Trabalho Prático Nº1 – Nível Aplicacional: Conceitos Introdutórios

Eduardo Benjamim Lopes Coelho e Henrique Gabriel dos Santos Neto

Universidade do Minho

Resumo O objetivo deste trabalho é a familiarização com conceitos básicos que envolve o desenvolvimento de aplicações em redes de comunicações, e na Internet em particular.

Questões

1. As aplicações em rede assentam normalmente em paradigmas cliente-servidor ou peer-to-peer

a) Explique em que se diferenciam ambos os modelos, salientando o papel das principais entidades envolvidas.

Num modelo cliente-servidor, o host do servidor tem de estar sempre ativo e tem um endereço de IP permanente. Para haver escalabilidade são necessários data centers. Por sua vez, os clientes comunicam com o servidor (não comunicam uns com os outros) e podem estar intermitentemente conectados, podendo assim alterar os seus endereços IP. Por outro lado, no modelo *peer-to-peer* (*P2P*) os sistemas finais (*peers*) são arbitários e não têm de estar sempre ativos, comunicando diretamente entre si. Os *peers* pedem serviços de outros *peers*, oferecendo, em troca, serviços aos outros. Deste modo, este modelo é auto escalável, na medida em que novos *peers* aumentam a capacidade de serviço, tal como demanda do serviço. Para além disso, os *peers* estão intermitentemente conectados, podendo assim mudar de endereços de IP. Deste modo, o modelo *peer-to-peer* tem uma gestão complexa.

b) Enuncie vantagens e desvantagens de cada paradigma e casos de aplicação.

Vantagens do modelo Cliente-Servidor sobre o modelo *Peer-to-Peer*

- Centralização : Ao contrário de *P2P*, onde não existe administração central, na arquitetura cliente-servidor existe controlo centralizado. Os servidores ajudam a gerir toda a rede. Os direitos de acesso e alocação de recursos são feitos pelos servidores.
- Armazenamento : No modelo cliente-servidor todos os ficheiros são armazenados no mesmo local. Deste modo, a gestão dos mesmos torna-se mais simples.
- Backups e Recuperações : Na arquitetura cliente-servidor, como todos os dados estão armazenados no servidor é mais simples realizar backups dos mesmos. Para além disso, na eventualidade de uma falha nos servidores, se se perderem dados, estes podem ser recuperados facilmente e eficientemente, enquanto que no modelo P2P é necessário fazer backups em todas as máquinas.
- Melhorias : No modelo cliente-servidor podem ser feitas melhorias mais facilmente, uma vez que basta modificar os servidores. Para além disso, novos recursos e sistemas podem ser adicionados fazendo as mudanças necessárias aos servidores.
- Acessibilidade: No modelo cliente-servidor, os servidores podem ser acedidos remotamente de várias plataformas.
- Segurança : No modelo cliente-servidor podem ser definidas regras de segurança e direitos de acesso durante a instalação de servidores.

– No modelo cliente-servidor, os servidores podem servir diferentes funções para diferentes clientes.

Vantagens do modelo *Peer-to-Peer* sobre o modelo Cliente-Servidor

– Congestão na rede: Demasiados pedidos dos clientes pode causar a congestão da rede, o que raramente acontece no modelo *Peer-to-Peer*. Para além disso, enquanto que na arquitetura Cliente-Servidor, a sobrecarga pode levar ao mau funcionamento dos servidores, no modelo *P2P*, a largura de banda da rede aumenta juntamente com o aumento dos *peers*/utilizadores (auto-escalável).
– A arquitetura cliente-servidor não é tão robusta como o modelo *P2P*, uma vez que, se o servidor avariar, toda a rede fica sem acesso. Para além disso, se um utilizador está a descarregar um ficheiro do servidor e esta operação é interrompida devido a algum erro, o download é cancelado. No entanto, se houvessem *peers*, eles poderiam fornecer os restantes *chunks* do ficheiro, não tendo de interromper a transferência do mesmo.
– Custo : Um sistema cliente-servidor é mais complexo de implementar e gerir, o que aumenta o seu custo.
– No modelo cliente-servidor, são necessários profissionais de informática para manter os servidores e gerir a rede em si.

2.

A Tabela 1 identifica tipos de aplicações amplamente usadas na Internet. Essas aplicações ou serviços apresentam diferente sensibilidade ao comportamento e desempenho da rede em si. Para cada tipo de aplicação (ou serviço), identifique qualitativamente os seus requisitos em termos de débito (throughput) necessário, atraso e suas variações (time sensitive) e perda de dados (loss sensitive). Dê exemplo concreto de aplicações da sua preferência que encaixem em cada tipo. Complemente a resposta quantificando os parâmetros em análise (referencie as suas fontes de informação).

Tipos de Aplicações	Débito (throughput)	Atraso e/ou Jitter (time sensitive)	Perda de dados (loss sensitive)	Aplicações
Web browsing	alguns kbs/s	não	sim	DuckDuckGo
Multimedia streaming	0,3 - 7 GBps/h	sim	não	Netflix
IP Telephony (VoIP)	100-300 kbs/s	sim	não	Teamspeak
File transfer/sharing	5kbps/s - 1Gbps/s	não	sim	bitTorrent
Interactive Games	400kbps/s - 2 Mbps/s	sim	não	Roblox
Video Conferencing	600kbps/s - 3Mbps/s	sim	não	Zoom

Web browsing [1]

Navegar pela internet pode exigir valores de débito bastante variados, uma vez que existem páginas *web* que exigem apenas algumas dezenas de *kbs* e outras páginas que exigem um débito de dezenas de Mbs. Atrasos na ligação não são graves uma vez que não afetam os resultados esperados ao carregar páginas web/pesquisar num motor de busca como o *DuckDuckGo*. Variações na latência da ligação também não causam problemas, pelo mesmo motivo.

[1] Webbrowsing - https://gobrolly.com/amount-data-and-bandwidth-required-web-browsing/

Multimedia Streaming [2]

Aplicações de streaming de multimédia são outro exemplo de serviços com uma ampla variação de débito necessário para a sua utilização. Por exemplo, consumir um vídeo na plataforma Netflix na qualidade mais baixa apenas exige um débito de 0,3 GBs/h, enquanto que assistir ao mesmo vídeo na qualidade máxima, já pode exigir 7 GBs/h. Este tipo de aplicações é extremamente sensível a atrasos e jitter na rede, uma vez que podem causar *buffering* do conteúdo a que está a ser assistido (conteúdo é colocado em pausa, enquanto a aplicação espera por mais dados para colocar em buffer), o que arruína a experiência para o utilizador. Para além disso, estes fatores também podem causar variações óbvias na qualidade do conteúdo, se a aplicação utilizar o protocolo *DASH*. Por outro lado, uma vez que, neste tipo de aplicações, a velocidade é priorizada em relação aos dados, a perda de dados é tolerada, afetando pouco a experiência do utilizador.

IP Telephony (VoIP) [3]

Aplicações *VoIP* normalmente não exigem um grande débito. Utilizando o Teamspeak como exemplo, podemos observar que esta aplicação necessita de um débito entre 100 a 300 kbs/s. No entanto, como o objetivo destas aplicações é a realização de chamadas de voz entre múltiplos utilizadores, atrasos e jitter na rede são extremamente graves, podendo tornar uma conversa de voz completamente incompreensível. Por outro lado, a perda de dados é, até certo ponto, tolerada, uma vez que, a perda de alguns pacotes pouco afeta a claridade de uma conversa de voz.

File transfer/sharing [4]
Aplicações para partilha e/ou transferência de ficheiros assentam, usualmente, em 2 modelos distintos: client-server e peer-to-peer. Tal como vai ser demonstrado mais abaixo, o débito exigido por estas aplicações depende, não só dos utilizadores e servidores envolvidos na transferência de ficheiros, mas também, do protocolo que a aplicação utiliza. Deste modo, o débito pode variar desde alguns kbps/s a vários Gbps/s. Como o objetivo deste tipo de aplicações é permitir a transferência de ficheiros aos seus utilizadores, atrasos e/ou jitter na rede não causam grandes problemas, uma vez que, apesar de poderem causar um maior tempo de transferência, o resultado final é sempre o pretendido (o ficheiro intacto). Ora, obviamente que a perda de dados não é desejada, uma vez que a perda de pacotes implicará as vários atrasos em retransmissões, pois caso contrário pode ocorrer corrupção e, consequentemente, a inutilidade de um ficheiro. Exemplos de aplicações de partilha/transferência de ficheiros são: BitTorrent, Mega, Telnet.

Interactive Games [5]

Jogos interativos normalmente não necessitam mais do que alguns Mbps/s de débito, tirando algumas exceções como, por exemplo, o novo Microsoft Flight Simulator, que descarrega gygbytes de dados dos seus servidores para alcançar gráficos visuais muito próximos da realidade. Por outro lado, o popular jogo online Roblox necessita entre 400 kbps/s a 2Mbps/s de débito. Neste tipo de aplicações, atrasos na rede e *jitter* afetam extremamente a sua jogabilidade, uma vez que causam inconsistências na experiência do jogador, podendo mesmo impedir a sua utilização. Devido aos gravíssimos problemas que a latência da rede pode causar, as aplicações focam-se na sua minimização, mesmo que isso cause alguma perda de dados.

[2] Netlifx - https://www.howtogeek.com/338983/how-much-data-does-netflix-use/
[3] Teamspeak - https://support.teamspeak.com/hc/en-us/articles/360002710657-How-much-bandwidth-does-TeamSpeak-require-
[4] bitTorrent - https://www.howtogeek.com/141257/htg-explains-how-does-bittorrent-work/
[5] Roblox - https://www.legendplayz.com/2021/08/roblox-internet-data-usage.html

Video conferencing [6]

As aplicações para conferências de vídeo, normalmente, necessitam de um débito na ordem dos Mbps/s para funcionarem corretamente. No caso do Zoom, que se tornou extremamente popular recentemente, o débito necessário é entre 600kbps/s a 3Mbps/s. Estas aplicações são extremamente sensíveis a ambos atrasos de rede e jitter, uma vez que estes podem causar distorção no vídeo e som, arruinando totalmente a experiência do utilizador. Porém, são tolerantes a perda de dados, já que a perda de alguns pacotes pouco afeta a qualidade da conferência, tanto no vídeo como na voz.

3.

Considere a topologia da Figura 1 onde será distribuído um ficheiro de tamanho X Gbits entre N nodos (hosts), Assuma que os débitos de download e upload do nodo i. são respetivamente d_i e u_i. Assuma ainda que: (i) os hosts estão dedicados à distribuição do ficheiro, i.e. não realizam outras tarefas; e (ii) o núcleo da rede (core) não apresenta qualquer estrangulamento (bottleneck) em termos de largura de banda, i.e., qualquer eventual limitação existe nas redes de acesso dos vários n_i. O valor de X deve ser indexado ao identificador de cada grupo de trabalho, i.e., $X = \frac{38}{10} = 3.8$

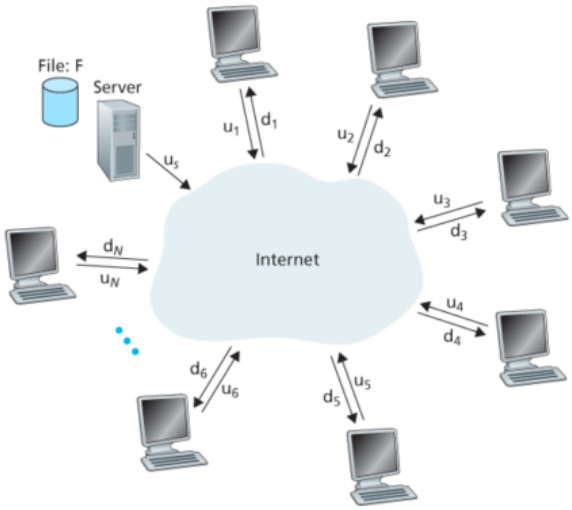

Figura 1: Distribuição do ficheiro F [Kurose, and Ross, 2016].

Sabendo que o servidor tem um débito de upload u_s=1Gbps, e que d_i=100Mbps, calcule, justificando, o tempo mínimo de distribuição de F pelos N nodos quando $N = 10$, $N = 100$ e $N = 1000$, e para débitos de upload ui de: a) 1Mbps; b) 5Mbps e c) 10Mbps, usando os modelos de distribuição: (i) cliente-servidor e (ii) peer-to-peer. Apresente os resultados numa tabela comparativa, bem como o processo de cálculo. Que conclusões pode tirar?

O tamanho do ficheiro é $F = \frac{38}{10} = 3,8\ Gb$.

[6] Zoom - https://www.reviews.org/internet-service/how-much-data-does-zoom-use/

Para calcularmos o tempo mínimo de transferência podemos estudar alguns pontos de estrangulamento em ambas as implementações.

Na implementação cliente servidor, os fatores limitantes são a taxa de upload do ficheiro, que neste caso diminui conforme o aumento do número de clientes conectados ao servidor e a taxa de download de cada cliente, podendo ser reduzidos na expressão:

$$máximo\left(\frac{N \times F}{u_s}; \frac{F}{d_{min}}\right)$$

Na arquitetura *P2P*, os fatores limitantes são a taxa de upload do servidor, a taxa mínima de download dos clientes e, adicionalmente, a taxa máxima de que o ficheiro é distribuído pelos clientes. Desta forma temos a expressão:

$$máximo\left(\frac{F}{u_s}; \frac{F}{d_{min}}; \frac{N \times F}{u_s + \Sigma u_i}\right)$$

Alinea	u_i (MBps)	Cliente - Servidor			P2P		
		10	100	1000	10	100	1000
a)	1	40,802	408,022	4080,219	40,802	370,929	2040,109
b)	5				40,802	272,015	680,036
c)	10				40,802	204,011	370,929

Tabela 1: Resultados

Como é possível observar, em todos os casos o método *P2P* é sempre igual ou mais rápido que o método cliente servidor, chegado a ter diferenças de uma hora em alguns casos. Adicionalmente, podemos também observar que como esperado, a abordagem *P2P* é mais rápida quanto melhor for a largura de banda dos *peers* ao contrário do método cliente-servidor, que é transparente a estas melhorias.

Desta forma, podemos confirmar as nossa conclusões prévias, e que de facto, o método *P2P* é mais escalável que o método Cliente-Servidor.

Conclusão

Com este guião prático, conseguimos estudar e familiarizar-nos melhor com os conceitos básicos no desenvolvimento de aplicações em redes e, consequentemente, sobre os vários problemas que estes implicam.

Na verdade, estes exercícios permitiram-nos conhecer melhor os diversos tipos de aplicações em rede, tal como os problemas e desafios que cada um nos apresenta, como, por exemplo, serviços de streaming de multimédia como a Netflix e a necessidade de haver um equilíbrio entre a latência com que o conteúdo nos é apresentado e a sua qualidade. Para além disso, foi uma ótima oportunidade para verificarmos, em primeira mão, as vantagens e desvantagens das várias arquiteturas de aplicações em rede, nomeadamente, as arquiteturas cliente-servidor e *peer-to-peer*.