1.

A - Menos RTTs até receber uma resposta para o cliente mas manter um número mais elevado de conexões em simultâneo é taxante!

2.

B - Prometo

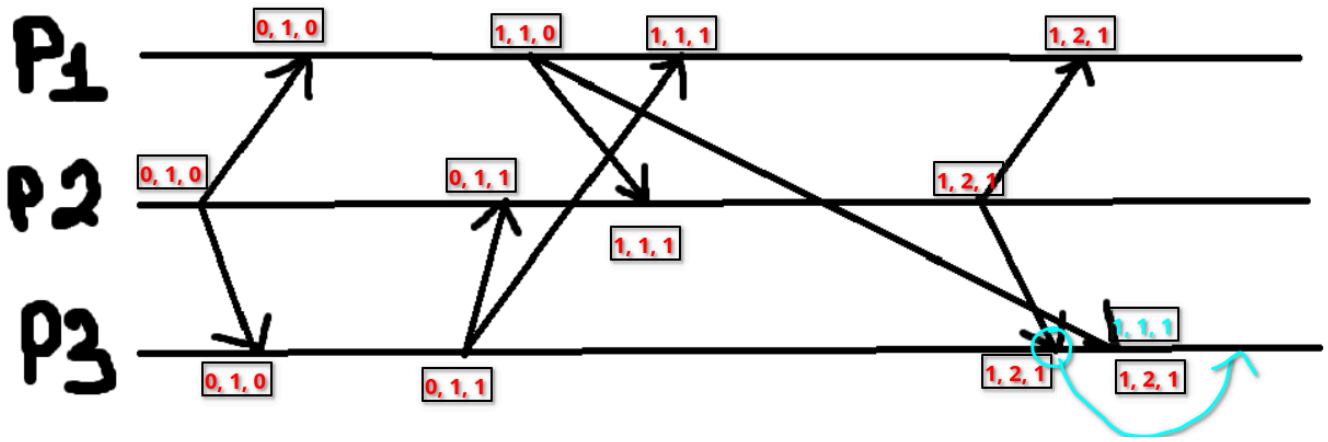
3.

A - Se as operações forem idempotentes, podemos repetir as vezes que quisermos, se a semântica usada for at-most-once, evitamos repetir as operações já bem sucedidas.

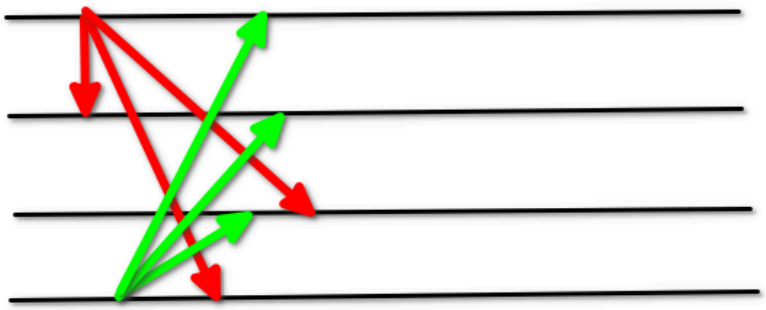
4.

Não há

5.



6.



- **Ordem Causal**:
  - Garante que, se $a \rightarrow b$, então a será entregue antes de b a TODOS os processos que os receberem. (Se $a \parallel b$, a ordem causal não impõe restrições)
- **Ordem Total**:
  - Garante que a ordem pela qual um recetor recebe todas as mensagens é igual para todos os processos no grupo. Ou seja, se P_i recebe a sequência $\{m_1, m_2\}$, todos os outros processos no grupo também receberão $\{m_1, m_2\}$.

Como os evento de envio vermelho e o verde são concorrentes, respeita a ordem causal, nem há restrições a cumprir! Mas a ordem pela qual os outros processos recebem as mensagens é bem diferente.

7.

Já saiu em 2024:

- Assumimos que A e B ambos possuem $K_{A_{\text{pub}}}$ e $K_{B_{\text{pub}}}$, bem como as respetivas chaves privadas.
- Queremos garantir:
  - Confidencialidade (apenas o B pode ler D)
  - Integridade (B pode ver que D não foi alterado e que veio de A)
  - Confirmação da receção e acesso (A deve receber uma prova de que B recebeu, desencriptou e leu D, com não-repúdio da receção por parte de B)

1. $A \rightarrow B : \text{TransactionID}, \{D\}^{K_s}, \{K_s\}^{K_{B_{\text{pub}}}}, \{H(D)\}^{K_{A_{\text{priv}}}}$
   - Gera um hash de D, $H(D)$ e assina com a sua chave privada $K_{A_{\text{priv}}}$
   - Gera uma chave privada da sessão K_s e assina com a chave pública de B, $K_{B_{\text{pub}}}$
   - Inclui TransactionID para identificar unicamente esta ligação
2. $B \rightarrow A : \text{TransactionID}, \{H(D')\}^{K_{B_{priv}}}$
   - Gera hash do documento recebido D', $H(D')$
   - Verifica se é igual ao $H(D)$ recebido

3. Resta agora a A verificar a confirmação de B.
    - Se a assinatura de B for válida, usando $K_{B_{\text{pub}}}$, consegue extrair $H(D')$ e comparar com o seu original.

8.

- **Confirmação de receção da resposta**: A mensagem de ACK tem o request ID da mensagem de resposta que está a ser confirmada. Ao receber este ACK, o server sabe que a resposta foi entregue com sucesso.
- **Gestão de recursos do server**: Para garantir at-most-once, o server mantém um registo das mensagens de resposta que enviou. Este registo faz com que não seja preciso re-executar uma ação para enviar a resposta associada, se o pedido for duplicado.
- **Libertação de memória do server**: A receção do ACK permite que o server apague a mensagem da resposta correspondente ao seu registo.

9.

Gnutella:

- Para procurar um ficheiro, Gnutella usa query flooding.
- Query flooding é o principal obstáculo para a escalabilidade do Gnutella. A rede pode ficar congestionada com mensagens de query, especialmente em redes grandes.

BitTorrent:

- O BitTorrent não se foca na procura eficiente, mas sim no download eficiente (fetching) de ficheiros. A abordagem do BitTorrent para localizar um ficheiro depende de um componente centralizado inicial, o tracker, e de um ficheiro de metadados, o .torrent file.
- Para um cliente obter um ficheiro:
    1. O cliente precisa de um .torrent file correspondente ao ficheiro que pretende descarregar, esse ficheiro contém metadados como o nome, tamanho, checksum e endereço IP do tracker
    2. O cliente contacta o tracker especificado no ficheiro .torrent. O tracker é um server que monitoriza os clientes ativos (seeders e leechers) de um determinado ficheiro
    3. O tracker devolve ao cliente uma lista de outros clientes ativos (peer set) que possuem partes ou a totalidade do ficheiro
    4. Com esta lista, o cliente estabelece ligações diretas com os outros peers e começa a descarregar pedaços (chunks) do ficheiro, enquanto simultaneamente lhes envia os chunks que já possui.
- A estratégia do BitTorrent é mais escalável para a localização de ficheiros, em comparação com o Gnutella, porque a função de procura (encontrar o tracker e o .torrent file) é

tipicamente feita através de métodos externos (como um browser) e a distribuição da lista de peers é centralizada no tracker. O tracker não armazena os dados do ficheiro nem participa na descarga, o que rediz a sua carga. O tráfego pesado de troca de dados ocorre diretamente entre os peers, sem envolver o tracker na transferência real do ficheiro. No entanto, o tracker ainda representa um single point of failure. Se o tracker falhar, novos clientes não vão conseguir encontrar peers para o ficheiro, embora os peers existentes possam continuar a comunicar entre si se já tiverem o peer set.