1) Na comunicação em redes, o endereço de rede (como um endereço IP) na camada de rede identifica qual máquina deve receber os dados, enquanto a porta na camada de transporte especifica qual aplicação ou serviço dentro dessa máquina deve processá-los. Podemos comparar o endereço de rede a um endereço de rua, que leva ao prédio certo, e a porta ao número do apartamento, que indica exatamente para quem a entrega é destinada. Assim, o endereço de rede garante que os dados cheguem ao dispositivo correto, e a porta direciona a entrega para o programa apropriado dentro desse dispositivo.
2) A camada de transporte dá sentido à pilha de protocolos porque ela é responsável por garantir que a comunicação entre aplicações em dispositivos diferentes aconteça de forma confiável e organizada. Enquanto as camadas inferiores (como a física, enlace e rede) cuidam de transmitir os dados pelo caminho até o destino, é a camada de transporte que monta, entrega, organiza e gerencia esses dados para que a aplicação final possa usá-los corretamente. Ela fornece serviços essenciais como controle de erro, controle de fluxo, e montagem dos dados em sequência, além de permitir a comunicação simultânea entre várias aplicações diferentes no mesmo dispositivo (usando portas). Sem a camada de transporte, a transmissão de dados seria apenas um fluxo bruto de bits, sem estrutura ou garantia de entrega, o que tornaria impossível o funcionamento confiável de aplicações como navegadores, e-mails, ou jogos online.
3) Uma aplicação do tipo one shot realiza uma comunicação rápida e única, enviando poucos dados e encerrando logo em seguida, sem necessidade de manter uma conexão prolongada. O TCP pode ser utilizado para esse tipo de aplicação, mas não é a melhor opção, pois ele exige a criação e finalização de uma conexão confiável, o que gera sobrecarga desnecessária para transmissões tão curtas. Já o **UDP** é mais apropriado para aplicações one shot, pois é um protocolo leve, sem conexão, que permite enviar dados rapidamente, embora não garanta entrega, deixando esse controle, se necessário, para a própria aplicação.
4) Em uma aplicação do tipo cliente-servidor usando **TCP**, o servidor primeiro executa as primitivas de soquete criando um soquete (`socket()`), associando-o a um endereço e porta (`bind()`), ficando pronto para receber conexões (`listen()`) e, em seguida, aceitando uma conexão de um cliente (`accept()`). Do lado do cliente, a aplicação cria um soquete (`socket()`), e depois tenta se conectar ao servidor usando o endereço IP e a porta apropriada (`connect()`). Uma vez que a conexão está estabelecida, tanto o cliente quanto o servidor usam primitivas como `send()` para enviar dados e `recv()` para receber dados. Quando a comunicação termina, ambos fecham seus soquetes com a primitiva `close()`. Dessa forma, o TCP gerencia toda a transmissão de dados de forma confiável entre cliente e servidor.
5) A transferência confiável de dados no TCP é garantida por um conjunto de mecanismos que asseguram que os dados enviados cheguem corretamente e na ordem ao destino. O TCP divide os dados em segmentos e utiliza os campos Sequence Number e Acknowledgement Number, junto com o sinalizador ACK, para controlar esse processo. O Sequence Number indica o número do primeiro byte de dados daquele segmento, permitindo que o receptor saiba a ordem correta dos pacotes recebidos. O Acknowledgement Number, por sua vez, informa qual é o próximo byte

que o receptor espera receber, ou seja, ele confirma o recebimento correto dos dados anteriores. O bit ACK, quando ativado, sinaliza que o número de reconhecimento (Acknowledgement Number) está válido e deve ser interpretado. Se algum segmento se perder ou chegar corrompido, o TCP detecta isso pela falta de confirmação (ACK) e o retransmite. Esse controle garante entrega confiável, ordenada e sem duplicações, mesmo em redes instáveis.

6) O gerenciamento de conexões do TCP envolve dois processos principais: o estabelecimento e o término da conexão, ambos utilizando os campos de controle SYN, ACK e FIN para coordenar e garantir uma comunicação confiável entre cliente e servidor.
   No estabelecimento da conexão, ocorre o chamado three-way handshake (aperto de mão em três etapas):
   O cliente envia um segmento com o campo SYN ativado (SYN = 1) para iniciar a conexão e indicar seu número de sequência inicial.
   O servidor responde com um segmento com os campos SYN e ACK ativados (SYN = 1, ACK = 1), reconhecendo o SYN do cliente e enviando seu próprio número de sequência.
   O cliente então envia um segmento com ACK ativado (ACK = 1), confirmando o recebimento do SYN do servidor. A partir daí, a conexão está estabelecida.
   No término da conexão, ocorre uma troca de mensagens chamada four-way handshake:
   A parte que deseja encerrar a conexão envia um segmento com o campo FIN ativado(FIN = 1), indicando que não quer mais enviar dados.
   A outra parte responde com um ACK (ACK = 1), confirmando o recebimento do FIN.
   Depois, essa parte também envia seu próprio FIN para encerrar sua transmissão.
   Por fim, o outro lado responde com ACK, completando o encerramento da conexão.
   Esses campos garantem que tanto a abertura quanto o fechamento da conexão sejam feitos de forma ordenada, sincronizada e confiável entre os dois dispositivos.
7) O controle de fluxo do TCP é um mecanismo que evita que o remetente sobrecarregue o receptor com mais dados do que ele pode processar ou armazenar em seu buffer. Para isso, o TCP usa o campo Window Size (tamanho da janela) no cabeçalho dos segmentos. Esse campo indica ao remetente quantos bytes de dados ele ainda pode enviar antes de precisar esperar por um novo reconhecimento (ACK).
8) O controle de fluxo no TCP tem como objetivo impedir que o remetente envie dados em uma velocidade maior do que o receptor pode processar, evitando o transbordamento do buffer de recepção. Isso é realizado por meio do campo Window Size no cabeçalho TCP, que informa quantos bytes o receptor está apto a receber naquele momento.
   Por outro lado, o controle de congestionamento busca prevenir a sobrecarga da rede com excesso de pacotes, o que poderia causar perdas e atrasos. Para isso, o TCP utiliza algoritmos adaptativos que ajustam dinamicamente a taxa de envio de dados conforme as condições da rede, promovendo uma transmissão mais eficiente e estável.
9) O controle de congestionamento do TCP visa evitar a sobrecarga da rede ajustando dinamicamente a taxa de envio de dados para prevenir perda de pacotes e garantir

uma transmissão eficiente. O algoritmo de iniciação lenta é utilizado no início de uma conexão TCP, começando com uma janela de congestionamento muito pequena (geralmente 1 ou 2 MSS) e aumentando exponencialmente a cada ACK recebido. Esse crescimento exponencial continua até que a janela atinja um valor limiar, o limiar de congestionamento (ssthresh), momento em que o crescimento da janela se torna linear. Esse processo permite que a conexão inicie de forma conservadora, adaptando-se de maneira gradual à capacidade da rede e evitando congestionamentos.

10) A figura ilustra a variação do tamanho da janela de congestionamento (congestion window) durante uma conexão TCP, demonstrando os comportamentos de crescimento exponencial (slow start), crescimento linear (congestion avoidance) e resposta a congestionamento. Inicialmente, a janela cresce exponencialmente até atingir um limiar (threshold), momento em que o crescimento passa a ser linear. Caso ocorra um timeout devido à perda de pacotes, o tamanho da janela é drasticamente reduzido para 1 KB, e o limiar é ajustado para metade do valor da janela anterior. A partir desse ponto, o crescimento volta a ser exponencial até alcançar o novo limiar, após o qual o comportamento linear é retomado. Esse mecanismo visa equilibrar o desempenho da conexão e o controle de congestionamento na rede, ajustando dinamicamente a taxa de envio de dados para evitar sobrecarga e perdas.