The Bufferbloat Problem and TCP: Fighting with Congestion and Latency.

Anatoliy Sivov
Yaroslavl State University
Yaroslavl, Russia
mm05@mail.ru

V.A. Sokolov (research supervisor)
Yaroslavl State University
Yaroslavl, Russia

Abstract—**The bufferbloat is a serious problem of the modern nets with complex organization. The persistent saturation of the net with an excessive buffering on hops results in a huge latency which can make interactive and realtime connections practically unusable. Most research in this area is devoted to Active Queue Management (AQM) which is used to keep the packet queue size correct on the bottleneck of the network path. This article gives the other look on the bufferbloat. It suggests to fight with the congestion – the reason of buffer saturation. It states that currently most widely used TCP loss-based congestion control algorithms have the problem which is shown in impossibility to correctly estimate the bandwidth in the presence of excessive buffering, and suggests to use the delay-based approach for congestion avoidance. It presents ARTCP which is delay-based and does not use burst sends limited by the congestion window, but transmits data on the adaptive rate estimated with pacing intervals.**

TCP; congestion; latency; bufferbloat; AQM; ARTCP.

I. Introduction

The bufferbloat is a term introduced by Jim Gettys which is very popular now and means the existence of excessively large and frequently full buffers inside the network. The bufferbloat influences the latency, an important parameter in the networking experience, very much. The latency consists of three kinds of delays: transmission delay, processing delay and queuing delay. The latter is the time the network packet spends waiting to be processed or transmitted. This time obviously depends on the queue size which can grow huge in the case of the bufferbloat.

The reason of this growth is a congestion the network path can suffer from. Paths between communicating endpoints are typically made up of many hops with links of different bandwidth. In the fast-to-slow transition hops we can have the situation when there are packets arrived to be queued or dropped, because they can not be immediately transmitted due to the previous arrivals being not processed (passed to the next hop) yet. The buffers (that carry the queue) are necessary in the "burst send" scenarios to maintain the flow of packets at the maximum rate and achieve the throughput. However, incorrect bandwidth estimation can lead to too many packets being sent, that results in buffers overpopulation or packets drops. If the buffers are too big, the packets are not dropped, and the queue grows, as well as the queuing delay. In this case we have unnecessary latency growth which does not lead to any benefits in throughput.

So, it is easy to conclude that in the presence of congestion in the net with excessive buffering the resulting delay can grow very high. As reported in [1], most network hops suffer from the excessive buffering. The other problem is that the network congestion is a usual case.

TCP is the most widely used transport protocol in the world. This protocol tries to use all the existing bandwidth to be efficient. To do that, it increments the transmission rate (actually, increases a congestion window) and provides the net with more and more data (that come out as burst sends limited by the congestion window) until a packet loss occurs. So, this protocol is one (and, possibly, main) of the reasons of congestion. TCP loss-based congestion avoidance algorithm surely uses the loss occurrence to slow down the transmission. This approach proved to be very efficient on reliable links with small buffers.

Today's networks generally do not meet the requirements listed above . With excessive buffering a packet loss in reliable links occurs on the complete saturation of the bottleneck buffer instead of the moment, when the congestion really occurred. This packet drop postponement results in bandwidth overestimation by TCP connection and consecutive packet losses that come from persistently full buffers. Moreover, this saturation of large buffers causes a delay increase, and big latency is surely experienced not on this TCP connection only, but on all network activities that share with it the same part of the network path.

To illustrate the situation, we can mention that 1 MB buffer is able to carry 8 seconds of data for 1 Mb/s connection. Provided that this is a bottleneck buffer and it stays full because of some big data transmission, we have 8 seconds queuing delay for any packet that passes this hop. This delay is much greater than common RTT for continental connections (about 100 ms). This huge latency makes interactive sessions (like ssh or web surfing) practically unusable and surely disallows the use of realtime networking like VoIP.

There are several approaches to fight with this problem. One of them is traffic prioritization or, more generally, Quality of Service (QoS) [2] which is out of scope of this article, because it does not help to improve the bufferbloat situation

The work is supported by the RFBR grant #11-07-00549-a

and does not decrease the latency for general applications. However, QoS is a very important mechanism which is used by realtime applications such as VoIP, IP-TV or online gaming.

The other approach is Active Queue Management (AQM) which attempts to set correct queue limits and either drop packets that overcome these limits or notify the sender about the congestion with ECN. These techniques try to eliminate the bufferbloat problem by removing the excessive buffering.

And finally, the last approach is to try to avoid the congestion without exhaustion of buffers. This is the most complex approach, because congestion can be of different nature. Despite the fact that congestion and excessive buffering are two different problems, and the best practice is to combine queue management and early congestion avoidance, congestion avoidance itself may solve the bufferbloat problem, if congestion is detected early, before buffers are saturated and latency heavily increased.

This paper consists of three sections. The first section very briefly discusses the current AQM solutions. The second section is devoted to the TCP congestion problem. It lists present-day TCP congestion avoidance algorithms and covers the latency increase problem. The last section introduces a version of ARTCP revised by the authors – the delay-based congestion avoidance algorithm that separates congestion avoidance and reliable delivery logic, and thus can be very efficient in lossy networks. The revised version does not use RTT for congestion avoidance, but uses pacing interval measurements only. So, it does not use indicators measured in the reverse direction and, unlike most other delay-based algorithms, behaves well, when a reverse direction path suffers from the congestion or delayed acknowledgments are used.

II. AQM SOLUTIONS

The problem of the excessive buffering is not new, it was discussed by John Nagle in 1985 [3] with an introduction of a "fairness" concept that suggested replacing a single FIFO with multiple queues (one per each source host) and service them in a round-robin fashion. This suggestion could limit the latency increase problem with one host and help to prevent a misbehaving host from making packet switch totally unusable for the others. Surely, there were no suggestions on single queue limitation and attempts to overcome the host bufferbloat problem.

The problem of bufferbloat became more serious today, when RAM is much cheaper and there are hops tuned for highspeed links (i.e. 1 Gb/s or even 10 Gb/s interfaces) that have to deal with relatively slow connections (about several Mb/s). So, this problem has been "rediscovered" today [1] and is actively discussed [4] by networking experts, giving a born for projects like bufferbloat.net.

Unfortunately, the excessive buffering can not be solved by simple static buffer management in several scenarios [4], and there is a need in Active Queue Management (AQM). The idea to detect congestion by the queue size and try to manage this size (and congestion) by dropping packets or marking them and, thereby, notifying the connection about congestion is not new. It is actively used in Random Early Detection (RED) algorithm presented by Van Jacobson and Sally Floyd in 1993 [5].

This AQM scheme monitors the average queue size and drops (or marks if ECN is used) packets based on statistical probabilities. It became a classic AQM implementation that was considered useful and strongly recommended for deployment in RFC 2309 [6]. Despite this fact, there is a strong criticism of RED [7, 8, 9, 12], mostly because of the difficulty in its configuration and the necessity of a rather large buffer in the bottleneck for RED to have time to react.

Nevertheless, RED is the most widely deployed and available AQM, which is the base for a wide variety of AQM algorithms. Some of them are presented in [10].

However, it should be mentioned that according to [4] "as much as 30 percent of residential broadband networks run without any queue management". Jim Gettys in his blog [11] names the main reason of "spotty deployment of AQM by network operators." It is a requirement of tuning and possibility of "trouble if not tuned properly". Also, he says with reference to Van Jacobson about two bugs in the classic RED discovered and proved by Kathy Nichols.

Van Jacobson considers [4], that RED can not be used with wireless nets because of the huge dynamic range presented by them. Also, he states there about the lack of the information helpful for determining the correct queue size RED can acquire: "I helped put people on the wrong track with RED which attempts to extract some information from the average queue size, but it turns out there's nothing that can be learned from the average queue size." He names the indicator used by BLUE and its successors "much more reliable" and pronounce his work with K. Nichols on RED Lite.

BLUE [12] is the AQM algorithm that uses the packet loss and link utilization history to manage congestion. BLUE maintains a single probability which it uses to mark (or drop) packets, when they are queued. If the queue is continually dropping packets due to a buffer overflow, BLUE increments the marking probability, thus increasing the rate at which it sends back congestion notification. Conversely, if the queue becomes empty or if the link is idle, BLUE decreases its marking probability.

BLUE, as any other single-queue AQM algorithm, treats all flows as a single aggregate. It brings unfairness to its behavior, because some single aggressive flow flooding the packets into the queue can push out some packets, that belong to other flows. To overcome this unfairness, the authors of BLUE presented the other AQM algorithm – Stochastic Fair Blue (SFB) which is a BLUE modification that hashes flows with a Bloom filter and maintains different mark/drop probabilities for every bin.

The other AQM algorithm, that attempts to save the protection provided by per-flow scheduling mechanisms, but combine it with its own simplicity, is RED-PD [13] which is the acronym for RED with Preferential Dropping. This is an AQM algorithm based on RED that uses the packet drop history at the router to detect high-bandwidth flows at the time of congestion and preferentially drops packets from these

flows. This AQM was designed to be effective at controlling high-bandwidth flows using a small amount of state and very simple fast-path operations in an environment dominated by end-to-end congestion control and a skewed bandwidth distribution among flows in which a small fraction of flows accounts for a large fraction of bandwidth.

Despite the intensive research in this area, there are still a lot of questions [4, 14, 15, 16] in AQM efficiency. Some of them are related to AQM performance, its behavior in heterogeneous, dynamic or unreliable environments; others consider security problems. One of such problems is DDoS attack vulnerability. Considerations of [17] states that the existing AQM algorithms are "rather vulnerable to spoofing DDoS attacks". However, this paper presents Resilient SFB (RSFB) algorithm as a solution of this problem, but there is surely the area for further research.

III. TCP CONGESTION AND QUEUING

TCP is the most widely used transport protocol. Designed to efficiently consume available bandwidth of dynamic nets with an unknown topology, it tries to achieve the goal by continuous monitoring of the network capacity. Until a packet loss occurs, TCP constantly increases its congestion window as a reaction on the acknowledgment. Unless limited by a receiver's window or application providing data for transmission, it will supply a link with more and more data being sent in a burst limited by the congestion window.

This technique is used to determine the bandwidth, starting with a small value and increasing it until a drop caused by congestion happens. As a reaction on the drop, TCP decreases its congestion window (halves it or drops to initial value depending on the use of "fast recovery"). After that, it increases its window again. The idea of this balancing determined by four intertwined essential TCP algorithms: slow start, congestion avoidance, fast retransmit, and fast recovery – defined in [18], is to keep the network operating near the inflection point, where the throughput is maximized, the delay is minimized, and a little loss occurs.

This point is known to be a bandwidth-delay product (BDP), the value that corresponds to the amount of data "in flight" exact for total link capacity consumption without congestion. BDP was a recommendation for the buffer size, and internet researchers and engineers had to advocate for sufficiently large buffers [1], because TCP uses burst transmissions and the lesser buffers could result in heavy losses.

However, these buffers lead to another problem: with their presence, they became a part of a "pipe" that TCP tries to fill to be efficient. The drop indicator used to estimate the available bandwidth triggers, when the bottleneck buffer is full. It results in three problems: TCP overestimates the link throughput, the latency increases due to the growth of the queuing delay, there is no space in full buffer to absorb the burstiness of the network.

Despite this fact, firstly recognized in 1989 [1], the packet loss remains to be the congestion indicator in modern TCP implementations. So, CUBIC [19] is a default congestion control algorithm in Linux. This congestion avoidance algorithm is loss-based, and thus it suffers from the pointed problems. For instance, [1] demonstrates the bufferbloat problem in CUBIC.

The reasons of such a behavior are understandable, and there is a strong interest in better behaving algorithms that take the delay indicator into account. The problems of existing such an efficient congestion control were, for example, discussed at the Linux Plumber Conference in 2011 [21], where the suggestion about refining a delay-based portion of hybrid congestion control algorithms was declared.

The idea of a delay-based congestion control is apparently not new, it was considered by Raj Jain in 1989 [22]. He suggested using a black-box model of the network, where changes in round-trip delays can be used as an implicit feedback, and stated that achieving congestion avoidance in heterogeneous networks is difficult, because the congestion feedback from one subnetwork may have no meaning to sources on other subnetworks.

The delay-based congestion control is promising for the networks where the latency is important, because delay-based algorithms tend to minimize queuing delays. It includes the networks with an active use of realtime data connections like VoIP or interactive sessions where latency degrades experience very much. Also, the connections with delay-based congestion control are known to be more efficient than loss-based in scenarios, where many connections share the same network path, because of being more fair and thus being more capable in an effective cooperation consuming a limited resource. However, these algorithms are less aggressive (that is considered as their plus from one point of view) and so, sharing the link with the greedy loss-based algorithm (like TCP CUBIC), they will get the less part of an available bandwidth.

These considerations underlie the growing interest in hybrid congestion avoidance algorithms. The research in this area resulted in the appearance of several hybrid congestion control algorithms in the second half of 2000's. Microsoft Research introduced Compound TCP [20] in 2005. And University of Illinois developed TCP-Illinois [23] in 2006. Both algorithms were designed for high-speed links as a reaction of TCP Reno too slow bandwidth utilization. They are rather aggressive and neither TCP-Illinois (see 'Background and motivation' section of [23]) nor Compound TCP (see Section III of [20]) were designed with a small latency goal in mind. Moreover, they are considered [24] rather unfair in some circumstances.

The other interesting congestion avoidance algorithm, that is to avoid latency increase problem, is TCP-NV [25] presented by a host networking team member from Google, the main developer of TCP Vegas [26] Lawrence Brakmo in 2010. The goal of this algorithm is to prevent queue build-up and packet loss, while making full utilization of the available bandwidth. TCP-NV is based on TCP-BIC [27] and behaves like it, unless the queue build-up is detected. On queue build-up detection, it decreases the congestion window proportionally to the amount of the detected build-up. The main problem in this approach, named by Brakmo, is a noise in RTT measurement caused by

such optimization techniques as TSO, LRO, and interrupt coalescence. Since TCP-NV is optimized for data-centers, it has to deal with these hardware tricks inherent to 10G NICs and has to filter affected measurements with various quirks.

The other problem RTT-based congestion control engines suffer from is the reverse congestion which results in ACK delivery delays and thus can be falsely treated by these engines as forward congestion indication. This problem was mentioned by Brakmo at the Linux Plumbers Conference in 2010 and is well illustrated for TCP Compound in [24]. Brakmo's suggestions for this problem were to prioritize pure ACKs in host and switches/routers and to measure reverse delay with the help of a new TCP option and adjust appropriately.

Besides hybrid congestion avoidance algorithms, there are some delay-based that worth mentioning. The most famous delay-based algorithm is TCP Vegas [26]. It can not rival with loss-based algorithms on the same link due to its lesser aggressiveness, does not fit high-speed links, is not efficient in nets with very variable delays and can lead to persistent congestion in some circumstances [28]. However, it is a sign algorithm, that showed vitality of the delay-based approach and gave a birth to the whole family of delay-based congestion avoidance algorithms.

FAST TCP [29] is one of Vegas-derived algorithms. It is a delay-based algorithm designed for networks with large bandwidth-delay product with responsiveness in mind, that seeks to maintain a constant number of packets in queues throughout the network by measuring the difference between the observed RTT and the “base RTT”, which is RTT observed without queuing. It is a promising algorithm, but its fairness is very sensitive to correct “base RTT” estimation [30] and, being TCP Vegas descendant, it suffers from some inherent problems [28]. Another issue of FAST TCP, because of which it can not be widely used, is patent restrictions imposed by its authors.

The other attitude to delay-based congestion avoidance is made by the authors of a new CDG algorithm [31] developed by CAIA, Swinburne University of Technology (Melbourne). They claim that the biggest limitation of the most delay-based congestion control algorithms is to infer congestion, when the path delay hits or exceeds certain thresholds. This approach suffers from the circumstance, that it is very hard to set the meaningful threshold, unless there is a reliable information about the path that packets will take. (It is mentioned above, for an instance, that FAST TCP suffers from relative unfairness in case of inability to accurately estimate path “base RTT” or TCP Vegas causes persistent congestion, when its threshold is set incorrectly.) They suggest that the threshold should be abandoned and “delay-gradient” congestion control, that relies on relative movement of RTT and adapts to particular conditions along the paths each flow takes, be used. The CDG algorithm is very interesting and seems to utilize the available bandwidth rather efficiently compared with loss-based CUBIC and NewReno algorithms [32]. However, available results of tests are ambiguous (especially from latency point of view) and there are few experiments to make a conclusion.

One of the queuing reasons (excluding congestion, when any queue will obviously become full) is the burstiness. To decrease this effect, TCP pacing may be worth applying. TCP pacing is the technique consisting in spreading the transmission of TCP segments across the entire duration of the estimated round trip time instead of having a burst at the reception of acknowledgments from the TCP receiver. Despite offering better fairness, throughput and low drop rate in some situations, pacing may be dangerous and, contrary to intuition, can cause significantly worse throughput because of congestion signal postponement and synchronized losses [33]. However, these results vary for different congestion control algorithms and pacing rates. From [34] it can be concluded that pacing can cause throughput degradation in mixed (both paced and non-paced flows) scenarios for various loss-based congestion control algorithms. On the contrary, [34] shows the better characteristics for delay-based congestion control algorithms, when pacing is used, including fairness enhancements due to the effect of reducing queue variation.

Let us summarize the above observations on TCP congestion and latency. Loss-based congestion avoidance algorithms saturate the bottleneck queue, that results in a great deal of queuing delays in the case of excessive buffering. Hybrid algorithms are promising in competing with loss-based in bandwidth utilization. However, their fairness must be revised, and they have a danger of issuing problems of both loss-based and delay-based approaches in different scenarios depending on the way, in which they combine loss-based and delay-based portions. In general, delay-based congestion control algorithms obviously have the best latency. These algorithms issue several difficulties: if they are designed to rely on a certain threshold, this threshold must be correctly estimated, otherwise the errors in estimation may result in algorithm work degradation and even persistent congestion, delay-based algorithms need rather precise time interval (i.e. RTT) measurement, the work of delay-based algorithm that uses RTT may be severely degraded by the reverse congestion, so forward-only indicators like “single trip times” are much more reliable. TCP pacing may lead to a worse behavior for loss-based congestion avoidance algorithms and even increase latency, but it improves characteristics of flows controlled by delay-based algorithms.

IV. Introducing ARTCP

Taking into account the above considerations, we introduce a revised variant of ARTCP – the TCP modification with delay-based congestion control that uses pacing and pacing intervals, which is a low latency loss-insensitive solution with a good throughput.

Adaptive Rate TCP (ARTCP) [35] does not use the congestion window at all. Instead, it uses pacing intervals measurement, so called “duty factor”, to determine the congestion and control the data flow speed. The idea of ARTCP is to remove burstiness by applying pacing with an adaptive rate. The pacing here is a key element, because it determines a sender's speed, and the analysis of its alteration is used to detect the congestion. To achieve this goal, we calculate pacing intervals in two points. The first point is a sender itself, here, pacing interval is just a time difference between the transmission of two successive segments, which is

determined by currently used transmission rate. The second point is a receiver. On its side, we calculate the time difference between two successive arrivals and send it back to the sender with an acknowledgment [36]. To prevent incorrect calculation of pacing intervals the sender provides the receiver with additional information, so called a “packet sequence number” (PS), so that the receiver could determine (invisible for it in some cases without this information) losses and packet reordering efficiently, and thus filter out packets that are not successive and avoid incorrect estimation of pacing intervals.

We treat these pacing intervals estimation as measures of transmission and reception rates. Thereby, we can use them on the sender as a congestion indicator. The situation when the reception rate is stabilized in the presence of the growing transmission rate can be certainly taken in as congestion. This variant is attractive comparing with the fallback RTT-based delay estimation, because it strictly uses forward direction time measures only, and thus, it is totally insensitive to the reverse congestion which causes troubles for most delay-based congestion avoidance algorithms.

Because ARTCP is not so sensitive to reverse direction delays, it has no limitations of RTT-based congestion avoidance algorithms that need immediate acknowledgment dispatch and behave badly, if delayed acknowledgments are used. So, with ARTCP we can delay acknowledgment in case receiver is expecting, that it will transmit data soon, or it is waiting for a start of the scheduled data transmission, and piggyback this acknowledgment, as well as pacing interval information, with these data, thus increase the efficiency of bandwidth utilization.

To behave well, ARTCP must follow the several states: slow start, initial multiplicative decrease, recovery, fine tuning, and multiplicative decrease. Slow start stage is to rapidly determine the available bandwidth. This state is the first after the connection setup, and there the sender increases its transmission rate exponentially. Here, we have to deal with two contradictory challenges: the first one is to occupy the available bandwidth as soon as possible, the second is to prevent big latency growth which is inescapable, if the slow start caused a congestion. Congestion is mandatory in order to find the throughput limit. Our goal is to examine it as early as possible and to drop the transmission rate accordingly to compensate the congestion.

Compared with loss-based TCP congestion avoidance algorithms, ARTCP has a better chance to achieve the smaller transmission rate oscillation in the slow start, because it has no need to saturate the bottleneck queue to discover the congestion. Instead, it examines the pacing intervals sent back by the receiver and checks whether they are not increasing, while transmission rate still continues increasing. Since we found the transmission rate we can rely, estimated with pacing intervals taken from the receiver, we have a bandwidth estimation. The current transmission rate will be surely greater than the estimated rate, because congestion is mandatory. So, we must compensate the congestion by reducing the transmission rate to the value that is less than the estimated one.

To perform the compensation, we do the initial multiplicative decrease which reduces the transmission rate, and goes to the recovery state in order to return to the estimated rate, when congestion compensation is over. Congestion compensation here is the amount of “superfluous” packets we sent, while the transmission rate is greater than our estimation. So, we must set the transmission rate lower than the estimation and increase it to the estimation in the way that allows the exact compensation we need.

It is a difficult question to what value it's better to drop the rate and how then it should increase. As a current solution, we suggest setting the rate as low as necessary, so that the recovery finishes as soon as possible, and increasing it linearly. This will allow the bottleneck queue size to decrease more rapidly, and thus have a propitious impact on the experienced latency. While linear increase is performing, we must constantly monitor the receiver's feedback, because the estimated bandwidth can be reduced by the other connection activities. The lack of doing that will result in bandwidth overestimation and in this case it will bring the network to the congestion. To detect bandwidth reduction in the recovery state, we can examine, whether the receiver's pacing intervals do not go down, while we proceed with transmission rate increasing. Since we achieved the estimated rate or the new estimation is discovered, we accept this rate as transmission rate and move to a fine tuning state.

Fine tuning is the most challenging state. It must adjust the transmission rate to changes in the available bandwidth and be able to rapidly react to heavy changes (decreases and increases). We suggest making adjustments in a probabilistic manner, and at the same time have the guards that can detect big changes of available bandwidth. Unless delay increase is detected (increase of the receiver's pacing intervals is observed), we set positive transmission rate speedup probability, so that the transmission rate can be increased. This allows us to increase the transmission rate, when the available bandwidth grows.

There is an interesting question, how we can inspect that the available bandwidth heavily increased. As a solution, we can increase the speedup portion, while we have a positive feedback from these actions, and reset it on any negative feedback. Applying these actions, we must be very careful not to provoke a congestion. To deal with this problem, we may address to the speedup portion increase only after we are assured about absence of negative impact on latency with acknowledgments of the data sent after previous increase. However, this speedup portion manipulation needs a further investigation, and we currently do not implement it, but deal with the probability of (mainly) a linear speedup.

At the same time we are looking for a delay increase. If it is detected, we set up slowdown probability, instead of speedup. As opposed to speedup, slowdown portion can be proportional to the current transmission rate, that makes us rather rapidly react on latency increases. In any case we must provide two mechanisms for slowdown: one – for a small tuning of the transmission rate, and another – for multiplicative decrease. The first is applied when we have a small receiver's pacing interval increase or have no decrease after the transmission rate

was increased. We call it “speedup undoing”. The other mechanism must be invoked if we observe big latency increase which is considered as a heavy congestion. In this case ARTCP yields the portion of its bandwidth share by exponential transmission rate drop. This drop is done by multiplicative decrease state which divides the current transmission rate by a certain ratio and then checks, whether it solved the latency problem. If not, it continues multiplicative decrease, otherwise it passes the connection to the fine tuning state.

This ends up a concise overview of ARTCP flow control states. As it is presented above, ARTCP uses the only indicator of congestion: latency increase which is examined by calculation of forward direction characteristics only. So, ARTCP is insensitive to packet losses and reverse congestion, what makes its idea very attractive. It is designed for lossy networks and solutions that require low latency, and it is supposed to be the right approach to the solution of the bufferbloat problem for links, where congestion is caused by TCP traffic. The main limitation of ARTCP is that it is a sender+receiver solution and it requires that both ends implement ARTCP, in the contrary to many sender-only TCP congestion avoidance solutions. However, ARTCP is designed to make it possible to use any TCP congestion avoidance algorithm as fallback and to switch to it on the fly, if the other end does not implement ARTCP [36].

[1] J. Gettys, K. Nichols, “Bufferbloat: Dark Buffers in the Internet” // Magazine Queue – Virtualization, vol. 9, issue 11, New York: ACM, 2011

[2] M. Marchese, QoS over heterogeneous networks, N.J. : John Wiley & Sons, 2007

[3] J. Nagle, On Packet Switches With Infinite Storage. // RFC 970, 1985.

[4] V. Cerf, V. Jacobson, N. Weaver, J. Gettys, “BufferBloat: What's Wrong with the Internet?” // Magazine Queue – Log Analysis, vol. 9, issue12, New York: ACM, 2011

[5] S. Floyd, V. Jacobson, “Random Early Detection gateways for Congestion Avoidance”, IEEE/ACM Transactions on Networking, V.1 N.4, August 1993, p. 397-413

[6] B. Braden, D. Clark, J. Crowcroft, B. Davie, S. Deering, D. Estrin, S. Floyd, V. Jacobson, G. Minshal, C. Partridge, L. Peterson, K. Ramakrishnan, S. Shenker, J. Wroclawski, L. Zhang, Recommendations on Queue Management and Congestion Avoidance in the Internet. // RFC 2309, 1998.

[7] V. Jacobson, K. Nichols, K. Poduri, “RED in a Different Light”, unpublished, 1999.

[8] M. May, J. Bolot, B. Lyles, “Reasons not to deploy RED”, technical report, 1999

[9] S. Harhalakis, N. Samaras, V. Vitsas, “An Experimental Study of the Efficiency of Explicit Congestion Notification” // PCI '11 Proceedings of the 15th Panhellenic Conference on Informatics, pp. 122-126

[10] S. Floyd, References on RED (Random Early Detection) Queue Management, web page, 2008 // url: http://www.icir.org/floyd/red.html

[11] J. Gettys, blog of the author of bufferbloat.net project, web page // url: http://gettys.wordpress.com/

[12] W. Feng, D. Kandlur, D. Saha, K. Shin, “BLUE: A New Class of Active Queue Management Algorithms”, U. Michigan Computer Science Technical Report (CSE–TR–387–99), 1999

[13] R. Mahajan, S. Floyd, D. Wetherall, “Controlling High-Bandwidth Flows at the Congested Router”, Proceedings to 9th International Conference on Network Protocols (ICNP), 2001

[14] S. Bohacek, K. Shah, G. Arce, M. Davis, “Signal Processing Challenges in Active Queue Management” // IEEE Signal Processing Magazine, 2004, pp. 69-79

[15] K. Shah, S. Bohacek, E. Jonckheere, “On the performance limitation of Active Queue Management (AQM)” // 43rd IEEE Conference on Decision and Control, 2004, pp. 1016-1022

[16] A. Kuzmanovic, “The power of explicit congestion notification” // ACM SIGCOMM Computer Communication Review, vol. 35, issue 4, New York: ACM, 2005

[17] C. Zhang, J. Yin, Z. Cai, “RSFB: a Resilient Stochastic Fair Blue algorithm against spoofing DDoS attacks” // International Symposium on Communication and Information Technology (ISCIT), 2009

[18] M. Allman, V. Paxson, E. Blanton, TCP Congestion Control // RFC 5681, 2009

[19] S. Ha, I. Rhee, L. Xu, “CUBIC: A New TCP-Friendly High-Speed TCP Variant” // ACM SIGOPS Operating Systems Review, vol. 42, issue 5, 2008

[20] K. Tan, J. Song, Q. Zhang, M. Sridharan, “Compound TCP: A Scalable and TCP-friendly Congestion Control for High-speed Networks” // 4th International workshop on Protocols for Fast Long-Distance Networks (PFLDNet), 2006

[21] S. Hemminger, “A Baker's Dozen of TCP bakeoff?” // Linux Plumbers Conference, 2011

[22] R. Jain, “A Delay Based Approach for Congestion Avoidance in Interconnected Heterogeneous Computer Networks” // Computer Communications Review, ACM SIGCOMM, 1989, pp. 56-71

[23] S. Liu, T. Basar, R. Srikant, “TCP-Illinois: A Loss and Delay-Based Congestion Control Algorithm for High-Speed Networks” // Proceedings of the 1st international conference on Performance evaluation methodolgies and tools, 2006

[24] D. Leith, L. Andrew, T. Quetchenbach, R. Shorten, K. Lavi, “Experimental Evaluation of Delay/Loss-based TCP Congestion Control Algorithms” // Proceedings of the 6th International Workshop on Protocols for Fast Long-Distance Networks (PFLDnet 2008), 2008

[25] L. Brakmo, “TCP-NV: Congestion Avoidance for Data Centers” // Linux Plumbers Conference, 2010

[26] L. Brakmo, L. Peterson, “TCP Vegas: End to End Congestion Avoidance on a Global Internet” // IEEE Journal on selected Areas in communications, vol. 13, 1995, pp. 1465-1480

[27] L. Xu, K. Harfoush, I. Rhee, “Binary Increase Congestion Control for Fast, Long Distance Networks” // INFOCOM 2004. Twenty-third AnnualJoint Conference of the IEEE Computer and Communications Societies, vol. 4, 2004, pp. 2514-2524

[28] R. La, J. Walrand, V. Anantharam, “Issues in TCP Vegas”, UCB/ERL Memorandum, No. M99/3, UC Berkeley, 1999

[29] C. Jin, D. Wei, S. Low, “FAST TCP: Motivation, Architecture, Algorithms, Performance” // INFOCOM 2004. Twenty-third Annual Joint Conference of the IEEE Computer and Communications Societies, vol. 4, 2004, pp. 2490-2501

[30] L. Tan, C. Yuan, M. Zukerman, “FAST TCP: Fairness and Queuing Issues”, IEEE Communications Letters, vol. 9, no. 8, 2005, pp. 762-764

[31] D. Hayes, G. Armitage, “Revisiting TCP Congestion Control using Delay Gradients” // Proceedings of 10th International IFIP TC 6 Networking Conference, 2011

[32] G. Armitage, “A rough comparison of NewReno, CUBIC, Vegas and ‘CAIA Delay Gradient’ TCP (v0.1)”, CAIA Technical report 110729A, 2011

[33] A. Aggarwal, S. Savage, T. Anderson, “Understanding the Performance of TCP Pacing” // Proceedings of the IEEE INFOCOM 2000 Conference on Computer Communications, 2000, pp. 1157 – 1165.

[34] D. Wei, P. Cao, S. Low, “TCP Pacing Revisited” // Proceedings of the IEEE INFOCOM 2006 Conference on Computer Communications, 2006, pp. 56-63

[35] I. V. Alekseev, V. A. Sokolov Compensation Mechanism for Adaptive Rate TCP. // 1-St International IEEE/Popov Seminar "Internet: Technologies A and Services". P. 68-75, October 1999

[36] A. Sivov, “ARTCP packet structure. Features of the Formation and Processing of ARTCP Headers in Linux Network Subsystem” // Modeling and Analysis of Information Systems, vol. 18, №2, Yaroslavl: Yaroslavl State University, 2011, pp. 129-138