Departamento de Ciência da Computação **João Paulo Bastos**
DCC023 - Redes de Computadores Matricula: 2013073440
Prof. Dorgival Neto 15/05/2017
TP1 - DCCNET

Introdução

Esse documento explica a implementação do protocolo de rede DCCNET, pertencente a camada de enlace de dados. A figura 1 trás a definição dos campos de um pacote DCCNET.

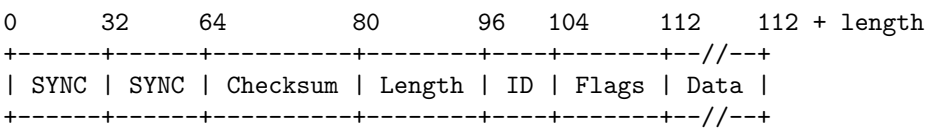

Figura 1: Definição do cabeçalho DCCNET

Onde SYNC é um string de 4 bytes com a representação hexadecimal de DC C0 23 C2, respectivamente.

O campo Flags possui comprimento de 1 byte, onde os 5 primeiros bits são reservados e os outros três bits seguem a seguinte lógica:

Flag	Máscara	Descrição
ACK	0x80	Confirmação de pacote de dados
END	0x40	Último pacote da transmissão
INVALID	0x3f	Bits reservados

Tabela 1: Tabela de Flags

Descrição da solução

A solução de baseia em uma abordagem na qual cada dispositivo que executa o programa age como emitente e receptor de dados, simultaneamente. Para isso é simulada a utilização de duas threads na forma de um pool de entidades que agem sobre os pacotes que percorrem a rede.

Para iniciar o fluxo de execução o programa deve ser executado em dois dispositivos, um fazendo a abertura passiva e o outro fazendo a abertura ativa conforme demonstram os comandos na figura 2.

```
# Abertura passiva
$./dcc023c2.py -s <porta> <input> <output>

# Abertura ativa
$./dcc023c2.py -c <ip_lado_passivo>:<porta_lado_passivo> <input> <output>
```

Figura 2: Comandos para iniciar a aplicação

Estrutura de dados

Para elucidar melhor a "simulacão de threads"apontada anteriormente, tenha em vista que foram criadas duas entidades do tipo dictionary com as respectivas responsabilidades de enviar e receber dados, respectivamente.

Segue da figura 3 que para o lado recebedor de dados interessa saber se um pacote com a flag END foi recebido - indicando que o outro lado não possui mais dados para enviar - e quais foram os últimos pacotes enviado e recebido. Sobre o último pacote recebido, só se guarda o registro de pacotes contendo dados; os pacotes de confirmação (ACK) não são arquivados.

```
recv_thread = {
    "recv_no_more": False, # Has a packet with flag END been received?
    "last_sent": {"checksum": None, "length": None, "id": None, "flags": None, "data": None, "packet": None},
    "last_recv": {"checksum": None, "length": None, "id": 1, "flags": None, "data": None, "packet": None},
}

send_thread = {
    "send_no_more": False, # Is there any more data in my input file?
    "last_sent": {"checksum": None, "length": None, "id": 0, "flags": None, "data": None, "packet": None},
    "last_recv": {"checksum": None, "length": None, "id": None, "flags": None, "data": None, "packet": None},
    "waiting_ack": False  # Did I just sent a data packet?
    }
}
```

Figura 3: Definição da estrutura de dados

Segue também que para o lado emitente interessa saber se ainda há dados no arquivo de entrada, manter o registro sobre o último pacote enviado e o último pacote recebido, bem como saber se um pacote de confirmação é aguardado ou não.

Fluxo de execução

Após iniciar a aplicação nos dois dispositivos um socket é instanciado em cada lado e ambos lêem um número de bytes menor ou igual a 2 ^ 16 - 1 dos respectivos arquivos de entrada, formando dois buffers de leitura. A partir de cada buffer, cada lado monta um pacote DCCNET seguindo as especificações apontadas em , calcula uma soma de verificação (checksum) sobre os dados do pacote e o envia para o outro lado.

Uma vez enviado o pacote inicial, o comportamento a seguir se repete até que seja detectado um pacote com a flag END ligada - indicando que o outro lado não tem mais nada a transmitir - e o mesmo seja validado por meio de um ACK. Além disso, é preciso também verificar que o lado que recebeu o pacote com a flag END não tem mais nada a enviar. Satisfeitas as condições mencionadas, a execução do programa termina.

O fluxo de execução se resume a receber bytes e reconhecer, através das flags de sincronização (SYNC) descritas em 1, se é possível formar um quadro de formato válido com eles. Uma vez recuperado o quadro, seu conteúdo é então decodificado e validado a partir da soma de verificação (checksum) e, caso o pacote seja dado como válido, o mesmo é por fim processado de acordo com seu tipo: dados ou confirmação. Caso seja inválido, o pacote é descartado e repete-se o processo descrito nesse parágrafo.

Quando o pacote é do tipo dados, a entidade recv_thread assume seu processamento e confere se o ID presente no cabeçalho do pacote é diferente do último ID recebido e armazenado na variável last_recv. Em caso positivo, é gerado então um quadro de confirmação dos dados e este é enviado de volta a quem enviou o quadro de dados. Em caso negativo, o pacote é descartado.

Quando o pacote é do tipo confirmação, a entidade send_thread assume seu processamento e verifica primeiro se um quadro de confirmação está sendo aguardado. Em caso positivo compara-se o ID presente no quadro com o último quadro enviado, presente na variável last_sent. Caso os IDs sejam iguais, a confirmação foi bem-sucedida e o processo de ler dados e enviar o pacote - descrito no início dessa seção - se repete. Caso os IDs sejam diferentes, o quadro em questão é a confirmação de um quadro de dados enviado em um momento anterior e que possivelmente já fora confirmado; dessa forma, descarta-se esse pacote. O mesmo comportamento se repete em caso de um quadro de confirmação não estar sendo aguardado mas mesmo assim o ID presente neste bater com o ID do último quadro enviado, bem como o valor do checksum. Nesse último caso, porém, uma ação adicional é executada: reenviar o último pacote de dados.

Problemas encontrados

O único problema enfrentado durante o desenvolvimento deste trabalho foi o fato de ter-se iniciado seu desenvolvimento no ambiente Windows - para o qual a interface de sockets da linguagem Python é uma espécie de adaptação ainda muito carente de estabilidade. Em decorrência disso, ao tentar executar um lado da aplicação localmente os pacotes enviados desapareciam e não geravam logs, warnings ou qualquer outro recurso que pudesse permitir rastreá-los. Tal problema foi contornado ao mudar o ambiente de desenvolvimento para a distribuição Arch do sistema operacional Linux.

Referências

[1] PETERSON, Larry; DAVIE, Bruce 2011. *Computer Networks - 5th Edition - A Systems Approach.* Elsevier

[2] MCMILLAN, Gordon: Socket Programming HOWTO. *Disponível em* `https://docs.python.org/2/howto/sockets.html` *Acesso em 14-05-2017.*