**Lesson 85**

**LAN. ENC28J60. TCP Server. Sending data**

In the  [**last lesson,**](http://narodstream.ru/stm-urok-84-lan-enc28j60-tcp-server-ustanavlivaem-i-razryvaem-soedinenie-chast-1/)  we became thoroughly familiar with the **TCP** data transfer  **protocol**  and learned how to establish and break the connection, as this is an essential condition for this protocol.

Now we have the following task before us - and how can we transfer something useful with this protocol. Otherwise, why is it needed then.

This is what we are going to do today.

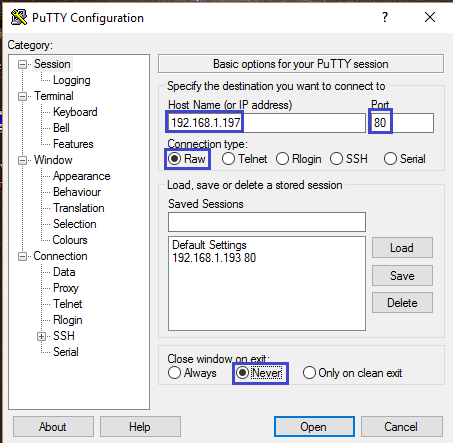
The project we will now call  **ENC28J60\_TCPS\_DATA** , it will be made on the basis of the project of the  [**previous lesson**](http://narodstream.ru/stm-urok-84-lan-enc28j60-tcp-server-ustanavlivaem-i-razryvaem-soedinenie-chast-1/)  named  **ENC28J60\_TCPS\_CONNECT** .

Run the project in the Cube MX, then, pre-generate it for Keil, run it, add our library files as usual, set up the programmer for autoloading, and we will remove the **optimization** , as we did in the last lesson, since the settings are not saved for us .

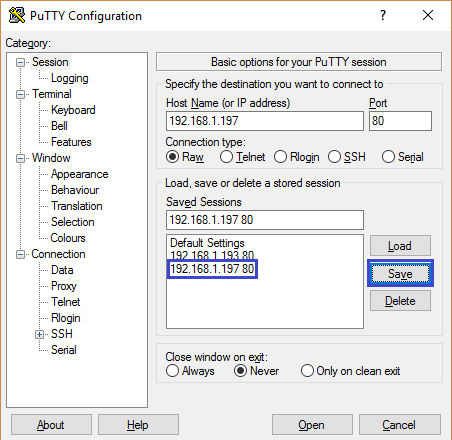
We'll compile our project, we'll run the controller and until we add a code, we'll try to send the data to the module.

To do this, we will already try another utility today -  **Putty** , which you can always download from the official site, and which is absolutely free.

First, as usual, run the terminal program, which connects to our controller, and also run the WireShark network traffic analysis utility, which also connects to our module. Then run the **Putty** utility  , in which we fill the IP address of the module and port **80** , select the RAW connection type, and also prevent the window from closing automatically

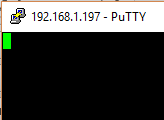


We can save these settings, add the invented name of our connection by clicking the **Save** button  , Then a line will appear with our saved settings in the connection list



After that, when starting the program, we can select this line and, by pressing the **Load** button  , restore all our settings and do not re-enter it every time.

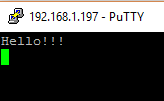
Now we press the **Open** button   and we will open a window with the command line



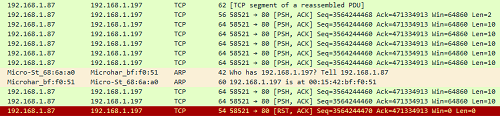
Also, when we open our window, the connection also opens, as indicated by the following lines in WireShark

image03

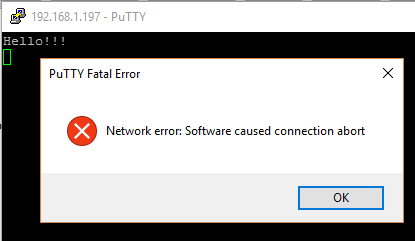
Let's try to enter some line and press  **Enter**



We will get the following few lines in Wireshark (click on the image to enlarge the image)

[](http://narodstream.ru/wp-content/uploads/2017/07/image05-3.png)

The thing is that the client tries to send data to us, but since the ACK flag is set, it means that the client also awaits confirmation from us. These attempts can not continue indefinitely and, without waiting for confirmation, the customer is tearing us apart. The same is reported about it and Putty



But it does not matter to us now, we will process all this later. But we can now analyze what the client sent us and with what flags. We also see that in addition to the ACK flag, the PSH flag is also present in the package, meaning that we have to immediately process the data.

Therefore, close the Putty utility and go to our project, and in it we will first try to display the incoming string in the terminal program as data in the package.

To do this, we function **tcp\_read**  file  **tcp.c**  first spend some adjustments.

Let's also display in the line in the terminal program instead of the MAC addresses of the source IP and the receiver, and also display the size of the data, and how this size is calculated, I'll tell you later. In the meantime, just create for this size a variable in our function and another variable for the counter

uint8\_t res=0;

**uint16\_t len\_data=0;**

**uint16\_t i=0;**

The lines for connecting to the packages are transferred higher, since we will need them now

uint16\_t i=0;

**ip\_pkt\_ptr \*ip\_pkt = (void\*)(frame->data);**

**tcp\_pkt\_ptr \*tcp\_pkt = (void\*)(ip\_pkt->data);**

Calculate the length of the data in the packet, subtracting from the length of the IP packet that is in the corresponding field including the data, the length of the IP header (20), and the length of the TCP header, which also includes options, and data are not included

tcp\_pkt\_ptr \*tcp\_pkt = (void\*)(ip\_pkt->data);

**len\_data = be16toword(ip\_pkt->len)-20-(tcp\_pkt->len\_hdr>>2);**

And now we will display in the terminal program everything that was going to be displayed

len\_data = be16toword(ip\_pkt->len)-20-(tcp\_pkt->len\_hdr>>2);

**sprintf(str1,"%d.%d.%d.%d-%d.%d.%d.%d %d tcp\r\n",**

**ip\_pkt->ipaddr\_src[0],ip\_pkt->ipaddr\_src[1],ip\_pkt->ipaddr\_src[2],ip\_pkt->ipaddr\_src[3],**

**ip\_pkt->ipaddr\_dst[0],ip\_pkt->ipaddr\_dst[1],ip\_pkt->ipaddr\_dst[2],ip\_pkt->ipaddr\_dst[3], len\_data);**

HAL\_UART\_Transmit(&huart1,(uint8\_t\*)str1,strlen(str1),0x1000);

Next, we create a condition that we will enter if there are data in the package, that is, the result will be nonzero

HAL\_UART\_Transmit(&huart1,(uint8\_t\*)str1,strlen(str1),0x1000);

**//Если есть данные, то покажем их в терминальной программе**

**if (len\_data)**

**{**

**}**

We display the data in the terminal program in the body of the condition and switch to a new line

if (len\_data)

{

**for (i=0;i<len\_data;i++)**

**{**

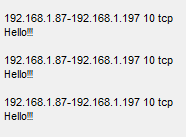
**HAL\_UART\_Transmit(&huart1,tcp\_pkt->data+i,1,0x1000);**

**}**

**HAL\_UART\_Transmit(&huart1,(uint8\_t\*)"\r\n",2,0x1000);**

}

Once again, we'll collect the code and patch the controller, then connect to the server and send the same line



We see that the data is coming, and for some reason, firstly the data is only a line, and then also a line feed with carriage return.

In general, the line has come to us, it remains for us to confirm this packet, and if such a line has come, we will also answer it with another line, thereby we will test the data transfer in the other direction.

In the **tcp.h** file **,** add one more operation code

#define TCP\_OP\_ACK\_OF\_RST 3

**#define TCP\_OP\_ACK\_OF\_DATA 4**

We will use this code in case we need to confirm the data packet.

Without exiting the data detection condition in the **tcp\_read** function  , we add the code according to which we will respond to the packet

HAL\_UART\_Transmit(&huart1,(uint8\_t\*)"\r\n",2,0x1000);

**//Если включен флаг подтверждения, то подтвердим приём данных**

**if (tcp\_pkt->fl&TCP\_ACK)**

**{**

**tcp\_send(ip\_pkt->ipaddr\_src, be16toword(tcp\_pkt->port\_src), TCP\_OP\_ACK\_OF\_DATA);**

**}**

Let's now **go** to the send function  **tcp\_send**  and add there one more local variable for the size of the TCP packet payload

static uint32\_t num\_seg=0;

**uint16\_t sz\_data=0;**

Add there a new reaction condition for our fourth option code

**else if (op==TCP\_OP\_ACK\_OF\_DATA)**

**{**

**}**

return res;

In the body of this condition, we will copy the code from the previous condition before the first sending of the data, and then we will simply change it in some places

First, at the very beginning of the condition body, we insert a row to store the size of the payload into a variable

//Заполним заголовок пакета TCP

**sz\_data = be16toword(ip\_pkt->len)-20-(tcp\_pkt->len\_hdr>>2);**

And here instead of a single we will add this size

tcp\_pkt->num\_ask = be32todword(be32todword(tcp\_pkt->bt\_num\_seg)**+ sz\_data**);

Also after this line we will add code that will show the size of the data (in this place, because if you do not put it, for some reason the controller hangs here, perhaps this is due to the fact that the line itself is transferred first and the line immediately translates the line and returns the carriage

tcp\_pkt->num\_ask = be32todword(be32todword(tcp\_pkt->bt\_num\_seg) + sz\_data);

**sprintf(str1,"sz\_data:%u\r\n", sz\_data);**

**HAL\_UART\_Transmit(&huart1,(uint8\_t\*)str1,strlen(str1),0x1000);**

While nothing else is touching, we'll try to collect the code and flash the controller and enter the same line and see if our confirmation comes, because if we make a mistake for example in the checksum, then the client will not like our confirmation (click on the image to enlarge the image)

[image08_0500](http://narodstream.ru/wp-content/uploads/2017/07/image08-3.png)

We went to 2 packages, and both of us are confirmed. Excellent!

Now let's answer this line in response to the customer. To do this, continue our body conditions. And in it we will create one more condition

enc28j60\_packetSend((void\*)frame,len);

**//Если пришло "Hello!!!", то отправим ответ**

**if (!strcmp((char\*)tcp\_pkt->data,"Hello!!!"))**

**{**

**}**

And in the body of this added condition we will add the code that will put in the data field a line with our answer and send the packet (we will put the flags the same as the client put when sending us a line)

if (!strcmp((char\*)tcp\_pkt->data,"Hello!!!"))

{

**strcpy((char\*)tcp\_pkt->data,"Hello to TCP Client!!!\r\n");**

**tcp\_pkt->fl = TCP\_ACK|TCP\_PSH;**

**sprintf(str1,"hdr\_len:%d\r\n",sizeof(tcp\_pkt\_ptr));**

**HAL\_UART\_Transmit(&huart1,(uint8\_t\*)str1,strlen(str1),0x1000);**

**len = sizeof(tcp\_pkt\_ptr);**

**tcp\_pkt->len\_hdr = len << 2;**

**len+=strlen((char\*)tcp\_pkt->data);**

**tcp\_pkt->cs = 0;**

**tcp\_pkt->cs=checksum((uint8\_t\*)tcp\_pkt-8, len+8, 2);**

**//Заполним заголовок пакета IP**

**len+=sizeof(ip\_pkt\_ptr);**

**ip\_pkt->len=be16toword(len);**

**ip\_pkt->cs = 0;**

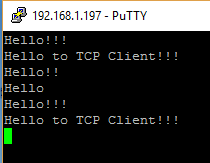
**ip\_pkt->cs = checksum((void\*)ip\_pkt,sizeof(ip\_pkt\_ptr),0);**

**len+=sizeof(enc28j60\_frame\_ptr);**

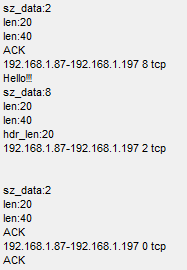
**enc28j60\_packetSend((void\*)frame,len);**

}

Once again, we'll compile the code and run the controller and check our "chat", which when trying to enter the string "Hello !!!" our server will respond with its own string, and when trying to enter another line (in our case, almost the same, but only with two exclamation marks, the server does not already answer the line



 In the terminal program, we also see that the server has already sent the string several times to us



Only now we see that for some reason the connection is not closed correctly

image13

Not only that, but the code also hangs. I have so far managed to apply a non-standard solution, since the cause of this tragedy has not yet been revealed.

We go into the function **tcp\_send** in the body of the condition corresponding to the closing of the connection (with the option **op == TCP\_OP\_ACK\_OF\_FIN** ) and in this body similarly we also insert this line after the line of entering the confirmation number

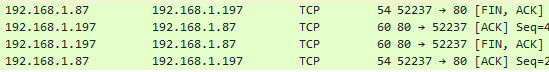
tcp\_pkt->num\_ask = be32todword(be32todword(tcp\_pkt->bt\_num\_seg) + 1);

**//передадим 0 в USART, иначе подвисает код**

**HAL\_UART\_Transmit(&huart1,(uint8\_t\*)0,1,0x1000);**

So it does not hang. If a more sensible way of finding is possible, then I will apply it in the late school bolle.

Now we'll clean up the code, precompiling it, check its operation. Now when Putty closes, the connection is completed correctly with two double handshakes



Then let's go back to the **tcp\_read** receive **function** and process one more option, if suddenly a packet with no data comes with the same flags. In this case, we will terminate the connection

**else if (tcp\_pkt->fl == (TCP\_PSH|TCP\_ACK))**

**{**

**//Если данных нет**

**if(!len\_data)**

**{**

**tcp\_send(ip\_pkt->ipaddr\_src, be16toword(tcp\_pkt->port\_src), TCP\_OP\_ACK\_OF\_FIN);**

**}**

**}**

else if (tcp\_pkt->fl == TCP\_ACK)

We will collect the code once again and check its operability.

Thus, today we have already advanced in the knowledge of the TCP protocol, learning how to receive and send small portions of data in server mode, by itself without the use of windows and long threads. This is an achievement. I think if in the future we need to transfer data of a longer length, then we will also cope with this task

s