**Lesson 84**

**Part 1**

**TCP Server. Install and break connection**

Today we are starting to get acquainted with another, more complex, but at the same time more reliable protocol of OSI transport layer model - **TCP** .

Apparently, due to its reliability, **TCP** ( **Transmission Control Protocol** ) is the most common protocol for data transmission.

The reliability of this protocol lies in the fact that each definite piece of data requires confirmation from the receiving party, thereby ensuring the guaranteed delivery of data to the recipient, as well as the assimilation of the order of messages. And the difficulty is that the data of several packets can be combined into segments, in windows, the windows are also sliding, the header is significantly larger and more complex than the UDP header, and before sending any data it is necessary to establish a connection, and after sending it to tear . And this is done by several packages in different directions. Also, there are still some protocol complications, but we will talk about them as they are applied when necessary.

TCP also provides the transfer of large data streams, integrating them into segments, thereby allowing the transfer of files and other streams.

Also, due to the complexity of the protocol, lessons on it will not take place in one or even in two stages.

The purpose of today's lesson is to organize a TCP server that will respond to the client's request for the connection with appropriate confirmation and also wait for confirmation from the client. Also, if necessary, to the request of the client to break the connection, our server must also react adequately and all this is confirmed.

Therefore, we need to first learn how the TCP connection happens and how it breaks.

But, before learning this, you first need to examine the composition of the TCP header and the purpose of its fields.

The TCP header has the following appearance (click on the image to enlarge the image)

[](http://narodstream.ru/wp-content/uploads/2017/07/image15.png)

Now, briefly about the purpose of the fields.

**The port of the poisoner**  is the value of the data source port,

**The destination port**  is the value of the data receiver port,

**The serial number** is the first byte number in the segment.

The sequence number performs two tasks:

If the SYN flag is set, then this is the initial sequence number (ISN) (Initial Sequence Number), and the first byte of data to be transmitted in the next packet will have a number equal to ISN + 1.

Otherwise, if SYN is not set, the first data byte sent in this packet has this sequence number

Since the TCP stream can generally be longer than the number of different states of this field, all operations with a sequence number must be executed modulo 232. This imposes a practical limitation on the use of TCP. If the transmission rate of the communication system is such that during the MSL (maximum segment lifetime) the serial number has overflowed, two segments with the same number belonging to different parts of the stream may appear in the network and the receiver will receive incorrect data.

**A confirmation number** is a number that indicates that all bytes have been received.

Used to guarantee the delivery of the package. As a rule, this is the number of the last byte in the segment, increased by 1, so the segment recipient gives information on the number of the next segment in the confirmation to the sender. Also, the sender, after receiving the confirmation number from the recipient, will be sure that this is due to the fact that all the sent bytes the recipient received.

**The length of the header** is the length of the entire header, including the options (optional part of the header). Only in contrast to UDP data in the length of the header is no longer included.

**Flags.**

**NS (ECN-nonce concealment protection)** - protection against accidental or malicious modification of ECN flags.

**CWR (Congestion Window Reduced)** - set by the sender to confirm receipt of the receipt of an overload signal.

**ECE (ECN-Echo)** - set by the receiver when receiving an overload signal from the router.

**URG (Urgent pointer field is significant or a pointer of importance)** - signals that the segment contains urgent data, which must be transmitted immediately. Used in conjunction with the field "Pointer to urgent data.

**ACK (Acknowledgment)** - used if the "Confirmation Number" field contains non-zero data, in other words, to confirm previously accepted information.

**PSH (Push)** - indicates that the received data must be immediately transferred to the application without writing to the buffer

**RST (Reset)** - is set if you need to forcefully break the connection.

**SYN (Synchronize)** - synchronization of sequence numbers. It is set to establish a connection.

**FIN ()** - Set if the connection is to be terminated correctly.

**Window size** - this indicates the number of bytes that the recipient can receive from the sender. A window can consist of several segments. It is usually used to transmit several segments with one confirmation to all these segments.

**Checksum** - the calculated value of the sum of header fields that include its mandatory, optional part, as well as all data, if any. As the header for the calculation of the checksum, as in the case of UDP, the header itself is not the same as the pseudo header, to which, like in the case of UDP, the same piece of information from the IP packet is attached.

**A pointer to urgent data** - this field speaks for itself. At the moment it is practically not used. 16-bit positive offset value from the sequence number in this segment. This field indicates the ordinal number of the octet, which ends with important (urgent) data. The field is only considered for packets with the URG flag set.

**Options (or parameters)**  are an optional part of the header. But unlike other types of protocols recently used almost always.

Examples of options or parameters.

The first byte always contains the type identifier of the options. For example, **2 - MSS** or the maximum segment size.

Next in the second byte is the number of bytes in the options, and in the remaining two bytes, it will be the length of the segment, if we use the MSS. Total we got in our case 4 bytes, so the second byte will be a four byte.

In the above case, the byte 3 and 4 will be the maximum segment length in the **big endian** format .

For completeness, I'll also provide a pseudo-header for calculating the checksum (click on the image to enlarge the image)

[](http://narodstream.ru/wp-content/uploads/2017/07/image01-3.png)

Let me remind you that in order to transfer data using the TCP protocol, we must establish a connection.

The TCP connection is provided by a method of a so-called three-fold handshake, that is, it consists of three stages:

1) The client sends a packet to the server with the **SYN** flag set  .

2) The server responds to the client with a packet with the **SYN** and  **ACK** flags set  .

3) The client responds to the server with a packet with the **ACK** flag set  .

Why do we need three stages, like two and enough, at first glance incomprehensible. I will not tell you what exactly provides the method of connection using a three-fold handshake. But such a solution allowed to avoid a number of problems when the packages were not passed.

The disconnection occurs in two stages, in contrast to the installation, but these two phases must be done both from the client side and from the server side. Thus, it will not be two, but four stages

1) The client sends the packet to the server with the **FIN**  and  **ACK** flags set  .

2) The server responds to the client with a packet with the **ACK** flag set  .

3) The server sends a packet to the client with the **FIN**  and  **ACK** flags set  .

4) The client responds to the server with a packet with the **ACK** flag set  .

The process of sending data via the TCP protocol will be considered in later lessons, I repeat that the purpose of our today's lesson is to establish a TCP connection and its correct break.

Therefore, we will start, as always with the project. We will create the project from the project of the [**last lesson**](http://narodstream.ru/stm-urok-83-lan-enc28j60-ntp-uznayom-tochnoe-vremya-chast-1/) named  **ENC28J60\_NTP** and call  **ENC28J60\_TCPS\_CONNECT** .

Run the project in Cube MX and, without touching, generate a project for Keil and open it in this programming environment.

Then we will connect all the files of our libraries and configure the programmer for auto-reloading.

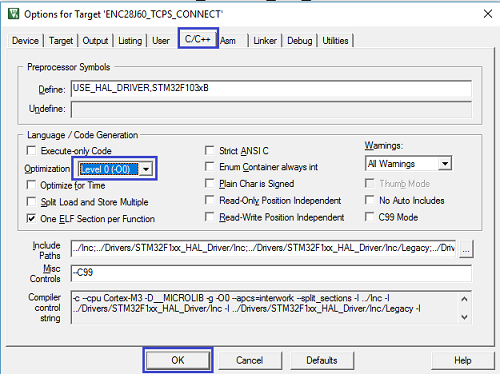
Also, let's turn off the optimization of the code, otherwise some values ​​will be lost during the execution.

This is done as follows.

You must first go to the project settings, for example, by clicking on this button on the Keil toolbar

image03

In the opened dialog, go to the " **C / C ++** " tab and select " **Level 0 (-O0)** " in the field with the heading " **Optimization** " on the left and then click " **OK** "



We will create two standard files for our new library, which will provide work on the TCP protocol with the following content

**tcp.h:**

**#ifndef TCP\_H\_**

**#define TCP\_H\_**

**//--------------------------------------------------**

**#include "stm32f1xx\_hal.h"**

**#include <string.h>**

**#include <stdlib.h>**

**#include <stdint.h>**

**#include "enc28j60.h"**

**#include "net.h"**

**//--------------------------------------------------**

**//--------------------------------------------------**

**#endif /\* TCP\_H\_ \*/**

**tcp.c:**

**#include** "tcp.h"

//--------------------------------------------------

**extern** UART\_HandleTypeDef huart1;

//-----------------------------------------------

**extern** **char** str1[60];

**extern** uint8\_t net\_buf[ENC28J60\_MAXFRAME];

**extern** uint8\_t macaddr[6];

**extern** uint8\_t ipaddr[4];

//--------------------------------------------------

Next, in the **tcp.c** file,  **we**  will create a function for receiving **TCP** packets

**//--------------------------------------------------**

**uint8\_t tcp\_read(enc28j60\_frame\_ptr \*frame, uint16\_t len)**

**{**

**uint8\_t res=0;**

**return res;**

**}**

**//--------------------------------------------------**

Let's create a prototype for this function, go to the file  **net.h**  and connect there our library  **at the bottom of the** file

#include "ntp.h"

**#include "tcp.h"**

We call our function in the function  **ip\_read in** file  **net.c**

else if(ip\_pkt->prt==IP\_TCP)

{

**tcp\_read(frame,len);**

}

And, since we are already in the right file, then at the same time in the checksum checksum function we add one more condition for calculating the amount for the TCP packet. In the third incoming argument for TCP we will use the value  **2**

if(type==1)

{

  sum+=IP\_UDP;

  sum+=len-8;

}

**if(type==2)**

**{**

**sum+=IP\_TCP;**

**sum+=len-8;**

**}**

Let's return to the function of receiving TCP packets in the **tcp.c** file   and displaying some data of the incoming package

uint8\_t res=0;

**sprintf(str1,"%02X:%02X:%02X:%02X:%02X:%02X-%02X:%02X:%02X:%02X:%02X:%02X; %d tcprn",**

**frame->addr\_src[0],frame->addr\_src[1],frame->addr\_src[2],**

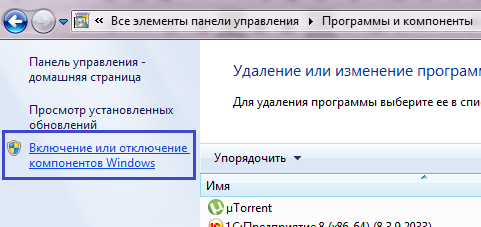
**frame->addr\_src[3],frame->addr\_src[4],frame->addr\_src[5],**

**frame->addr\_dest[0],frame->addr\_dest[1],frame->addr\_dest[2],**

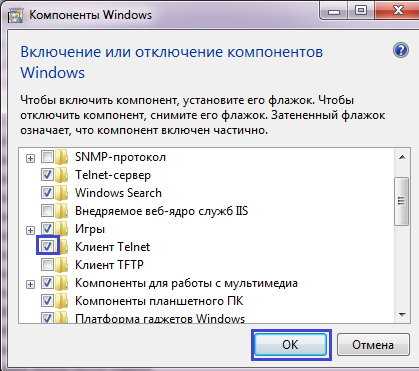
**frame->addr\_dest[3],frame->addr\_dest[4],frame->addr\_dest[5],len);**

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

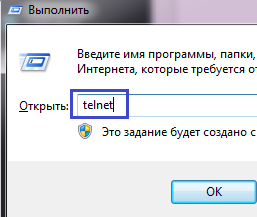
Let's see now whether this package will come to us. Let's start the WireShark utility first, filtering it as usual on the  **MAC** address of our module, and launch the telnet utility. If someone in the Windows OS does not have it, then it can be installed in the next place. Run " **programs and components** " from the control panel, including the image there, for example, in the form of small icons. Then, on the left, select " **Turn Windows components on or off** "



In the dialog that opens, put a checkmark next to " **Telnet Client** " and click  **OK**



The client is installed and will be available for launch from the command line " **Run** ", which is available by the " **Win + R** "



We will collect and write our code to the controller, at the command line of the telnet utility we try to connect to our module using TCP protocol by typing the following command

image00

As we see, we will not be able to connect, since we have not yet written the procedure for answering the package, but nevertheless, attempts to connect to us have come to the terminal programs

image01

We can also see these connection attempts in WireShark

image02

In the [**next part of the**](http://narodstream.ru/stm-urok-84-lan-enc28j60-tcp-server-ustanavlivaem-i-razryvaem-soedinenie-chast-2/) lesson, we will write the entire organization code for all the answers and queries for setting up and breaking the connection and check it in practice.

**Lesson 84**

**Part 2**

# TCP Server. Install and break connection

In the [**previous part of the**](http://narodstream.ru/stm-urok-84-lan-enc28j60-tcp-server-ustanavlivaem-i-razryvaem-soedinenie-chast-1/) lesson we got acquainted with the TCP protocol and its header, also studied the specifics of the installation and disconnection of the TCP connection, created the project and adopted the TCP package.

Now we will think about how we can respond to this package. Let's create another function in tcp.c to send the TCP packet above

**//--------------------------------------------------**

**uint8\_t tcp\_send(uint8\_t \*ip\_addr, uint16\_t port, uint8\_t op)**

**{**

**uint8\_t res=0;**

**uint16\_t len=0;**

**return res;**

**}**

**//--------------------------------------------------**

In the incoming arguments there will be a pointer to the IP address of the recipient, the port of the receiver, and the code of the operation.

In the **tcp.h** file  **,**  create a structure for the TCP header

**//--------------------------------------------------**

**typedef struct tcp\_pkt {**

**uint16\_t port\_src;//порт отправителя**

**uint16\_t port\_dst;//порт получателя**

**uint32\_t bt\_num\_seg;//порядковый номер байта в потоке данных (указатель на первый байт в сегменте данных)**

**uint32\_t num\_ask;//номер подтверждения (первый байт в сегменте + количество байтов в сегменте + 1 или номер следующего ожидаемого байта)**

**uint8\_t len\_hdr;//длина заголовка**

**uint8\_t fl;//флаги TCP**

**uint16\_t size\_wnd;//размер окна**

**uint16\_t cs;//контрольная сумма заголовка**

**uint16\_t urg\_ptr;//указатель на срочные данные**

**uint8\_t data[];//данные**

**} tcp\_pkt\_ptr;**

**//--------------------------------------------------**

I commented on each field of the structure, although I could not do this, we already learned a good title.

Let's add a macro for the number of our local port for accessing us from the outside using the TCP protocol

#include "net.h"

**//--------------------------------------------------**

**#define LOCAL\_PORT\_TCP 80**

**//--------------------------------------------------**

Port 80 is usually used in the case of a web server, perhaps in the future we will implement it.

Next, we will create several readable macros for flags and operation codes

} tcp\_pkt\_ptr;

**//--------------------------------------------------**

**//флаги TCP**

**#define TCP\_CWR 0x80**

**#define TCP\_ECE 0x40**

**#define TCP\_URG 0x20**

**#define TCP\_ACK 0x10**

**#define TCP\_PSH 0x08**

**#define TCP\_RST 0x04**

**#define TCP\_SYN 0x02**

**#define TCP\_FIN 0x01**

**//--------------------------------------------------**

**//операции TCP**

**#define TCP\_OP\_SYNACK 1**

**#define TCP\_OP\_ACK\_OF\_FIN 2**

**#define TCP\_OP\_ACK\_OF\_RST 3**

**//--------------------------------------------------**

Further, in the **tcp\_read** function in the  **tcp.c** file  **,**  we call the function to send a packet with the response code to the connection attempt, provided that we receive a packet with the **SYN** flag  , and only with this flag, there will not be any others. Previously, of course, we will connect to our headings

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

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

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

**if (tcp\_pkt->fl == TCP\_SYN)**

**{**

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

**}**

Then in the **tcp\_send** function   we will create a variable for the header number and also connect to all the packets

uint16\_t len=0;

**static uint32\_t num\_seg=0;**

**enc28j60\_frame\_ptr \*frame=(void\*) net\_buf;**

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

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

Next, we add a condition in which we filter out the response to the connection attempt

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

**if (op==TCP\_OP\_SYNACK)**

**{**

**}**

In the body of this condition, we will first fill the TCP packet header

if (op==TCP\_OP\_SYNACK)

{

**//Заполним заголовок пакета TCP**

**tcp\_pkt->port\_dst = be16toword(port);**

**tcp\_pkt->port\_src = be16toword(LOCAL\_PORT\_TCP);**

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

**tcp\_pkt->bt\_num\_seg = rand();**

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

**tcp\_pkt->size\_wnd = be16toword(8192);**

**tcp\_pkt->urg\_ptr = 0;**

**len = sizeof(tcp\_pkt\_ptr)+4;**

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

**tcp\_pkt->data[0]=2;//Maximum Segment Size (2)**

**tcp\_pkt->data[1]=4;//Length**

**tcp\_pkt->data[2]=0x05;**

**tcp\_pkt->data[3]=0x82;**

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

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

}

Now everything in order.

First, we enter the port numbers of the sender and the receiver into the appropriate header fields, then we enter the number in the confirmation number that came to us in the segment number field, increased by 1, not forgetting the overturn of bytes. For one, we increase it because there is no data in this kind of received packet, so we pass the number of the next expected byte. In the segment number, we can safely enter any value, since this is our first sent packet in this thread. Then we set the corresponding flags that are needed to confirm the connection request, then we enter the window size, you can enter any up to 65535, but I decided to enter the same as in the server request. Then, in the urgent data field, we enter zero, since we do not use it. Then calculate the length equal to the length of the header, increased by the size of the options, then enter this length in the corresponding bits, shifting by 2 points. Then we enter the options. As options, we give the maximum segment size, putting the corresponding option code in the first byte, the number of bytes in the options in the 2nd byte, and in 3 and 4 bytes, we will have the length itself, respectively inverted, that is **0x0582** .

In general, since we have already studied the title closely, it will not be difficult to understand the code for filling it.

Then we reset the field with the checksum and then count it again and put it in the same field.

Next we fill the IP header

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

**//Заполним заголовок пакета IP**

**sprintf(str1,"len:%d\r\n", len);**

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

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

**sprintf(str1,"len:%d\r\n", len);**

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

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

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

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

**ip\_pkt->verlen = 0x45;**

**ip\_pkt->fl\_frg\_of=0;**

**ip\_pkt->ttl=128;**

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

**ip\_pkt->prt=IP\_TCP;**

**memcpy(ip\_pkt->ipaddr\_dst,ip\_addr,4);**

**memcpy(ip\_pkt->ipaddr\_src,ipaddr,4);**

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

Well, there's nothing to explain at all, we've already filled the IP header many times.

Then fill the Ethernet header and send our packet to the client

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

**//Заполним заголовок Ethernet**

**memcpy(frame->addr\_dest,frame->addr\_src,6);**

**memcpy(frame->addr\_src,macaddr,6);**

**frame->type=ETH\_IP;**

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

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

Next, let's display in the terminal program the length of our packet and the type of our sent TCP packet

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

**sprintf(str1,"len:%d\r\n", len);**

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

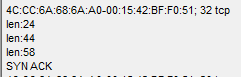
**HAL\_UART\_Transmit(&huart1,(uint8\_t\*)"SYN ACK\r\n",9,0x1000);**

}

We will collect our code and try again to send the package, but not forgetting that the port we now have 80

image05

Also we see everything in the terminal program



In WireShark we also see that our confirmation is correct. And we see this because the client responded to our confirmation by acknowledgment with the **ACK** flag  (click on the picture to enlarge the image)

[image07_0500](http://narodstream.ru/wp-content/uploads/2017/07/image07-2.png)

That is, we have established a connection.

Let's also, for completeness of the picture in the function **tcp\_read,**  display in the terminal program the fact that we received confirmation from the client

  tcp\_send(ip\_pkt->ipaddr\_src, be16toword(tcp\_pkt->port\_src), TCP\_OP\_SYNACK);

}

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

**{**

**HAL\_UART\_Transmit(&huart1,(uint8\_t\*)"ACK\r\n",5,0x1000);**

**}**

We will collect the code once again, we will sew the controller and try to connect again. The truth before this will have to be disconnected, which is correct to do so we will not be able to answer, because we can not answer because we do not have the same code yet and we did not write the code for further disassembly. Well, what to do, disconnect incorrectly. To break the connection, in telnet, you must first return to the command line mode. To do this, enter the keyboard shortcut Ctrl + "]". Before we get an invitation to the command line, where we enter the command "c" without any parameters at all, which means our desire to break the current connection

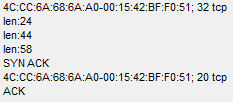
image08

In WireShark we will also see an incorrect disconnection

image10

Well, what to do, one way or another the connection is broken, and we can try to connect again.

Now we should see in the terminal program this



This means that the confirmation packet reached us and we received it.

Well, now we need to learn how to professionally and disconnect. We will be disconnected also on the initiative of the client. Therefore, in the function of receiving the package, we process the corresponding flags

  tcp\_send(ip\_pkt->ipaddr\_src, be16toword(tcp\_pkt->port\_src), TCP\_OP\_SYNACK);

}

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

**{**

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

**}**

Then we return to our function of sending the TCP packet tcp\_send and process the following condition with our sent operation code

  HAL\_UART\_Transmit(&huart1,(uint8\_t\*)"SYN ACK\r\n",9,0x1000);

}

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

**{**

**}**

And in the body of this condition, we copy all the code completely from the condition above, and then slowly adjust only certain lines in it.

First of all, we insert another line after entering the ports in the appropriate fields, which temporarily places the value of the confirmation number from the received packet into a variable, since in the next line it will be intercepted

tcp\_pkt->port\_src = be16toword(LOCAL\_PORT\_TCP);

**num\_seg = tcp\_pkt->num\_ask;**

Then, in the line where we enter the confirmation number in the corresponding field, instead of a random value, we will already use the value stored. That is, the confirmation number from the received packet in the sent packet will become the segment number

tcp\_pkt->bt\_num\_seg = **num\_seg**;

Further we put in the field of flags only one flag instead of two - the confirmation flag

tcp\_pkt->fl = **TCP\_ACK**;

We will no longer have the options, we do not need them, we still complete the connection, so we save 4 bytes and do not add them to the length of the header

**len = sizeof(tcp\_pkt\_ptr);**

Well, accordingly, we remove the code for adding options

~~tcp\_pkt->data[0]=2;//Maximum Segment Size (2)~~

~~tcp\_pkt->data[1]=4;//Length~~

~~tcp\_pkt->data[2]=0x05;~~

~~tcp\_pkt->data[3]=0x82;~~

After sending the package, we will remove all the code to the end of the condition body and show the following line in the terminal program

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

**HAL\_UART\_Transmit(&huart1,(uint8\_t\*)"ACK OF FIN\r\n",12,0x1000);**

After sending the package, we will have to send another request for disconnection, since this procedure is two-way.

To send us this request again, we just need to replace the field with the flags, as well as recalculate the checksum, and also run through the length of the entire package

    HAL\_UART\_Transmit(&huart1,(uint8\_t\*)"ACK OF FIN\r\n",12,0x1000);

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

**len = sizeof(tcp\_pkt\_ptr);**

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

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

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

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

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

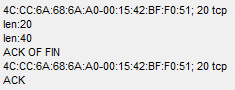
  }

  return res;

}

We'll collect the code, we'll tell the controller, and if we still have the connection, then we'll try to disconnect in the same way with the help of the telnet command.

If everything is fine, then in the terminal program we will see this



And in WireShark we will see this (click on the image to enlarge)

[image14_0500](http://narodstream.ru/wp-content/uploads/2017/07/image14.png)

Judging by this information, we have achieved a considerable result by installing and breaking the TCP connection correctly. I think the lesson on transferring the payload via the TCP protocol will be much easier for us to study, since we have studied the heading, and have already established some of the theory in practice. Subsequently, we also, I think, write and the client, who already himself will request permission from the server to connect, disconnect and transfer data.