Type of the Paper (Original Paper)

Controlling data flows in computer networks

Ahmad AbdulQadir AlRababah[1]*

[1]Faculty of Computing and Information Technology in Rabigh, King Abdulaziz University, Rabigh 21911, KSA.

*Correspondence: e-mail: aaahmad13@kau.edu.sa ; ahd_68@yahoo.com

Abstract: In computer networks, loss of data packets is inevitable, in particular, because of the buffer memory overflow of at least one of the nodes located on the path from the source to the receiver, including the latter. Such losses associated with overflows are hereinafter referred to as congestion of network nodes. There are many ways to prevent and eliminate overloads; these methods, in the majority, are based on the management of data flows. A special place is taken by the maintenance of packages, taking into account their priorities. The article considers a number of original technical solutions to improve the quality of control and reduce the required amount of buffer memory of network nodes. The ideas of these solutions are quite simple for their implementation in the development of appropriate software and hardware for telecommunication devices.

Keywords: data transmission, data stream, input output buffers, telecommunication devices, data packets, blocks of memory, switching matrix, high priority packets, bitstaffing.

---

1. Introduction

One of the known ways to control the flow of data is explained in Fig. 1, on which a fragment of the computer network is shown and the trace of the data stream transmitted through it is indicated[1]. The packets are transferred from the node A - the data source (transmitter) to the node B - the data receiver through the intermediate nodes, for example, switches and / or M1-M3 routers[2].

2. Materials and Methods

Method 1: Control the flow of data adjusting the length of pauses between packets

Prototype Mode 1

In this example, the node M2 is overloaded, its input buffer memory (in the following, for brevity, the input buffer) is completely or almost completely filled with incoming data packets. New packages, at least some of them, are lost due to lack of free space in the buffer[3].

During the data transfer, the receiver notices a persistent shortage of arriving packets (for example, by tracking their sequence numbers) and sends a control packet containing the XOFF command to suspend the data stream to the data source A. The address of the data source is known to the receiver, since the data packets coming to it contain information about the addresses (or directly addresses) of devices A and B[4]. Sending requests for retransmission of lost packets is also sent.

When the XOFF command is received, the data source completely stops sending packets and resumes it, either after some time specified in the data exchange protocol, or after receiving the renewal of transmission from the XON command receiver [1].

This method has several drawbacks.

First, data flow control is quite crude (the flow is either there or it is not). The delay in the execution of commands can lead to unjustified idle of the transmitter and the periodic occurrence of new overloads, in which some of the packets[5], including those belonging to other flows, will be lost[6].

Secondly, with prolonged overload, the receiver sends the transmitter a series of identical stopping commands, which clogs the communication channel with a large number of repetitive service packets[7].

Thirdly, the commands for suspending the transmitter are generated by the receiver only if the number of packets rejected due to buffer overflows is large enough. Otherwise, if one responds to insignificant packet losses, the transmitter will receive and execute suspense commands without any special reason.

Fourth, the suspension of the transmitter increases the average and maximum packet delay on the route, which can reduce the quality of service (QoS) parameters specified in the contract between the user and the provider [8].

The idea of method 1

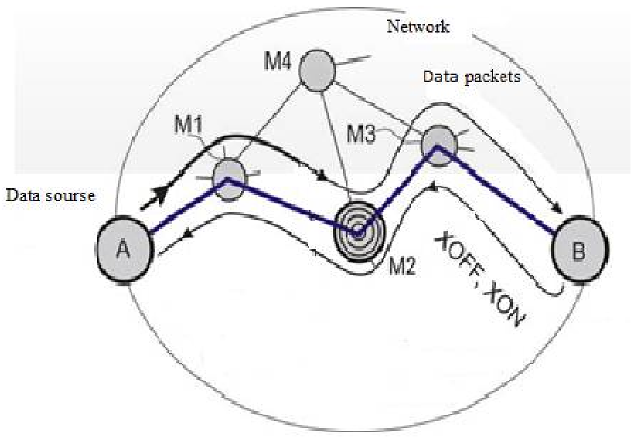

Figure 1. Traditional way to control the flow of data

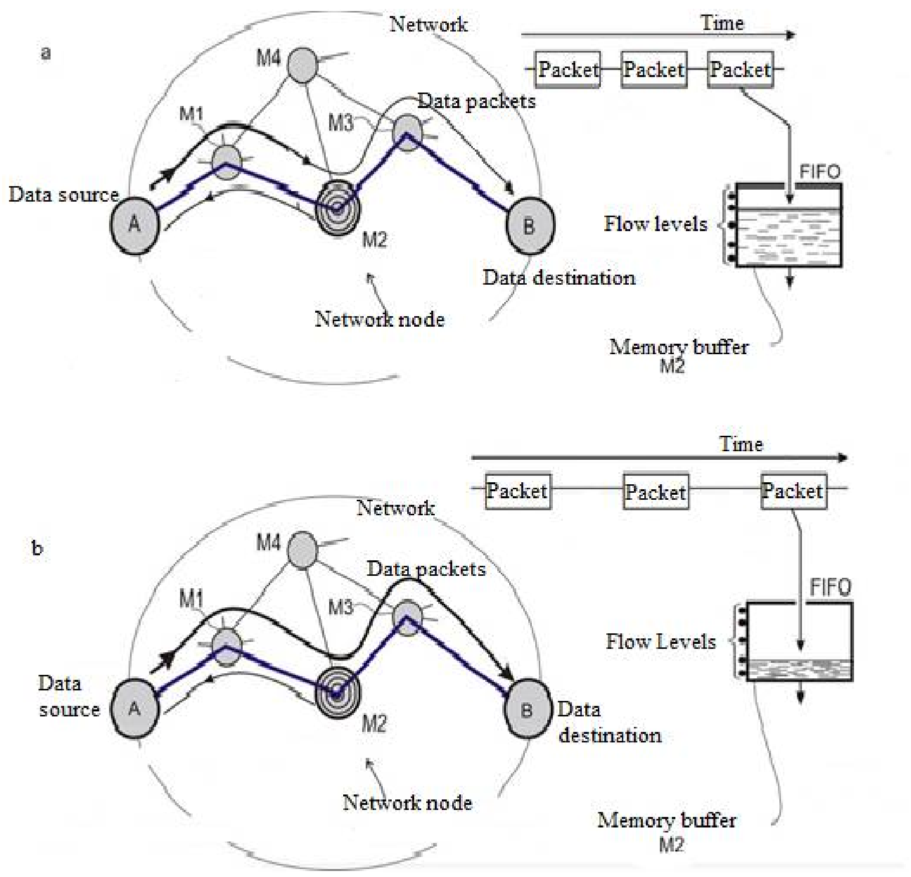

Figure 2. Improved way to control the flow of data: a - if there is a danger of overflow of the buffer memory of the node M2; b - during normal operation

The proposed solution (Figure 2) largely eliminates these shortcomings due to a smooth and "advanced event" adjustment of the data transmission rate by the source. The speed is controlled by changing the length of pauses between packets: the longer the pause, the lower the data transfer rate, and vice versa[9]. Note that the presence of a pause does not mean that there is no signal in the communication line - the signal is present constantly, but there are no flag codes indicating the beginning of the packet, or vice versa - a continuous stream of these codes is transmitted[10].

In the one shown in Fig. 2, and the pause situations between packets transmitted on the route A-B are relatively small, or in other words, the data rate of the data placed in the packets is relatively large, in the sense that the buffer memory level of the intermediate node M2 is steadily increasing, which may result in buffer overflow[11]. Buffer memory for clarity is shown in the figure as a tank with liquid replenished by the input stream of packets, while the output stream tends to reduce the level of its filling[12].

In this case, the node M2 registers the operation of the second upper level sensor (the comparator of read and write addresses of the buffer memory block). This means that the level of filling is close to critical, therefore, it is necessary to reduce the rate of data flow to the buffer. To reduce the speed, the node M2 sends to the node A a service packet, a command to increase the pauses between packets[13].

In response to this command, node A increases the duration of pauses between packets (Figure 2, b). The degree of increase can be stipulated in the protocol of data exchange between nodes of the network or in the explicit form indicated in the service package. After increasing the pauses, the buffer memory level of the M2 node starts to decrease, if there is no other reason for its increase[12]. Upon reaching the central or lower mark, the node M2 sends to the node A the command to reduce the duration of the pauses, the level of the buffer filling again begins to increase, etc.

Thus, in an ideal case, the buffer memory of node M2 does not overflow and does not emptied, the speed of data output from the buffer memory remains constant, the rate of data arrival adapts to it, making slow fluctuations inherent in conventional automatic control systems[14].

If there are several data sources, then to prevent overload the work of the most active one, but not the most priority, is slowed down; if the sources are equally active, then the impact on those with low priorities is primarily affected[8].

In the development of the described method, it is proposed to take into account, not only the level of the buffer completion, but also the dynamics of its change when forming commands for decreasing or increasing the intensity of the flow[15]. This allows eliminating unnecessary flow control commands when the buffer fill level is high, but the history of the process is such that there is a steady tendency to stop its growth and the subsequent decrease (and vice versa). Essentially, along with absolute reference, the rate of change in the rate (acceleration) of the motion of the level of buffer memory filling is considered[16].

Method 2: Managing the flow of data by notifying the packet source of causes of overload

Prototypes of method 2

Let's continue our consideration of the known methods of data flow control (Fig. 3) using the same network model as before (Figure 1, 2). The data source A transmits a series of data packets to the receiver B. In response to each packet or to a group of packets, the receiver B sends the ACK or NACK response packets to the source A. The ACK response acknowledges the successful reception; the NACK response is a request to retransmit a single packet or group of packets[14].

The first prototype of method 2. In principle, even such a simple feedback (using ACK or NACK response packets) allows detecting and eliminating network congestion on the A TO B path[17]. Indeed, if the data source is increasing the packet rate or at some fixed rate starts to receive an excessive number of retransmission requests, then, most likely, at least one of the nodes of the route entered the overload mode[18].

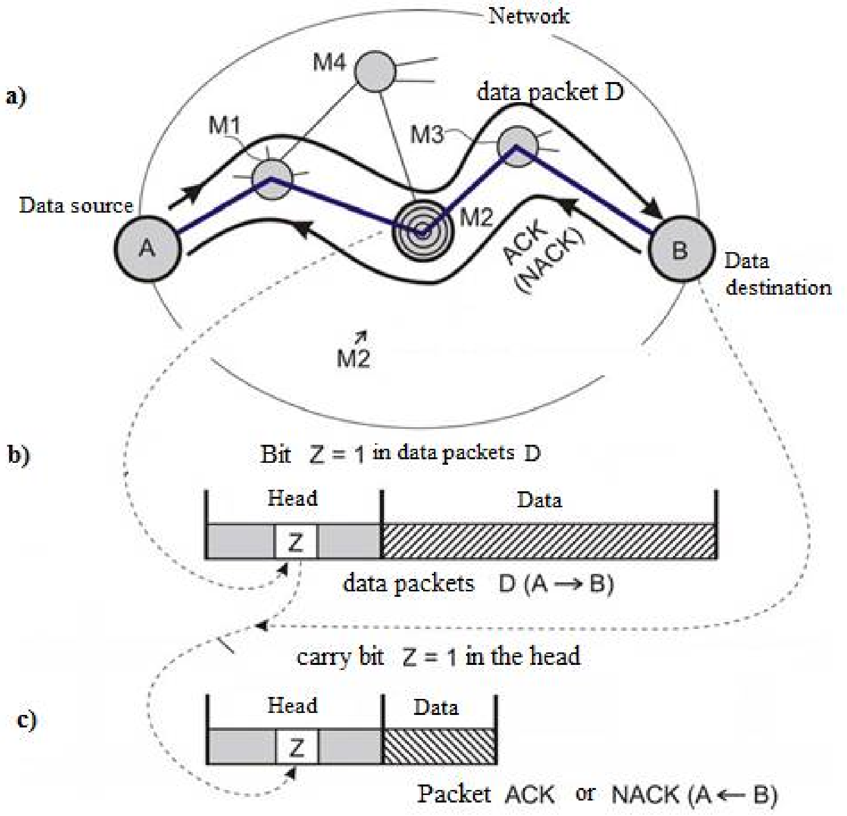

Figure 3. Informing the data source A about the upcoming or available overload of the input buffer of the M2 node of the network: a - packet propagation paths b, c - packet structure D and ACK (NACK)

In this case, the data source drastically reduces the packet transmission rate or (and) increases their length to reduce the share of the overhead bits that make up the headers in the data stream[19]. In the future, the data source gradually either by random trial and error increases the data transmission speed, moving to the permissible upper limit, taking into account some permitted speed increase margin. Such a method is called a "slow start".

Of course, packet loss is possible, not related to the overload of network nodes, for example, due to uncorrectable errors caused by interference in the communication line, but in this case we are not interested in such losses[20].

The considered method of data flow control does not prevent the forthcoming loss of packets, but allows reacting only to the accomplished fact of overloading of the intermediate node of the network or the data receiver[21]. This is its main defect.

The second prototype of method 2. The idea is to warn in time the data source A about the threat of overloading one or several nodes along the route A In the propagation of data packets D. This warning is the bit Z included in the header of the ACK or NACK response packet[22] (Figure 3, c).

In the example shown in Fig. 3, the processor of node M2 anticipates overload, observing the steadily increasing level of buffer filling, as it was shown on its model, shown in the right part of Fig. 2, a; (other events are possible, such as predecessors of congestion[23].

In packages passing through the node M2, more precisely, in the header of each of them, there is information sufficient for its routing, for example, in the form of IP addresses of the source and the data receiver[24]. Viewing this information allows the M2 node to identify the "culprit" of the expected overload, from which the most intensive flow of packets originates. There can be several such.

Suppose that the main "culprit" of the impending congestion is the data source A. This source, like all others, transmitting data packets D, sets the Z bits to zero. With normal data transmission on the route A TO B, these bits remain in the zero state[25].

If the conditions for the upcoming overload are detected and knowing that the largest number of packets per unit of time originate from the source A, the node M2, when transmitted along the route A TO B, marks all packets or a part of them with their Z = 1 bits that inserts in the headers, as shown in Fig. 3, b. The data receiver B returns the received Z = 1 bits to source A, including them in the headers of the response packets ACK and NACK (Figure 3, c).

Finally, data source A receives bits Z = 1 and sharply reduces the data transfer rate to node M2[26]. Further, the data source A gradually restores the original data flow parameters or even exceeds the previously reached data transmission rate until a new series of bits Z = 1, etc., is detected (here, too, the "slow start" mentioned earlier is applied). Having determined the allowable upper speed limit, the data source takes a small step down to create some margin, guaranteeing the route from overload[27].

This way of preventing or eliminating overloads is satisfactory, but not optimal. Its disadvantage is that, without knowing the reason for the overload of node M2, the source of data A is unable to adequately respond to it. So, the "natural reaction"

- a sudden and sharp decrease in the data transfer rate - is unacceptable for many applications. But if, for example, the data source A knew that the reason for the upcoming overload was that the processor of the M2 node could not cope with header stream processing, then it could, without reducing the transfer rate of payload data, increase the packet length to reduce the intensity of this flow[28].

The problem solved by the method 2 discussed below is thus not only to prevent the source of data on the impending overload, but also to inform him of its cause. Then the source could choose the most appropriate "line of behavior" in this situation[29].

The idea of method 2

The problem is solved by extending the single-digit sign Z to several bits. Let us explain what has been said by example, accepting some assumptions.

Suppose that route A-B (Figure 3) is a virtual telephone link between devices A and B, for example, between computers or IP telephones. The technology of VoIP (Voice over IP) is used. Devices A and B contain codecs such as AMR (adaptive multi rate)[30]. The codec generates compressed speech fragments every 20 msec and encodes data from one of eight speeds in the range from 4.75 to 12.2 kbps. Further, as before, one-way data transfer from device A to device B is considered[31].

After the connection A-B is established, the data source generates packets, each of which contains a header and a data field. The data field of the packet is filled with fragments of speech from the codec output, and then the packet is sent along the communication line to node M2[32]. The codec, if the bandwidth of the A-B channel allows, is initially set to the maximum coding rate to ensure the highest speech intelligibility recovered from the data input to receiver B. The Z bits of the sent packets are set to zero.

In the event of detection of the danger of overload by some node located along the A TO B route, this node (in our example, the M2 node) inserts some indication Z in the headers of packets originating from the most active source (A), as described earlier, taking into account that this feature contains not one, but at least two bits. This attribute is returned to the source; as a result, the processor of node A receives information about the reason for the upcoming overload.

The node M2 may experience overloading for at least one of the following reasons.

1. Narrowing the bandwidth (bandwidth) of the channel A TO B due to the appearance of a "bottleneck." This can happen, for example, because a part of the dedicated link A TO B of the linkage between the nodes M2 and M3 (Figure 3) has decreased. This decrease may be due to various reasons. Let's name two of them.

- The previously unobtrusive competing data flow along the route M4 M2 M3, which uses the same channel M2 to M3, as the route A TO B, has increased to a significant level earlier. As a result, the M2 node redistributed the strip of this channel to the detriment of the route A TO B .

- The M2 node has changed the type of signal modulation in the M2 to M3 channel, reducing the transmission rate due to the deterioration of the signal-to-noise ratio in this channel.

2. The M2 node processor for some reason or other has stopped coping with the volume of work on analyzing packet headers following the route A TO B.

The first and second reasons above for the approaching overload are displayed respectively by the codes Z = 012 and Z = 102, the absence of an overload hazard corresponds to the code Z = 002, both causes simultaneously generate the code Z = 112. The code Z = 112 can Form one node if it simultaneously observes both reasons for the upcoming overload, or by two or more nodes located along the A to V.

So, the node M2 can insert the Z = 102 codes into the headers of the A B packets that pass along the route, because the processor of this node cannot cope with the volume of work on the analysis of headers. These packets are transmitted to the M3 node, which is supposed to reveal a decrease in the M3 to B channel bandwidth allocated to the A to B route. In this case, the M3 node replaces the Z = 102 codes in the packets passing through it with Z = 112. These codes, as described, reach the receiver B and return to the data source A as part of the headers of the response packets (Figure 3, c).

The optimal response of the data source to the identified causes (1 or 2) of overloading may be this.

The narrowing of the channel bandwidth A to B (reason 1) should cause a corresponding decrease in the total data rate (both useful and service) of the source A. To estimate the rate reduction, it would be desirable to use a multi-bit code Z in which this degree is reflected. However, in this case there is no such possibility, therefore the processor of the data source A switches its codec to the mode of the lowest encoding speed (out of eight possible - from 4.75 to 12.2 kbps). If the packet length is unchanged, and the lowest

The frequency of their succession decreases due to the increase in pauses between them. At the same time, the delay in the formation of the packet increases due to the increase in the time it is filled with compressed fragments of speech. Thus, the data transfer rate (both useful and service data) is reduced by source A, and if the narrowing of the band is not too large, then there is no danger of overloads.

In the future, to restore the high quality of voice transmission, the coding rate and, correspondingly, the packet repetition rate gradually increase to the experimentally detected limit, in which there is still no danger of overloading the network nodes on the A to B

Alternative response of the data source to the narrowing of the channel band A to B also provides for using the lowest encoding rate. When this keeps the packet repetition rate, and their length decreases. The rate of transfer of useful data decreases, the service data flow remains unchanged.

Finally, the strongest reaction is possible, at which the coding rate is set to the minimum, and the length of the packets increases to such an extent that their average delay approaches the permissible limit (not more than 100 ms [3]), after which, during a telephone conversation, begins eavesdropped. Such a reaction is the maximum that can be done in this situation.

After exiting the crisis, the coding rate gradually increases, and the length of the packets decreases with this in time (to reduce the delay of their transmission along the route A to B). This process of two-dimensional optimization of flow parameters is completed when the boundary is reached, after which the risk of overloading again arises.

Overloading the processor of one or more nodes on the A to B route (reason 2) is eliminated by reducing the intensity of the header stream that it (they) has to process. For this, while maintaining a high coding rate, the data source increases the length of the transmitted packets to such an extent that their average propagation delay along the A to B path does not exceed the previously mentioned allowable limit (100 ms).

Thus, the correct response to the overload warning in many situations allows to eliminate the danger of overflow of input buffers and, what is essential, to maintain high quality of voice transmission.

Method 3: Control of the flow of data with compensation of the inertia of the feedback loop

Prototype of method 3

One of the simplest ways to control the flow of data transmitted between the nodes of the network J1 and J2 (Figure 4, a) is as follows.

In steady state, data packets are accumulated in the output buffer of node J1 for transmission along a certain route, possibly through other network nodes (not shown in the figure) to the input buffer of node J2. Both buffers are executed in the form of blocks of memory of type FIFO.

The flow of data packets passing through the system from the left to the right has the character of "machine-gun queuing", since the series of packets are transmitted by the J1 node via the communication line only with the permission of the receiver, node J2, which "causes fire to the extent possible". The instantaneous packet transfer rate inside the series is C; the average speed is less than the instantaneous one and depends on the average ratio of the pauses between packets to the length of the series. The unevenness of the arrival of packets in the buffers of the nodes J1 and J2 causes fluctuations in the levels of their filling. The challenge is to protect these buffers from overflow or emptying.

Further, this task is solved only with respect to the input buffer of the node J2, however, the output buffer of the node J1 can be protected in a similar way by introducing feedback from the source of the packets sent to it (in the figure this source and its feedback are not shown). Such a successive chain with feedbacks between neighboring elements can be arbitrarily long. Each transmitting port thus issues a stream of packets to the communication line only if there is a transmission permission previously received from the destination of the XON command.

The input buffer of node J2 contains a pointer to the threshold level F of its filling. In this example, the input buffer of node J2 contains Q packets. At the moment the current level Q overcomes the threshold level filling in the F upward side (Q < F), the node J2 transfers to J1 the packet with the XOFF command of the transmission suspension. Similarly, at the moment overcoming the current level Q filling the threshold level F downwards (Q>= F), the J2 node sends a packet to the J1 node with the XON resumption command.

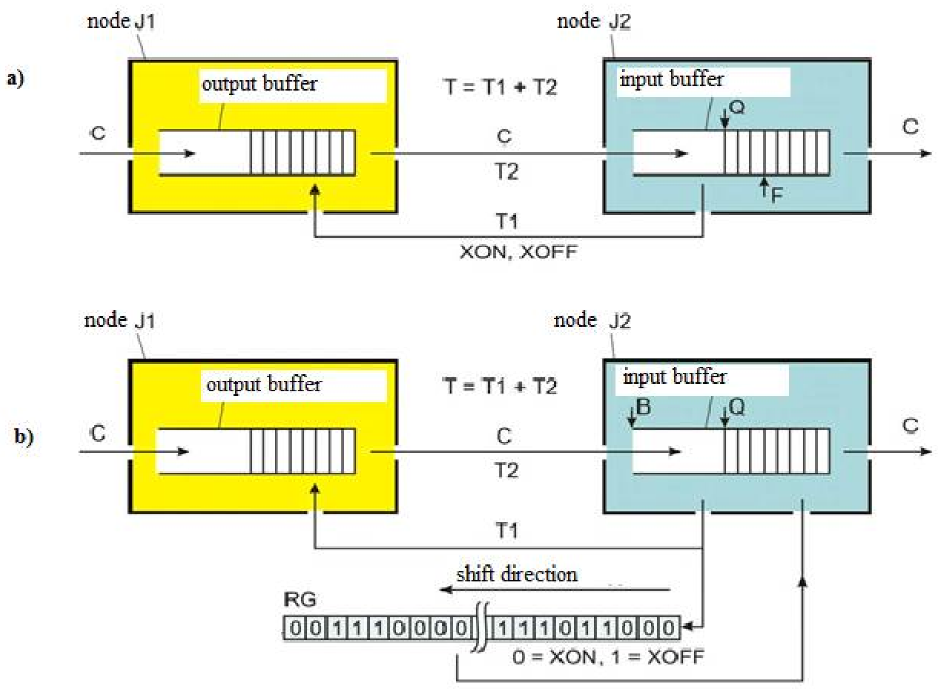

Figure 4. Flow control scheme: a - traditional; b - the proposed

The problem is that flow control can be very inertial.The response time of the system to the XON and XOFF commands is determined by the delay T = T1 + T2, where

T1 - the time from the instant the command is generated by the node J2 until the previously stopped process of sending packets by the node J1 resumes or the previously activated process of issuing packets by the node J1 is suspended;

T2 - the time of packet transmission from the output buffer of node J1 to the input buffer of node J2.

Thus, if the increasing filling level of the input buffer of the node J2 has overcome the threshold value F, then the generated XOFF command will stop the flow of packets at the input of the node J2 only through the time T. During this time, the input buffer of the node J2 continues "by inertia "To replenish.

Similarly, the first packet after issuing the XON command to resume the previously-stopped stream will arrive at the input buffer no earlier than the time T. During this time, the level of filling the input buffer of the node J2 "by inertia" is reduced due to the outflow of data from it.

If the capacity of the input buffer of node J2 is small, then the inertia of the control can lead to overflow or emptying. In the worst case, after the moment of exceeding the threshold level F (Q < F, the command XOFF is issued) and at the time no outflow of data from the input buffer of the node J2 during the time T in this buffer "by inertia" will come with C*T packets.

Similarly, if there was no inflow of data, after the moment of crossing the threshold level F in the direction of decrease (Q =>F, the command XON is issued) and with continuous data flow from the input buffer of node J2 during the time T from this buffer "on inertia "will be selected C*T packets.

Thus, to protect against overflow and emptying, the input buffer of node J2 should be designed to store at least 2C*T packets; the threshold F must correspond to its middle.

The resulting estimate of the minimum buffer size is disappointing. Some switches contain several hundred buffers, so the actual task of reducing their volume is actual. In high-speed networks, the T value reaches tens and hundreds of microseconds. The value of C is of the order of 10 Gbit / s. As a result, the buffer size 2C T = 2 1010 10–4 is several megabits. The goal of the next solution is to reduce the buffer size by half thanks to smoother flow control.

The idea of method 3

Smoothness of control is achieved by fragmentation of series of packets and more intelligent algorithm of forming commands XON and XOFF to resume and stop transmission of the stream.

The circuit shown in Fig. 4, b, [4] contains the same components and has the same parameters (T, C, Q), which have just been discussed. The volume of the input buffer of the node J2 is denoted by B. The new element of this node - the history memory of the control - is shown for clarity in the form of a shift register RG, although it can be executed programmatically using a set of memory cells.

For definiteness, suppose that the flow of ATM cells is transmitted via the communication channel [5]. (The term "cell" is equivalent to the term "packet".) This stream is continuous - after the last bit of the previous cell, the first bit of the next is transmitted. The length of the cell is 53 bytes. The cells follow the line of communication with a period of 40 ns. This does not mean that the proposed idea is applicable only to ATM technology - it is easy in the following description to operate with strictly prescribed quanta of time with duration of 40 ns.

Suspension of the flow in this case is conditional (a continuous stream of cells follows the connection line always) and means that the output of the nodes J1 accumulated in the output buffer really stops, but instead of them, bypassing this buffer, empty cells of the same length are output into the communication line, as well as cells with data. Empty cells can be inserted once or form more or less lengthy sequences. Blank cells are rejected by the J2 node and do not enter its input buffer.

Suppose that the time T = T1 + T2 = 2 μs, that is, corresponds to the passage of 50 cells. The rate of issuing commands XON or XOFF is equal to the rate of arrival of cells (empty and non-empty) at the input of node J2, that is, commands are issued every 40 ns. The commands issued by the node J2 in response to each incoming cell on the communication line affect the input stream after a time of 50 cells - this is the inertia of the control loop.

Simultaneously with issuing the XON or XOFF command from node J2 to node J1, it is stored as the corresponding bit (0 or 1) in the right-hand bit of the shift register RG, the remaining bits are shifted one position to the left, the leftmost bit is pushed out of the register. Thus, in the RG register, the history of issuing control commands for the next 50 cycles (the periods of succession of the cells) is displayed.

Each XON or XOFF command when entering J1 is responsible for making a decision to issue one (regular) cell either from the output buffer of this node (when receiving the XON command) or from a source of empty cells to bypass the output buffer (upon receipt command XOFF).

The code in the RG register is analyzed by the J2 node. Counting the number of zeroes contained in it, the node predicts the number of cells with data that will go to its input buffer within the next 50 cycles. The single bits in this register correspond to the number of empty cells that will arrive at the input of node J2 during this period and will be destroyed by them.

The formation of XON or XOFF commands is as follows. Let NON be the number of zero bits in the RG register, B the size of the input buffer of the node J2, Q the current size of the queue. Then:

if Q + NON   B, then the XOFF command is generated; otherwise, the XON command.

Indeed, in the worst case, when there is no outflow of data from the input buffer of the node J2, the expected level of its filling is equal to the current level of Q, increased by the number of NON cells that are actually already in transit and will surely be received in the next 50 cycles. The expected level of buffer filling Q + NON should not exceed its size B. If this condition is met, then the thread should not be suspended, so the XON command is generated. In the opposite situation, when the predicted level of buffer filling exceeds the volume of the buffer, stop the flow for at least one clock cycle, that is, generate the XOFF command.

The commands, of course, will have an effect only after 50 clock cycles, but due to the "smallness" of their action and the integration of many commands in time, the total effect is expressed in that the fluctuations in the buffer fill level become smaller, and the necessary buffer memory capacity is reduced by two times.

So, in the steady state, the average level of buffer memory of the J2 node is close to V / 2, in the 50-bit RG register the average number of zeros and ones is approximately the same. Suppose that B = 50, the average level is 25. Then the stocks in relation to overflow and emptying the buffer will be 25 cells in each direction. This is consistent with the fact that the average number of arriving cells expected in the nearest time interval T is 50/2 = 25.

In the prototype (Figure 4, a), in the worst case (in the absence of data outflow from the buffer), at the time T, 50 cells arrive at the input of the buffer of node J2. Similarly, in the opposite situation, in the absence of data flow to the buffer, the level of its filling during the time T will decrease by 50 cells. Therefore, to create the necessary reserves of 50 cells in each direction, a buffer with a volume of 100 cells is needed, which is twice as large as when using method 3 (Figure 4, b).

Expanding the scope of the method 3

Previously, the idea of reducing the amount of buffer memory of the receiver when building a data transfer system between nodes of a computer network was considered. However, this idea can find wider application.

As an example, consider the circuit of the commutator (Fig. 5). As usual, to simplify the description, we assume that the data streams propagate only in one direction - from left to right. In fact, to construct a switch operating with flows of both directions, it is necessary to apply the same circuit deployed in the opposite direction, superimpose the resulting circuit to the original one, and combine the corresponding external inputs with the outputs.

The switch contains three input buffers # 1 - # 3, a switching matrix, a processor and three output buffers ## 1 - ## 3. Comparing Fig. 5 with Fig. 4b, one can note the similarity between the block structures used in both schemes. Some designations also coincide, therefore further are not explained. The signals GO_1 - GO_3 from the rightmost cell of the corresponding input buffer of type FIFO are given a data packet, with the queue moving one position to the right.

Data packets from independent sources, for example, from computer network nodes, enter the input buffers of the switch. As a result, buffers create queues of packets waiting to be sent to the output buffers. The directions of packet transmission are detected by the processor based on the analysis of address information contained in their headers.

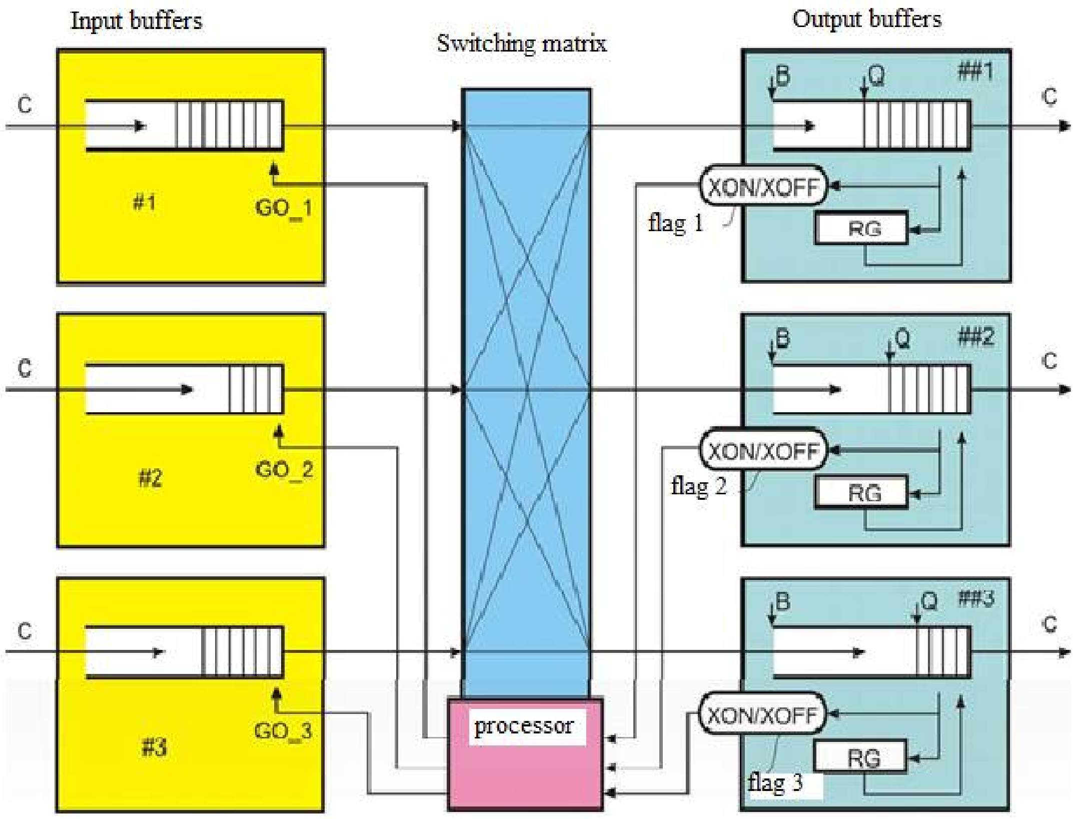

Figure 5. Structure of the switch, the first option

The packets are transferred from the input buffers to the output through the switching matrix under the control of the processor. Packets of some types are sent simultaneously to all output buffers or to some subset of them. The switching matrix allows simultaneous transmission of packets in different independent directions. For example, simultaneously with the transfer of a packet from the buffer # 1 to the buffer ## 3, transmissions along the directions # 2 ## 1 and # 3 ## 2 can be carried out.

In the output buffers, queues of packets awaiting delivery to the corresponding communication lines are also created. In each of these buffers, the previously discussed method of preventing overflows and devastations of the queue is applied (Fig. 4, b). However, in this case (Figure 5), the output buffer "does not know" from which directions and in what order the data is expected to arrive, ie, it does not have information about which input buffers and which sequence should be sent the results of the queue state forecasting - the XON or XOFF commands.

Therefore, the output buffers form the XON / XOFF flag bits (flag 1-flag 3), irrespective of which input buffer will be affected. The flags are polled by the processor and used by the processor to control the transmission of data through the switching matrix.

Looking through the outputs of buffers # 1 - # 3, the processor monitors a lot of packets, ready to be sent to the buffers ## 1 - ## 3. The decision to send each of these packets is accepted by the processor only if the flag of the corresponding output buffer is set to the enabling state - XON. Then the processor creates the required path through the switching matrix and initiates the issuance of the packet by the command (signal) GO_i (i = 1, 2, 3).

The structure of the switch (see Figure 5) has a drawback that is not related to the application of the proposed method for managing data flows.

If the packet type provides its transfer to a group of several output buffers, the processor does not wait for the entire group to receive data at the same time to speed up the process. It transmits copies of this package sequentially, as the output buffers that make up the group appear. In this case, until the complete distribution of the packet across the whole group of output buffers, this packet is not removed from the input buffer and therefore prevents the progress of the queue in it.

A similar situation (blocking of the input queue) can be observed when sending a normal packet addressed to only one output buffer. If the output buffer is not ready for data reception for a relatively long time, then the packet remains at the output of the input buffer, and the queue in it does not advance, but only grows with the arrival of new packets. This queue may contain packages that could be serviced, since the corresponding output buffers are ready to receive data, but they are all prevented by the priority packet waiting for maintenance and blocking access to the rest of the packets to the switching matrix.

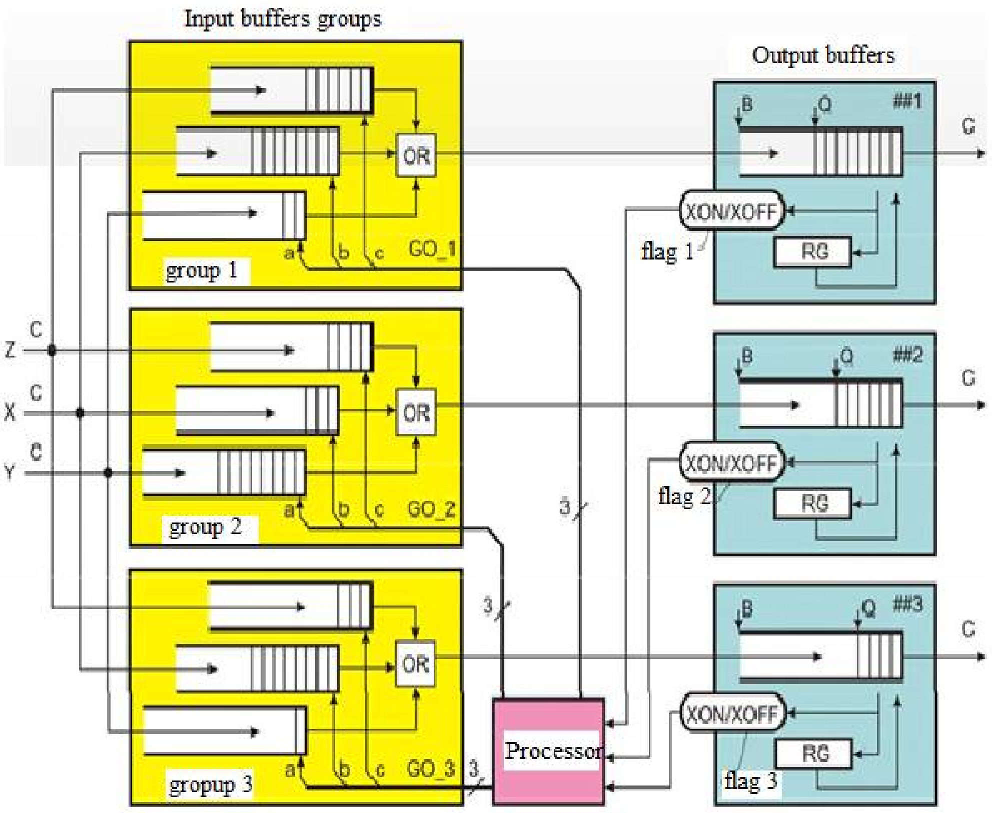

Figure 6. Switch structure, second option

Blocking of input queues is eliminated in the scheme shown in Fig. 6. In comparison with the previously considered circuit (Figure 5), the input buffers are replaced by buffer groups, the switching matrix is excluded. Each group of input buffers accumulates more than one queue for the number of input channels of the switch. Each group of input buffers transfers data to the corresponding output buffer.

Packets coming from the input channels Z, X and Y are sorted. Packets of channel Z, which should get into the output buffer ## 1, are written to the upper buffer of group # 1. Packets of the Z channel, intended for sending to the line through the buffer ## 2, are written to the upper buffer of group # 2. Packets of channel Z, which should be sent to the output buffer ## 3, are written to the upper buffer of group # 3. Packages from the input channels X and Y are sorted similarly.

The processor analyzes the flags 1 to 3 and, in the presence of the readiness of one or more output buffers, receives one or more GO_i signals (i = 1, 2, 3) to receive data. Each of these commands is addressed to one group of input buffers. Since in this example the group contains three buffers, the command contains three bits that indicate from which queue the next data packet should be issued via the OR gate. Commands (a, b, c) = (0, 0, 1), (a, b, c) = (0, 1, 0) and (a, b, c) = (1, 0, 0) correspond to the issuance The data packet from the upper, middle and lower case of the selected group. The queue number can be transmitted from the processor with binary code with its decoding in groups of input buffers, but this possibility is not considered to simplify the figure.

If one of the output buffers is not ready to receive data for a relatively long time, this does not affect the transmission of packet flows through other output buffers. For example, the output buffer ## 1 may not be ready to receive data (flag 1 in the XOFF state), then the GO_1 signal remains zero for this time (0, 0, 0), preventing the issuance of packets from group 1. Other groups remain in normal operating mode, i.e., as far as possible under the control of the processor, data is transferred to the corresponding output buffers.

3. Discussion

Accelerated transmission of high priority packets through the switch. The switch shown in Fig. 7, is an improved version of the previously considered structure (Fig. 6). Comparing Fig. 6 and Fig. 7, one can note that some of the previously considered elements in Fig. 7 are not shown, although they may be present in the circuit. At the same time, new elements have been introduced, the functions of which do not violate the work of the previously considered schemes. The purpose of introducing new elements is to accelerate the transfer of high-priority packets through the switch.

Just like in the previous scheme, the switch contains three groups # 1 - # 3 input buffers of type FIFO. The outputs of these buffers in each group are connected through the first logical OR and the L1-L3 packet converters with the inputs of the output buffers ## 1 - ## 3. In each group of input buffers, the second logical OR is added, through which bypass paths (without queue) pass high-priority covenants, if they enter buffers.

Switches SW1 to SW3 translate packets either from the corresponding queues located in the buffers ## 1 - ## 3, or from the workarounds. In the first case, the key is set to LP (low priority), in the second - to HP (high priority). Coordination of actions of all components of the multiplexer is performed by one or several processors (in Figure 7 processors are not shown).

In general, the proposed idea is as follows: As in the previous scheme (Figure 6), the packets arriving from the input channels Z, X and Y are sorted. Packets of channel Z, which are addressed to buffer ## 1, are written to the upper buffer of group # 1. Packets of channel Z, addressed to buffer ## 2, are written to the upper buffer of group # 2. Finally, the Z channel packets addressed to buffer ## 3 are written to the upper buffer of group # 3. Packages from the input channels X and Y are sorted similarly.

Then the packets are moved along the corresponding input queues, through the first logical OR elements and the lower channels of the converters L1 to L3 are transmitted to the output buffers ## 1 - ## 3 and in the order of their arrival are output from them to the output lines Q, R and S via the keys SW1 to SW3, which are in the LP state.

This "natural" sequence of events is violated with the arrival of a high-priority packet, for example, in the upper buffer of group # 1. All new arrivals in the buffers, packets are checked for priority. Suppose first that the number of priority levels is two, and the high-priority packet came at a time when all other packets on the switch have low priorities. The priority level of the package is indicated in its header.

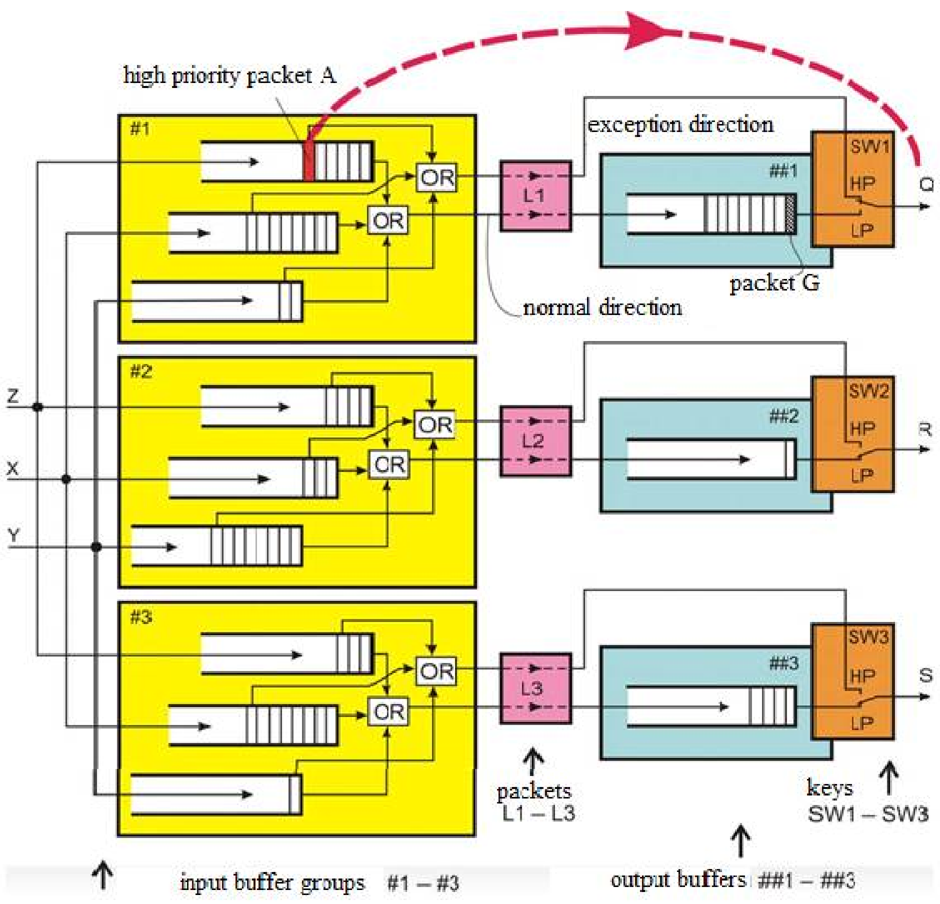

Figure 7. Switch structure with accelerated maintenance of high-priority packets

In the known switch structures, a simple and understandable reaction to the entry of a high-priority packet was adopted:

• If the desired output link is not used, then the high priority packet immediately, without delay, begins to be issued to it;

• If the communication line is busy transmitting a low-priority packet, the delivery of a high-priority packet is delayed until it is released.

The latter circumstance leads to delays in switching high-priority packets, which for some applications is highly undesirable or even unacceptable. In the worst case, a high-priority packet may be a little late at the time of issuing a low priority, which may have a significant length, for example, 1500 bytes.

The proposed solution allows interrupting transmission of the low-priority packet practically at any stage, then to transmit a high-priority packet and only then resume the interrupted transfer. Nested interrupts are possible if the number of priority levels exceeds two. Let us consider this solution in more detail.

Suppose that in each transmitted packet (Figure 8), in addition to the address and other information, its priority P and length N are indicated. The codes P and N can be located, for example, in two adjacent bytes, with three bits defining one of the eight priority- and the remaining 13 bits are the length of the packet (in bytes) in the range from some fixed minimum length U to the maximum, equal to (U + 213 - 1) bytes. All transmitted packets pass through converters L1-L3 (Figure 7), where each of them is bit-oriented and is preceded by a unique flag.

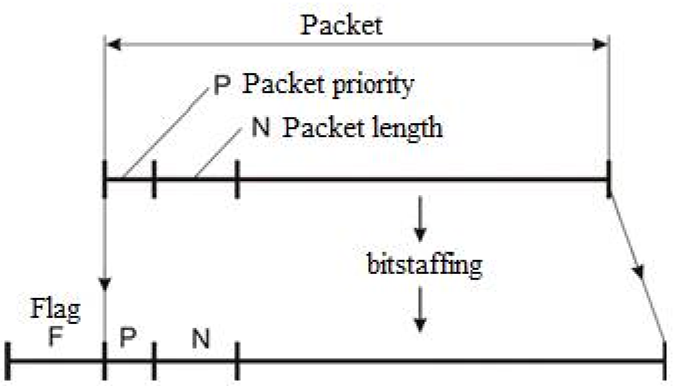

Figure 8. Conversion of packets by blocks L1-L3 (Figure 7)

Recall that bit staffing allows you to exclude from the data stream a random copy of the unique code selected as the frame start flag F. In this example, F = 01111110.

In Fig. 9, and the "true" flag F of the beginning of the frame (circled in a rectangular frame) is inserted into some sequence of bits. The problem is that, most likely, this sequence also contains codes 01111110, which can be considered as false flags. In order to prevent the transmission of false flags to the far side of the communication channel, they are intentionally reversibly distorted, for example, according to the algorithm proposed in [7].

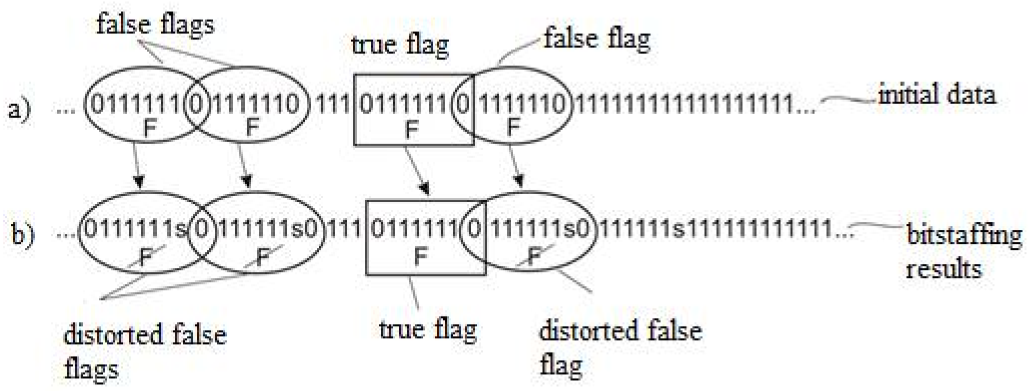

Figure 9. Improved bitstaffing: a - the initial sequence of bits with the "true" flag of the beginning of the frame introduced into it; b - the same sequence after excluding false flags from it

This algorithm is as follows. The original sequence of bits with the "true" flag inserted into it is viewed through a sliding seven-bit window in order to detect in it the code 0111111, almost coincident with the flag. If such a code is detected and is not a component of the "true" flag, then it is supplemented by a single bit of s, regardless of the value of the subsequent bit (Figure 9, b). Such a procedure is called bitstaffing.

Bitstaffing does not apply to "true" flags, so they become unique, since all false flags are deliberately distorted by bits of s.

On the far side of the communication channel, the reverse operation is performed - bits s (following the sequences 0111111, which are not constituent parts of the "true" flags) are destroyed.

In contrast to the classical bitstaffing used in the HDLC protocol, the variant proposed in [7] allows us to reduce the redundancy introduced into the initial bitstream by half. Indeed, for a single random sample, the probability of detecting a 7-bit code (0111111) in a random data stream is 1/27 = 1/128. In the classical version of bitstaffing, the probability of detecting a 6-bit code (011111) in a random data stream is 1/26 = 1/64. In other words, the insertion of redundant bits in the classical version of bitstaffing is carried out twice as often as in the version proposed in [7].

Suppose that in the initial state, a low-priority packet is sent to the line from the output buffer ## 1 of the switch (Figure 7). The SW1 switch is set to LP. As shown in Fig. 10, a, at some time T0, a high priority packet arrives from the upper channel of the packet transformer L1, bypassing the output queue. The transmission of the low priority packet terminates in the nearest bit interval, the SW1 switch goes to the HP state and the first flag bit of the high priority packet is placed in place of the not transmitted bit. Then all the bits of this packet are transmitted (Fig. 10, b).

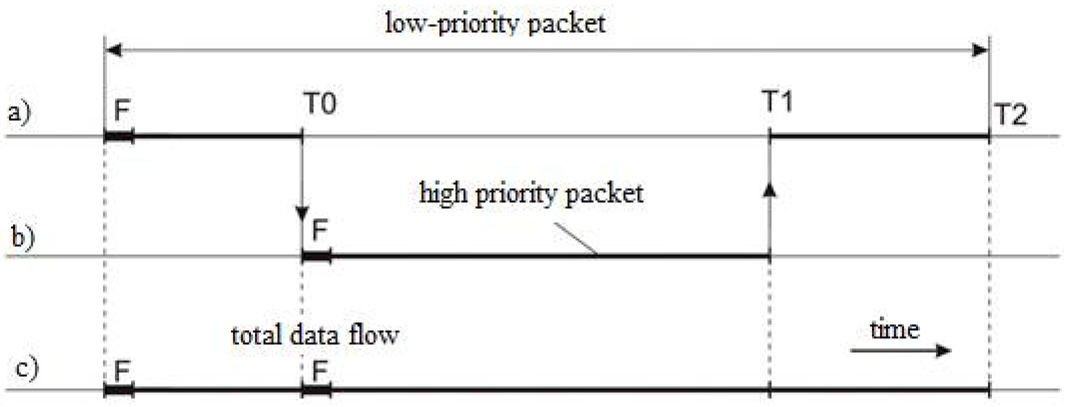

Figure 10. Interruption of low-priority data stream high priority: a - low-priority data packet; b - high-priority data packet; c is the total data flow in the line

At the time T1, the last bit of the high priority packet is transmitted. The key SW1 returns to the LP position. Following the last bit of the high-priority packet, all the bits of the previously suspended low-priority packet are transmitted. The total data flow (Fig. 10, c) can be divided on the far side of the communication channel into two components corresponding to Fig. 10, a and b, due to the uniqueness of the flags F and the presence of the P and N fields in the packet headers.

To simplify the analysis of code situations by the receiver, one can accept the condition that the low-priority packet flag is protected from interrupts, i.e., not crashed when switching to a high-priority packet transmission. In other words, if a high-priority packet has entered the SW1 key during the low priority packet transmission, it is delayed and its transmission begins only after the low-priority packet flag is fully transmitted. In the worst case, the delay is eight bit intervals.

With a greater number of priority levels, the described process of switching data flows acquires the nature of nested interrupts widely used in microprocessor technology. As shown in Fig. 11, the transmission of packets can repeatedly go from one priority level to another and back.

In the period T0 - T1, the packet Y0 of the zero (lowest) priority level is transmitted to the line. At time T1, this transmission is interrupted due to the arrival of the Y1 packet of the first (higher) priority level. The transmission of the packet Y1, in turn, is interrupted at the time T2, after which the Y2 packet of the second priority level is fully transmitted. The end of the transmission of this packet is marked by a period.

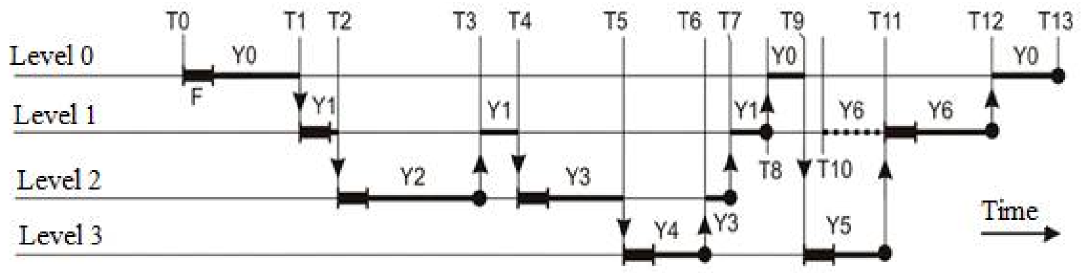

Figure 11. Transmission of data packets Y0 to Y6 using a four-level priority system

At time T3, the switch returns to the transmission of the packet Y1, but at the time T4 the transmission is again interrupted by the higher priority packet Y3, which in turn is interrupted by the Y4 packet at the time T5. This packet has the highest priority; therefore its transfer cannot be interrupted under any circumstances.

Further, at the moments T6 to T8, in the order of decreasing priorities, the transmissions of the packets Y4, Y3, Y1 are completed, and the transmission of the packet Y0 resumes. At time T9, this transmission is again interrupted by a Y5 packet having the highest priority. At time T10, the Y6 packet is ready for dispatch, but it is performed only starting from the moment T11, when the transmission of the Y5 packet is complete. At the moments T12 and T13, the transmission of the packets Y6 and Y0 is completed.

5. Conclusions

High-priority packets are "wedged" into low-priority packets, without waiting for the end of their transmission. This allows reducing delays in high-priority packets even with low-priority packets of long length. The increase in the intelligence of telecommunication devices became possible to apply more sophisticated algorithms and original flow control schemes in comparison with the known ones. This allows solving the following tasks:

- reduce the likelihood of overflow and emptying of buffer blocks located along the distribution routes of packets and, ultimately, improve the quality of computer networks;
- reduce the required amount of buffer memory;
- improve the efficiency of servicing high-priority packets.

The article considers a number of original solutions to these problems at a level sufficient for the development of new generations of telecommunication devices and systems.

References

1. Fujioka, Y., M. Kameyama, and M. Lukac. A dynamically reconfigurable VLSI processor with hierarchical structure based on a micropacket transfer scheme. in Information and Digital Technologies (IDT), 2017 International Conference on. 2017. IEEE.
2. Drumm, D.E., Computer input device using an orientation sensor. 1991, Google Patents.
3. Sansyzbaevich, I.S., et al. Development of algorithm flow graph, mealy automaton graph and mathematical models of microprogram control mealy automaton for microprocessor control device. in Control and Communications (SIBCON), 2017 International Siberian Conference on. 2017. IEEE.
4. Sidler, D., et al. Accelerating pattern matching queries in hybrid CPU-FPGA architectures. in Proceedings of the 2017 ACM International Conference on Management of Data. 2017. ACM.

5. Wu, E.-I., Control device for vehicle hiring and control system using same. 2017, Google Patents.
6. Aslam, M.H., et al., Exploring the effect of LUT size on the area and power consumption of a novel memristor-transistor hybrid FPGA architecture. Arabian Journal for Science and Engineering, 2016. 41(8): p. 3035-3049.
7. de Rochemont, L.P. and A.J. Kovacs, Hybrid computing module. 2016, Google Patents.
8. Maeda, T. and R. Matsubara, Storage apparatus and failure location identifying method. 2017, Google Patents.
9. Kim, S. and R. Lu, The Pseudo-Equivalent Groups Approach as an Alternative to Common-Item Equating. ETS Research Report Series, 2018.
10. Kaushansky, D., et al., Programmable test instrument. 2017, Google Patents.
11. Tan, C.J., et al. Review on Firmware. in Proceedings of the International Conference on Imaging, Signal Processing and Communication. 2017. ACM.
12. Cabillic, G. and J.-P. Lesot, Selective compiling method, device, and corresponding computer program product. 2017, Google Patents.
13. Wiśniewski, R., Prototyping of Concurrent Control Systems, in Prototyping of Concurrent Control Systems Implemented in FPGA Devices. 2017, Springer. p. 99-116.
14. Durand, Y., et al. A Programmable Inbound Transfer Processor for Active Messages in Embedded Multicore Systems. in 2017 Euromicro Conference on Digital System Design (DSD). 2017. IEEE.
15. Vladimirov, S. and R. Kirichek, The IoT Identification Procedure Based on the Degraded Flash Memory Sector, in Internet of Things, Smart Spaces, and Next Generation Networks and Systems. 2017, Springer. p. 66-74.
16. Ye, J., A novel ship-borne positive pressure solid phase extraction device to enrich organo chlorinated and pyrethroid pesticides in seawater. Se pu= Chinese journal of chromatography, 2017. 35(9): p. 907-911.
17. Anderson, J.L. and T.J. Balph, Memory interface device with processing capability. 1981, Google Patents.
18. Vasumathi, B. and S. Moorthi, Implementation of hybrid ANN–PSO algorithm on FPGA for harmonic estimation. Engineering Applications of Artificial Intelligence, 2012. 25(3): p. 476-483.
19. Wiśniewski, R., Modelling of Concurrent Systems in Hardware Languages, in Prototyping of Concurrent Control Systems Implemented in FPGA Devices. 2017, Springer. p. 117-137.
20. Pearlson, K.E., C.S. Saunders, and D.F. Galletta, Managing and Using Information Systems, Binder Ready Version: A Strategic Approach. 2016: John Wiley & Sons.
21. Ruiz, P.A.P., B. Kamsu-Foguem, and D. Noyes, Knowledge reuse integrating the collaboration from experts in industrial maintenance management. Knowledge-Based Systems, 2013. 50: p. 171-186.
22. Han, Y.Y., et al., Unexpected increased mortality after implementation of a commercially sold computerized physician order entry system. Pediatrics, 2005. 116(6): p. 1506-1512.
23. Archer, C.J. and G.R. Ricard, Administering registered virtual addresses in a hybrid computing environment including maintaining a cache of ranges of currently registered virtual addresses. 2016, Google Patents.
24. Jagadish, H., et al., Big data and its technical challenges. Communications of the ACM, 2014. 57(7): p. 86-94.
25. Rafi, D.M., et al. Benefits and limitations of automated software testing: Systematic literature review and practitioner survey. in Proceedings of the 7th International Workshop on Automation of Software Test. 2012. IEEE Press.
26. Al-Rababah, A. and N. Hani. Component linked based system. in Modern Problems of Radio Engineering, Telecommunications and Computer Science, 2004. Proceedings of the International Conference. 2004. IEEE.
27. Rodríguez, P., et al., Continuous deployment of software intensive products and services: A systematic mapping study. Journal of Systems and Software, 2017. 123: p. 263-291.
28. Taylor, S.J., R. Bogdan, and M. DeVault, Introduction to qualitative research methods: A guidebook and resource. 2015: John Wiley & Sons.
29. Ciccozzi, F., et al., Model-Driven Engineering for Mission-Critical IoT Systems. IEEE Software, 2017. 34(1): p. 46-53.
30. AlRababah, A.A., A new model of information systems efficiency based on key performance indicator (KPI). management, 2017. 4: p. 8.

31. Al Ofeishat, H.A. and A.A. Al-Rababah, Real-time programming platforms in the mainstream environments. IJCSNS, 2009. 9(1): p. 197.
32. Choi, J. and R.A. Rutenbar, Video-rate stereo matching using Markov random field TRW-S inference on a hybrid CPU+ FPGA computing platform. IEEE Transactions on Circuits and Systems for Video Technology, 2016. 26(2): p. 385-398.