**1. Qual é a diferença entre controle de fluxo e controle de congestionamento do TCP? Explique os dois métodos em detalhes incluindo todas as fases.**

Controle de Congestionamento: Mecanismos para evitar que um transmissor sobrecarregue a rede. Ou seja, a origem da transmissão envia os dados na velocidade que a REDE suportar.

Controle de Fluxo:Na camada de transporte é o mecanismo de "Janela Deslizante" e tenta evitar que um transmissor sobrecarregue o Receptor. Ou seja, a origem da transmissão preocupa-se apenas em transmitir na mesma velocidade do destino (fim-a-fim), independente se a rede aguenta ou não.

Em resumo, o controle de fluxo preocupa-se apenas com o fim-a-fim (origen - destino), enquanto o controle de congestionamento preocupa-se com a rede como um todo

**2. Quando aprendemos o controle de congestionamento em TCP admitimos que o remetente TCP sempre tinha dados para enviar. Agora considere o caso em que o remetente TCP envie uma grande quantidade de dados e então fique ocioso em t1 (já que não há mais dados para enviar). O TCP permanecerá ocioso por um período de tempo relativamente longo e então irá querer enviar mais dados em t2. Quais são as vantagens e desvantagens do TCP utilizar os valores cwnd essthresh de t1 quando começar a enviar dados em t2? Que alternativa você recomendaria? Por que?**

Se as estimativas de RTT e RTO estiverem em boa forma, é possível afirmar, com grande certeza, que, se o cronômetro de retransmissão expirar (timeout), um pacote foi perdido. Pacotes se perdem na rede por duas razões: são danificados em trânsito, ou a rede está congestionada e, em algum lugar, não há buffer suficiente. Na maioria dos enlaces, a perda de pacotes devido a danificação dos dados é rara (<< 1%). Logo, é muito provável que a perda de um pacote seja conseqüência direta de congestionamento na rede.

Congestion avoidance e slow start são algoritmos independentes com objetivos diferentes. Mas, quando ocorre congestionamento, baixa-se a taxa de transmissão de pacotes na rede, e então invoca-se o slow start para recomeçar o processo de aumento da janela e taxa de transmisão.

O congestion avoidance e o slow start requerem que duas variáveis seja monitoradas para cada conexão: a janela de congestionamento, **cwnd** , e a janela limite para o algoritmo slow start, **ssthresh.** O algoritmo combinado funciona da seguinte maneira:

1. Durante a inicialização de uma conexão, a cwnd é igual a um segmento e a ssthresh, a 65536 bytes;
2. A rotina de emissão do TCP sempre envia o mínimo entre a cwnd e a janela anunciada pelo receptor;
3. Quando ocorre congestionamento (indicada por timeout ou o recebimento de ACKs duplicados), metade do valor atual da janela de transmissão (o mínimo entre a cwnd e a janela anuciada pelo receptor) é armazenado em ssthresh. Além disso, se o congestionamento foi causada por timeout, a cwnd passa a valer um segmento (ou seja, slow-start);
4. Quando novos dados forem confirmados pelo nó destino, a cwnd é aumentada, mas a maneira como isto é feito depende se está sendo feito o slow-start ou o congestion avoidance.

Se a cwnd for menor ou igual a ssthresh, o TCP está em slow start; caso contrário, ele está realizando o congestion avoidance. O slow start prossegue até que a janela de transmissão do TCP esteja com metade do tamanho de quando ocorreu o congestionamento (guarda-se metade do valor da janela que causou problema no passo 3), e, então, passa-se para a fase de congestion avoidance.

O slow start faz a cwnd começar valendo um segmento e ser incrementada de um segmento toda vez que um ACK é recebido. Como mencionado anteriormente, isto abre a janela exponencialmente: um segmento é enviado, então dois, quatro, e assim por diante. O congestion avoidance, por sua vez, faz com que a cwnd seja incrementada por **segsize\*segsize/cwnd** toda vez que um ACK for recebido, onde segsize é o tamanho do segmento (segsize e cwnd são mantidos em bytes).

Isto faz com que a cwnd tenha um aumento linear, comparado com o aumento exponencial do slow start. O aumento da cwnd deve ser de, no máximo, um segmento a cada round-trip time (independente de quantos ACKs sejam recebidos neste RTT), enquanto que o slow start incrementa a cwnd baseado no número de ACKs recebidos em um RTT.

## Recomendaria Fast retransmit e fast recovery

Antes de descrever a mudança, note que o TCP é obrigado a gerar uma confirmação imediata (um ACK duplicado) quando um segmento fora de ordem é recebido. A finalidade deste ACK duplicado é indicar ao emissor que um segmento foi recebido fora de ordem e qual o número de seqüência esperado.

Partindo do fato que não se sabe se um ACK duplicado foi causado por um segmento perdido ou somente uma reordenação de segmentos, espera-se que um pequeno número de ACKs duplicados sejam recebidos antes que qualquer atitude seja tomada. É assumido que, se for somente uma reordenação de segmentos, só serão recebidos um ou dois ACKS duplicados antes do segmento fora de ordem alcançar o destino e ser processado, o que implicará em um novo ACK.

Se três ou mais ACKS duplicados forem recebidos em seguida, é um forte indício que um segmento foi perdido. O TCP realiza, então, a retransmissão imediata do que aparenta ser o segmento perdido, **sem** esperar que o cronômetro de retransmissão expire (timeout). Este é o algoritmo de fast retransmit. Em seguida, congestion avoidance, e não slow start, é feito. Este é o algoritmo de fast recovery.

A razão pela qual não se faz slow start nesse caso é que o recebimento de ACKs duplicados diz mais do que simplesmente um segmento foi perdido. Sabe-se que o destino só pode gerar ACKs duplicados quando outro segmento for recebido, isto é, o segmento deixou a camada física e está no buffer do destino. Logo, ainda temos dados trafegando entre os dois nós. Então, não é aconselhável reduzir o fluxo abruptamente usando o slow start.