Network Protocols for Applications of Shared Virtual Reality

Jiri Hnidek

Technical University of Liberec
Studentska 2
Czech Republic 461 17 , Liberec

jiri.hnidek@tul.cz

ABSTRACT

Files are usually used for exchange of 3D data between graphical applications, but this approach is not feasible for applications of shared virtual reality. Thus a network protocol is used for this purpose. Two antithetical requirements are claimed for such protocol. Protocol has to be partially or completely reliable. Neither delay jitter nor too high delay are acceptable. This paper explains and analyzes new version of protocol called Verse. This improves shortcomings found in UDP, TCP, SCTP and DCCP transport protocols. The Verse protocol was designed for sharing 3D data between applications of shared virtual reality. This paper also contains results of experiments comparing suitability of network protocols for application of shared virtual reality.

Keywords
Shared Virtual Reality, Network Protocol, Transport Protocol, Delay, Delay Jitter, Verse

1. INTRODUCTION

Applications of shared virtual reality (ASVR) require transferring of information (e.g. position of avatar [KCJ0]) with small delay and delay jitter, because flickering of object motion is disruptive for an observer of virtual reality. Let consider situation, when users of ASVR cooperate in this environment and try to create large scene (e.g. city with buildings). When users create these objects, then movements of all entities (objects, vertexes) is unpredictable. Each user should see what other users are doing in real-time and movement of shared entities should be smooth as much as possible. Many users of ASVR can create large traffic. Moreover some activities of users can cause burst traffic (uploading of existing object, sculpt painting, etc.).

UDP protocol is usually used for sending real-time data. On the contrary TCP is usually used for transferring static 3D data, because it is reliable stream protocol. When users of ASVR want to edit shared geometry and topology of 3D objects, then partial reliability as well as low latency is required.

It will be proved that any transport protocols as is can not meet those needs. It will be shown that new Verse protocol can effectively meet both needs.

2. CONDITIONS OF EXPERIMENTS

A special client-server application was developed for testing all above network protocols (Fig. 9). Network protocols were tested in real network environments, but comparison of protocols required different approach.

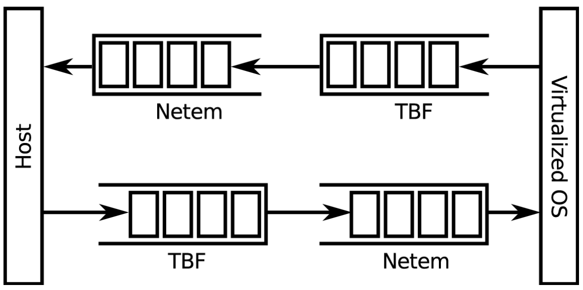


Figure 1. Each operating system had TBF engress qdisc and Netem ingress qdisc. Connection of Netem and TBF qdisc allowed to simulate real network conditions.

It was necessary to set exact parameters of link between client and server applications for tested protocols. For this reason the server application run on virtualized Linux operating system and the client application run on host. Virtual link between host and virtualized OS was modified with Linux traffic control to simulate real network conditions . The Netem [Hem05] queueing disciple (qdisc) was used for setting delay and delay jitter. The TBF qdisc was used for setting limited bandwidth. It is important to note, that unmodified link between host and virtualized OS had delay 0.5 ms and average delay jitter was 0.03 ms. Configured values of delay and delay jitter were 10 times higher than values on unmodified link. This delay simulated local are network. The bandwidth of the link was limited using TBF qdisc to 256 kb/s. The MTU of link was 1500 B.

3. METHODS OF EXPERIMENTS

Particle system was used to simulate wide range of users working with 3D data. Two particle systems were pre-generated. The particle system containing 100 particles was used for testing network protocols at modified virtual link, because it generated decent traffic and visualization. The particles system with 1000 particles was tested in real network environment. The particle system is simple simulation of bouncing balls, but it is assumed that movement of particles simulates some type of unpredictable movement and thus only position of particles was sent through network. Graph at Fig. 2 shows time slope of particles in scene.

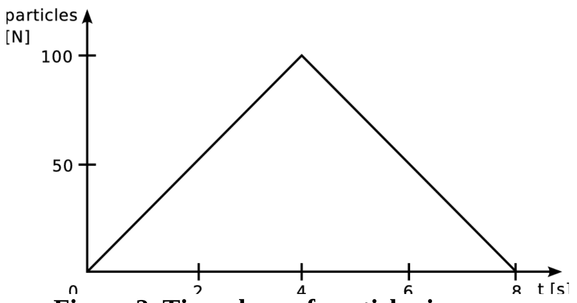


Figure 2. Time slope of particles in scene. Duration of pre-generated particle system was 8 seconds using 25 frames per seconds (FPS).

The particle systems were generated with 25 frames per second (FPS). The illusion of slow motion depends on many factors (resolution, distance of user from the screen, etc.). In the worst case this illusion is broken, when delay of received particle is longer then 40 ms. Such particle is visualized as problematic delay.

The main communication between client and server is started by client application by a request to send particles. When server receives client request, then it starts to send positions of all moving particles every 40 milliseconds back to the client. The PMTU is used during this simple handshake to discover maximal size of packet. The position of each moving particle is added to the packet in a simple message containing particle id, frame number and vector of position. When there are no other particles that could be added to the packet or packet is full, then packet is send to the client. The client application tries to receive packets with positions of particles and it visualizes differences between received and pre-generated particle system. First 10 received blind packets are used to compute average delay between client and server before real data transfer.

4. TESTING OF TRANSPORT NETWORK PROTOCOLS

UDP and TCP

UDP [Pos80] was the first tested protocol. UDP is unreliable datagram protocol widely used in gaming applications for its low latency. UDP is not congestion aware and generated traffic can cause congestion collapse. We can see at Fig 4 that particles transported with UDP had low delays except the period, when congestion occurred. Average delay was bigger than 40 milliseconds and packet loss was visually noticeable during connection. On the other hand tests proved that using pure UDP in ASVR is not feasible, because lost data are not resent and it leads to inconsistency of shared data in ASVR. UDP protocol could be used for ASVR, but re-sending of lost packets has to be solved at the application layer.

Contrary TCP protocol [Pos81] is not widely used in gaming applications, because of the consequences of its reliability mechanism. When one single packet is lost, then proceeding of all following packets is blocked until the lost packet is resent as we can see at Fig 5. Such behavior lead to a sudden stop of motion and high delay up to 1 second. Tests of TCP protocol proved that using TCP in ASVR is possible only in situations where bandwidth is bigger than the highest generated bitrate, there are no other concurrent transmission and RTT is much smaller than 1/FPS. It is usually not possible to guarantee such conditions in real networks and thus re-transmission of lost data leads to very big delays.

Al-Regib and Altunbasak proved in [ARA04], that combination of UDP and TCP connection can be effectively used for streaming of large 3D data sets. This approach can be also used in ASVR for loading large data sets, but it does not solve all specific problems of ASVR, where many users share the same 3D data set.

We can see at Fig. 4 and Fig. 5 that tests of UDP and TCP protocols on real network produced similar results as tests at modified virtual link.

SCTP

SCTP [Ste07] is a modern message based transport protocol. It can act as reliable or partially reliable protocol. SCTP does not provide reliable order delivery, because it is based on message. When reliable variant of SCTP is used, then there is no need to wait for re-transmission of previous lost packets. This feature could be considered as great benefit for ASVR, because there is smaller average delay of received particles. On the other hand this caused flickering of received particles. Someone can argue that this is visually more confusing than big delays of TCP. Flickering could be theoretically removed by adding a time stamp to each message, but re-transmission of obsolete particle position is not effective approach. Every re-transmission of obsolete particle position makes congestion worse. We can see at Fig. 6 that average delay of reliable variant of SCTP is bigger than average delay of TCP.

Partial reliable variant of SCTP can specify time to live (TTL) of each message. It means that sender tries to re-transmit lost message only for specified time. When the TTL of message is reached, then unsent message is dropped. Partially reliable variant of SCTP removed flickering of received particles, when TTL was smaller then 0.5/FPS, but important feature of reliability was lost. Using partially reliable variant of SCTP in ASVR is not acceptable for the same reason as pure UDP protocol. The results of tests (Fig. 7) gave similar result as test of UDP protocol.

DCCP

DCCP [KHF06] is a congestion friendly unreliable datagram protocol. Congestion control of DCCP protocol should be better than congestion control implemented at application layer on top of UDP, because DCCP can send ECN capable packet that helps to detect congestion without dropping packets. Tests of DCCP protocol proved that implementation of this protocol in Linux is not ready yet for practical deployment. When some error occurs (e.g. many packets loss), then memory of machine can go out of the limit [LF09]. Thus results of DCCP test were not comparable with tests of other transport protocols.

5. VERSE PROTOCOL

All tested transport protocol failed in some way. Unreliable or partially reliable protocols do not resend lost packets. Reliable protocols try to resend all lost data and it causes high delays. Verse protocol uses different approach, because it tries to resend only actual data and obsolete data are dropped.

Verse protocol [BSS06] [SB07] is an application protocol designed for sharing 3D data in ASVR. It uses UDP protocol as a transport layer. UDP it is widely used datagram protocol and it allows implementation of own effective resend mechanism at application layer.

Principles of Verse Protocol

The Verse protocol uses client-server architecture. It means the Verse server holds data and distributes changes of shared data between connected clients. For example, when a client sends a message containing new position of the object to the server, then server changes local position of the object and re-transmits this change to all clients interested in this object. From this point of view the Verse protocol behaves like a network protocol used in gaming applications [WCC+09]. The Verse protocol allows much more. Applications can share not only the object transformations but also geometry and topology of objects, materials, textures, UV coordinates etc. On the other side, the Verse protocol does not allow to use multicast connections because each client is interested in different set of objects.

Resend Mechanism of Verse Protocol

Basic principles of resend mechanism will be described on the example (Figure. 3). It is assumed, that the Verse packet with ID = 31 contained information about object position. This packet was sent from the sender to the receiver. The receiver received this packet and sent the acknowledgment packet back to the sender. The sender could be client and receiver could be server and vice versa.

After a while the sender sent new packet. This packet had ID = 32 and contained new position of the object. This packet was lost. The receiver detected this loss, when the packet with ID = 33 was received, because the receiver expected packet with ID = 32. After this, the receiver sent acknowledgment containing information about reception of packet 33 and loss of the packet 32. The next method of detection of packet loss is detection by sender using timeouts.

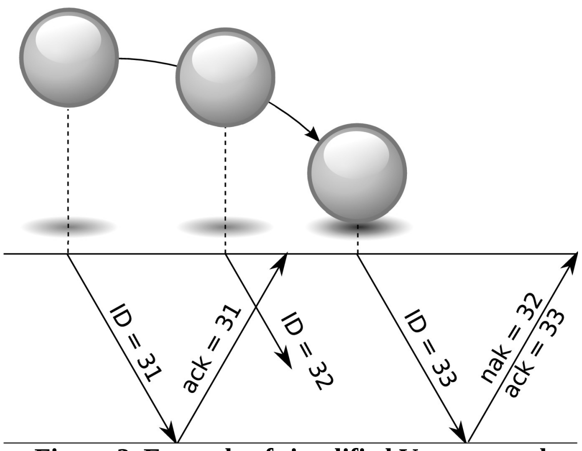


Figure 3. Example of simplified Verse resend mechanism. Each packet sent from the sender has unique ID and it contains position of sphere object. Acknowledgment packet sent from the receiver to the sender contains positive or negative acknowledgment of received packet.

Let's assume, that this acknowledgment was received by the sender. How the sender processed this acknowledgment? If the sender resent content of the lost packet 32, then the receiver would receive obsolete position of the object and it would lead to inconsistency of shared data. On the other side the lost packet could contain some useful and still valid information (for example, information about position of some other object, command to delete an object etc.). Therefor sender must pick non-obsolete data from lost packet and pack them to a new packet.

It is important to note, that the most of packet loss is caused by congestion in the network. Because network equipments try to use fair scheduling for data flows, then most of congestion is caused by traffic from the sender. Thus most of the packet loss could be effectively detected by methods described above.

If packet loss was detected only by sender using timeout, then this behavior would lead to high delays. Let's consider that retransmission timeout interval (RTO) is computed using the following formulas. Smoothed RTT (SRTT) is computed with:

$$\alpha \cdot RTT + (1-\alpha) \cdot SRTT \rightarrow SRTT \quad (1)$$

where RTT is round trip time measurement from the most recently acknowledged payload packet. The RTO is then:

$$RTO = \beta \cdot SRTT \quad (2)$$

Suggested value of constant α is 0.9 and suggested value of constant β is 2. RTT of packet could be in range of 1-100 ms in real network environment. If SRTT is 10 ms, then not-lost packets are delivered with average delay 5 ms and lost packet would be delivered with average delay 25 ms. Proposed approach used in Verse protocol allows to deliver lost packet with average delay 15 ms. If 3D scene is visualized with 60 FPS, then delay between two frames is 16.7 ms. It is obvious, that Verse protocol has high chance to resend lost packet just in time.

If TCP was used on transport layer, then the packet loss would be solved very ineffectively from our point of view. If the packet 34 was lost, then processing of all following packets would be suspended until content of the packet 34 would be delivered. Such behavior had negative effect on fluency of particle movements in tests of TCP protocol.

The real Verse protocol is more complicated, then example described above. Verse uses two types of packets. Payload packets contain payload data. Acknowledgment packets contain acknowledgments of payloads packets.

Each payload packet has unique ID (Payload ID). A sender increments the counter of sent packets every time it sends a payload packet. When the counter reaches value 2^{32} , then the counter is reset to zero value.

When payload packet is received, then receiver sends an acknowledgment packet to the sender. This acknowledgment packet has unique ID (AckNak ID) and it contains at least one message with the acknowledgment of the received payload packet. This packet could contain more acknowledgment messages (will be described later). The uniqueness of AckNak ID is guaranteed by the same mechanism as in the case of Payload ID. The receiver should send at least one acknowledgment packet for two received payload packets. The receiver should decrease or increase the ratio of acknowledgment packets, when sender detects acknowledgment packet loss. Thus the sender negotiate the ratio of acknowledgment packets.

Negative acknowledgment informs the sender, that one ore more packets were lost. The receiver detects packet loss, when expects receiving of payload packet with ID = N , but payload packet with ID > N is received. The host sends an acknowledgment packet containing all the ACK and NAK messages from the previous acknowledgment packets and following sequence:

$$nak(N), \ldots, nak(ID-1), ack(ID) \quad (3)$$

When delayed packets (considered as lost) are received, then it is possible to process non-obsolete data from these packets, but it is easier to drop them.

Delivery of acknowledgment packet is uncertain, because an unreliable datagram protocol on the transport layer is used. Therefore, probability of delivery of an acknowledge packet to other side must be maximized. All the ACK and NAK messages from previous acknowledgment packets are added to further packets, including payload packets. It is clear, that adding the ACK and NAK messages to packets should be limited somehow. The ACK and NAK messages could not be added to the packet infinitely, because traffic with low packet loss and high delay could produce long sequence of ACK and NAK messages. In this manner ACK and NAK messages would fill the whole packet in a short time.

To avoid infinite increase of the ACK and NAK messages, acknowledgment of acknowledgment has to be added to the Verse resend mechanism. The ID of the last acknowledged payload packet is added to the packet sent to the peer. This ID is called Ank ID. When the receiver receives such packet, then it is necessary to send only ACK and NAK messages for payload packets greater than Ank ID. The sender sends packets with the Ank ID until a newer acknowledgment packet is received.

The next mechanism of limiting sequence of ACK and NAK messages is compression of this sequence. Let's consider the following sequence of ACK and NAK messages:

$$\begin{gathered} ack(31), ack(32), nak(33), nak(34), \\ nak(35), ack(36), ack(37), ack(38) \end{gathered} \quad (4)$$

Such sequence could be split into the several subsequences containing only ACK messages:

$$AckSeq_i = \{ack_0(N_i), \ldots, ack_{n_i}(N_i + n_i)\} \quad (5)$$

and NAK messages:

$$NakSeq_i = \{nak_0(N_i), \ldots, nak_{n_i}(N_i + n_i)\} \quad (6)$$

where n_i+1 is the number of ACK or NAK messages in each subsequence.

Because numbers of received payload packets are constantly increasing, then original sequence could be compressed to the following sequence:

$$ack(31), nak(33), ack(36), ack(38) \quad (7)$$

in general *m* subsequences could be compressed to the following sequence:

$$\begin{aligned} &ack_0(N_0), nak_0(N_1), ack_0(N_2), \ldots \\ &\ldots, ack_0(N_{m-1}), ack_{n_{m-1}}(N_{m-1} + n_{m-1}) \end{aligned} \quad (8)$$

It is necessary to send an empty payload packet every 2 seconds to the receiver, when there is no payload data to send. It is used for computing current RTT. Empty payload packets also work as a keep alive packets. When the host does not receive any packet from its peer during 30 seconds, then this connection is considered as closed.

Tests of Verse protocol

The Verse protocol was tested in similar client-server application where transport protocols were tested. The Verse server run again on virtualized OS. Position of moving particles were sent to the Verse server from special Verse client running on the same virtualized OS. When the Verse server received positions of particles, then it tried to send these positions to the second Verse client running at host OS. The link between host and virtualized OS was modified in the same way as it is described in section 2. We can see at Fig. 8 that average delay of Verse protocol was comparable with average delay of UDP. The congestion was longer and delay was bigger than with UDP, because messages containing position of particles are not so simple as messages used for tests of transport protocols. It is price for flexibility and partial reliability of Verse protocol.

6. RELATED WORK

Work of Terrence L. Disz et. al. [DPPS95] contains first experiments with CAVE to CAVE communication. They proposed very ambitious plans of object sharing, but it was only plan and this project was canceled. Chen-Chi Chet et. al. [WCC+09] proposed Game Transport Protocol (GTP) with 4 schemes of re-transmission, but every type of retransmission scheme can re-transmit whole packet. This approach is very similar to SCTP protocol and it's inefficiency for ASVR was proved. Harcsik et. al. [HPG07] tested transport protocol and its efficiency for network games. These test were specific for network games using thin streams – they consists of small packets sent at low packets rates.

7. CONCLUSION AND FUTURE WORK

Sharing of 3D data over lossy networks is quite challenging problem. UDP, TCP, SCTP and DCCP transport protocols were tested and compared in special client-server application. It was proved that no transport protocol as is can guarantee low delays together with reliable or semi-reliable transport. Basic principles of new Verse protocol were introduced. This protocol was also tested in the client-server application. Proposed approach used in Verse protocol gave significantly better results for ASVR than simple ad-hoc solutions based on transport protocols, because Verse resend mechanism re-sends only actual data and obsolete data are dropped. New Verse protocol allows to use compression and thus further minimize congestion and delays.

Future work will be focused on design and implementation of reliable congestion control for datagram transport varying packet size with fixed sending rate of packets. Next target will be implementation of prioritization, queueing and scheduling of data, that are going to be sent to the receiver. In this way it will be able to increase probability of delivering data with high priority.

8. ACKNOWLEDGMENTS

This research is realized under the state subsidy of the Czech Republic within the research and development “Advanced Remediation Technologies and Processes Center” 1M0554 – Program of Research Centers PP2-DP01 supported by Ministry of Education. I would like to thanks to Pavel Satrapa and David Kmoch for useful suggestions and careful reading of this paper.

9. REFERENCES

[ARA04] Al-Regib, G. Altunbasak, Y. 3TP: 3-D models transport protocol. In Web3D ’04: Proceedings of the ninth international conference on 3D Web technology, pages 155–162, New York, NY, USA, 2004. ACM.

[DPPS95] Disz, T.L. Papka, M.E. Pellegrino, M. Stevens, R. Sharing Visualization Experiences among Remote Virtual Environments. In International Workshop on High Performance Computing for Computer Graphics and Visualization, pages 217–237. Springer-Verlag, 1995.

[KHF06] Kohler, E. Handley, M. Floyd, S. Designing DCCP: congestion control without reliability. In SIGCOMM ’06: Proceedings of the 2006 conference on Applications, technologies, architectures, and protocols for computer communications, pages 27–38, New York, NY, USA, 2006. ACM.

[KCJ07] Kempf, J., Chander, A., and Jo, M. Optimizing avatar environmental update in shared virtual reality environments. In Proceedings of the First International Conference on Immersive Telecommunications (ICST, Brussels, Belgium, Belgium, 2007), ImmersCom '07, ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering), pp. 1–6.

[Hem05] Hemminger, S. Network Emulation with NetEm. In Linux Conf Au, April 2005.

[HPG07] Harcsik, S., Petlund, A., Griwodz, C., and Halvorsen, P. Latency evaluation of networking mechanisms for game traffic. In Proceedings of the 6th ACM SIGCOMM workshop on Network and system support for games (New York, NY, USA, 2007), NetGames '07, ACM, pp. 129–134.

[Pos80] Postel, J. RFC 768: User Datagram Protocol, aug 1980.

[Pos81] Postel, J. RFC 793: Transmission Control Protocol, sep 1981. Updated by RFCs 1122, 3168.

[BSS06] Brink, E. Steenberg, E. Svenson, G. The Verse Networked 3D Graphics Platform. In SIGRAD '06: Conference proceedings: The Annual SIGRAD conference: Special theme: Computer Games, pages44-48, Skövde, Sweden. 2006.

[SB07] Steenberg, E. Brink, E. The Verse Specification. http://verse.blender.org/, 2007.

[Ste07] Stewart, R. RFC 4960: Stream Control Transmission Protocol, sep 2007.

[WCC+09] Wu, C.C. Chen, K.T. Chen, C.M. Huang, P. and Lei, C.L. On the challenge and design of transport protocols for MMORPGs. Multimedia Tools Appl., 45(1-3):7–32, 2009.

[LG09] Linux Foundation. Networking ToDo List. http://www.linuxfoundation.org/collaborate/workgroups/networking/todo, 2009

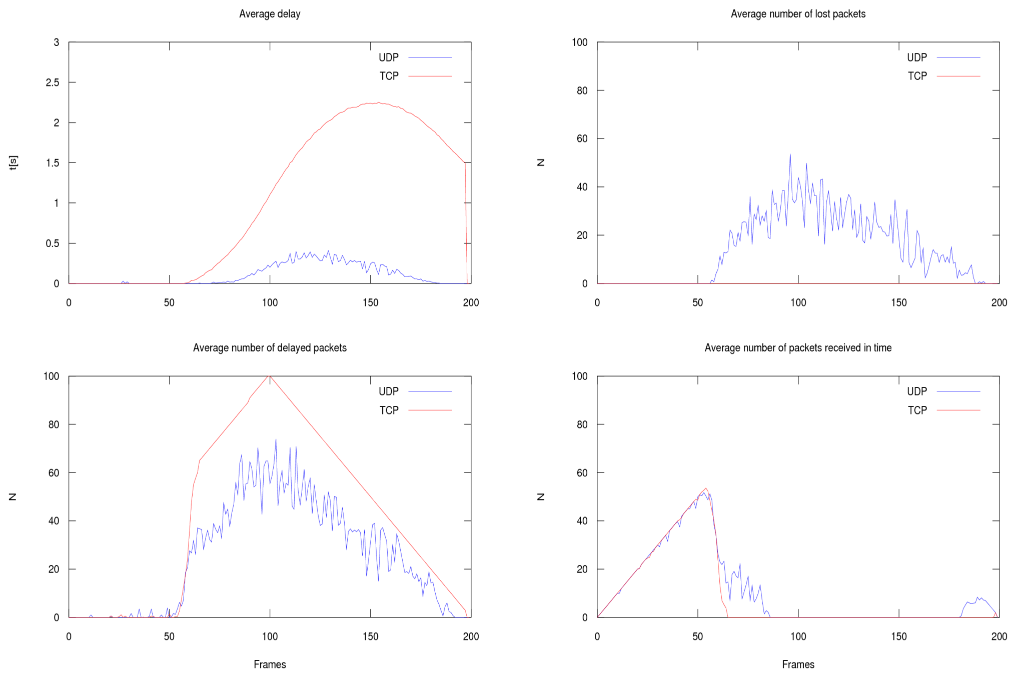


Figure 4: Results of experiments with 1000 particles on real WAN network. The link had delay about 5 ms (delay jitter 1 ms) and the bandwidth was about 1900 kb/s.

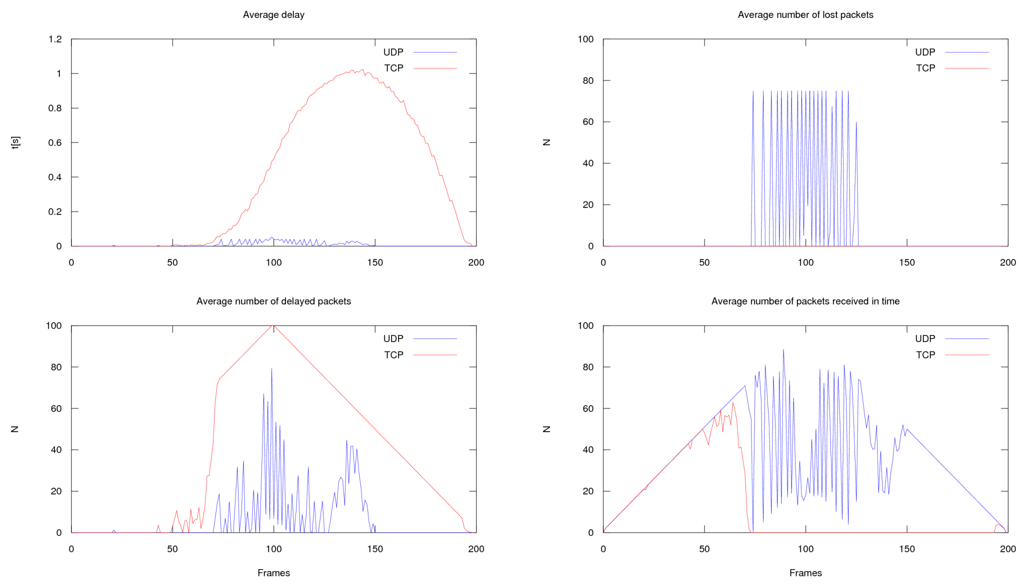


Figure 5*:* Test of UDP protocol proofed, that UDP had average delay quite low. Delay between two frames was 40 milliseconds (~25 FPS). Some of particles lost after 100 th frame has never been resend and this packet loss caused inconsistency of data between client and server.

Test of TCP protocol proofed, that this transport protocol is not feasible for ASVR. When generated traffic exceeded bandwidth, then delay of received particles grooved to 1 second.

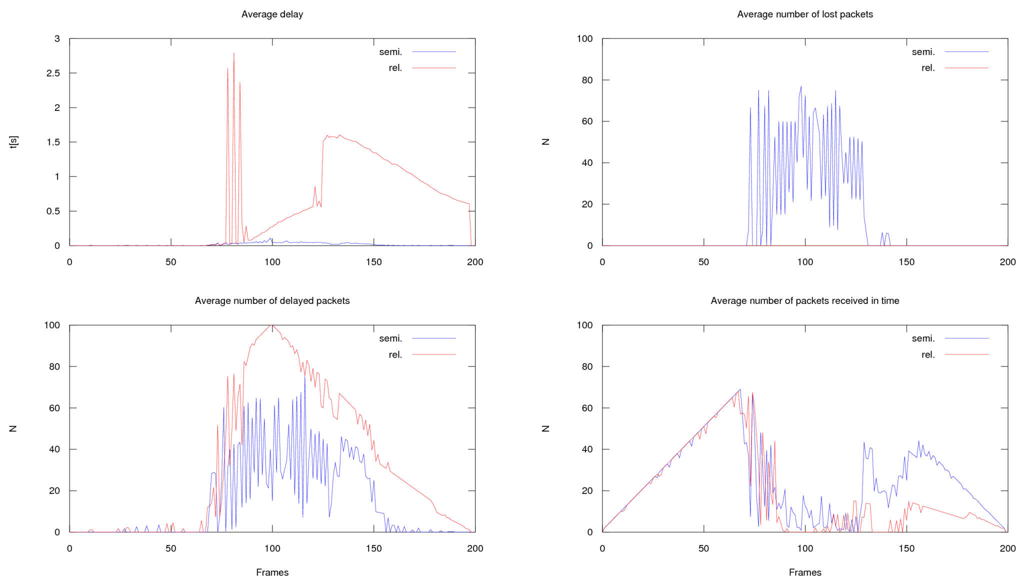


Figure 6: Test of semi-reliable variant of SCTP protocol gave similar results as UDP protocol. Average delay was quite low, but lost packets were not resend and it caused inconsistency in data between server and client, when transmission of particles was finished.

Test of reliable variant of SCTP protocol. This protocol gave similar results as TCP protocol. When generated traffic exceeded bandwidth of the link between client and server, then delay grooved up to 3 seconds.

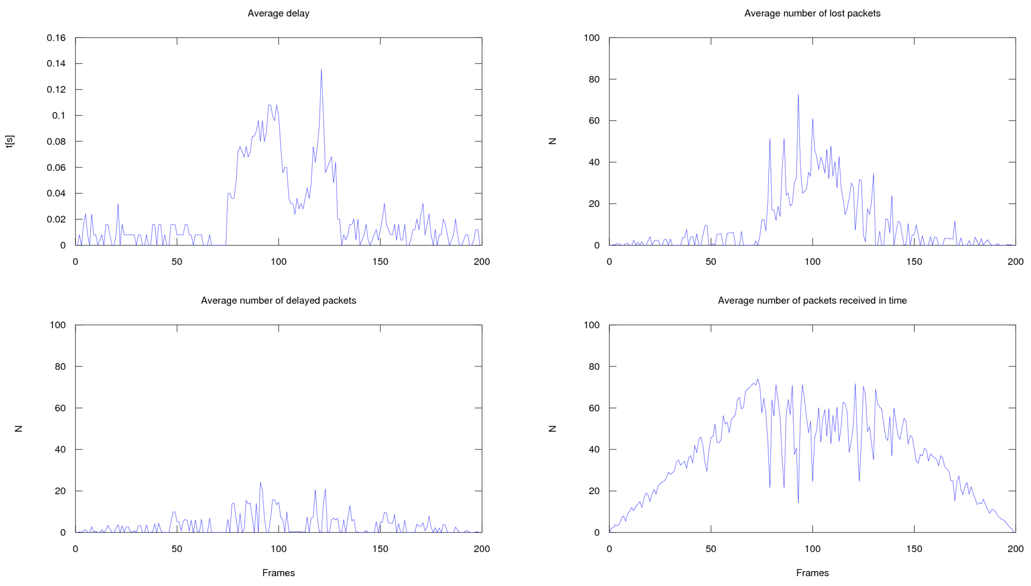
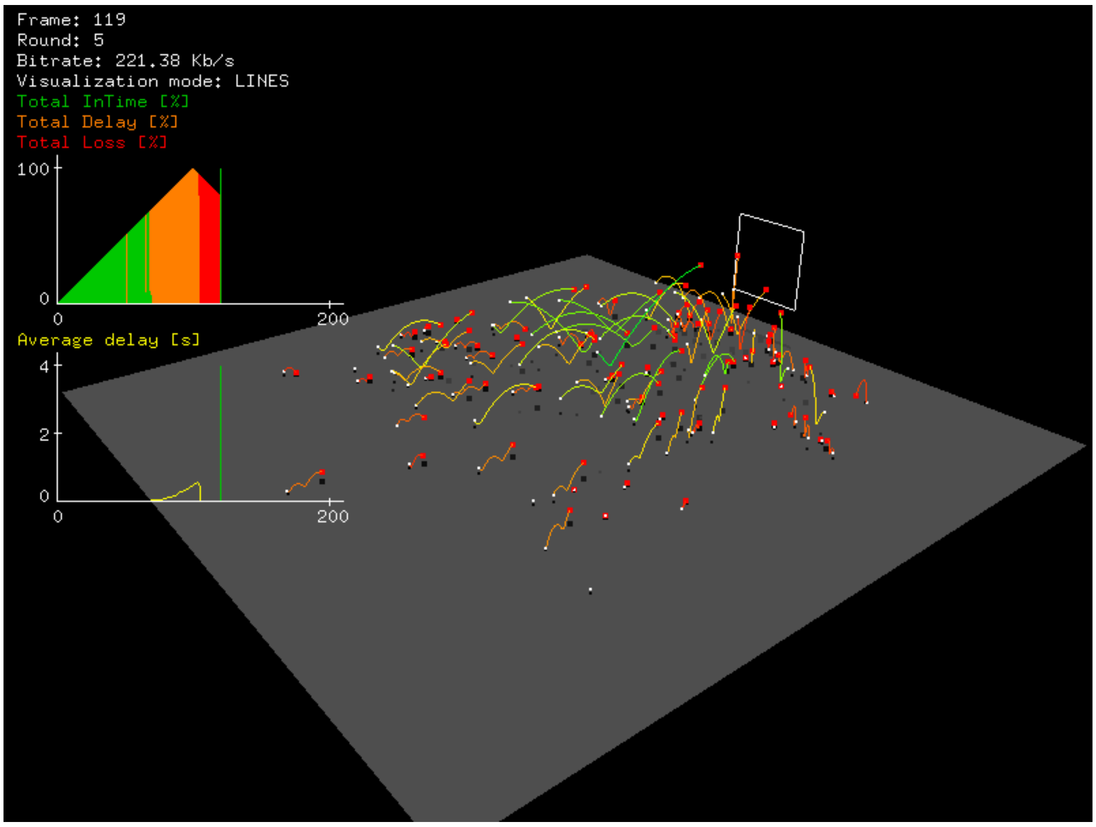


Figure 7: Test of Verse protocol has bigger average delay, then UDP or semi-reliable variant of SCTP protocol, because Verse protocol has to transfer more data (header, acknowledgment commands, node commands, etc.). When transferring of particles was finished, then position of all particles was the same at the client and the server.



Frame: 119
Round: 5
Bitrate: 221.38 Kb/s
Visualization mode: LINES

Figure 8: Screenshot of client visualizing delay of received particles. TCP protocol is used in this case. This screenshot was captured during congestion. Thus all received particles are delayed. Red points visualize current received positions and white points visualize expected positions of particles. Colored line between these two points visualize delay of received particle.