Protocolo de Ligação de Dados

Relatório

Universidade do Porto

Faculdade de Engenharia

FEUP

Redes de Computadores

3^o ano

Mestrado Integrado em Engenharia Informática e Computação

Turma 4

Carolina Moreira	201303494	up201303494@fe.up.pt
Daniel Fazeres	201502846	up201502846@fe.up.pt
José Peixoto	200603103	ei12134@fe.up.pt

4 de Novembro de 2016

Resumo

(dois parágrafos: um sobre o contexto do trabalho; outro sobre as principais conclusões do relatório)

1 Introdução

2 Arquitetura

3 Estrutura do código

4 Casos de uso principais

5 Protocolo de ligação lógica

6 Protocolo de aplicação

A camada de aplicação é responsável pela leitura/escrita dos dados do ficheiro a enviar/receber. Do lado do emissor, procede-se à segmentação do ficheiro em pacotes de dados que vão sendo numerados e enviados para a camada de ligação de dados, por forma a serem encaixados em tramas de informação e posteriormente enviados através da porta de série. Do lado do receptor, é feita a compilação e escritura dos dados recebidos num ficheiro em disco nomeado de acordo com a informação recebida nos pacotes de controlo. Quer no emissor quer no receptor, recorre-se à codificação das etapas sob a forma de máquinas de estado.

6.1 Envio de um ficheiro

A camada de aplicação pode interpretar os argumentos opcionais passados através da interface de linha de comandos para ler do disco um ficheiro para uma estrutura de dados que armazena os dados, o nome e o tamanho do ficheiro. Opcionalmente, só os dados de um ficheiro serão lidos do `stdin` para a estrutura de dados referida e será atribuido um nome de ficheiro predefinido.

```
struct file {
    const char* name;
    size_t size;
    char* data;
};
```

Após a leitura do ficheiro, a camada de aplicação entra numa máquina de estados com quatro estados ordenados: abertura de ligação, envio de pacote de controlo inicial, envio de pacotes de dados, envio de pacote de controlo final e fecho da ligação.

Método usado na abertura de ligação

```
int llopen(char *port, int transmitter);
```

Método usado para o envio de pacotes de controlo inicial e final

```
int send_control_packet(struct connection* connection, struct file *file,
        byte control_field);
```

Método usado para o envio de pacote de dados

```
int send_data_packets(struct connection* connection, struct file* file,
        size_t* num_data_bytes_sent, size_t* sequence_number);
```

Método usado para o envio dos pacotes para a camada de ligação de dados

```
int transmitter_write(struct connection* conn, byte* data_packet, size_t size);
```

Método usado no fecho da ligação

```
int llclose(const int fd);
```

6.2 Receção de um ficheiro

7 Validação

8 Elementos de valorização

8.1 Selecção de parâmetros pelo utilizador

Quando o programa é invocado pela linha de comandos de forma errónea ou sem quaisquer parâmetros adicionais, são mostrados os argumentos opcionais disponíveis que permitem configurar a execução do programa, nomeadamente o modo de operação (`receiver` ou `transmitter`), leitura do ficheiro a enviar do disco ou proveniente de dados redirecionados do `stdin`, selecção da baudrate, determinação do tamanho máximo de bytes de dados enviados em cada frame e do número de tentativas na recuperação de erros.

8.2 Implementação de REJ

Quando ocorrem erros no processamento de tramas recebidas na camada de ligação de dados, é enviada de forma preemptiva ao `timeout` uma trama com confimação negativa (`REJ`) que permite a retransmissão da trama de informação.

```
else if (ret == BADFRAME_CODE) {
    /*
    * Send 'bad frame' acknowledgment.
    */
    byte c_out = data_reply_byte(conn->frame_number, FALSE);
    if (f_send_frame(conn->fd, FRAME(c_out)) != SUCCESS_CODE)
        break;
}
```

8.3 Verificação da integridade dos dados pela Aplicação

Após receção com sucesso do primeiro pacote de controlo pela camada de aplicação, é armazenado o tamanho expectável em bytes do ficheiro a receber, comparando-o no fim da sessão com o valor real de bytes dados recebidos. São também guardados os números de pacotes de dados perdidos e duplicados.

```
void receiver_stats()
{
    fprintf(stdout, "Receiver statistics\n");
    fprintf(stdout, "\treceived file bytes/file bytes:%zu/%zu\n",
            received_file_bytes, real_file_bytes);
    fprintf(stdout, "\tlost packets:%zu\n", lost_packets);
```

```
    fprintf(stdout, "\tduplicated packets:%zu\n", duplicated_packets);
}
```

9 Conclusões

Referências

[1] Andrew S. Tanenbaum, David J. Wetherall, *Computer Networks*, Prentice Hall, 5th edition, 2011.