P2: Simple crosslayer protocol

Ad Nunes Ribeiro (11200620), Eduardo Dias Defreyn (11203780),
Ricardo do Nascimento Boing (14200760)
Bacharelado em Ciência da Computação -- Universidade Federal de Santa Catarina (UFSC)
Campus Reitor João David Ferreira Lima, 88.040-900 -- Florianópolis -- SC -- Brasil

adnunesribeiro@gmail.com; eduardo_dududex@hotmail.com; ricardoboing.ufsc@gmail.com

1. Simple crosslayer protocol

O grupo desenvolveu e implementou um protocolo sem fragmentação. Na Fig. 1 é apresentada uma representação do protocolo. Os dois primeiros campos correspondem a porta de quem está enviando e recebendo a mensagem, respectivamente. Já o campo seguinte corresponde a um byte de identificação, para sinalizar se a mensagem é um ACK ou uma mensagem normal. Já os três últimos campos correspondem ao identificador, tamanho e conteúdo da mensagem, sendo que no caso de um ACK o tamanho é zero e o conteúdo é nulo.

PORTA		TIPO	MENSAGEM		
ORIGEM	DESTINO	ACK/MSG	ID	TAM.	CONTEÚDO
0 - 3	4 - 7	8	9 - 12	13 - 16	17 - MTU

Figura 1. Representação do protocolo desenvolvido pelo grupo SOS.

2. Classe SOS e SOS_Communicator

Os três primeiros campos do protocolo são implementados na classe SOS. A função dessa classe é receber um update da NIC e tratá-lo. O tratamento ocorre a partir da verificação de para qual porta essa mensagem é destinada. De acordo com essa porta, e com uma lista de observadores (SOS_Communicator), a classe realiza um notify para esses observadores. Em caso de haver observadores, o SOS envia um ACK para o endereço/porta de quem enviou a mensagem.

Já a classe SOS_Communicator é responsável por receber uma mensagem, para uma dada porta, e encaminhar a mensagem para a aplicação. Inicialmente, no construtor do SOS_Communicator, é realizado um attach na classe SOS para receber mensagens para uma dada porta (definida pelo usuário e passada como parâmetro pelo construtor).

Para receber uma mensagem, a aplicação chama o método receive, da classe SOS_Communicator, ficando bloqueada em um semáforo até o recebimento de um update (que da um "p" no semáforo). Já para enviar uma mensagem, o método send da classe SOS_Communicator deve ser chamado, o qual enviará a mensagem, criará uma Thread para

o TIMEOUT, bloqueará a Thread do usuário em um semáforo, e a Thread do usuário ficará a espera de um ACK ou do TIMEOUT.

2.2. Time-out e retransmissão de pacotes

No método send, da classe SOS_Communicator, uma Thread é criada para verificar o TIMEOUT. A Thread do usuário, que criou a Thread para o TIMEOUT, ficará bloqueada até receber uma mensagem de ACK ou der TIMEOUT. A Thread do TIMEOUT é criada recebendo como parâmetros um ponteiro para verificar se um ACK foi recebido e outro ponteiro para settar o TIMEOUT, caso ocorra. Em caso de um ACK for recebido antes de um TIMEOUT for completado, a Thread deixa de fazer a verificação para que seja finalizada.

2.3. Confirmação de recebimento (ACK)

O ACK é enviado pela classe SOS, em caso de haver um observador esperando pela mensagem recebida. Em caso de uma mensagem ser recebida e ninguém estiver esperando por ela, não será feito o envio do ACK.

3. Traits

Um Traits<SOS> foi criado para definição dos RETRIES e TIMEOUT.

```
template<> struct Traits<SOS>: public Traits<void>
{
    static const bool enabled = (Traits<Build>::NODES > 1);

    static const unsigned int RETRIES = 20;
    static const unsigned int TIMEOUT = 5; // s
};
template<> struct Traits<TSTP>: public Traits<Network>
```

4. Testes

Os testes estão contidos na pasta tests_sos.

4.1. Teste ack_received

Esse teste demonstra o envio de uma mensagem e o recebimento de um ACK. Também demonstra a utilização correta de uma porta e o endereçamento.

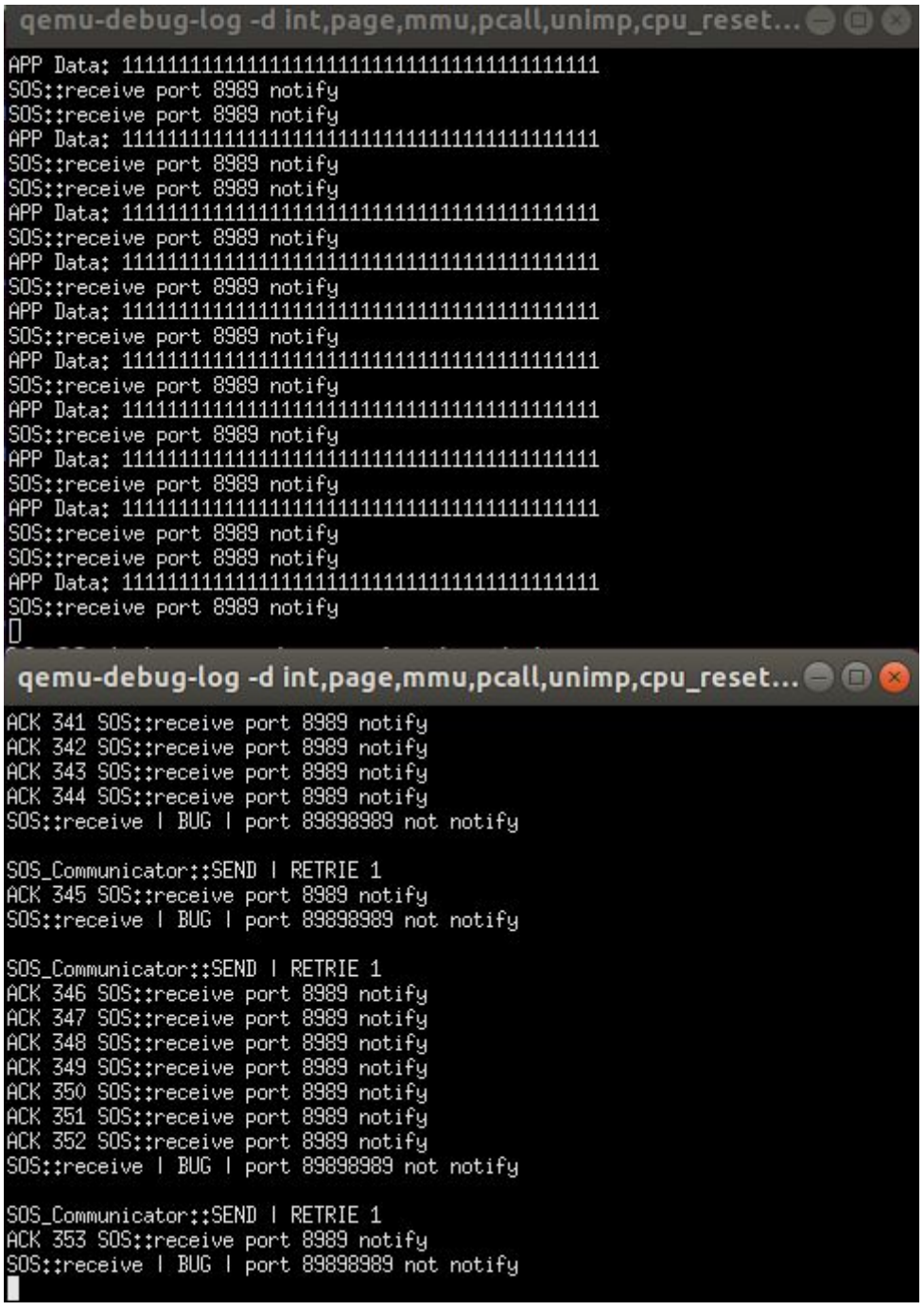

Figura 2. Resultado da execução do teste ack_received.

4.2. Teste ack_not_received_port

Esse teste demonstra a utilização incorreta de uma porta. Ou seja, é feito o envio de uma mensagem para uma porta que não está esperando por mensagens.

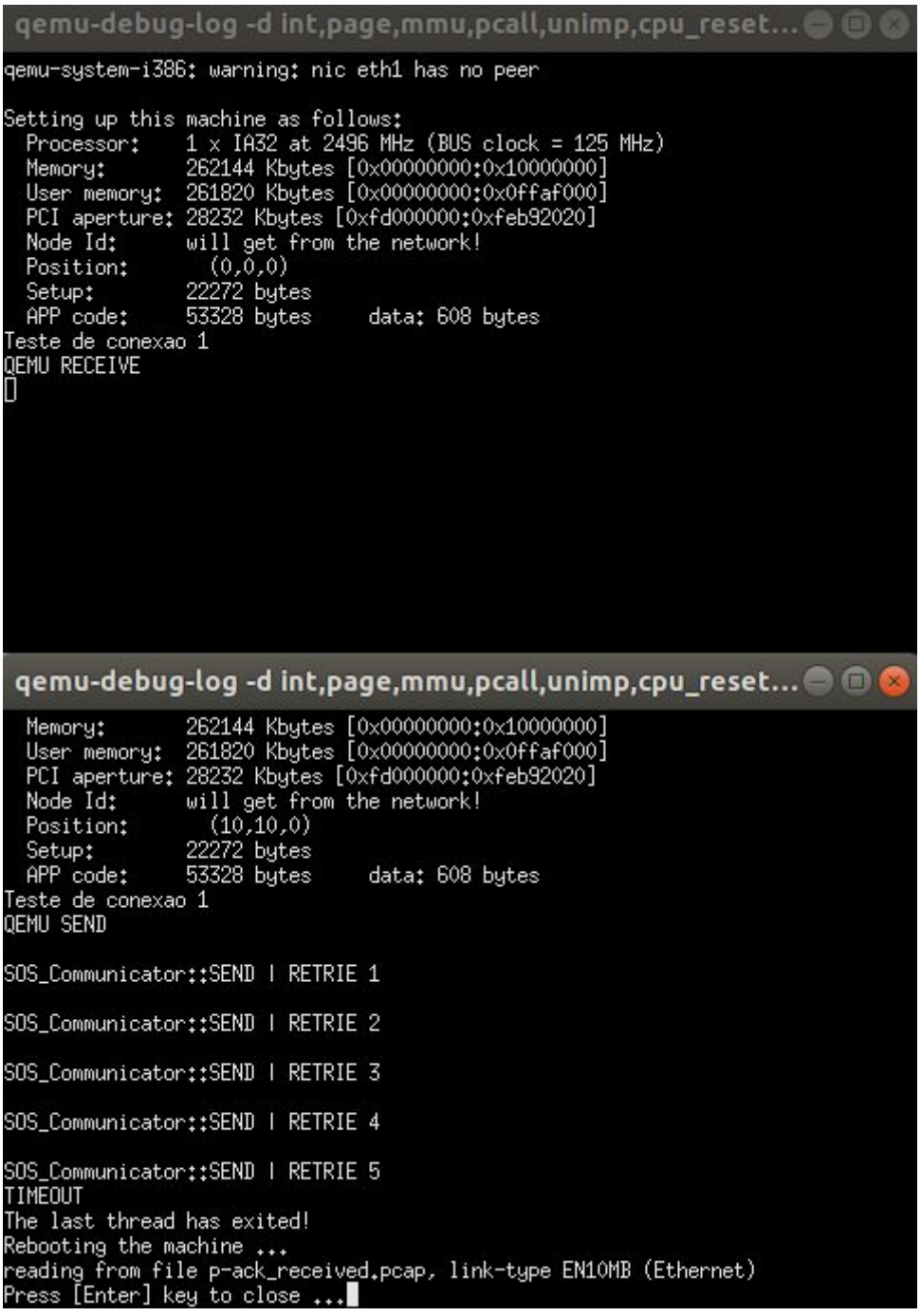

Figura 3. Resultado do teste ack_not_received_port.

4.3. Teste ack_not_received_endereco

Esse teste demonstra a utilização incorreta de um endereço. Ou seja, é feito o envio de uma mensagem para um endereço que não está esperando por mensagens.

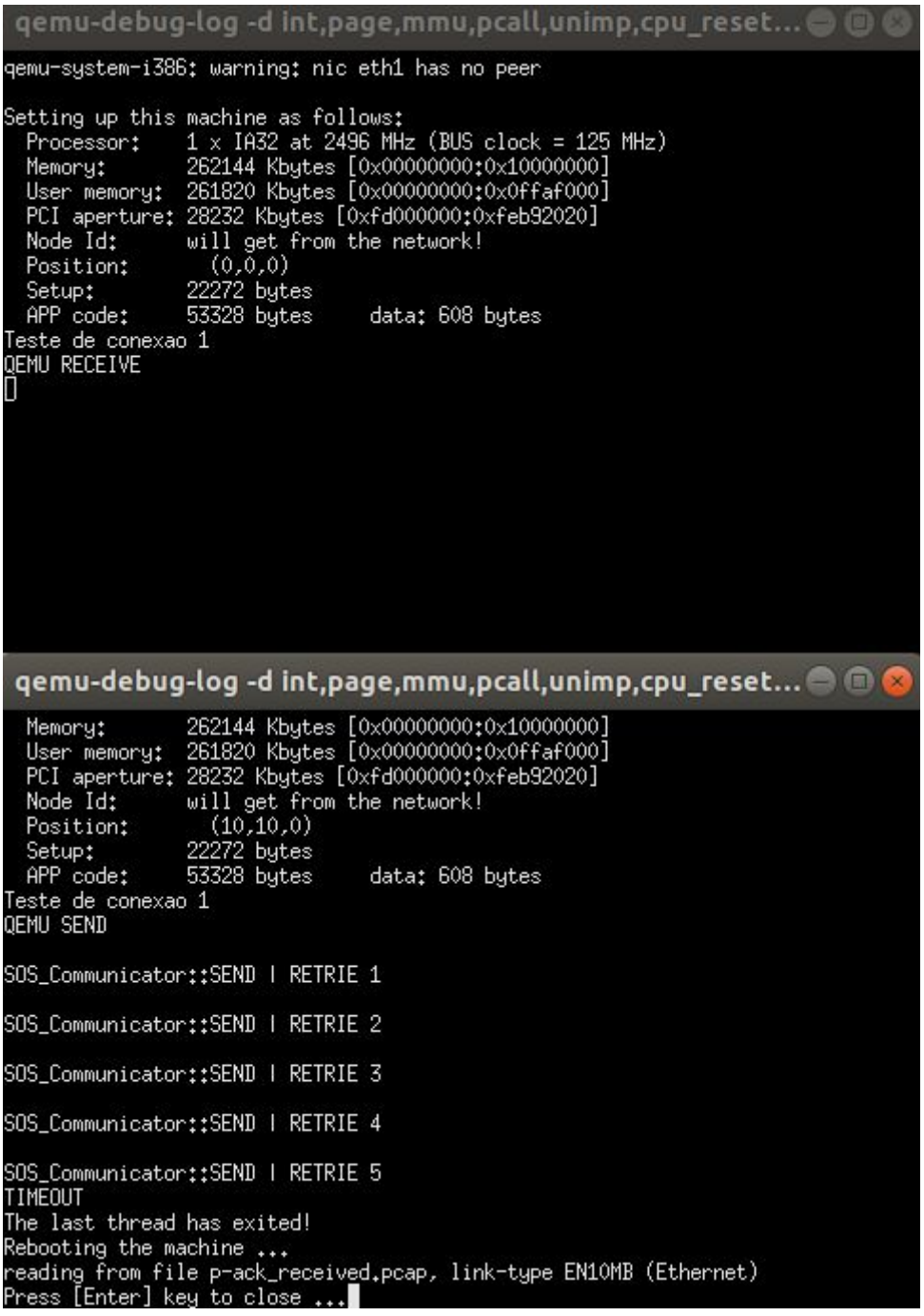

Figura 4. Resultado do teste ack_not_received_endereco.

5. Bug

Existe um bug na quebra do pacote em algumas regiões do código, ocasionado pelo tipo char. As Fig. 5 e Fig. 6 mostram um trecho do código, do método SOS::receive (arquivo ip.cc), onde os 4 bytes utilizados para identificação de uma das portas leva a obtenção de bytes a mais. Esse problema leva ao não recebimento do pacote pela aplicação, já que não é realizado um notify corretamente. O teste da Fig. 2 mostra o resultado desse bug.

```
int SOS::receive(char data[])
{

    NIC_Address* src = new NIC_Address("00");

    char pacote[mtu()+header];
    nic_receive(src, pacote, mtu()+header);

    // Copia mensagem
    for (unsigned int c = 0; c < mtu(); c++) {
        data[c] = pacote[header +c];
    }

    // Porta do receiver
    char port_char[4];
    port_char[0] = pacote[4];
    port_char[1] = pacote[5];
    port_char[2] = pacote[6];
    port_char[3] = pacote[7];

    unsigned int port = atol(port_char);

    char pacote_ack[header];

    OStream cout;
    char* t = data;
    if (notify(port, &t)) {
```

Figura 5.

```
} else {
    cout << "SOS::receive | BUG | port " << port_char << " not notify " << endl;
}
```

Figura 6.