**Lesson 87**

**Part 1**

**LAN. ENC28J60. TCP WEB Server. We pass the page a little more**

Today we will continue the topic of the **HTTP protocol** , which is very actively used in our time for data transmission.

In the  [**previous lesson**](http://narodstream.ru/stm-urok-86-lan-enc28j60-tcp-web-server-peredayom-maluyu-stranicu-chast-2/) we got acquainted with this protocol, with its header, with the types of messages in the header, and also we could compose a small server from our controller paired with the module on the **ENC28J60**chip and transfer the page to the client who requested it from the browser. True, our page consisted of a single TCP packet, but nevertheless it was correctly transmitted.

Let's try to transfer a larger page today, which will consist of any number of packages. This has its own specifics, but, I think, we will slowly sort this out. Also we traditionally again optimize and improve our project a little, with what, actually, and we will begin.

We added an additional incoming argument to the **eth\_send** function found in the **net.c** file that was created for the transmission of Ethernet packets, also corrected calls to this function, thus noticing that we only call this function in two places in the project, although we send these packets constantly. In the wasteland, as it turned out, we do not use this function, but we are addressing directly to the low-level function of frame transmission, which introduces some redundancy into the code, since it is necessary. that is in our function, constantly writing in all places of code. Let's fix this.

For this, we will first need a project that we call **ENC28J60\_HTTPS\_LARGE** , and it will be made on the basis of a draft [**of the previous sessions**](http://narodstream.ru/stm-urok-86-lan-enc28j60-tcp-web-server-peredayom-maluyu-stranicu-chast-2/) **ENC28J60\_HTTPS** .

Launch our project in **Cube the MX** , generate project for **the Keil** , open it up, configure the programmer to reload, just in case, disconnect and connect the optimization of our files in the project.

Let's start with the file directly **net.c** . We find there the function **icmp\_request** and at the very bottom we delete the following lines of code

~~//Заполним заголовок пакета Ethernet~~

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

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

~~enc28j60\_packetSend((void\*)frame,len + sizeof(enc28j60\_frame\_ptr));~~

Instead, we add the following

**//отправим пакет Ethernet**

**eth\_send(frame,ETH\_IP,len);**

In this file there is nothing more to change. Now go to **arp.c** , find the function **arp\_request** in it .

Delete this

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

~~frame->type = ETH\_ARP;~~

~~enc28j60\_packetSend((void\*)frame,sizeof(arp\_msg\_ptr) + sizeof(enc28j60\_frame\_ptr));~~

And we'll put this in here

**// send the Ethernet packet**

**eth\_send (frame, ETH\_ARP, sizeof (arp\_msg\_ptr));**

There is one more place - file **ntp.c** function **ntp\_request** .

Delete (not all, not forgetting, leave the necessary)

~~//Заполним заголовок пакета Ethernet~~

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

memcpy(frame->addr\_dest,ntpprop.macaddr\_dst,6);

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

~~enc28j60\_packetSend((void\*)frame,len + sizeof(enc28j60\_frame\_ptr));~~

We paste

**//Отправим пакет Ethernet**

memcpy(frame->addr\_dest,ntpprop.macaddr\_dst,6);

**eth\_send(frame,ETH\_IP,len);**

Now we'll work with the main topic of the lesson. To do this, go to the file **tcp.c** and try to separate the request from the client of the main page from the requests of other documents. To do this, in the function **tcp\_read,** we fix the code that monitors the query string, eliminating a space from there, so that we can now work with all the HTTP requests, and not only with the request of the server's main page. Also, a little correct and comment

//Если строка **"GET /", то значит это запрос HTTP**

if (strncmp((char\*)tcp\_pkt->data,**"GET /", 5**) == 0)

Let's go into the body of this condition and add the following branch there, separating the request of the main page from the requests of other documents

if (strncmp((char\*)tcp\_pkt->data,"GET /", 5) == 0)

{

**//Если пробел, то это запрос главной страницы**

**if((char)tcp\_pkt->data[5]==' ')**

**{**

**}**

**else**

**{**

**}**

  tcpprop.data\_size = strlen(http\_header) + sizeof(index\_htm);

Go to the file  **tcp.h**  and add there several options for documents. While we will have only two - the main page and the page with the error of existence of the document on the server

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

**//Варианты документов HTTP**

**#define INDEX\_HTML 0**

**#define E404\_HTML 1**

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

Also, in the same header file, add one more property for the TCP properties

  volatile uint16\_t cnt\_rem\_data\_part;//количество оставшихся частей данных для передачи

**volatile uint8\_t http\_doc;//вариант документа для передачи**

} tcp\_prop\_ptr;

At the same time, at the beginning of this structure, we will also add a field for storing the MAC address, as well as the IP addresses of the future recipient, since even though they are equal to the address of the sender who sent us the request, only they often have the property of getting lost

typedef struct tcp\_prop {

**uint8\_t macaddr\_dst[6];//MAC-адрес получателя**

**uint8\_t ipaddr\_dst[6];//IP-адрес получателя**

We return to the file **tcp.c** and in the function **tcp\_read we** immediately initialize the data of the structure field with addresses

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

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

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

tcpprop.port\_dst = be16toword(tcp\_pkt->port\_src);

Add one more version of the HTTP response header to answer a request for a nonexistent document

const char http\_header[] = {"HTTP/1.1 200 OK\r\nServer: nginx\r\nContent-Type: text/html\r\nConnection: keep-alive\r\n\r\n"};

**const char error\_header[] = {"HTTP/1.1 404 File not found\r\nServer: nginx\r\nContent-Type: text/html\r\nConnection: keep-alive\r\n\r\n"};**

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

Our header differs from the previous one only with the **404** document existence error code and the " **File not found** " error message itself .

And after announcing the array with our main page, add another array with an error page. True, the browser most likely, after seeing the appropriate header of the HTTP response from the server, will either issue its page, or take some more actions, but nevertheless we will have this page in service

0x3e,0x0a,0x3c,0x2f,0x62,0x6f,0x64,0x79,0x3e,0x3c,0x2f,0x68,0x74,0x6d,0x6c,0x3e};

**const uint8\_t e404\_htm[] = {**

**0x3c,0x68,0x74,0x6d,0x6c,0x3e,0x0a,0x20,0x20,0x3c,0x68,0x65,0x61,0x64,0x3e,0x0a,**

**0x20,0x20,0x20,0x20,0x3c,0x74,0x69,0x74,0x6c,0x65,0x3e,0x34,0x30,0x34,0x20,0x4e,**

**0x6f,0x74,0x20,0x46,0x6f,0x75,0x6e,0x64,0x3c,0x2f,0x74,0x69,0x74,0x6c,0x65,0x3e,**

**0x0a,0x20,0x20,0x3c,0x2f,0x68,0x65,0x61,0x64,0x3e,0x0a,0x3c,0x62,0x6f,0x64,0x79,**

**0x3e,0x0a,0x3c,0x68,0x31,0x20,0x73,0x74,0x79,0x6c,0x65,0x3d,0x22,0x74,0x65,0x78,**

**0x74,0x2d,0x61,0x6c,0x69,0x67,0x6e,0x3a,0x20,0x63,0x65,0x6e,0x74,0x65,0x72,0x3b,**

**0x22,0x3e,0x34,0x30,0x34,0x20,0x45,0x72,0x72,0x6f,0x72,0x20,0x46,0x69,0x6c,0x65,**

**0x20,0x4e,0x6f,0x74,0x20,0x46,0x6f,0x75,0x6e,0x64,0x3c,0x2f,0x68,0x31,0x3e,0x0a,**

**0x3c,0x68,0x32,0x20,0x73,0x74,0x79,0x6c,0x65,0x3d,0x22,0x74,0x65,0x78,0x74,0x2d,**

**0x61,0x6c,0x69,0x67,0x6e,0x3a,0x20,0x63,0x65,0x6e,0x74,0x65,0x72,0x3b,0x22,0x3e,**

**0x20,0x54,0x68,0x65,0x20,0x70,0x61,0x67,0x65,0x20,0x79,0x6f,0x75,0x20,0x61,0x72,**

**0x65,0x20,0x6c,0x6f,0x6f,0x6b,0x69,0x6e,0x67,0x20,0x66,0x6f,0x72,0x20,0x6d,0x69,**

**0x67,0x68,0x74,0x20,0x68,0x61,0x76,0x65,0x20,0x62,0x65,0x65,0x6e,0x20,0x72,0x65,**

**0x6d,0x6f,0x76,0x65,0x64,0x2c,0x20,0x3c,0x62,0x72,0x20,0x2f,0x3e,0x68,0x61,0x64,**

**0x20,0x69,0x74,0x73,0x20,0x6e,0x61,0x6d,0x65,0x20,0x63,0x68,0x61,0x6e,0x67,0x65,**

**0x64,0x2c,0x20,0x6f,0x72,0x20,0x69,0x73,0x20,0x74,0x65,0x6d,0x70,0x6f,0x72,0x61,**

**0x72,0x69,0x6c,0x79,0x20,0x75,0x6e,0x61,0x76,0x61,0x69,0x6c,0x61,0x62,0x6c,0x65,**

**0x2e,0x3c,0x2f,0x68,0x32,0x3e,0x0a,0x3c,0x2f,0x62,0x6f,0x64,0x79,0x3e,0x3c,0x2f,**

**0x68,0x74,0x6d,0x6c,0x3e};**

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

Let's return now to the body of our condition and fill the empty code code of the document request condition, and the line for calculating the length of the document that we will send in response will be deleted, since it will be different for our variants

//Если пробел, то это запрос главной страницы

if((char)tcp\_pkt->data[5]==' ')

{

**tcpprop.http\_doc = INDEX\_HTML;**

**//сначала включаем в размер размер заголовка**

**tcpprop.data\_size = strlen(http\_header);**

**//затем размер самого документа**

**tcpprop.data\_size += sizeof(index\_htm);**

}

else

{

**tcpprop.http\_doc = E404\_HTML;**

**//сначала включаем в размер размер заголовка**

**tcpprop.data\_size = strlen(error\_header);**

**//затем размер самого документа**

**tcpprop.data\_size += sizeof(e404\_htm);**

}

~~tcpprop.data\_size = strlen(http\_header) + sizeof(index\_htm);~~

Now we also need to separate the options for populating the field with the TCP packet data in the send function of the **tcp\_send\_http\_one** page  . Two lines with the filling of the field with the data will be replaced by the following code

//Отправляем страницу

**if (tcpprop.http\_doc==INDEX\_HTML)**

**{**

**strcpy((char\*)tcp\_pkt->data,http\_header);**

**memcpy((void\*)(tcp\_pkt->data+strlen(http\_header)),(void\*)index\_htm,sizeof(index\_htm));**

**}**

**else**

**{**

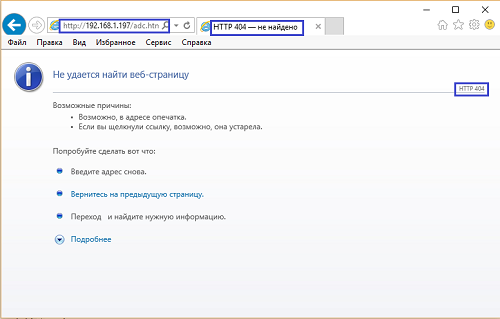
**strcpy((char\*)tcp\_pkt->data,error\_header);**

**memcpy((void\*)(tcp\_pkt->data+strlen(error\_header)),(void\*)e404\_htm,sizeof(e404\_htm));**

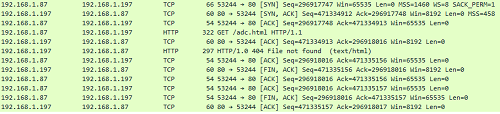
**}**

len = sizeof(tcp\_pkt\_ptr);

Now we can check our code by compiling it by flashing the controller. First we check that we also have the main page transfer, and then in the address bar of the browser we enter the request for a nonexistent document from our server. The result will be like this (click on the image to enlarge the image)

[](http://narodstream.ru/wp-content/uploads/2017/08/image00-1.png)

As I warned, the browser instead of our page offered its own. Let's see also the transfer of our page to WireShark (click on the picture to enlarge the image)

[](http://narodstream.ru/wp-content/uploads/2017/08/image01-1.png)

We see. that the response was sent to the client correctly and was correctly recognized.

Now let's proceed to the solution of our main task - to transfer to the client a page consisting not of one, but of two or more packages.

Again, go into the header file **tcp.h** and make some corrections to the structure of TCP, changing the name of the field for the number of remaining pages

volatile uint16\_t cnt**\_rem**\_data\_part;//количество оставшихся частей данных для передачи

The old name will also come in handy, but we'll add it later, otherwise it will be hard for us to find the occurrence of this variable in the text of the code, which we will now do by mistake, correcting the old field name with a new field in the code of the tcp.c file, We are only in the **tcp\_read** function  **so far** . To do this, we will compile the code and make all this by mistake.

Then we return to our structure and above this field we insert another one with the old name, but also with another comment

**volatile uint16\_t cnt\_data\_part;//общее количество частей данных для передачи**

volatile uint16\_t cnt\_rem\_data\_part;//количество оставшихся частей данных для передачи

That is, in the future, to work correctly the code will need to know not only the number of remaining packages that we need to transfer, but also remember the total number of such packages.

We now return to our feature **tcp\_read** file **tcp.c** and think about what might happen next situation in which the total amount of data that we are going to transfer to the client will be divided without a remainder on the maximum segment size. At the same time, we can meet with such a problem, in which the size of the remaining part will be zero, and the number of packets will be overestimated by one. So let's do some fighting with this

tcpprop.last\_data\_part\_size = tcpprop.data\_size % tcp\_mss;

**//борьба с неправильным расчётом, когда общий размер делится на минимальный размер сегмента без остатка**

**if(tcpprop.last\_data\_part\_size==0)**

**{**

**tcpprop.last\_data\_part\_size=tcp\_mss;**

**tcpprop.cnt\_rem\_data\_part--;**

**}**

Also, we will save the total number of packets for transmission until their number is incremented

  tcpprop.cnt\_rem\_data\_part--;

}

**tcpprop.cnt\_data\_part = tcpprop.cnt\_rem\_data\_part;**

Let's also add a new argument to the function that prepares the header of the TCP packet, the length for calculating the checksum, since this length often differs from the length of the header due to the fact that there may be data in the packet that are also involved in the calculation of the checksum

void tcp\_header\_prepare(tcp\_pkt\_ptr \*tcp\_pkt, uint16\_t port, uint8\_t fl, uint16\_t len**, uint16\_t len\_cs**)

Also at the end of this function, we fix the function of calculating the checksum using this argument

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

}

We will also add this argument in all places of this function call. Since we have so far used this function to prepare packet headers in which there was no data, the last two arguments in them will be the same

  tcp\_header\_prepare(tcp\_pkt, port, флаги, len**, len**);

Now we add a function that will pass the first packet of the HTTP multi-packet response, since until now we had only one function, in which we transmitted a single packet. In the function that we will now add, we will first send a TCP confirmation packet to the HTTP client request. The function code will be placed immediately after the function of sending a single HTTP packet

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

**/\*Отправка первого пакета многопакетного ответа HTTP\*/**

**uint8\_t tcp\_send\_http\_first(enc28j60\_frame\_ptr \*frame, uint8\_t \*ip\_addr, uint16\_t port)**

**{**

**uint8\_t res=0;**

**uint16\_t len=0;**

**uint16\_t sz\_data=0;**

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

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

**//Отправим сначала подтверждение на пакет запроса**

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

**tcpprop.seq\_num = tcp\_pkt->num\_ask;**

**tcpprop.ack\_num = be32todword(be32todword(tcp\_pkt->bt\_num\_seg) + sz\_data);**

**len = sizeof(tcp\_pkt\_ptr);**

**tcp\_header\_prepare(tcp\_pkt, port, TCP\_ACK, len, len);**

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

**ip\_header\_prepare(ip\_pkt, ip\_addr, IP\_TCP, len);**

**//Заполним заголовок Ethernet**

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

**eth\_send(frame,ETH\_IP,len);**

**return res;**

**}**

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

And further we will pass our package, which is the first part of our common HTTP response

eth\_send(frame,ETH\_IP,len);

**//Отправляем первую часть страницы**

**if (tcpprop.http\_doc==INDEX\_HTML)**

**{**

**strcpy((char\*)tcp\_pkt->data,http\_header);**

**memcpy((void\*)(tcp\_pkt->data+strlen(http\_header)),(void\*)index\_htm,tcp\_mss-strlen(http\_header));**

**}**

**else**

**{**

**strcpy((char\*)tcp\_pkt->data,error\_header);**

**memcpy((void\*)(tcp\_pkt->data+strlen(error\_header)),(void\*)e404\_htm,tcp\_mss-strlen(error\_header));**

**}**

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

**len = sizeof(tcp\_pkt\_ptr);**

**len+=tcp\_mss;**

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

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

**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);**

**//Заполним заголовок Ethernet**

**eth\_send(frame,ETH\_IP,len);**

We will have exactly as many data as the maximum size of our segment, that is, 458, so we use the variable **mss** boldly  .

Decrement the remaining number of parts, since we have already transferred one

eth\_send(frame,ETH\_IP,len);

**//будем считать, что одну часть отправили, поэтому количество оставшихся частей декрементируем**

**tcpprop.cnt\_rem\_data\_part--;**

It remains for us in this function only to determine whether the last part will be the next part or not, and in accordance with this, assign the desired structure variable to the desired status

  tcpprop.cnt\_rem\_data\_part-;

**if (tcpprop.cnt\_rem\_data\_part> 1)**

**{**

**tcpprop.data\_stat = DATA\_MIDDLE;**

**}**

**else**

**{**

**tcpprop.data\_stat = DATA\_LAST;**

**}**

  return res;

}

That is, if the number of remaining parts is one, then we will use the transfer function of the last packet in the next transfer, and if more - then the middle package.

Let's try to transfer this page, which together with the HTTP header will fit into two packages, so to speak, let's go to the goal gradually. For this we need to prepare such a page. We will cook it as well, as in the last lesson, using the **makefsdata** utility  . I will not show the process, as we have already seen. Replace the page array with a new one

**index\_htm**

Process the status of  **DATA\_FIRST**  in the **tcp\_read** function

    tcp\_send\_http\_one(frame, ip\_pkt->ipaddr\_src, tcpprop.port\_dst);

  }

**else if(tcpprop.data\_stat==DATA\_FIRST)**

**{**

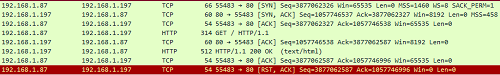
**tcp\_send\_http\_first(frame, ip\_pkt->ipaddr\_src, tcpprop.port\_dst);**

**}**

}

//Иначе обычные данные

We will collect the code, we will tell the controller, we will request our page in the browser, and although it does not show it, since we have not yet transferred it, but still see what we have already transferred, we can at least use the Wireshark utility (click on the picture for increase the image)

[](http://narodstream.ru/wp-content/uploads/2017/08/image02-2.png)

We see that our package was sent to the client and the latter confirmed his acceptance. So all the checksums in our order and the number of segments and confirmations, too. Otherwise, the client will never confirm receipt of such a packet (it is checked).

Let's go into the file  **tcp.h**  and for the future we'll still protect ourselves from some trabblah, which, due to our actions, we will not see.

We will add one more field for the segment number to the TCP property structure, since the only thing sometimes happens is that the code is running and, as a result, it is not always stored unchanged

volatile uint32\_t seq\_num;//порядковый номер байта

**volatile uint32\_t seq\_num\_tmp;//порядковый номер байта временный**

Let's return to the file **tcp.c** and at the end of the function **tcp\_send\_http\_one** we will **enter the** value in our new field

eth\_send(frame,ETH\_IP,len);

**//Подготовим номер на будущее - на пакет с желанием завершить соединение**

**tcpprop.seq\_num\_tmp = be32todword(be32todword(tcpprop.seq\_num)+tcpprop.data\_size);**

tcpprop.data\_stat=DATA\_END;

Now go into the body of the function  **tcp\_send\_http\_dataend**  and fix in it some code

~~tcpprop.seq\_num = tcp\_pkt->num\_ask;~~

**tcpprop.seq\_num = tcpprop.seq\_num\_tmp;**

Later, this field is still useful.

We will not write the code for the middle packet, since our answer consists of two packages, we will do this later.

In the meantime, we will deal with the function of transmitting the last HTTP response packet. Add it after the function that we just wrote

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

**/\*Отправка последнего пакета многопакетного ответа HTTP\*/**

**uint8\_t tcp\_send\_http\_last(enc28j60\_frame\_ptr \*frame, uint8\_t \*ip\_addr, uint16\_t port)**

**{**

**uint8\_t res=0;**

**uint16\_t len\_tcp=0, len=0;**

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

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

**return res;**

**}**

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

Call this function at the end of the **tcp\_read** function  in the **TCP** packet processing  with confirmation

else if (tcp\_pkt->fl == TCP\_ACK)

{

  if (tcpprop.data\_stat==DATA\_END)

  {

    tcp\_send\_http\_dataend(frame, ip\_pkt->ipaddr\_src, tcpprop.port\_dst);

  }

**else if (tcpprop.data\_stat==DATA\_LAST)**

**{**

**HAL\_UART\_Transmit(&huart1,(uint8\_t\*)"LAST\r\n",6,0x1000);**

**tcpprop.data\_stat=DATA\_COMPLETED;**

**tcp\_send\_http\_last(frame, tcpprop.ipaddr\_dst, tcpprop.port\_dst);**

**}**

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

}

return res;

Let's return to our new function  **tcp\_send\_http\_last**  and continue to write its code.

Prepare the TCP header and data

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

**//Отправляем последнюю часть страницы**

**//Заполним заголовок пакета TCP**

**tcpprop.seq\_num = be32todword(be32todword(tcpprop.seq\_num)+tcp\_mss);**

**len\_tcp = sizeof(tcp\_pkt\_ptr);**

**if (tcpprop.http\_doc==INDEX\_HTML)**

**{**

**memcpy((void\*)tcp\_pkt->data,(void\*)(index\_htm+(tcp\_mss\*(tcpprop.cnt\_data\_part-1))-strlen(http\_header)),tcpprop.last\_data\_part\_size);**

**}**

**else**

**{**

**memcpy((void\*)tcp\_pkt->data,(void\*)(e404\_htm+(tcp\_mss\*(tcpprop.cnt\_data\_part-1))-strlen(error\_header)),tcpprop.last\_data\_part\_size);**

**}**

**len=len\_tcp + tcpprop.last\_data\_part\_size;**

**tcp\_header\_prepare(tcp\_pkt, port, TCP\_PSH|TCP\_ACK, len\_tcp, len);**

We also use the **PSH** flag here  , since the post-package package requires the client to immediately process and confirm it.

Now prepare the IP header and send our package

tcp\_header\_prepare(tcp\_pkt, port, TCP\_PSH|TCP\_ACK, len\_tcp, len);

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

**ip\_header\_prepare(ip\_pkt, ip\_addr, IP\_TCP, len);**

**//Заполним заголовок Ethernet**

**memcpy(frame->addr\_dest,tcpprop.macaddr\_dst,6);**

**eth\_send(frame,ETH\_IP,len);**

Next, we will update the value of the segment number for the next TCP packet. This we will have a request for disconnection

  eth\_send(frame,ETH\_IP,len);

**//продвинемся по номеру сегмента**

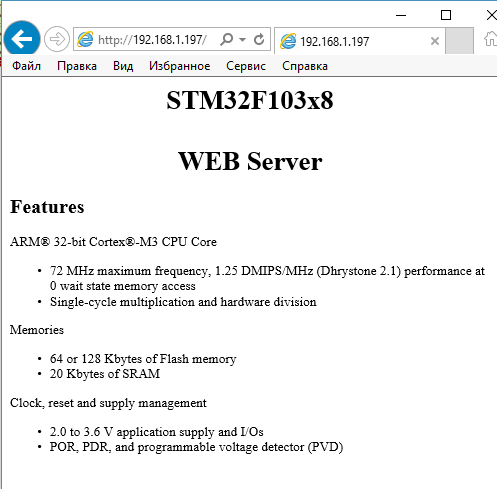
**tcpprop.seq\_num\_tmp = be32todword(be32todword(tcp\_pkt->bt\_num\_seg)+tcpprop.last\_data\_part\_size);**

**tcpprop.data\_stat=DATA\_END;**

  return res;

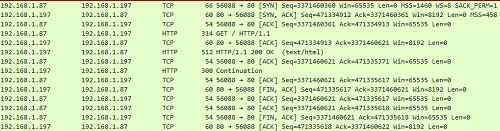
}

We will collect the code, we will tell the controller and try again to request the page in the browser



Everything has been reflected in us.

Also see the process of transferring data to Wireshark (click on the image to enlarge the image)

[](http://narodstream.ru/wp-content/uploads/2017/08/image04-1.png)

Here, too, everything is fine! However, to get this done it was necessary to sweat, play with the numbers of segments, with the checksum, since if the latter is incorrect, the client will never give a confirmation for the packet sent to him, although he often (and almost always) sends packets with an incorrect control amount. Well, nothing can be done about it. The client is always right!

In the [**next part of the**](http://narodstream.ru/stm-urok-87-lan-enc28j60-tcp-web-server-peredayom-stranicu-pobolshe-chast-2/) lesson, we will write the code to send the client a page consisting of an unlimited number of TCP packets.

**Lesson 87**

**Part 2**

**LAN. ENC28J60. TCP WEB Server. We pass the page a little more**

In the [**previous part of the**](http://narodstream.ru/stm-urok-87-lan-enc28j60-tcp-web-server-peredayom-stranicu-pobolshe-chast-1/) lesson, we carried out a certain optimization of the project and wrote a code that differentiates the request from the client's main page from the rest of the requests, and also passed the response to the client's request for a non-existent document. We also gave the client a page consisting of two TCP packets.

Further work is to transfer the document to three or more packages. To do this, create a function to transfer one of the medium document packages. Add it between the transfer functions of the first and last packets

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

**/\*Отправка среднего пакета многопакетного ответа HTTP\*/**

**uint8\_t tcp\_send\_http\_middle(enc28j60\_frame\_ptr \*frame, uint8\_t \*ip\_addr, uint16\_t port)**

**{**

**uint8\_t res=0;**

**uint16\_t len\_tcp=0, len=0;**

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

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

**return res;**

**}**

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

Call this function in the last condition of the **tcp\_read** function

else if (tcp\_pkt->fl == TCP\_ACK)

{

  if (tcpprop.data\_stat==DATA\_END)

  {

    tcp\_send\_http\_dataend(frame, ip\_pkt->ipaddr\_src, tcpprop.port\_dst);

  }

**else if (tcpprop.data\_stat==DATA\_MIDDLE)**

**{**

**tcp\_send\_http\_middle(frame, tcpprop.ipaddr\_dst, tcpprop.port\_dst);**

**}**

  else if (tcpprop.data\_stat==DATA\_LAST)

We continue to write our function.

As always, calculate the segment number and save it in the structure in the corresponding field, and also enter the length of the TCP header into the corresponding variable

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

**//Отправляем одну из средних частей страницы**

**//Заполним заголовок пакета TCP**

**tcpprop.seq\_num = be32todword(be32todword(tcpprop.seq\_num)+tcp\_mss);**

**len\_tcp = sizeof(tcp\_pkt\_ptr);**

Then fill in the data field, applying copy operations from FLASH-memory, and also applying a cunning calculation of the displacement along the array with the document, depending on the total number of parts and the remaining number of parts (for which we at the beginning of the conception and came up with a new field in the structure). Also, imagine that the page with an error is also very large. Although this is not the case, it may be useful in the future to send large documents that are not the main page

len\_tcp = sizeof(tcp\_pkt\_ptr);

**if (tcpprop.http\_doc==INDEX\_HTML)**

**{**

**memcpy((void\*)tcp\_pkt->data,(void\*)(index\_htm+(tcp\_mss\*(tcpprop.cnt\_data\_part-tcpprop.cnt\_rem\_data\_part))-strlen(http\_header)),tcp\_mss);**

**}**

**else**

**{**

**memcpy((void\*)tcp\_pkt->data,(void\*)(e404\_htm+(tcp\_mss\*(tcpprop.cnt\_data\_part-tcpprop.cnt\_rem\_data\_part))-strlen(error\_header)),tcp\_mss);**

**}**

Further we calculate the length for the calculation of the checksum and call the function of preparing the TCP packet

  memcpy((void\*)tcp\_pkt->data,(void\*)(e404\_htm+(tcp\_mss\*(tcpprop.cnt\_data\_part-tcpprop.cnt\_rem\_data\_part))-strlen(error\_header)),tcp\_mss);

}

**len=len\_tcp + tcp\_mss;**

**tcp\_header\_prepare(tcp\_pkt, port, TCP\_ACK, len\_tcp, len);**

Then we prepare the IP packet, then the Ethernet packet, the last one is sent to the client

tcp\_header\_prepare(tcp\_pkt, port, TCP\_ACK, len\_tcp, len);

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

**ip\_header\_prepare(ip\_pkt, ip\_addr, IP\_TCP, len);**

**//Заполним заголовок Ethernet**

**memcpy(frame->addr\_dest,tcpprop.macaddr\_dst,6);**

**eth\_send(frame,ETH\_IP,len);**

Reduce the remaining number of packets for transmission to the client

eth\_send(frame,ETH\_IP,len);

**//будем считать, что ещё одну часть отправили, поэтому количество оставшихся частей декрементируем**

**tcpprop.cnt\_rem\_data\_part--;**

And at the end of the function, we determine whether the next packet will be in our possession or not, in accordance with this we will take the necessary solution

  tcpprop.cnt\_rem\_data\_part--;

**if(tcpprop.cnt\_rem\_data\_part>1)**

**{**

**tcpprop.data\_stat=DATA\_MIDDLE;**

**}**

**else**

**{**

**tcpprop.data\_stat=DATA\_LAST;**

**}**

  return res;

}

Let's prepare now a page that, together with the header, will fit into three packages, and we will change our array with the main page

**index\_htm 3 packages**

We will collect the code, let's check the controller and check that our page is displayed in the browser to the client's request. I will not show it here, I think you will take my word for it.

Now we will do the same with the pages in 4, 5 and 6 parts (packages).

I will list the compositions of their arrays

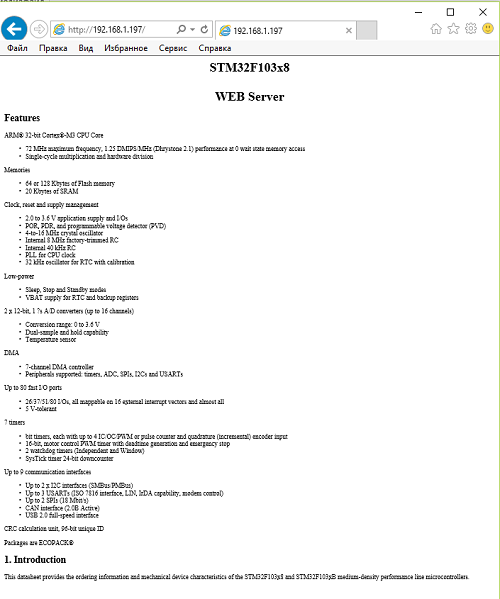
**index\_htm 4 packages**

**index\_htm 5 packets**

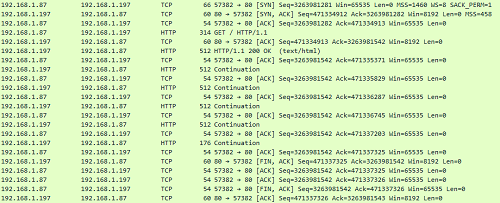
**index\_htm 6 packets**

After each change, we check the result in the browser and in Wireshark.

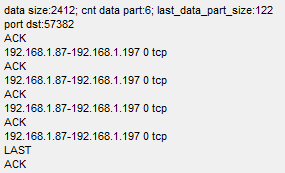
And here is our biggest page, which consists of 6 packages (click on the image to enlarge the image)

[](http://narodstream.ru/wp-content/uploads/2017/08/image05-1.png)

Let's see also the process of transferring the client of our last document to WireShark (click on the picture to enlarge the image)

[](http://narodstream.ru/wp-content/uploads/2017/08/image06-1.png)

Also we will make sure that our document consists of 6 parts, looking into the output of data in the terminal program



Thus, in our lesson we managed to transfer, using the HTTP protocol, a document consisting, it is possible to say from an unlimited number of packages. Also, without disrupting the traditions, we still slightly optimized our project, which saved us some memory, and also made our project more readable.

Thank you for attention!