

**SIMULATION THE ALGORITHM OF MULTIMEDIA DATA  
INTEGRATION IN PACKET BASED DIGITAL CHANNEL**

*This paper presents the algorithm and simulation model for conveyor-module method of multimedia data integration designed for a packet based digital channel. The latency control issues considered for real time segment transition along with dynamic packet fragmenting. An algorithm of syntax analysis applied towards the endless multiplexed data flow mapped on a symbol chain of an abstract grammar. The simulation model performed on Python in Linux Ubuntu operating system.*

*Keywords: multimedia data, packet channel, abstract grammar, algorithm simulation.*

В. ТИХОНОВ, А. ТАХЕР, Е. ТИХОНОВА  
Одесская национальная академия связи им. А.С.Попова

**МОДЕЛИРОВАНИЕ АЛГОРИТМА ДЛЯ ИНТЕГРАЦИИ  
МУЛЬТИМЕДИЙНЫХ ДАННЫХ В ЦИФРОВОМ ПАКЕТНОМ КАНАЛЕ**

*В статье описаны алгоритм и модель симуляции для метода конвейерно-модульной интеграции мультимедийных данных в цифровом пакетном канале. Рассмотрены вопросы контроля задержки при передаче сегментов реального времени совместно с динамической фрагментацией пакетов. Применен алгоритм синтаксического анализа для бесконечного мультиплексированного потока данных представленного цепочкой символов абстрактной грамматики. Модель симуляции выполнена на языке Python в операционной системе Linux Ubuntu.*

*Ключевые слова: мультимедийные данные, пакетный канал, абстрактная грамматика, алгоритм симуляции.*

**1. Introduction**

Advanced researches on next generation network (NGN) technologies face difficulties in time delay mitigation when transporting real time data (voice, video, telemetry etc.) over packet based networks. Conventional methods and related protocols of resource reservation (RSVP, NSIS) on IP layer constructed for IntServ/DiffServ models of QoS provision proved not enough scalable and relevant to meet new challenges, particular with respect to high dynamic object interaction in sensor networks and machine-to-machine architecture (M2M). Last years, a number of dedicated protocols emerged to overcome the aforesaid issues primarily designed on layer 2 of the open system interconnection reference model (OSI). To compromise packet and circuit switching techniques across an autonomously software driven network (SDN), an open flow protocol and related unified architecture have been originated in [1]. This framework implies that all the underlying switching hardware (packet and circuit switches) will be driven by an external control plane including network operating system with on top applications and open flow protocol to manage data flow tables of switches. The channel resource scheduling considered in [2] with time division switching fabric introduced based on a slotted random access bus. This method corresponds to the hybrid packet/time slotted circuit switched scheme (HPTS) presented in [3] where real time data segments follow a circuit path with no packet loss or jitter; instead, the other ones are following packet switched path being statistically multiplexed.

Deterministic unified Ethernet with asynchronous and synchronous traffic provision described in [4]. It operates using a global sense of time and a schedule which is shared between network components. This approach enables customer convergence of real-time controls messages with regular best effort packet data over the common Ethernet network. The time-scheduled information is located apart from other segments being immune from disturbance. This resulted in guaranteed latency of critical traffic. The five real-time techniques considered in [5] based on Ethernet: Ethernet-Powerlink (EPL), Ethernet/IP, Sercos-III, Profinet-IRT and EtherCAT. The real-time methods differ in their tolerance to Ethernet standard. Such hardware driven tools as Profinet-IRT or EtherCAT use special FPGA which are not typical on conventional Ethernet controllers. Conversely, EPL or Ethernet/IP use standard hardware platform. It was concluded that non-conformance to the Ethernet TCP/IP standard is not necessarily consequence while developing real-time Ethernet systems. To provide an adequate overall response time it is quite enough to benefit standard mechanisms as switched full-duplex in Fast or Gigabit Ethernet along with UDP, QoS data prioritization, VLAN segmentation and time synchronization via IEEE1588 specification. *However, new demands of dynamic resource management in distributed automated control systems require further investigations. This work aims to design and simulate a uniformed algorithm of multimedia data integration in an arbitrary packet based channel to provide an effective latency control for critical traffic.*

**2. Conveyor-module method of multimedia data integration**

According to the TCP/IP protocol suite encapsulation, a protocol data unit (PDU) on the OSI layer 2 (L2) contains an L3 packet as a payload. For instance, the Ethernet frame carries IP-packet or its fragment allocated within maximal slot of 1500 bytes; the ATM cell bears a payload data segment of constant size 48 bytes. The

multilayered encapsulation scheme causes excessive overhead traffic and channel performance underuse. To improve the throughput utilization, an enhanced method of conveyor transporting modules proposed in [6]. The figure 1 shows the principle of this method. The sending party schedules transmitting multiproduct data segments onto the three queues of bytes:

- real-time flow (RTF) provided by reserved channel bandwidth and virtual circuit labels;
- logical link flow (LLF) provided by logical channel labels without resource reservation;
- IP-packet data flow (IPF) compliant with IP-protocol.

The common byte sequence consolidates the segments and/or fragments of these products. This sequence further partitioned into transporting modules due to the size of payload field for the data link layer technology (e.g. Ethernet, ATM, FR etc.). Fragments of each data type in resulting blocks should be delimited by predefined markup tags. These blocks are transferred to the adjacent network object by the data link layer frames. The receiving network node extracts payloads content from the incoming frames and forms separate streams from data fragments of 3 types. Each of the resulting streams is serviced by the corresponding rules: an IP packets stream is routed in accordance with the IP-protocol; the segments of RTF and LLF streams are forwarded by virtual and logical label switched paths in accordance with special protocols, which is supposed to expand the network layer: virtual circuits switching protocol (VCP) for real time data flow (RTF) and logical channels switching protocol (LCP) for non-real time logical data flow (LDF).

The data segments of all types thus forwarded to the appropriate outgoing ports of the node, where, in turn, they are organized in three different queues. This splits conventional IP-layer on three sub-layers: IP sub-Layer (IPL), Real Time data sub-Layer (RTL), Logical link Sub-layer (LSL). The essence of VCP is that any virtual connections for RTF transfer will solely established if required channel capacity available; again, the VCP primarily forwards real time data into the channel. Thus, real time data queue will never get full, and therefore, RTF data never lost or indefinitely waiting for service. The LDF and IP-packet queues are served under the “best-effort” policy while LDF uses the quick logical channel switch in contrast to IP-packet routing.

Conventional IP-traffic (including traditional VoIP, multimedia over IP, TCP etc.) is processed at every node as usual in packet-switching mode. The real-time traffic is handled in quasi-circuit-switching mode simulated due to the dynamically established virtual connections and resource reservation. The Logical Link Control 2 (LLC2) protocol also establishes a data link layer connection by sending a corresponding request and receiving confirmation, after which the data is transmitted.

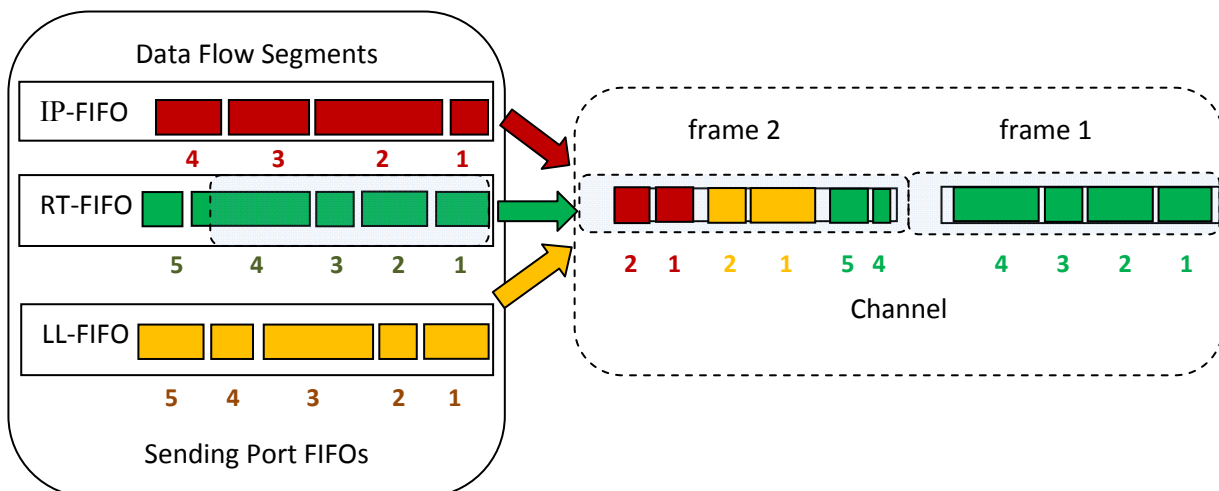


Figure 1 – The diagram of conveyor-module queuing

### 3. The algorithm of conveyor-module method

To simulate the conveyor-module queuing, the following simplified algorithm proposed, Fig. 2. The real time data segments of RT-FIFO are generated within the main iteration cycle. Three types of RTF segments are defined; these segments are to be included into circulated conveyor transporting modules (CTM). The size of CTM is optional (i.e. 24 hexadecimal symbols). The first real time flow segment (RTF1) has repeating factor  $K1=1$  (e.g. must appear in any CTM); the RTF2 factor is 2 (appears in any second CTM module); the RTF3 factor 3 lets allocate this type of segment at each third CTM module. The sizes of RTF modules optionally defined as  $D1=2$ ;  $D2=4$ ;  $D3=6$  hexadecimal symbols. The iteration variable is predefined as  $n=1$ ; the number of iteration is optional 6.

The packet data of IP-FIFO is simulated in the text file which opened for reading by the nickname “fin” (file for input). The count variable  $L$  accumulates the overall number of symbols occupied by real time segments within any distinct line of output data, which has been written in output file with nickname “fout” (file for output). Each output line simulates a distinct conveyor transporting module (CTM). At the beginning of any iteration the variable  $L$  is set to value 2; this correlates with ongoing writing two-symbol header of the first in line real time segment (i.e. C1, C2, C3). For ease data interpretation in output file, any written line begins with the current

iteration number of CTM= $n$  (i.e. CTM=1, CTM=2 etc).

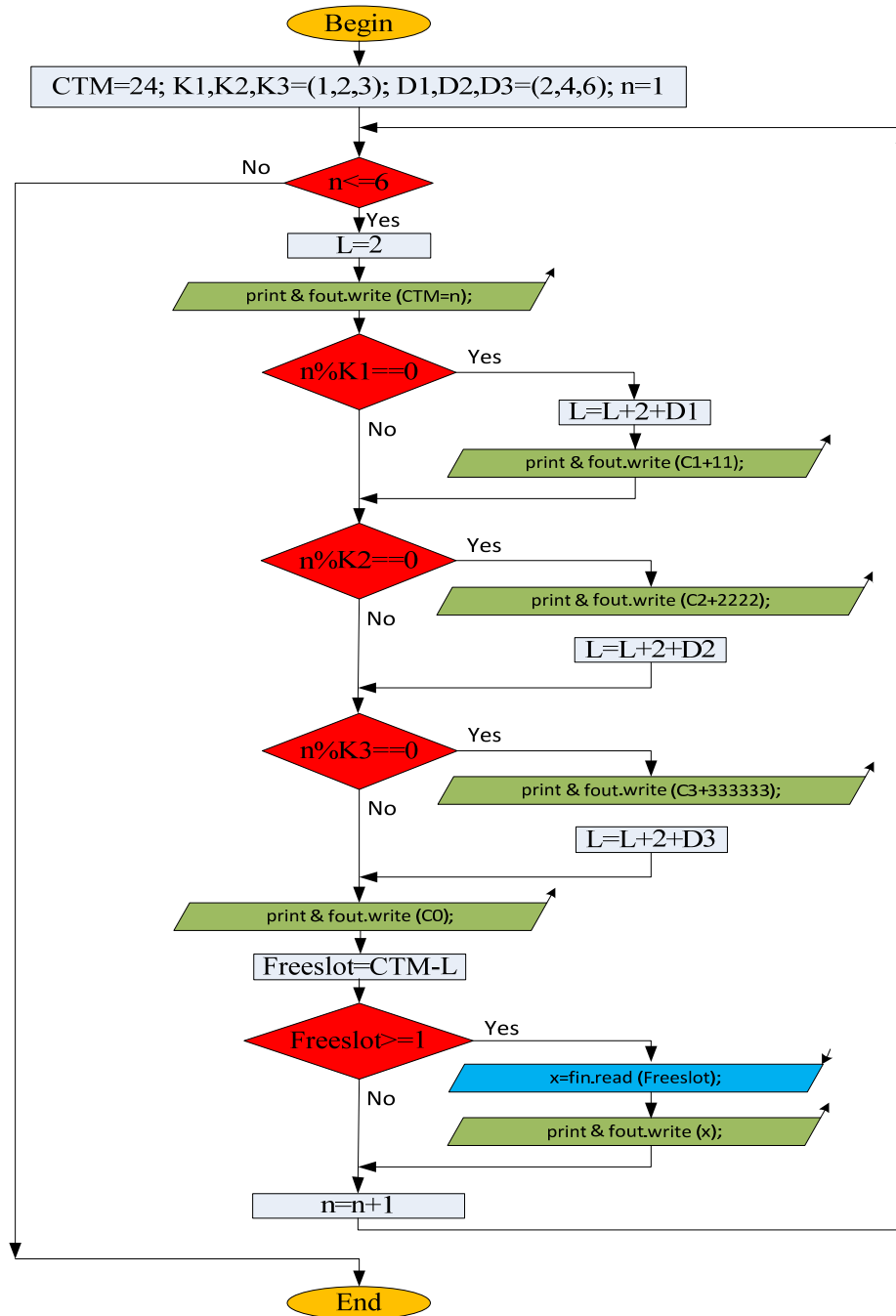


Figure 2 – The flowchart of conveyor-module algorithm.

The body of iteration cycle consists of two main parts:

Real time segment allocation (first three blocks “If”;

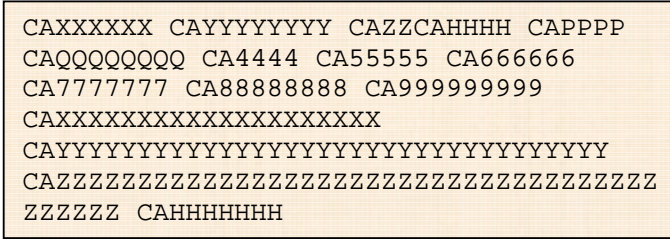
Packet data fragment allocation starting with the block “*print & fout.write (C0)*” up to the code “*n=n+1*”.

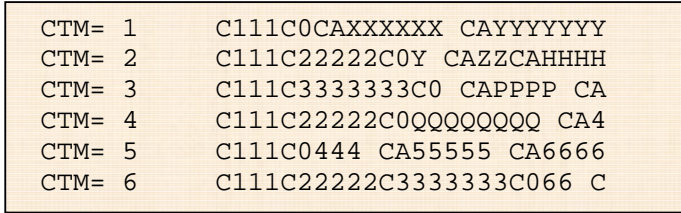
Any real time segment of RTF flow is checked for necessity to be inserted into the current CTM module; this operation performed due to the “rest” function, i.e.  $rest(n/K1) \rightarrow n\%K1$  (it is 0 when  $n=1, 2, 3, \dots$ );  $rest(n/K2) \rightarrow n\%K2$  (it is 0 when  $n=2, 4, 6, \dots$ );  $rest(n/K3) \rightarrow n\%K3$  (it is 0 when  $n=3, 6, 9, \dots$ ). These figures are valid for options:  $K1=1, K2=2, K3=3$ . For simplicity, three types of real time segments are defined as C111, C22222 and C3333333 respectively; the symbol combinations C1, C2 and C3 are segments headers. The symbol collections 11, 2222, 3333333 are segment bodies; these can be arbitrary varied in semantic and length. After printing a segment into the output file *fout.write* the count variable L incremented in the number of printed symbols (i.e.  $L=L+2+D3$ ) where 2 positions consider the header C3, and the D3 is the length of the RTF segment number 3.

The packet data allocation starts with the printing delimiter command C0 to separate packet data from real time segments within a conveyor transporting module CTM. Next, the unallocated space in CTM is calculated as *Freeslot* variable:  $Freeslot=CTM-L$  (i.e. CTM size is,  $n=3$ ; so two segments of real time data inserted into the CTM: the first one and the third; the total occupied slot in the current CTM is  $(2+2)+(2+6)=12$  hexadecimal digits; now,  $Freeslot=24-12=10$ digits; so, 12 digit more can be dynamically allocated from the packet queue. If *Freeslot*

$\geq 1$ , then *Freeslot* symbols are taken from input file *fin* into the variable *x*:  $x = fin.read(Freeslot)$ ; next, these symbols are printed into the output file *fout*: *print & fout.write(x)*. Figure 3-a shows the content of the input file *fin* with the packet data queue. Packets are delimited due to the command *CA* at the beginning of any packet. Packets are delimited due to the command *CA* at the beginning of any packet. Figure 3-b shows the content of the output file *fout* with multiplexed data. Six conveyor transporting modules CTM are created, each contains 24 symbols (including spaces presented in the input file).

To integrate multimedia data (real time segments and packet data fragments) in CTM modules, the two-layered formal grammar is used. The first grammar layer is built on the alphabet of 16 hexadecimal symbols; among those, one symbol is syntax sign (symbol “C”, or command symbol), and 15 other symbols are grammar letters. On the second grammar layer, two-symbol combinations C0 – CF create 16 syntax signs of the layer two. The sign “CC” is meta-syntax sign (to substitute the syntax symbol “C” in the data sequence (while coding data, the “C” letter is substituted for “CC”); conversely, while decoding multiplexed data, any combination “CC” is substituted for letter “C” of the second layer alphabet. The other 15 syntax signs of layer 2 grammar are used as tags to mark up the text converting the sequence of letters into a semantically structured text formed by the words and sentences. To delimit distinct words in the text (i.e. fragments of packets), the tag “C0” is used. Tag “CA” delimits packets; tags C1–C8 delimit eight simple types of RTF segments; tag “C9” is used to extend the length of RTF segment headers.

a) 

b) 

**Figure 3: a) Input file *fin* with the packet data queue (IP-FIFO);  
b) Output file *fout* with multiplexed data**

```
# NUX script for multimedia data multiplexing
CTM=24
K1, K2, K3 = (1, 2, 3)
D1, D2, D3 = (2, 4, 6)
n=1
fin = open("/home/victor/Desktop/InPack.txt", 'r') # Open file InPack.txt for reading input data
fout = open("/home/victor/Desktop/OutPack.txt", 'w') # Open file OutPack.txt for writing output data
x = fin.read(1)
while n<=6:
    L=2
    print '\n CTM="%2d"%(n),          #          \n - carriage return (print in next line);
          fout.write('\n CTM="%2d"%(n),)
    if (n%K1 == 0):
        L=L+D1
        print '\t C1'+'11',          # function n%K1 is the rest of n/K1
          fout.write ('\t C1'+'11',)          # output data into the dfile opened as fout
    if (n%K2 == 0):
        print 'C2'+'2222',
          fout.write ('C2'+'2222',)
        L=L+D2
    if (n%K3 == 0):
        print 'C3'+'333333',
          fout.write ('C3'+'333333')
        L=L+D3
    print 'C0',
    fout.write ('C0',)
    Freeslot = CTM-L
    # print 'Freeslot=', Freeslot
    if Freeslot >= 1 :
        x = fin.read(Freeslot)          # captures first N characters in file f
        print x
        fout.write(x)
    n=n+1
fin.close()
fout.close()
```

**Figure 4 – The core fragment of the Python script for simulation algorithm.**

Figure 4 presents the core fragment of the Python script to simulate the algorithm of multimedia data integration in packet based digital channel. The simulation performed in Linux Ubuntu operation system.

### Conclusion

An algorithm proposed to simulate conveyor-module method of multimedia data integration designed for a packet based digital channel. The two-layered formal grammar applied for data multiplexing. Proposed algorithm realized in Python programming language and exercised in Linux Ubuntu operation system.

### References

1. S. Das, G. Parulkar, N. McKeown, P. Singh, D. Getachew and L. Ong, "Packet and Circuit Network Convergence with OpenFlow." (Last visited June 15, 2016) [http://yuba.stanford.edu/~nickm/papers/Openflow-OFC10\\_invited.pdf](http://yuba.stanford.edu/~nickm/papers/Openflow-OFC10_invited.pdf).
2. E. Iannone, Telecommunication networks, CRC Press, 2011.
3. S. Bjornstad, A. Kimsas. "Hybrid packet/time slotted circuit switched scheme (HPTS)", in Conference Proceedings Transparent Optical Networks (ICTON 2008), Athens, June 22-26, 2008, pp. 97-100 (Volume 3).
4. "Deterministic Ethernet. Time-Scheduled Ethernet Standards." (Last visited June 15, 2016) <https://www.tttech.com/technologies/deterministic-ethernet/?gclid=CLO0--WZjM0CFUsNewodS8UO2g>.
5. "Industrial Ethernet book." (Last visited June 15, 2016) <http://www.iebmedia.com/index.php?id=4878&parentid=63&themeid=255&showdetail=true>.
6. O. Tykhonova, "Method of real time data transmission with delay control over IP-network", Scientific works of ONAT n.a. O.S.Popov, No.2, 2014 pp. 207-213.

Рецензія/Peer review : 3.6.2016 р. Надрукована/Printed :27.6.2016 р.  
Стаття рецензована редакційною колегією