegante para que os peers não precisem ter informações sobre todos os peers na rede.

No pior caso, todos os peers precisarão ser consultados. Em média, N/2 mensagens são enviadas.

Atalhos podem ser adicionados, ou seja, um peer conecta não apenas a seu sucessor e antecessor, mas a alguns outros peers no círculo. Isso tende a reduzir o número de mensagens trocadas. A DHT pode ser projetada para que o número de mensagens trocadas seja O(logN).

Peer churn

Peers podem sair e entrar na rede sem aviso.

Um peer deve enviar mensagens para seus vizinhos para checar se estão vivos.

Se o sucessor de um peer sair, esse peer deve atualizar seu sucessor com o segundo sucessor. Então, ele pergunta para o seu novo primeiro sucessor qual é o sucessor, para que atualize o seu segundo sucessor.

Quando um peer entra na rede, ele pergunta para o primeiro peer da rede quem seria o seu sucessor e antecessor.

A mensagem chega até esses peers, e eles enviam essa informação para o novo peer. Eles também enviam essa informação para seus antecessores e sucessores. Então, o novo peer entra na rede e notifica seu antecessor de que ele deve mudar seu sucessor.