Management of Digital Streams of an Autonomous System by the Raw Socket Ethernet Channel Virtualization Method in Linux OS

Victor Tikhonov[1], Serhii Nesterenko[2], Olena Tykhonova[1], Oleksandra Tsyra[1], Olha Yavorska[1] and Vladyslav Hlushchenko[1]

[1]*Department of Computer Engineering and Information Systems, State University of Intelligent Technologies and Telecommunications, Kuznechna Str. 1, Odesa, Ukraine*

[2]*Odesa Polytechnic National University, Shevchenko Avenue 1, Odesa, Ukraine*

{victor.tykhonov, elena.tykhonova}@suitt.edu.ua, sa_nesterenko@ukr.net, aleksandra.tsyra@gmail.com, yavorskayao7@gmail.com, vlad.gluschenko.97@gmail.com

Keywords: Autonomous System, Digital Stream Management, Channel Virtualization, Ethernet, Raw Socket, Linux.

Abstract: The issues of multimedia data transfer in packet-based networks have been considered in this work. Known approaches to digital streams management in autonomous systems studied with respect to the Internet of things traffic requirements and sensor networks real time telemetry provisioning. An original method of telecommunication channel virtualization presented based on Ethernet Raw Socket technique. An algorithm of digital flows integrating described for data multiplexing by conveyor transporting modules which payload the 802.3 Ethernet frame instead of conventional IP-packets. Computer model exhibited for simulation the process of joint multimedia data transmission via Ethernet frames in the form of four Python scripts under the Linux OS. Scripts SEND and RECEIVE implement the physical layer of Raw Socket interface. The scripts MUX and DEMUX perform multiplexing and de-multiplexing of integrated multimedia data on the data link layer. The results of the work intended for next generation networks application and Big Data distributed systems.

1 INTRODUCTION

The problem statement and the research purpose.

The global telecommunication industry has achieved outstanding success, which the main is the modern Internet. According to the International Telecommunication Unit (ITU) concept, a next generation network (NGN) should integrate multimedia services based on the Internet Protocol (IP) Multimedia Subsystem (IMS) platform. This concept considers a certain compromise between the existing information infrastructure of the Internet and new challenges of increasing quality of service (QoS) requirements, as well as the quality of content perception (QoE) [1]. One of the IMS concerns is ensuring the QoS aware transmission of real-time traffic (audio/video conferences, digital telemetry of sensor networks, etc.). This issue occurs due to stochastic data delay fluctuations while transporting IP-packets, as well as packet loss due to communication channel congestions. The above factors result in voice/video deterioration, end user connection instability, and slows down the data exchange rate in real-time control systems [2].

The improvement of NGN transport is carried out by coherent optical communication (COC) technologies and multiprotocol label switching (MPLS) transport profile (MPLS-TP), enhanced data flow control based on modern network operation researches. Among them, the software defined networking (SDN) concept has being actively developed in recent years. The SDN technology provides virtualization of given network physical topology within an autonomous telecommunication transporting system (AS) by network resources dynamic reconfiguration in accordance with the current data flow traffic. The overall data link capacity is virtually split to parallel channels by central controller of AS. The SDN networks studied by domestic and foreign scientists, and significant results have been achieved in this direction [3-10]. Enhancement the IMS-traffic transmission has been

also studied under the concept of multipath routing [11, 12]. Known approaches for telecom network and systems management mainly focus given potentials of embedded technologies and standard protocols. The QoS real-time data delivering means network resources reservation by virtual connection establishment among terminal network entities.

Though it goes beyond a distinct AS administration policy due to the fixed IP predetermined for IP-networks interoperability. Nevertheless, IP (in two standard versions IPv4 and IPv6) remains mandatory protocol for the Internet AS-to-AS interaction according to the NGN/ITU model, because of the most modern network services and applications are still based on IP. Thus, the real-time end-to-end data transport via the multilayer IP-packet encapsulation suffers an excessive overhead and the lack of efficiency. So, novel principles needed for multimedia IP packet-data transporting to overcome emerged challenges.

Given this fact, a fairly competitive mechanism to advance QoS aware end-to-end multimedia traffic transmission in the next few decades will be enhancement the local packet transfer function within any distinct autonomous transporting system, and first of all, at the L2/OSI data link layer [13].

The most common data-link-layer interfaces in modern digital network devices support either wired twisted pair Ethernet or/and Wi-Fi radio-Ethernet. To our opinion, exactly the Ethernet standards' utilization is one of the most promising direction for further development the Internet packet-based transport function for real-time multimedia applications.

The Ethernet/Wi-Fi technology operates with protocol data units (PDU) of three main types: line coding symbols (elementary protocol data units), block coding symbol groups (conventional "bytes" ranging in size from 2 to 8-12 or more bits) and, finally, frames channel coding. These three coding types and their corresponding protocol data units (PDU) define de facto three conditional data-link layer sublayers. The first two of them (line coding and block coding) are usually referred to as the so-called MAC sublayer - the access to the media-environment data transmission (twisted copper pair, fiber optic or radio broadcast). Ethernet frames are an object of the LLC sublayer - the control of data transfer from the data link to the network layer. There are a fairly large number of Ethernet technology standards supported by most network equipment manufacturers.

IEEE 802.3 is considered the basic standard that regulates the structure of an Ethernet frame. An Ethernet 802.3 frame, from the point of view of the LLC sublayer, has a header of 3 fields (the 6 octets frame destination MAC-address in the Ethernet local network; the 6 octets source MAC-address; the 2 octets "Type/Length" field); payload field of 46 octets minimal and 1500 octets maximal size; the 4-octets frame checksum field, which is often included into the frame header. In general, the overhead fields of an Ethernet frame at the LLC sublayer are 6+6+2+4=18 octets. If the frame payload is less than 46 octets, the payload field must be padded to 46 octets with non-content buffer information (padding).

In addition, two more service fields are added to the Ethernet frame at the MAC sublevel: preamble for physical synchronization of the frame receiver with the transmitter symbol sequence (7 octets - 56 bits of the form 101010...10); Frame Start Delimiter (FSD) in the form of octet 10101011. The appearance of the last 1 in the FSD octet means that the next character is the first bit destination MAC-address. Between any two consecutively transmitted Ethernet frames an Inter Frame Gap (IFG) must be inserted in the form of the signal absence for a period of at least 12 conditional octets; the physical size of the IFG depends on the nominal standardized bitrate in the Ethernet channel (starting from 10/100 Mbps and further up to 1/10/100 Gbps or more).

The typical payload of an Ethernet frame is a standard IP-packet. However, for the virtualization of communication channels at the L2 level, a non-trivial task arises of splitting an integral frame into separate virtual parts (transport containers) of variable length within the overall standardized frame payload size of 1500 octets. Some network operating systems (in particular, Linux) provide for the possibility of non-standard use of Ethernet frames and encapsulated protocols at different OSI levels using the so-called sockets (logical connections). In particular, an applied programmer is granted to artificially generate the TCP/UDP transport segments, IP-packet headers, and Ethernet frames as a whole. The non-trivial frames generation on the transmitting side of the channel (and the non-standard interpretation of these frames on the receiving side) corresponds to the lowest level of programming specialized application interfaces (the so-called Raw Ethernet sockets). At the same time, for the normal operation of Raw Ethernet sockets, it is necessary to correctly form the destination MAC-address (DMAC), as well as the source MAC-address (SMAC) in the first 12 octets of the frame header. The other two fields of the standard Ethernet frame (2 octets of the Type/Length field and 4 octets of the checksum field) may be freely formed and added to the frame payload [14].

The objective of this work is further developing of digital flows management based on multimedia data integration via the scalable LLC-sublayer multiplexing and routing-less IP-packets transportation through an autonomous system.

To achieve this, the task was set to jointly transfer data of different types with different sender and recipient addresses in one raw socket Ethernet frame. To experimentally verify this approach, a software package was developed in the python/Linux environment, which consists of four main modules: SEND – the raw socket cyclic frame transmission; RECEIVE – the raw socket periodic frame reception; MUX and DEMUX – the integrated stream segments multiplexing and demultiplexing. For testing the actual physical transmission of raw socket frames, two types of interaction are implemented: the master-slave Ethernet link via the 8-wire copper twisted-pair cross cable patch-cord, and radio-link via the Wi-Fi access point. The more detail results of this study are discussed below.

2 INTEGRATION THE MULTIMEDIA STREAMS IN VIRTUAL CHANNELS OF RAW SOCKET ETHERNET

The Figure 1 shows the principles of multimedia data integration over a packet network by virtualizing an IP-over-Raw Socket Ethernet link. The two OSI levels are presented here: the L3 network layer (IP-packets queue of three different multimedia streams F1, F2, F3), the L2 data-link layer （divided into two Ethernet technology sub-levels MAC and LLC). The data blocks of three different IP-packet's payloads are formed by the segments DS1, DS2, DS3 (each can include any arbitrary part of the corresponding IP-packet). Besides, the correspondent command segments CS1, CS2, CS3 are formed for individual treatment of these three data streams. The sizes of DS payload segments and CS command segments are variable.

Further, at the L2 LLC sub-level, a queue of segments (CS, DS) of individual streams is reshaped into payload transporting modules of a given size (Payload in Figure 1) within the allowable payload of an Ethernet frame (it was indicated above that this size can exceed 1500 octets due to the simplified Raw Header Socket Ethernet). The method of converting an IP-packets queue into a sequence of Payload_TM transport modules can be chosen arbitrarily depending on the individual requirements for the individual QoS requirements of multimedia streams. Finally, at the L2 MAC layer, the Payload_TM transport modules fit into Raw Socket Ethernet frames.

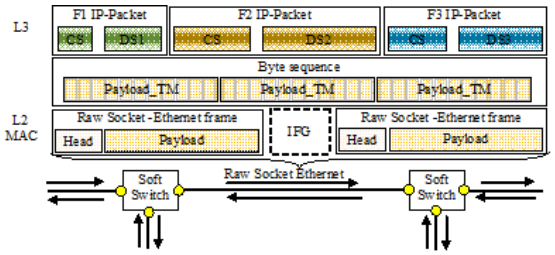

Figure 1: Transfer of multimedia traffic over the network IP/Raw Socket Ethernet.

The Raw Socket Ethernet frames circulate in both directions of a duplex link with 802.3 Ethernet network interfaces (or Wi-Fi radio-Ethernet); at the same time, instead of conventional Ethernet switches of the L2/OSI level, it is proposed to use modified flexible Soft-Switch switches to support the conditional L2.5/OSI level. The peculiarity of such a Soft-Switch is that an autonomous system enabled for faster interior IP-packets transfer by simple switching the separate DS units over the Raw Socket Ethernet virtual channels using the CS-labels; therefore, no routing function is needed more within the given autonomous system.

This way of transmitting IP packets is like MPLS technology (Multi-Protocol Label Switch) for the fast packet data transmission within an MPLS domain. However, the Ethernet over MPLS frame requires an additional header with a flow switch label (placed before the main frame header). Beyond, any conventional frame of Ethernet/MPLS solely contains a single IP-packet.

This fact is critical when transporting small IP-packets in real time applications and services (IP telephony, telemetry traffic in IoT sensor networks, etc.), since it significantly reduces the information efficiency of communication channels due to excessive header redundancy. The Pv6 packet redundancy with 40 octets header is even more sensitive.

Figure 2 shows structuring the overall data sequence into distinct command segments CS and data segments DS. For this, two 8-bit meta-commands are used: the command separator "11111111" (the "FF" byte in the hex-code); data separator "00000000" (byte "00" in hex-code). To recognize bytes "FF" or "00" in data segments, the byte-stuffing mechanism is used: two commands

("FF01" and "FF02") are reserved to replace the data characters "FF" and "00".

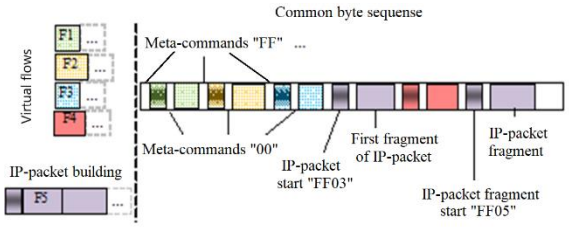

Figure 2: Structuring a data sequence by virtual channels of a frame Raw Socket Ethernet.

The packet fragmentation and defragmentation mechanism assume that any IP-packet of the incoming multimedia streams queue is divided into fragments of arbitrary size due to the three reserved 16-bit commands: packet start (byte "FF03" in the 16-bit code); packet end (byte "FF04"); packet fragment start (FF05). The packet fragment start command is removed after the packet arrives the input queue of the adjacent switch; the packet start/end commands are remaining. With these commands, the flow analyzer extracts individual packets from the queue.

3 COMPUTER SIMULATION THE INTEGRATED DATA STREAM TRANSMISSION

As mentioned above, the physical layer of the computer model for transmitting integrated multimedia streams is built since a special mode of operation of the standard Ethernet 802.3 technology (the so-called "raw sockets"). In this mode of operation, the frame structure, in particular its header, is formed in accordance with the requirements of application interface developer. To do this, each network adapter being interacting with any other one through the Raw Socket software interface must support the four software modules: SEND (transmitting Raw Socket frames to an Ethernet channel); RECEIVE (receiving/interpreting the frame); MUX (multiplexing the multimedia data flows); DEMUX (demultiplexing the multimedia data flows). The SEND and RECEIVE modules operate on the physical layer. Both modules can operate independently at one computer in two Linux terminals. The MUX/DEMUX operate on L2/OSI, All the 4 modules coded in Python/Linux [13, 14].

Figure 3 briefly shows the algorithm of SEND program module. The first block of the diagram imports the necessary modules (*time*, *socket*, *binaskii*) and creates the socket itself (*socket ()*).

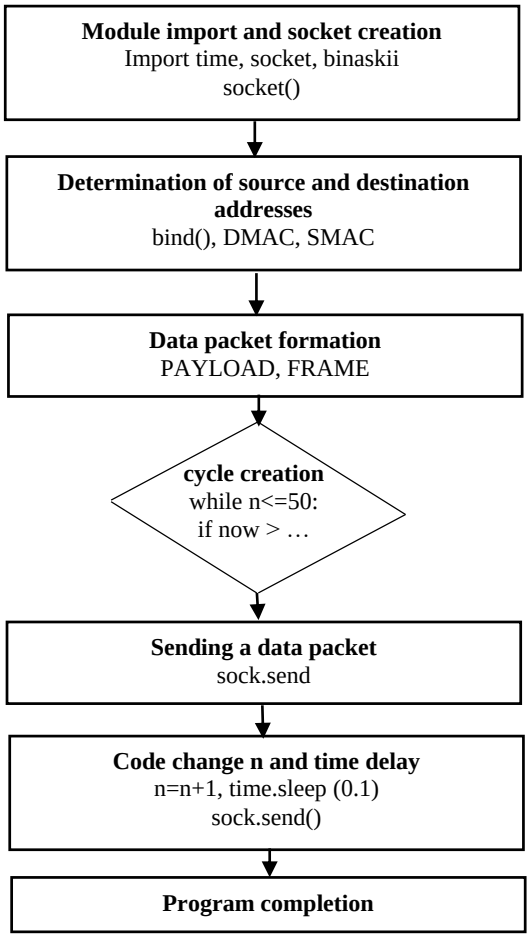

Figure 3: The algorithm of the SEND program module.

The second block of the algorithm determines frame transmission parameters (interface identifier, S/D MAC addresses). The third block forms the payload of the frame, as well as the structure of the entire frame. Next, a loop is created to repeatedly transmit the frame a certain number of times (for example, 50 times). This option is used exclusively at the stage of offline debugging and testing of the SEND/RECEIVE modules. For complete simulation the software model including the MUX/DEMUX modules, the SEND module performs a modified frame payload for each loop iteration.

The RECEIVE software module allocates the Raw Socket frame into memory buffer and extracts its Payload (which was generated on the transmitting side by the MUX module for multiplexing data streams). The payload content is further handled by the DEMUX module on the receiving side.

Briefly consider the key operators of the MUX software module. The first block of statements defines the parameters of the conveyor transporting modules (CTM) that is payload for the Raw Socket frame, as well as the initial value of the iteration variable *n*. Next, the operator

fin=open("/home/user/Desktop/InPack.txt",'r')

opens file *"/home/user/Desktop/InPack.txt"* for reading; this file contains a sequence of packets. The operator

fout=open("/home/user/Desktop/OutPack.txt",'w')

opens the file *"/home/user/Desktop/OutPack.txt"* for writing; this file contains CTM modules generated for output.

The operator *x=fin.read(1)* sets the read pointer in the file fin to the first position. The operator *while n<=6:* opens a loop of iterations over an iteration variable *n* from *n=1* (specified above) to 6; the iteration loop body is determined by the same offset of all first line loop operators.

The last statement in the loop is *n=n+1*. The *L=2* operator defines the initial length of the real-time data block *L=2*; this value is due to the fact that each CTM necessarily contains a two-character tag *"C0"* (end of real-time data block).

The operator *print '\n CTM='"%2d"%(n),* prints the current CTM value on the monitor. The script *\n* means jump to a new print line; the script *"%2d"%(n),* formats printing the module number *n* as an integer in two positions; comma at the end means that the next print statement will output the data at the same line.

The operator *fout.write('\n CTM='"%2d"%(n),)* prints CTM module number to the source file *OutPack.txt* with internal virtual name *"fout"*.

The Figure 4 shows the module DEMUX script for demultiplexing aggregated streams. The operator

f = open ("/home/user/Desktop/ OutPack.txt", 'r')

opens file *OutPack.txt* for reading; its internal is taken *"f"*.

The operator *x=f.read(1)* sets the file pointer of *"f"* to the first character of this file; the initial length of the character string *x* equals zero (i.e. the string of characters *x* is empty).

The operator *print ' '*, displays a space on the monitor screen (comma at the end means that the next print will be carried out on the same line through one space).

The operator *while len(x) > 0*: opens the main iteration loop until the size of the character string *x* is greater 0 (that is, until all non-empty characters in the open file *OutPack.txt*, denoted by *f* in the module, run out). The operator *x=f.read(1)* reads one character from the current position of the pointer (in this case, from the first position); the value of this character is assigned to variable *x*.

The operator *if (x == 'C'):* checks whether *x= "C"*; if so, then a following code executed:

```
f.seek(-1,1)
x = f.read(2)
if ((x == 'CA') or (x == 'C0')).
```

In this block, the operator rolls back the pointer by one position (i.e., sets the pointer to the first character of the file again). Next, operator *x = f.read(2)* reads two characters from this current pointer position. Finally, the operator *if ((x == 'CA')* or *(x == 'C0'))*: checks if these two characters are mark-up tags of type CA (beginning of packet) or C0 (end of real-time data block in CTM). If the "CA" or "C0" tag is indeed read, then this tag is printed on the monitor screen.

```
# Begin DEMUX
f=open("/home/user/Desktop/OutPack.txt", 'r')
x=f.read(1)          #set file pointer to the first symbol
print ' ',
while len(x)>0:
   #Do iteration cycle until len(f.read(1))>0,
   #e.g. reads    file to the end
   x=f.read(1)
   if (x=='C'):
      #print x,
      f.seek(-1,1)
      x=f.read(2)
      if ((x=='CA') or (x=='CO')):
         print x,
         for k in range (1,10):
         x=f.read(1)
         if x=='C':f.seek(-1,1); break
            print x,
```

Figure 4: Python script of the program module SEND.

Next, a built-in loop is introduced to read the following characters (which are the letters of the package): for *k* in range (1, 10): The increment *k* from *1* to *10* is taken arbitrary (consider the packet fragment size not exceed 10 characters).

The operator *if x == 'C': f.seek(-1,1);* checks whether the next character read is reserved letter *"C"*; if not, the next character is printed on the monitor screen (operator *print x,*); comma at the end means the following characters will be printed side by side on the same line. If the character *"C"* is read, then the file pointer is shifted for one position back, and the built-in loop for *k* in range (1, 10) ends by command *break*. Then the main loop continues (*while len(x) > 0:*). In this case, the operation of

reading characters from the file *f* is carried out with the current pointer (which gradually moves until the end of the file is reached); under these conditions, the main loop ends.

3 CONCLUSIONS

Novel principles proposed for digital streams management in autonomous telecommunication transporting systems at the OSI data link layer. This approach aims further improvement the information efficiency of network equipment and communication channels utilization, as well as to meet increased QoS requirements for real-time applications.

To achieve this, the original method of digital streams management in autonomous systems introduced on the base of multimedia data integration on Logical Link Control sublayer by virtualization the Raw Socket Ethernet channel.

The method provides scalable multiplexing and switching-mode IP-packets transportation via an autonomous system avoiding the interdomain routing processes. Computer model of integrated multimedia streaming presented on Linux Python. The results of the work intend for next generation networks application in the concept of the Internet of Things.

REFERENCES

[1] G. Bertrand, "The IP Multimedia Subsystem in Next Generation Networks," Gilles Bertrand, 2007. [Online]. Available: http://www.rennes.enst-bretagne.fr/ ~gbertran/files/ IMS_an_overview.pdf.

[2] V. Hol and M. Irkha, "Telekomunikatsiini ta informatsiini merezhi: navchalnyi posibnyk," Kyiv: ISZZI KPI im. Ihoria Sikorskoho, 2021, 250 p.

[3] M. Mousa, "Software Defined Networking concepts and challenges," 2016. [Online]. Available: https://www.researchgate.net/publication/312569297_Software_Defined_Networking_concepts_and_challenges.

[4] P. Vorobiienko, L. Nikitiuk, and P. Reznichenko, "Telekomunikatsiini ta informatsiini merezhi: Pidruchnyk [dlia vyshchykh navchalnykh zakladiv]," K.: SAMMIT-Knyha, 2010, 708 p.

[5] L. Kriuchkova, "Metodolohiia Upravlinnia infokomunikatsiŷnymy merezhamy zi zminnymy strukturamy v umovakh destruktyvnykh vplyviv," Dyssertatsyia na soyskanye uchenoi stepeny doktora tekhnycheskykh nauk po spetsyalnosty 05.12.02. Hosudarstvennyi unyversytet telekommunykatsyi, Kyiv, 2017.

[6] M. Klymash, O. Shpur, V. Bahrii, and A. Shvets, "Metod dyferentsiiovanoho multypotokovoho keruvannia trafikom u transportnykh prohramno-kerovanykh merezhakh," Natsionalnyi universytet Lvivska politekhnika. Radioelektronika ta telekomunikatsii, no. 796, 2014, pp. 60-68.

[7] O. Aouedi, K. Piamrat, and B. Parrein, "Intelligent Traffic Management in Next-Generation Networks," Future Internet 2022, 14(2), 44; [Online]. Available: https://doi.org/10.3390/fi14020044.

[8] M. Al-Jameel, T. Kanakis, S. Turner, A. Al-Sherbaz, and W. Bhaya, "A Reinforcement Learning-Based Routing for Real-Time Multimedia Traffic Transmission over Software-Defined Networking," Electronics, 2022. [Online]. Available: https://doi.org/10.3390/electronics11152441.

[9] M. Yanev, S. McQuistin, and C. Perkins, "Does TCP new congestion window validation improve HTTP adaptive streaming performance?" NOSSDAV'22, June 17, 2022, Athlone, Ireland, pp. 29–35. [Online]. Available: https://doi.org/10.1145/3534088.3534347.

[10] M. Seliuchenko, "Modeli ta alhorytmy pidvyshchennia yakosti obsluhovuvannia u telekomunikatsiinykh prohramno-konfihurovanykh merezhakh," Dysertatsiia na zdobuttia naukovoho stupenia kandydata tekhnichnykh nauk: 05.12.02 – telekomunikatsiini systemy ta merezhi; Lviv, 2016, 156 p.

[11] M. Dibrova, "Sposib bahatoshliakhovoi marshrutyzatsii v kompiuternykh merezhakh velykoi rozmirnosti," Avtoreferat dysertatsii, KhNURE, 2017.

[12] O. Lemeshko and K. Arus, "Model vidmovostiikoi marshrutyzatsii z realizatsiieiu riznykh skhem rezervuvannia resursiv merezhi v umovakh multypotokovoho trafiku," Materialy vseukrainskoi naukovo-praktychnoi konferentsii "Suchasni problemy Telekomunikatsii i pidhotovka fakhivtsiv v haluzi telekomunikatsii - 2014" SPTEL, 2014, Lviv: NU "Lvivska politekhnika", 2014, pp. 15-20.

[13] V. Tikhonov, O. Tykhonova, O. Tsyra, O. Yavorska, A. Taher, O. Kolyada, S. Kotova, O. Semenchenko, and E. Shapenko, "Modeling the conveyor-modular transfer of multimedia data in a sensor network of transport system," Eastern-European Journal of Enterprise Technologies, 2018, vol 2, no. 2 (98), pp. 6-14.

[14] V. Tikhonov, E. Tykhonova, O. Yavorskaia, V. Berezovskyi, A. Nehalchuk, and V. Hlushchenko, "Postroenye symuliatora dlia peredachy dannykh telemetryy cherez modyfytsyrovannyi ynterfeis Ethernet," Materialy 73 NTK profesorsko-vykladatskoho skladu, naukovtsiv, aspirantiv ta studentiv (Odesa, 12-14 December 2018), pp. 100-103.