Resposta questão 1)

Escolhi reutilizar a camada de Rede e a de Enlace.

Para conseguir criar uma arquitetura que tenha resiliência e escalabilidade, resolvi usar como escopo a arquitetura P2P, onde existe um servidor (o que garante a resiliência, a arquitetura continua de pé mesmo sem a presença dos pares) que contém o arquivo a ser compartilhado inicialmente. À medida que os pares vão sendo formados, os clientes (os pares) também passam a fazer papel de servidor já que eles passam a compartilhar os arquivos uns com os outros, aumentando assim a escalabilidade. Dessa mesma forma, acontece com os serviços de streaming de vídeos armazenados, um servidor detém um arquivo em vídeo, e os pares vão chegando e fazendo o download desse arquivo e passam a fazer o upload dele também para outros pares que chegam, fazendo assim o papel de cliente e servidor simultaneamente.

Para o serviço de tradução de nomes em endereços, utilizarei como escopo o protocolo DNS.

Para o serviço de e-mail, utilizarei como base o protocolo SMTP.

Resposta questão 2)

Considerando um canal subjacente não confiável sujeito a erros de bits e perdas de pacotes, existem dois problemas de comunicação entre transmissor e receptor para serem solucionados e fazer com que o meu protocolo proveja uma transmissão confiável de dados: o primeiro problema são os erros de bits e o segundo problema é a perda de pacotes.

Para solucionar o problema de erros de bits faz-se necessário: verificar se algum bit foi perdido; que o destinatário envie um feedback para o remetente indicando se o pacote foi recebido integramente ou não; e a retransmissão dos pacotes que foram recebidos com erros.

Para verificar se algum bit foi perdido, é enviado um número correspondente ao complemento de 1 do segmento transmitido (sequência de 16 bits), no campo checksum, para o receptor onde a verificação é feita somando o número recebido no campo checksum com o segmento recebido (soma de verificação - checksum), caso a soma dê todos os bits iguais a 1, não houve perda de bits, caso contrário, houve perda de bits.

Para que o transmissor saiba se houve erros de bits ou não, é preciso que o receptor envie um feedback de volta para o transmissor. Caso o feedback seja positivo (ACK), o pacote foi recebido perfeitamente, caso contrário, o feedback seja negativo (NACK), o pacote recebido tinha erros, o transmissor reenvia o pacote para o receptor. No entanto, como faz-se necessário aumentar a utilização do protocolo em relação ao rdt 1.0, vários pacotes são enviados em sequência, sem esperar que o feedback do último pacote enviado chegue (paralelismo), aumentando assim a fração de tempo em que o transmissor está enviando bits para o canal(utilização). Então, para sabermos a que pacote o feedback se refere e evitar duplicação de pacotes, adiciona-se um novo campo no pacote de dados do remetente que contém um número de sequência. Se o número de sequência do pacote recebido é igual ao número de sequência do pacote armazenado antes desse, é uma retransmissão, caso não, é um novo pacote. Os números de sequência variam de 1 a N, sendo N o tamanho

do buffer disponível. Sendo assim, solucionamos também o problema de que o protocolo será usado por dispositivos com pouca memória, basta escolher um valor de N adequado.

Se um pacote de número de sequência ‘n’ chegar sem perda de bits e estiver na ordem certa (caso o pacote armazenado antes desse tenha o número de sequência igual a ‘n-1’) o destinatário armazena a parte de dados do pacote e o envia à camada superior e envia um ACK com o número de sequência ‘n’ para o transmissor; nos outros casos, o pacote estando na ordem errada ou o pacote chegando danificado, o pacote é descartado e um ACK do elemento ‘n-1’ será enviado ao transmissor fazendo com que ele envie o pacote com número de sequência n. Dessa forma, resolvemos casos de perda de bits e perda de pacotes.

Um temporizador de contagem regressiva com o tempo médio estipulado do atraso de um pacote ser enviado, processado e seu feedback chegar é usado para ter controle de perda de pacotes. Caso o feedback que chegue corresponda ao pacote na ordem correta, ele para o temporizador e espera outros feedbacks chegarem e faz a mesma verificação. Por exemplo, se o último ACK recebido foi de número de sequência 2, o próximo ACK a chegar deveria ter o número de sequência igual a 3, se o ACK recebido tenha o número de sequência igual a 3, ou o pacote 3 chegou integramente, então o temporizador é pausado (ele já havia sido iniciado quando o pacote foi enviado). Mas, caso o ACK recebido tenha o número de sequência igual a 2, o pacote 3 chegou com perda de bits, ou ele não chegou, ou ele sofreu um atraso longo então o temporizador é iniciado e o transmissor segue esperando um ACK 3. Existe também a possibilidade de nenhum ACK ser recebido, então o temporizador iniciado ao enviar o pacote chegaria a zero e quando o temporizador termina, chega a 0, o transmissor reenvia todos os pacotes que já tinham sido enviados a partir do 3 em sequência.

FSM remetente:

Considerando que o primeiro número dos pacotes a serem analisados (base) é 1 e que o próximo a ser analisado é o de número 1. Quando há uma chamada da camada de cima passando dados para a camada de transporte, verificaremos se o primeiro número dos pacotes a serem analisados mais o tamanho do buffer é menor que o número de sequência do próximo pacote a ser enviado, se for verdade então há espaço no buffer e um pacote é criado com o próximo a ser analisado como número de sequência e é enviado. Se a base for igual ao próximo a ser analisado ligamos o temporizador. Ao final de tudo isso, atualizamos o valor do próximo a ser analisado somando 1 para que o processo se repita com outro conjunto de dados mais tarde. Se não houver espaço no buffer, ou seja, se a primeira verificação não for atendida, os dados são recusados.

Caso o remetente tenha recebido um feedback não corrompido, o valor da base é atualizado para o valor do número de sequência do ACK que chegou + 1, ou seja, o próximo que deve ser enviado, se ele coincidir com o que o próximo a ser analisado, correu tudo bem, todos os pacotes chegaram sem erros e em ordem, então o temporizador é pausado, caso haja outra entrada de dados, tudo se repete. Se não coincidir, algum pacote foi perdido, então inicia o temporizador.

Acabando o tempo do temporizador, ele é inicializado de novo e todos os pacotes de base até o último criado são enviados. Ou seja, o próximo a ser enviado e todos os outro que vem depois dele e que já estão criados serão enviados novamente.

Caso o remetente tenha recebido um feedback corrompido, ele não faz nada. Isso será tratado posteriormente quando um feedback válido chegar, ou quando o temporizador terminar.

FSM destinatário:

Inicialmente o número de sequência esperado é igual a 1 e é criado um pacote ACK com o número de sequência igual a 0

Recebendo um pacote que não foi corrompido e o número de sequência corresponde ao esperado, os dados são extraídos do pacote e enviados para a camada de cima, após isso, um pacote ACK é criado com o número de sequência esperado e enviado. Depois disso, o número de sequência esperado é atualizado somando um para que o processo se repita mais tarde com o próximo pacote da fila.

Caso o pacote recebido tenha sido corrompido ou o número de sequência não seja o esperado (pacote fora de ordem), a última mensagem ACK criada é reenviada para o transmissor.