UNIVERSIDADE FEDERAL DO RIO DE JANEIRO – UFRJ

INSTITUTO DE MATEMÁTICA – IM
DEPARTAMENTO DE CIÊNCIA DA COMPUTAÇÃO – DCC

Trabalho 2 – Camada Enlace

Disciplina: Teleprocessamento e Rede
Professor: Flávia Delicato

Júlio César Machado Bueno	106033507
Marcus Vinicius Rabelo da Silva	107363206
Jonas Tomaz Alves Da Silva	106089655

Introdução

Para este trabalho, consideramos uma emulação do serviço provido pela camada de enlace como não-orientado a conexão (sem o estabelecimento de um “handshaking”) e não-confiável (sem garantia de recebimento correto dos quadros enviados).

Este relatório é referente a uma emulação da camada de enlace, que utiliza os serviços da emulação da camada física, que é foi implementada de acordo com as orientações dadas.

Sendo assim, é feita a implementação de um comutador da camada de enlace, que coordena a comunicação entre os hosts de maneira transparente para estes. Algumas simplificações foram consideradas, como:

1. A ausência de TTL na tabela de comutação, a conexão de todos os hosts ao comutador uma vez que não existem diversas LANs conectadas ao comutador.
2. A ausência de diferentes portas para tratamento paralelo da comunicação entre os hosts já que uma mesma porta deve ser usada para a execução do comutador e dos hosts.

Além disso, o projeto foi concebido utilizando C++ dado a forte correlação entre os protótipos das funções auxiliares da camada de enlace dados e os protótipos da linguagem.

Visão geral da Camada de Enlace

De acordo como modelo “bottom-up”, a camada de enlace é a segunda do modelo híbrido. Portanto, resumidamente, a função desta camada é transmitir um datagrama da camada de rede, logo acima no modelo, de um nó de enlace para outro. Dentre os serviços que presta para a camada de rede estão:

1. **Enquadramento**: deve-se determinar quais as informações adicionais ao payload definido devem ser usadas e como organizar todas as informações, inclusive o próprio payload dentro do quadro. Por fim, deve-se ter mecanismos para definir início e fim de quadro.
2. **Controle de fluxo**: evitar que o nó receptor não receba quadros a uma taxa maior do que a que é capaz de processar.
3. **Controle de acesso**: para redes broadcast este serviço é necessário.
4. **Entrega confiável**: garantir que quadros enviados sejam recebidos com sucesso.
5. **Detecção de erro**: mecanismo para detectar possíveis erros de transmissão.

Projeto da Camada de Enlace

Descrição das estruturas de dados definidas

enlace.cpp

- **unsigned char local_addr**: endereço MAC local.
- **int flag**: identifica o recebimento de um quadro.
- **int loss_prob**: recebe a probabilidade de erro desejada.
- **char buffer_in[]**: guarda um frame recebido.
- **queue<string> buffer_out**: guarda os frames a serem transmitidos.
- **string buffer_in_temp**: guarda o frame final recebido.
- **string buffer_out_temp**: guarda o frame sendo transmitido.

comutador.cpp

- **int sock**: socket usado na comunicação do comutador com os hosts.
- **int flag**: indica qual quadro recebido.
- **int port**: porta usada na comunicação do comutador com os hosts.
- **char buffer**: guarda byte recebido.
- **char buffer_in[]**: guarda um frame recebido.
- **map <unsigned char, string> tabela_broadcast**: mapeia todos os hosts que estão conectados ao comutador.
- **map <unsigned char, string> tabela_comutador**: tabela de comutação.
- **struct sigaction handler**: estrutura para definição da ação para tratamento de sinal.
- **struct sockaddr_in fromAddr**: endereço IP da máquina que está transmitindo um quadro.
- **struct sockaddr_in destAddr**: endereço IP da máquina que está recebendo um quadro.

Descrição das funções oferecidas (API da camada de Enlace)

- **int L_Activate_Request (unsigned char, int, char *)**
  Inicializa a camada de enlace e da camada física. Seus parâmetros são:

  1. o endereço MAC (da emulação) da máquina local

2. a porta do comutador
3. endereço IP do comutador.

Retorna 1 em caso de sucesso e 0 caso contrário.

- **void L_Data_Request (unsigned char, char *, int)**
  Solicita a transmissão de um quadro. Seus parâmetros são:
  1. o endereço MAC (da emulação) destino
  2. dados a serem transmitidos
  3. número de bytes.

- **int L_Data_Indication (void)**
  Testa se há um quadro recebido dentro do nível de enlace. Retorna 1 caso exista e 0 caso contrário.

- **int L_Data_Receive (unsigned char *, char *, int)**
  Busca no nível de enlace os dados do último quadro recebido. Seus parâmetros são:
  1. ponteiro para o endereço MAC do emissor
  2. ponteiro para os dados do quadro recebido.
  3. tamanho máximo esperado para o campo de dados do quadro.

  Retorna o número de bytes do campo de dados ou -1 em caso de falha.

- **void L_MainLoop (void)**
  Caso exista um quadro a ser transmitido, transmite um byte para o nível físico. Caso exista byte para receber no nível físico, recebe o byte.

- **void L_Set_Loss_Probability (float)**
  Estabelece a taxa de perda de quadros dada pelo usuário.

- **void L_Deactivate_Request(void)**
  Termina o funcionamento do nível físico e do nível de enlace.

Descrição de possíveis funções adicionais (suporte a API principal)

- **void l_Recebe_Byte(void)**
  Recebe um byte e armazena no buffer da camada de enlace. Avisa quando todos os bytes de um quadro foram recebidos. Descarta o quadro caso não passe na validação.

- **bool l_Valida_Quadro(const char *)**
  Valida um quadro recebido; quadros válidos são os destinados, diretamente ou por broadcast, para a máquina local e não-corrompidos dado que passaram no teste de detecção de erro e o número aleatório gerado é menor que a probabilidade de erro definida.

- **void l_Transmite_Byte()**
  Transmite um byte do quadro e verifica se terminou a transmissão do quadro.

- **int Calcula_Codigo_Erro(const char *frame)**
  Resolve o código de detecção de erro a partir de frame.

- **int Activate_Request(int)**
  Inicializa o nível físico do comutador. Seu parâmetro é a porta combinada para a comunicação do comutador com os hosts.

- **void SIGIOHandler(int signalType)**
  Função de tratamento do sinal SIGIO, sinal disparado sempre que uma E/S assíncrona está pronta.

- **void Deactivate_Request(void)**
  Finaliza o funcionamento do comutador.

- **void Data_Request(char *, char *, int)**
  Usada para o caso de o endereço MAC da máquina destino estar na tabela de comutação. Solicita a transmissão de um byte para a máquina especificada como destino. Os parâmetros são:
  1. dados a serem transmitidos
  2. endereço IP e porta (combinada ser a mesma para todos os hosts e comutador) da máquina destino.

- **void Data_Request_Broadcast(char *data, unsigned char MAC_src)**
  Usada em casos de requisição de broadcast ou de o endereço da máquina destino não estarem na tabela de comutação. Solicita uma transmissão broadcast. Os parâmetros são:
  1. dados a serem transmitidos
  2. endereço MAC do emissor.

- **void Transmite_Byte(char c, char *addr, int porta)**
  Realiza a transmissão de um byte para máquina destino. Os parâmetros são, respectivamente, o byte a ser transmitido e o endereço IP e porta (combinada ser a mesma para todos os hosts e comutador) da máquina destino.

- **int Data_Receive (unsigned char *addr, unsigned char *dest_addr, char *data, int size)**
  Busca os dados do último quadro recebido. Os dois primeiros parâmetros são ponteiros para as variáveis que irão conter, respectivamente, o endereço MAC do destino e os dados do quadro recebido. O último parâmetro é o tamanho máximo esperado para o campo de dados do quadro. Retorna o número de bytes do campo de dados ou -1 em caso de falha.

- **void Recebe_Byte(void)**
  Recebe um byte e armazena num buffer.

- **char P_Data_Receive(void)**
  Recebe um byte e o retorna.

- **int Data_Indication (void)**
  Testa se há um quadro recebido. Retorna 1 caso exista e 0 caso contrário.

- **int P_Data_Indication(void)**
  Testa se há um byte recebido. Retorna 1 caso exista e 0 caso contrário.

- **int main(int argc, char *argv[])**
  Ativa o comutador e faz a transmissão e recebimento de quadros entre os hosts.

Implementação (emulação) da camada de Enlace

Estratégias de implementações adotadas

Os quadros são definidos como vetores de caracteres, organizado da seguinte forma:

1. os três primeiros caracteres definem os algarismos do endereço MAC destino.
2. os três caracteres seguintes são os algarismos do endereço MAC local.
3. três caracteres para os algarismos do tamanho do quadro, que determinamos ser 999 (especificamos TAM_FRAME_MAX como 1000 devido ao terminador \0 de strings em C); size caracteres para os dados.
4. Os três últimos caracteres para o código de detecção de erro.

O código de detecção de erro é avaliado realizando a soma dos caracteres da string que representa um quadro, seguida da operação de mod 1000 para que haja, no máximo, três algarismos confirmando a integridade do vetor.

Para recebimento de quadros, são utilizados dois buffers; um para armazenar os caracteres do quadro a ser recebido, que é um vetor de caracteres, e outro para armazenar o

quadro inteiro, que é uma string. A detecção de que o quadro foi inteiramente recebido é feita por simples contagem de bytes a partir da definição dada para a estrutura de um quadro.

Após o quadro ser recebido, este passa pela validação, que calcula o código de detecção de erro e o verifica se é igual ao último campo do quadro, em seguida verifica se um número aleatório gerado é menor do que a taxa de erro. A validação aprova o quadro caso seja aprovado em ambos os testes. Caso seja inválido ele é retirado do buffer de bytes e não é armazenado no buffer de quadro.

Para a transmissão de quadros, há também dois buffers; um para armazenar os quadros a serem transmitidos (a taxa de requisição para transmissão de quadros pode ser maior do que a taxa de transmissão de um quadro) e o outro para armazenar o quadro cujos bytes estão sendo transmitidos. O primeiro é uma estrutura queue nativa do C++ e o último, uma string. Os caracteres transmitidos vão sendo apagados da string.

O funcionamento do programa do comutador é semelhante a enlace.cpp para recebimento e transmissão de quadros.

Em comutador.cpp, fazemos uso de duas tabelas: uma para armazenar os endereços IP dos hosts que são conectados ao comutador e outra para representar a tabela de comutação.

A tabela de hosts conectados é preenchida através do recebimento de uma mensagem especial configurada pela string “hi” destinada ao endereço MAC virtual 0 uma vez que este não poderá ser usado por nenhum host.

A tabela de comutação da emulação funciona da mesma forma que uma tabela de real, a menos da existência do TTL e da capacidade de tratamento paralelo da comunicação entre os hosts. Ou seja, um destino não presente na tabela de comutação implica broadcast, e o endereço MAC fonte é armazenado na tabela de comutação.

Estruturas do programa de teste e resultados obtidos

O programa teste (enlace-teste.cpp) é executado nas máquinas que simularão os hosts e a máquina que simular o comutador deve executar comutador.cpp.

O programa enlace-teste.cpp é executado na seguinte sequência:

1. L_Activate_Request() é chamada para inicializar a camada de enlace.
2. É passada a mensagem especial para conexão com o comutador.
3. A taxa de erro é estabelecida.
4. É oferecida a opção de se enviar uma mensagem
5. Em seguida, dentro de um laço, no máximo MAX_OPS operações (recebimento ou transmissão) são realizadas. No laço, é verificado pela L_Data_Indication() se há quadro disponível para recebimento, e, caso haja, este recebido pelo uso de L_Data_Receive().
6. Em seguida, é verificado se há algo digitado no teclado, com uso de select() com time_out de 1 milissegundo; caso haja, a mensagem é transmitida, com uso de L_Data_Request().
7. Por último, L_Main_Loop() é chamada para um possível recebimento ou transmissão de byte. Os resultados obtidos: quando há recebimento de mensagem destinada ao host, na

tela aparece: “Recebi <mensagem> do host <número do host>”; quando o recebimento não é destinado ao host ou o quadro tem erro, na tela aparece: "Quadro invalido recebido: <motivo>", onde <motivo> pode ser “probabilidade”, “código de erro” ou “endereço”.

Dificuldades encontradas

Uma dificuldade encontrada foi implementar a emulação da camada de enlace sem ter conhecimento das características da camada física. Isso tornou necessário o estudo da camada física para adequar a implementação de uma camada física inexistente. Ou seja, realizar a implementação do enlace nos moldes das estruturas de dados e constantes sem a implementação da camada física.

Outra dificuldade foi descobrir o fato de certas strings enviadas serem perdidas, não sendo recebidas. Isto decorreu devido ao requisito do sistema não ser bloqueante. Como resolução do problema foi feito o uso do sinal SIGIO que, quando disparado, uma entrada estivesse disponível, permitia o uso do programa sem bloqueio.

O ocorrido é que, dentro do programa do comutador, l_Transmite_Byte() é chamada várias vezes dentro de um laço for. Antes, tínhamos somente l_Transmite_Byte() dentro do laço. Desta forma, não havia tempo dos receptores estarem prontos para o recebimento de um byte após o outro. Isso se deve ao fato que a execução de um laço for é muito mais rápido do que uma operação de E/S e, portanto, não havia tempo de o sinal SIGIO ser disparado como planejado. A solução dada foi chamar deixar a execução propositalmente mais lenta com o uso da função usleep() após cada transmissão de um byte de modo a permitir que haja tempo para a operação de E/S.

Conclusões

O trabalho de emulação cumpriu a sua tarefa de emular a dinâmica da camada de enlace. Foi possível compreender aspectos de dependência das camadas todos os processos relacionados. Com relação ao uso, o trabalho foi testado com seis máquinas, uma delas executando o programa do comutador, cada host foi capaz de enviar e receber strings de tamanhos variados diretamente para outro host ou fazendo broadcast. Desta forma, podemos considerar que a emulação apresentou o resultado esperado.

Referências Bibliográficas

- J. F. Kurose, K. W. Ross, Computer Networking: a Top-Down Approach, Addison-Wesley, 5a edição, 2010
- Linux Man Pages. Disponível em: <http://linux.die.net/man/>.
- Linux Man Pages. Disponível em: <http://www.kernel.org/doc/man-pages/>