UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
CEIOT INC14 - LABORATÓRIO DE REDE DE SENSORES SEM FIO I
ATIVIDADE DE LABORATÓRIO

Professor: Guilherme Luiz Moritz e Ohara Kerusauskas Rayel
Tema: Comunicação UDP entre nós Contiki
Data: 31 de março de 2022

Atividade - Comunicação UDP entre nós Contiki

Os protocolos da camada de transporte (UDP e TCP) têm a seguinte função em uma rede IP:

- Ocultar a rede física;
- Ocultar a complexidade de processamento de pacotes;
- Abstrair diferentes tecnologias e arquiteturas de rede;
- Prover transporte de dados entre nós.

Um protocolo de transporte deve resolver os seguintes problemas:

- Endereçamento (múltiplos serviços no mesmo nó);
- Estabelecimento de conexão;
- Encerramento de conexão;
- Controle de Fluxo;
- Controle de Erro.

Em redes IP, estes problemas são resolvidos com dois protocolos distintos, que são desenvolvidos com propósitos diferentes. Suas características serão enumeradas nas seções subsequentes.

1 TCP

A camada IP que é implementada logo abaixo da camada TCP provê troca de quadros de mensagens de maneira não confiável. Esta troca de mensagem não garante a entrega do quadro e não provê conexão. Sua função básica é prover roteamento de quadros, quando um problema ocorre, o quadro é descartado sem retransmissão. A tarefa de verificação de estado e retransmissão dos quadros IP é de responsabilidade do protocolo TCP.

Desta maneira, o TCP é um protocolo orientado a conexão que garante à aplicação a entrega de um fluxo de dados ordenado e sem erros. É função do TCP dividir o fluxo em quadros, retransmitir os quadros perdidos e reordenar os quadros recebidos fora de ordem.

2 UDP

Já o UDP (User Datagram Protocol) é um protocolo da camada de transporte que não é orientado a conexão como no caso do TCP. Ele é recomendado para aplicações que dão prioridade a velocidade e eficiência, ao invés da confiabilidade. Nestes casos, um protocolo que não é orientado a conexão pode ser usado, assim não existe a necessidade de se estabelecer uma conexão antes de se enviar um pacote, simplesmente se envia o pacote.

Depois do UDP ter enviado o pacote para a rede (via protocolo IP), ele esquece do pacote. Portanto, o UDP não garante que o pacote chega no destino. A maioria das aplicações que usa UDP simplesmente espera por qualquer resposta resultante de pacotes que enviou por UDP. Se uma resposta esperada não chega em um certo período de tempo, a aplicação ou envia o pacote novamente, ou simplesmente desiste.

O UDP utiliza um modelo de transmissão simples, sem diálogos de *handshake* implícitos para prover confiabilidade, ordenação ou integridade dos dados. Resumindo: UDP provê um serviço não-confiável e seus pacotes podem chegar fora de ordem, duplicados ou simplesmente não chegar.

Para redes de sensores sem fio o protocolo TCP não é desejável, já que por ser orientado a conexão obrigaria que os nós da rede ficassem "acordados" aguardando conexão, o que não é compatível com as características de baixo consumo desejáveis nesse tipo de rede. Além disso, os mecanismos implementados pelo TCP para gerar confiabilidade utilizam mais recursos de processamento, o que também não é desejável em redes de sensores.

3 Envios de Pacotes UDP

Nesta atividade será desenvolvido um *firmware* que utiliza o protocolo UDP para alterar o estado dos leds do kit da disciplina. Para maiores informações sobre envio e recebimento de datagramas UDP veja [**?** , Capítulo 4.4] (até 4.4.3). Então, siga as intruções abaixo:

1. Caso ainda não o fez, sincronize o seu fork com o fork de upstream (utfprwsn) conforme procedimento da aula sobre Git;
2. Utilize como base o arquivo `examples/udp-ipv6/udp-client-atividade4.c`
3. Defina as seguintes macros no *firmware* do nó:

```
#define LED_TOGGLE_REQUEST (0x79)
#define LED_SET_STATE (0x7A)
#define LED_GET_STATE (0x7B)
#define LED_STATE (0x7C)
```

4. Observe a linha que cria o endereço IPv6 do servidor no código e a altere de acordo:

```
uip_ip6addr(&ipaddr, 0xfe80, 0, 0, 0, 0x0212, 0x4b00, 0x07b9, 0x5e8d);
```

5. Observe as linhas que configuram a conexão UDP com o servidor, configure as portas de acordo.

```
/* new connection with remote host */
client_conn = udp_new(&ipaddr, UIP_HTONS(CONN_PORT), NULL);
udp_bind(client_conn, UIP_HTONS(CONN_PORT));
```

6. Observe que há um timer configurado para chamar a função `timeout_handler()` a cada 15 segundos. Altere o conteúdo desta função para que ela envie 1 byte de conteúdo `LED_TOGGLE_REQUEST` para o IP do servidor a cada 5 segundos. Utilize a função `uip_udp_packet_send`, conforme exemplo:

```
uip_udp_packet_send(client_conn, buf, sizeof(payload));
```

7. Observe que a função não deve tentar enviar pacotes UDP antes que o nó receba um IP global do DAG Root. Utilize o seguinte trecho de código para verificar o fato:

```
if(uip_ds6_get_global(ADDR_PREFERRED) == NULL) {
      PRINTF("Aguardando auto-configuracao de IP\n");
      return;
}
```

8. Imprima uma mensagem indicando o IP e a porta de destino, utilizando o seguinte trecho de código:

```
PRINTF("Cliente para [");
PRINT6ADDR(&client_conn->ripaddr);
PRINTF("]:%u", UIP_HTONS(client_conn->rport));
```

9. Observe que o processo `udp_client_process` executa a função `tcp_ip_handler` toda vez que ela recebe um evento do tipo `tcpip_event`. Altere esta função para que a mesma verifique os pacotes recebidos e ao receber um pacote de 1 byte contendo a macro `LED_GET_STATE` responda com um pacote UDP com o conteúdo conforme a tabela:

byte	0	1
Valor	`LED_STATE`	Macro de indicação de estado dos leds

Dica - Função de eco:

```
static void
tcpip_handler(void)
{
       char i=0;
       #define SEND_ECHO (0xBA)
       if(uip_newdata()) //verifica se novos dados foram recebidos
       {
           char* dados = ((char*)uip_appdata); //este buffer eh padrao kdo Contiki
              PRINTF("Recebidos %d bytes\n", uip_datalen());
              switch (dados[0])
              {
              case SEND_ECHO:
              {
                     uip_ipaddr_copy(&client_conn->ripaddr, &UIP_IP_BUF->srcipaddr);
                     client_conn->rport = UIP_UDP_BUF->destport;
                     uip_udp_packet_send(client_conn, dados, uip_datalen());
                     PRINTF("Enviando eco para [");
                     PRINT6ADDR(&client_conn->ripaddr);
                     PRINTF("]:%u\n", UIP_HTONS(client_conn->rport));
                     break;
              }
              default:
              {
                     PRINTF("Comando Invalido: ");
                     for(i=0;i<uip_datalen();i++)
                     {
                            PRINTF("0x%02X ",dados[i]);
                     }
                     PRINTF("\n");
                     break;
              }
              }
       }
       return;
}
```

10. Observe as macros `#define UIP_IP_BUF` e `#define UIP_UDP_BUF` definidas no início do código. Elas criam um ponteiro para início do datagrama IP ou UDP contido no datagrama completo recebido pela biblioteca μIP;

11. Altere a função `tcp_ip_handler` para que ela altere o estado dos leds do kit a partir do recebimento do pacote UDP especificado na tabela. O conteúdo da Macro de indicação deve ser equivalente às macros utilizadas na API do led. Quando o comando for bem sucedido, enviar ao nó requisitante um pacote conforme Item 9

byte	0	1
Valor	`LED_SET_STATE`	Macro de indicação de led

4 mDNS

1. Caso não se deseje adicionar e alterar o IP dos nós manualmente cada vez que eles se alteram, é possível utilizar mDNS.

2. O primeiro passo é ativar ativar o protocolo mDNS. Para isto, é preciso ativar as macros `RESOLV_CONF_SUPPORTS_MDNS` e `RESOLV_CONF_MDNS_INCLUDE_GLOBAL_V6_ADDRS` no arquivo /core/net/ip/resolv.c

3. No processo udp_client_process, observe a configuração da comunicação UDP:

```
static resolv_status_t status = RESOLV_STATUS_UNCACHED;
while(status != RESOLV_STATUS_CACHED) {
       status = set_connection_address(&ipaddr);

       if(status == RESOLV_STATUS_RESOLVING) {
       PROCESS_WAIT_EVENT_UNTIL(ev == resolv_event_found);
       } else if(status != RESOLV_STATUS_CACHED) {
       PRINTF("Can't get connection address.\n");
       PROCESS_YIELD();
       }
```

```
}
/* new connection with remote host */
client_conn = udp_new(&ipaddr, UIP_HTONS(CONN_PORT), NULL);
udp_bind(client_conn, UIP_HTONS(CONN_PORT));
```

A função `set_connection_address` procura resolver o IPv6 do nó `contiki-udp-server .local` (observe que você pode alterar o nome a ser resolvido!), retorna se obteve sucesso ou não e altera uma estrutura de endereçamento IP (resolvida de acordo com o mDNS do destino) que foi passada por referência. A função `udp_new` cria uma estrutura de envio de pacotes UDP para um par IP/Porta específico. Já a função `udp_bind` registra o processo para receber um evento sempre que um pacote UDP é recebido de um endereço configurado pela estrutura `client_conn`;

4. Neste ponto, o resto da atividade executa inalterada do exemplo da Seção 3.

5 Envios de Pacotes UDP complexos

1. Na Seção 3, somente pacotes com campos de 1 byte foram enviados. Nesta seção utilizaremos pacotes complexos;

2. Em adição às macros definidas no Item 3 da Seção 3, crie as seguintes macros:

```
#define OP_REQUEST (0x6E)
#define OP_RESULT (0x6F)
```

3. Escreva código que, ao pressionamento do botão do kit, envie o seguinte pacote ao servidor. Dica: Declare duas structs para enviar e receber os pacotes. Se inspire no código fonte do servidor da atividade: examples/udp-ipv6/udp-server-atividadeAdvanced.c e examples/udp-ipv6/protocol.h

byte	0	1-4	5	6-9	10-13
Valor	`OP_REQUEST`	OP1	Operação	OP2	Fc
Tipo	`uint8_t`	`int32_t`	`uint8_t`	`int32_t`	`float`

4. O campo operação deve assumir um dos sequintes valores:

```
#define OP_MULTIPLY (0x22)
#define OP_DIVIDE (0x23)
#define OP_SUM (0x24)
#define OP_SUBTRACT (0x25)
```

5. Ao recebimento do pacote recebido, o servidor enviará o seguinte retorno:

byte	0	1-4	5-8	9-12	13
Valor	`OP_RESULT`	IntPart	FracPart	FPResult	CRC
Tipo	`uint8_t`	`int32_t`	`uint32_t`	`float`	`uint8_t`

6. Os valores retornados são:

   - FPResult = $(\text{OP1}\ \Phi\ \text{OP1}) * Fc$, onde Φ é a operação escolhida;
   - IntPart = é a parte inteira de FPResult;
   - FracPart = a parte inteira de $(\text{FPResult} - \text{IntPart}) * 10000$;
   - CRC é a soma de todos os 13 primeiros bytes do pacote;

7. Escreva uma função que receba a resposta do servidor com resultado. Imprima os campos recebidos no terminal;

8. Confira a integridade do pacote utilizando o byte de CRC;