Advances in Engineering Software 157–158 (2021) 103010

Contents lists available at ScienceDirect

Advances in Engineering Software

journal homepage: www.elsevier.com/locate/advengsoft

Research paper

Performance evaluation of massively parallel and high speed connectionless vs. connection oriented communication sessions

Zoltán Gál [a,*], Gergely Kocsis [a], Tibor Tajti [b], Robert Tornai [a]

[a] University of Debrecen Faculty of Informatics Debrecen P.O. Box 400 H-4002 Hungary
[b] Eszterházy Károly University Institute of Mathematics and Informatics Hungary

ARTICLE INFO

Keywords:
High speed networking
High performance computing
Parallel communication
Internet
Congestion control
Traffic engineering
Statistical analysis
Scale independence

ABSTRACT

In this paper we focus on the fast communication issues of the Big Data processing tasks shared between High Performance Computing systems. In our performance evaluation framework we designed and developed two traffic measurement tools in order to answer some theoretical questions related to congestion control in practice. The first one is based on iperf and tcpdump softwares to capture data flows of TCP and UDP sessions. Classification aspects of the measurement cases were: homogeneity of the traffics, number of parallel communication sessions, and implementation types of the TCP congestion control algorithm. Dozens of parallel traffic scenarios were executed in a dumbbell topology to evaluate effects of the massively parallel communication sessions in wireline local and metropolitan area networks. Since we found that connection oriented data transfer sessions have limited performance features during communication, we implemented a second communication tool named Fast Manager of File Transfer (FMFT). This application with transfer rate monitoring and regulation capability is based on parallel connectionless data transfer sessions supervised by a common connection oriented control session and provides better transfer rate than the classical file transfer mechanisms using TCP services. Methodology of the statistical analysis and highlights of this heterogeneous parallel communication mechanism are explained, too.

1. Introduction

Main issues of the Big Data (BD) processing include not just computation or storing huge amount of data in the High Performance Computing (HPC) machines but high speed transmission of these data between different nodes of the infocommunication systems as well. Best effort based datagram delivery of the protocol data unit streams provides usable time critical services just in networks having minimal bandwidth in the scale of $n * 10$ Mb/s. Although IntServ and DiffServ QoS mechanisms make possible for time critical data flows to be forwarded under reasonable conditions, high speed transmission of the big data in LAN/WAN environments remains hot topic. Different implementations of the TCP congestion control mechanism with various efficiency of the transmission speed were developed by research institutes, standardization institutes, and ICT companies in the last decades. The requirements for proper congestion control modules are defined by RFCs, e.g. [8,9]. The QoS strategies applied in LAN environment are weakly usable in wide area data networks producing low usage efficiency of the communication path traversing autonomous systems belonging to different ISPs. Comparison and analysis of the high speed communication mechanisms existing today makes possible to set configuration for best transmission performance.

Big Data processing requires high-speed networks and high computing power too [1]. Very large data sets can be processed in shared memories. Hierarchical structure of the memory types involves intensive communication among these modules. Grid or cluster based high performance computation systems use IP based communication to offer common virtual memory system for the BD applications. Network type file systems offer access to the data with reliable transfer capability. When the client-server system is connected by IP technology, the transfer of raw data requires significant time sometimes even acting as the bottleneck of efficient BD processing [2]. Fast packet switching transmission mechanisms are necessary for BD operations, making hard the decision of server operators to manage data movement services optimally. Fast delivery of the BD contents makes time sensitive applications degraded by the lack of bandwidth on the communication path.

* Corresponding author.
E-mail addresses: zgal@unideb.hu (Z. Gál), kocsis.gergely@inf.unideb.hu (G. Kocsis), tajti.tibor@uni-eszterhazy.hu (T. Tajti), tornai.robert@inf.unideb.hu (R. Tornai).

https://doi.org/10.1016/j.advengsoft.2021.103010
Received 15 November 2019; Received in revised form 22 February 2021; Accepted 15 April 2021
Available online 16 May 2021

For our performance evaluation framework we engineered and developed an own measurement tool based on *iperf* and *tcpdump* software to acquire statistics about data flows of TCP and UDP sessions. Classification aspects of the measurement cases were: homogeneity of the traffics (TCP only, UDP only, heterogeneous TCP and UDP), number of parallel communication sessions (1…100) and type of the TCP congestion control algorithm (16 different implementations). More than four hundred traffic scenarios were executed in a dumbbell topology containing routers and Linux machines. Statistical analysis methods were used to evaluate effects of the aspects mentioned above in the wireline local and metropolitan area networks. All the research results of this framework serve as design and development consideration of a high speed reliable file transfer application based on UDP based sessions managed by a common TCP control session.

The structure of the paper is the following: In chapter two we give a short overview of the BD communication issues and possible solutions given in the literature. Chapter three contains characteristics of the transport layer services based on packet switching including features of the connection oriented and connectionless mechanisms, respectively. In chapter four measurement scenario and performance analysis of high number of parallel traffics controlled by sixteen different congestion control mechanisms is presented. In chapter five we described details of a new fast file transfer application (FMFT: Fast Manager of File TransferM M) together with its communication performance characteristics. At the end we conclude and we give possible continuation of this work in chapter five. Because of high number of measurements, a set of plots is attached in annex.

Current work is partially published before in the proceedings of PARENG 2019 conference [3]. However, at several points we have extended the current paper with new aspects. These observations are concerning the second traffic analysis tool being a new server - client software pair and the corresponding GUI we have developed. Section 5 in this paper is newly introduced and gives details of the fast file transfer application (FMFT). Characteristics of high speed connectionless data traffic parallel sessions managed through a common connection oriented control session is also presented here.

2. Related works of the parallel vs. serial communication

Two methods exist to transmit data between digital devices: serial and parallel transmissions. Serial data transmission sends data bit-by-bit through a communication channel. Parallel data transmission sends multiple data bits in the same time through different channels. Data bits of the protocol data unit are sent in a given order rule applied by both the sender and receiver nodes. In the process of serial transmission, a given bit can be sent just after the previous bit was forwarded on the channel. This rule should be maintained in both timing variants of this communication: synchronous or asynchronous.

Most of the digital computer network technologies are based on serial transmission mechanisms because in practice long-distance data delivery tasks should be executed by the peer entities. The protocol data unit (PDU) has special record structure with frame check sequence field at the tail part to detect any bit error during the transmission and correct it if the error affects just a single bit. The number of error combinations is a $O(L^k)$, where L is the size of PDU in bits and k is the number of bits affected by error. In practice no correction just error detection is applied in case of multiple bit errors in the PDU. Fortunately, majority of channel errors are single bit based, making possible to detect and correct such frames.

Parallel transmission sends multiple bits simultaneously. In practice digital channels are basebands providing no parallel forwarding possibilities on the physical channel. Because the low efficiency of the upper logical network layer mechanisms, the physical channel may remain underutilized for variable period of time. Starting with network layer toward the application layer communication mechanisms use queue pairs to manage PDU transmission between the sender and receiver protocol entity. In practice such queue pairs exist in parallel because several simultaneous sessions are running on the network nodes. Parallel transmission can transfer data quicker than serial sessions on the same logical level but requires more than one parallel channels. Having more than one channel, the sent data segments may arrive out of sending order to the destination. To solve reordering of such segments, extra task is needed both sides.

Finding algorithms with low computation processing level for PDU management remains an open question in the network and transmission logical levels for high speed communication services. Big Data processing requires high volume of data to be delivered between client and high performance computation systems. This implies usage of applications providing fast file transmission compatible with the existing IP based stacks. Usage of multiple physical interfaces makes possible to enhance the file transmission rate with multipath TCP (MPTCP) [4]. The drawback of this method is that if there is only one interface available, the bandwidth is lower compared to the case without using MPTCP. In practice the majority of the client nodes with multicore processors have only one high speed LAN interface card, including usage of fast file transfer applications based on MPTCP [5]. These applications should take into consideration best effort property of the IP layer services. Because UDP services are datagram based, unbalanced usage of the IP layer services on the client node level can be provided for UDP in detriment of TCP services. Of course care should be taken regarding maximum usage of the physical channel by the multiple UDP sessions in order not to starve all TCP sessions running on the common communication path toward the server node.

3. Characteristics of the transport layer communication services

Transport layer protocols have big impact on the packet switched based communication services. Data transmission rate depends on three critical factors: the transmission speed of the links, the end-to-end latency, and the protocol efficiency. Depending on the complexity and intelligence of the connection type reliable or unreliable data delivery solutions are provided. In case of Internet both transport service types are using unreliable classical IP networks layer mechanism, delivery guarantees imply extra communication in the control plane making longer delivery time of the application data units in case of connection oriented services. However, low latency data unit forwarding can be provided with reduced reliability, time sensitive applications like voice and video prefer to use connectionless network services.

3.1. Overview of the connection oriented services

Most of the applications on the Internet use Transmission Control Protocol (TCP) transport layer mechanism for data exchange. TCP is a mature and efficient protocol which ensures that the sent data arrives to the receiver. TCP is adaptive speed control protocol and has few tuneable attributes. One of these parameters is the congestion control algorithm type which can be chosen for the connection. These congestion control algorithms regulate the sending of data packets in order to avoid inefficiency caused by congestion. On the core links of the communication path huge amount of consecutive flows exist each having influence on the other flows to provide fair usage of the total bandwidth. Several such algorithms have been developed, so we can choose one of them which gives the best result in our environment. Since congestion control is part of the transfer control in TCP over IP, changing the congestion control means that we will use another TCP variant. Since understanding and the usage of the theoretical results is an essential property of engineering, below we describe essential characteristics of various congestion control mechanisms of the TCP.

BBR - Bottleneck Bandwidth and RTT congestion control algorithm

This algorithm has been developed at Google and deployed to be used by YouTube. It is a model based algorithm which is measuring bottleneck bandwidth and round-trip propagation time. When it starts

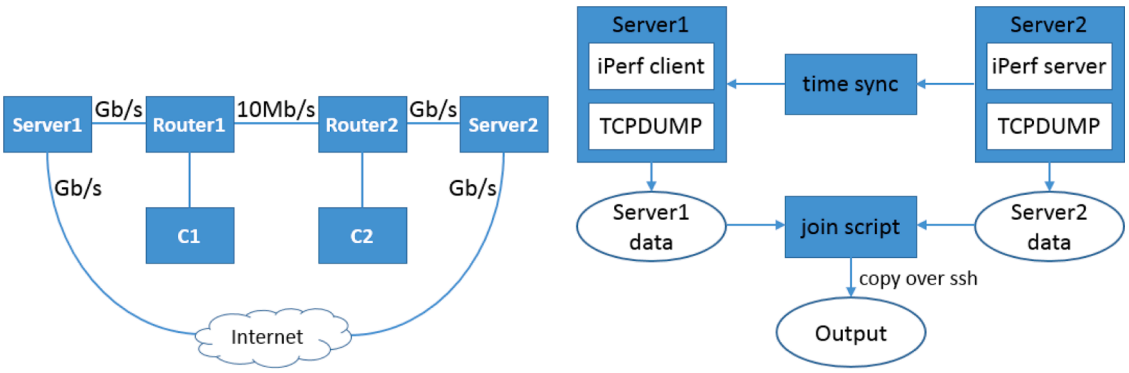

Fig. 1. Architecture elements of Tool 1. *(Left):* Architecture elements of the measurement tool. *(Right):* Dumbbell topology of the measurement.

sending data stream, BBR first tries to raise the transfer rate. After observing from acknowledgement (ACK) loss that the pipe is full, it drains the queue and changes the state to probe bandwidth. Monitoring the round trip time (RTT) during the transmission is key element of the algorithm. This is the main state of BBR, spending most of the time in it. Since the network bottleneck bandwidth and the round-trip propagation time can vary dynamically through time, the algorithm must recalculate them periodically. So from the ACKs it recalculates the two-way RTT of the path (RTProp). If it is not updated, then BBR goes to probe RTT state. In that state it sets the TCP window to a low size to check whether a lower RTT is possible. Disadvantage of BBR is the missing of scalability feature and slaughtering concurrent loss-based flows in an ecosystem of TCP flows [6,7].

BIC - Binary Increase Congestion control algorithm

BIC has been optimized for fast, long distance networks [10]. According to the paper written by the developers of BIC, it has focused on the following features: scalability, RTT fairness, TCP friendliness, fairness and convergence. The BIC algorithm is based on the idea to determine the TCP window size with a search algorithm. It uses binary search increase to fast approximation of the optimal TCP window size and uses additive increase for fine convergence. These two increase strategies are combined with multiplicative decrease when the RTT increases. Linux up to kernel version 2.6.18 uses BIC by default [10].

CDG - CAIA Delay-Gradient congestion control algorithm

The Linux module implementation is based on the paper [11]. The Linux implementation does not make major changes, however there are several smaller differences, improvements. CDG works based on the delay gradient. The delay gradient is used as a congestion indicator and the flow control is based on loss of segments. It is sensitive and reduces the amount of sent segments when packets are lost due to congestion but it tolerates packets lost due to non-congestion network events. The algorithm uses the moving average of the gradients of minimum and maximum RTT values within a specified window for smoothing.

CUBIC - TCP CUBIC: Binary Increase Congestion control algorithm

Linux kernel 2.6.19 and later uses CUBIC by default [12]. This congestion control algorithm changes the function for increasing the TCP window size to be more aggressive, so it can reach more effective window size faster compared to earlier versions (e.g. TCP-Reno). On high-speed networks it can cause big effect, since without this aggressive increase a TCP stream might not reach or even get close to the optimal window size before it finishes. The idea is similar to BIC-TCP, however the aggressive increase in case of BIC-TCP can be ineffective for low speed networks or when RTT is short. CUBIC utilizes multiplicative decrease and fast convergence in increase using heuristics which can adapt to cases e.g. when a new parallel stream starts on the same network.

DCTCP - DataCenter TCP (DCTCP) congestion control algorithm

DCTCP is an enhancement of Reno algorithm, it is designed for data

Table 1
Sampling data set of the transport layer traffic.

Sending time stamp [s]	Receiving time stamp [s]	Port ID	Maximum Transfer Unit [B]	Packet ID
1545154926.170711	1545154926.173984	50286	1500	0000
1545154926.170714	1545154926.174111	50286	1500	02d4
1545154926.170717	1545154926.174235	50286	1500	05a8
1545154926.170724	-1	50286	1500	087c
1545154926.170727	1545154926.174485	50286	1500	0b50
1545154926.170730	-1	50286	1500	0e24

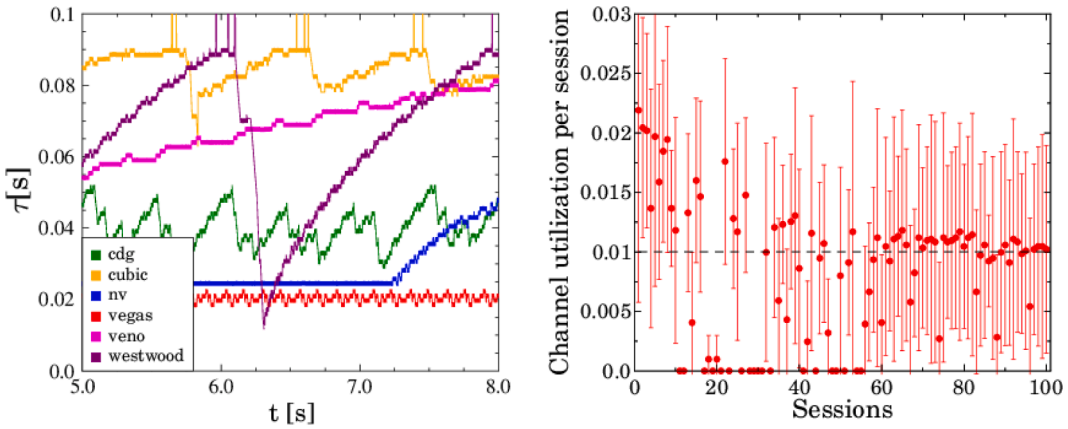

Fig. 2. (left) The time amount needed to deliver frames versus time based on the aggregated time series of two competing sessions in case of different TCP versions. (right) Contribution of each segment to the total channel utilization in case of 100 sessions of cubic TCP. Red dots are representing the averages over measurement, while error bars illustrate quite large fluctuation of values using 1s timescale. The dashed line indicates the theoretical unbiased value.

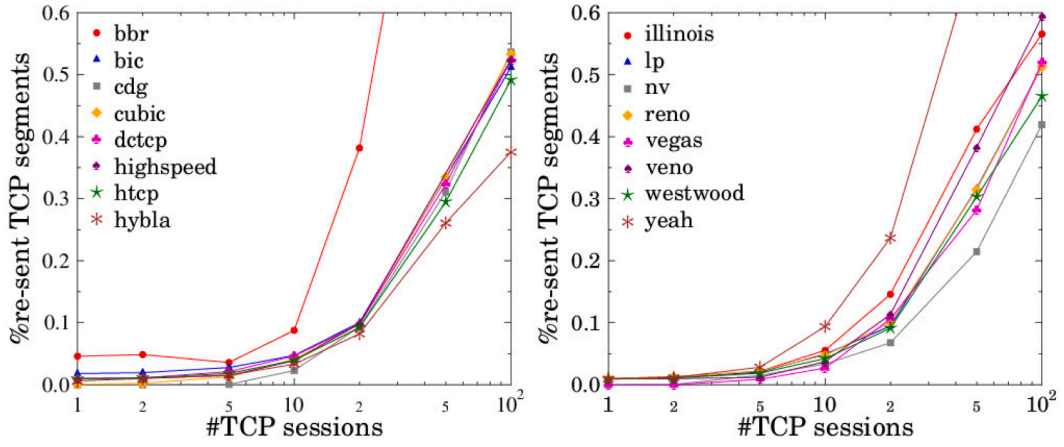

Fig. 3. The ratio of the re-sent TCP segments relative to the amount of delivered segments as a function of the number of TCP sessions. All the curves belong to different TCP versions.

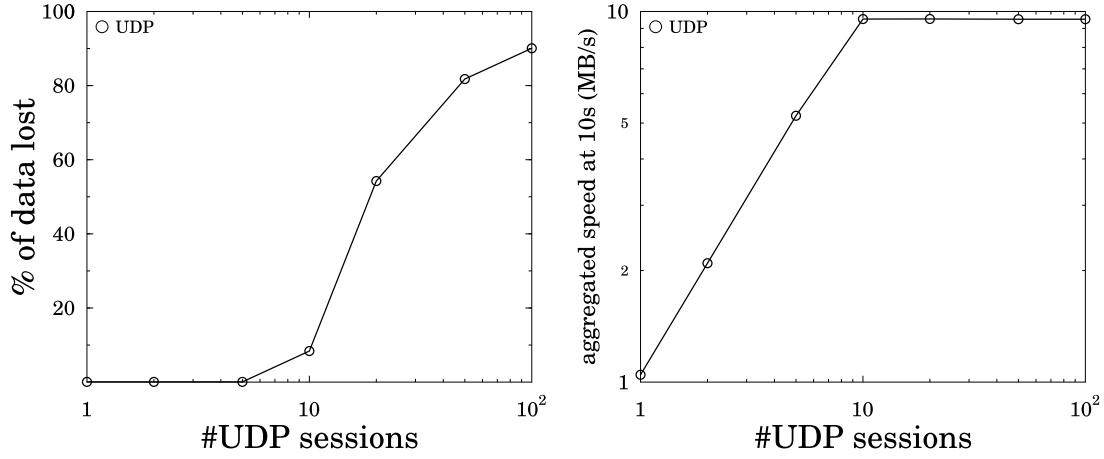

Fig. 4. (left) Data loss in percentages as a function of the number of sessions. (right) Aggregated speed of the sessions as a function of the session number. Our findings show that in contrary to homogeneous TCP session sets homogeneous UPD sessions do not utilize the maximal possible bandwidth.

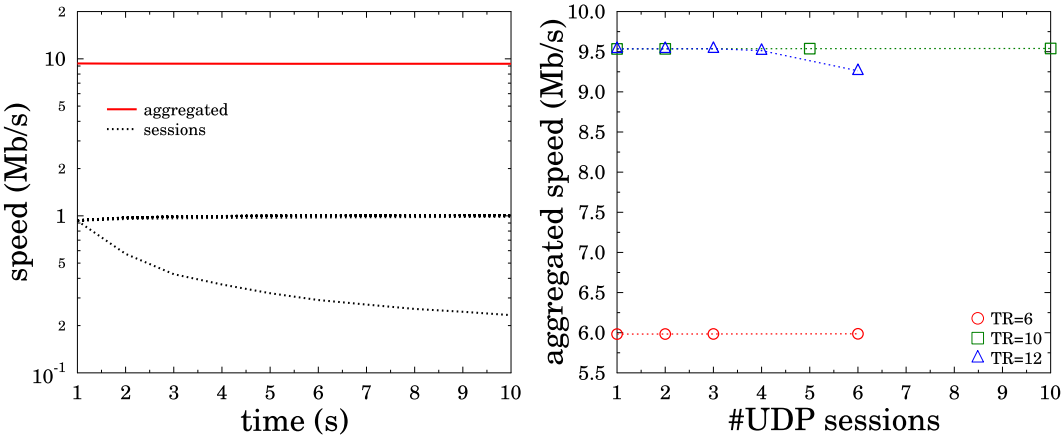

Fig. 5. (left) Separate and aggregated speed of 10 UDP sessions as a function of time. (right) The aggregated speed at 10 s in our measurements as functions of the session number. Values for the same markers are for the same aggregated transmit speed.

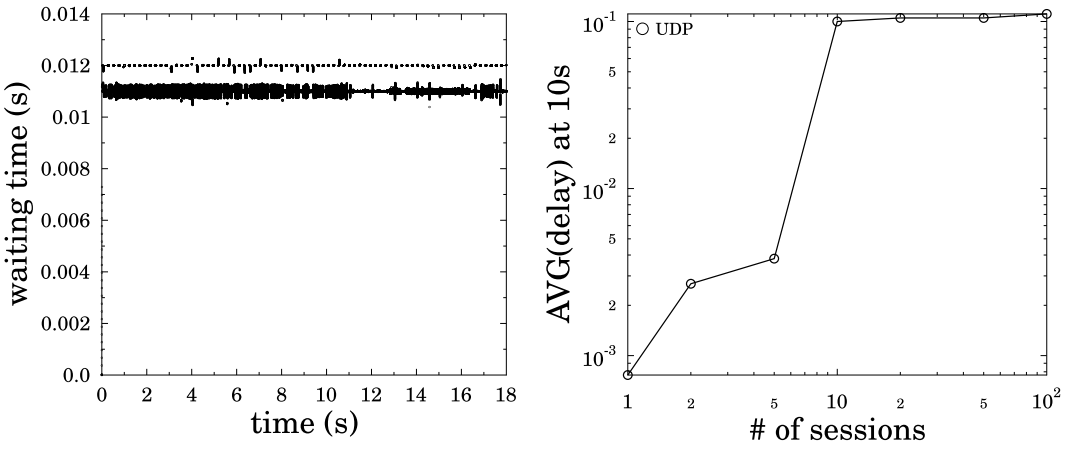

Fig. 6. (left) Waiting times of packages for 50 sessions. (right) The average delay of sessions as a function of session numbers. While below 10 sessions the delay increases by the number of sessions, over 10 it stays constant.

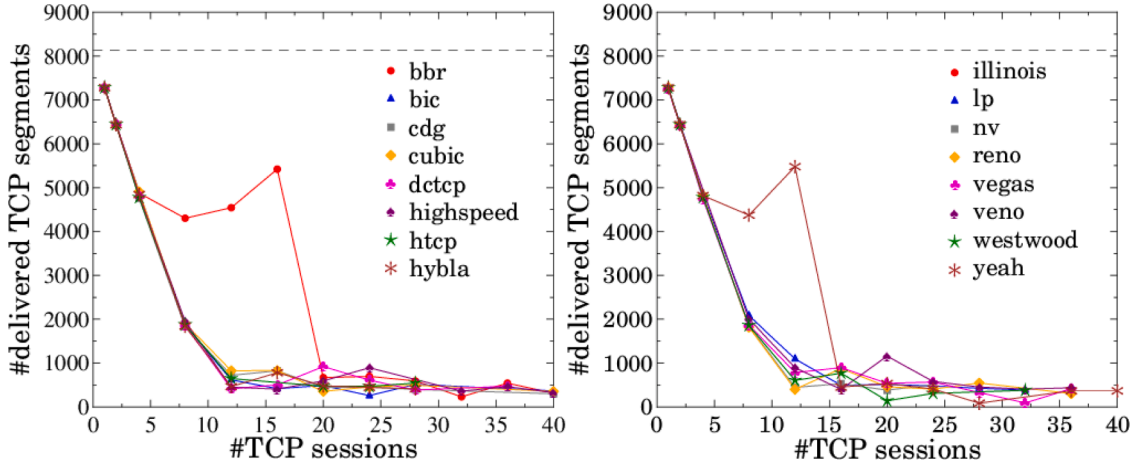

Fig. 7. The number of delivered TCP segments as a function of the number of TCP/UDP sessions. Dashed line indicate the maximum number of delivered TCP segments within 10 seconds. For several parallel sessions, the proportion of TCP traffic is approximately constant.

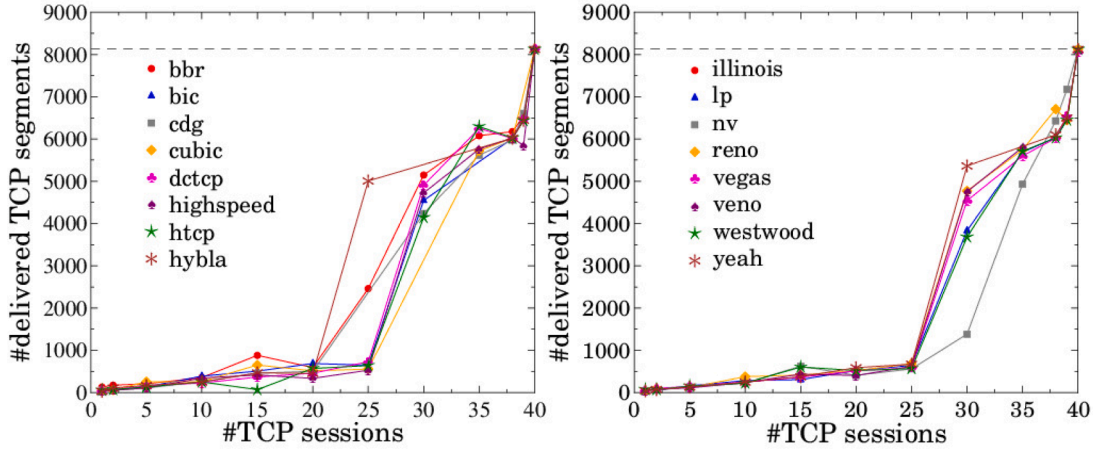

Fig. 8. The number of delivered TCP segments for 40 TCP+UDP sessions (their ratio is changing along the horizontal axis). The interaction of TCP and UDP sessions results in non-trivial behaviour.

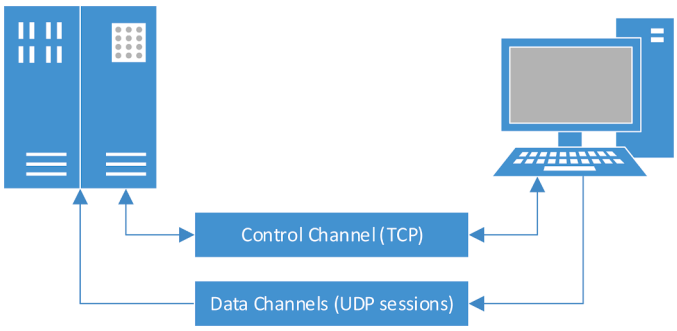

Fig. 9. Architecture view of the server - client application (Tool 2). Data content forwarding is provided by a number of concurrent connectionless sessions (UDP) and control service of the file content delivery is supervised by a single connection oriented (TCP) session.

centers with the goals to achieve high burst tolerance, low latency, and high throughput [13,14]. DCTCP is more than a congestion control algorithm, since it introduces protocol modifications to TCP. Marking of the packages based on the RED marking feature of modern switches is used to provide active queue management. The way to send the ACK packets has been changed to send back Explicit Congestion Notification (ECN) with flagging Congestion Experienced (CE) codepoints which ensures quick notification about queue overloads. This mechanism has high burst tolerance, low latency and high throughput with slightly loaded buffers of the backend switches.

High Speed TCP congestion control algorithm

Sally Floyd's High Speed TCP congestion control algorithm (RFC 3649) is based on increasing/decreasing the TCP window size to offer more realistic packet drop rates [15,16]. It modifies additive-increase and multiplicative-decrease parameters in function of current window size to faster reach of high speed in TCP slow-start and to quickly recover from occasional slow-down periods. For compatibility with earlier TCPs it uses their parameter values when the packet loss is higher than a specified limit.

Table 2
Parameter ranges of the sliders of the GUI.

Parameter	Minimum value	Maximum value
Port number	1024	65535
Chunk size	1	65535
Transfer rate	1	1000
Start chunk size	1	65535
Chunk size step	1	10000
End chunk size	1	65535

HTCP - H-TCP congestion control algorithm

H-TCP is a congestion control protocol for high-speed and long distance networks [17]. It uses bandwidth estimation and changes the TCP window increase/decrease parameters according to the estimated minimal and maximal bandwidth. It takes into consideration that the parameter for additive increase should be small in slower networks and should be large in high-speed and long distance networks in order to achieve fast adaptation to the available bandwidth.

Hybla - TCP-HYBLA congestion control algorithm

This algorithm aims to achieve higher speeds compared to other TCP versions on heterogeneous networks which encounter higher error rates and longer round-trip propagation times., e.g. on satellite link [18]. It uses modified rules for the congestion window increase based on an analytical study of the congestion window dynamics. Additionally, this algorithm uses the SACK option and timestamps.

Illinois - TCP congestion control algorithm

The TCP-Illinois is a congestion control for high-speed networks. It uses packet loss information to determine the window size increase/decrease [19]. The more standard AIMD (additive increase/multiplicative decrease) window change function is not optimal for high-speed networks, since it takes long after start or when recovering from a network hang/slowdown. TCP-Illinois maintains fairness and stability as

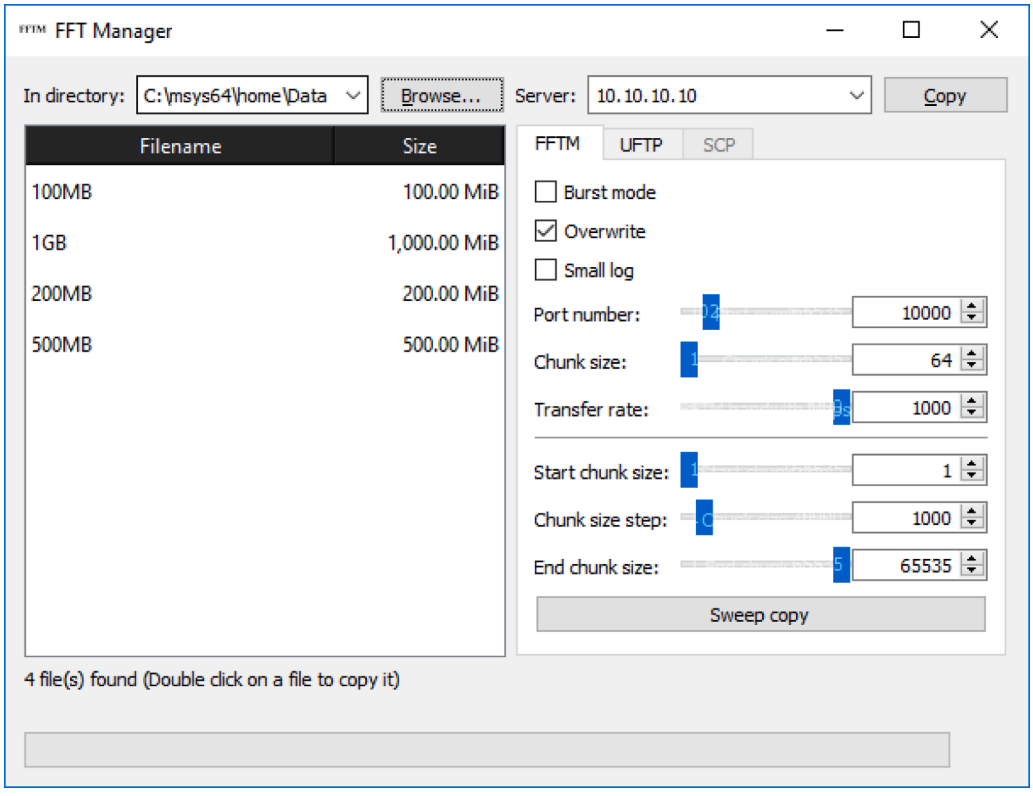

Fig. 10. The FMFT Windows GUI application

Table 3
Size of files transmitted and sampled with FMFT tool.

Host	File$_1$ (MB)	File$_2$ (MB)	Segments, m
Client$_1$	102.431	444.951	1... 19
Client$_2$	198.442	323.299	1... 33

well as router independence. It manages the factor for the additive increase (α) so that the factor is larger when far from congestion and smaller when close to congestion, while the factor for the multiplicative decrease (β) is smaller when far from congestion and larger when close to congestion. It keeps the features of TCP-NewReno except the AIMD algorithm.

LP - TCP Low Priority (TCP-LP) congestion control algorithm

While most TCP congestion control algorithms aim to reach highest possible bandwidth while trying to keep fairness, the main goal of TCP Low Priority is to take fairness as priority and utilize the network only to a fair share so minimally disturb other applications using the same network connection [20]. TCP-LP uses AIMD with an addition of an inference phase and use of a modified back-off policy.

NV TCP New Vegas congestion control algorithm

TCP-NV (New Vegas) is a successor of TCP-Vegas optimized for use in data center. It detects congestion before packet losses occur. It allows high-speed and should be used only when all connections are using TCP-NV congestion control.

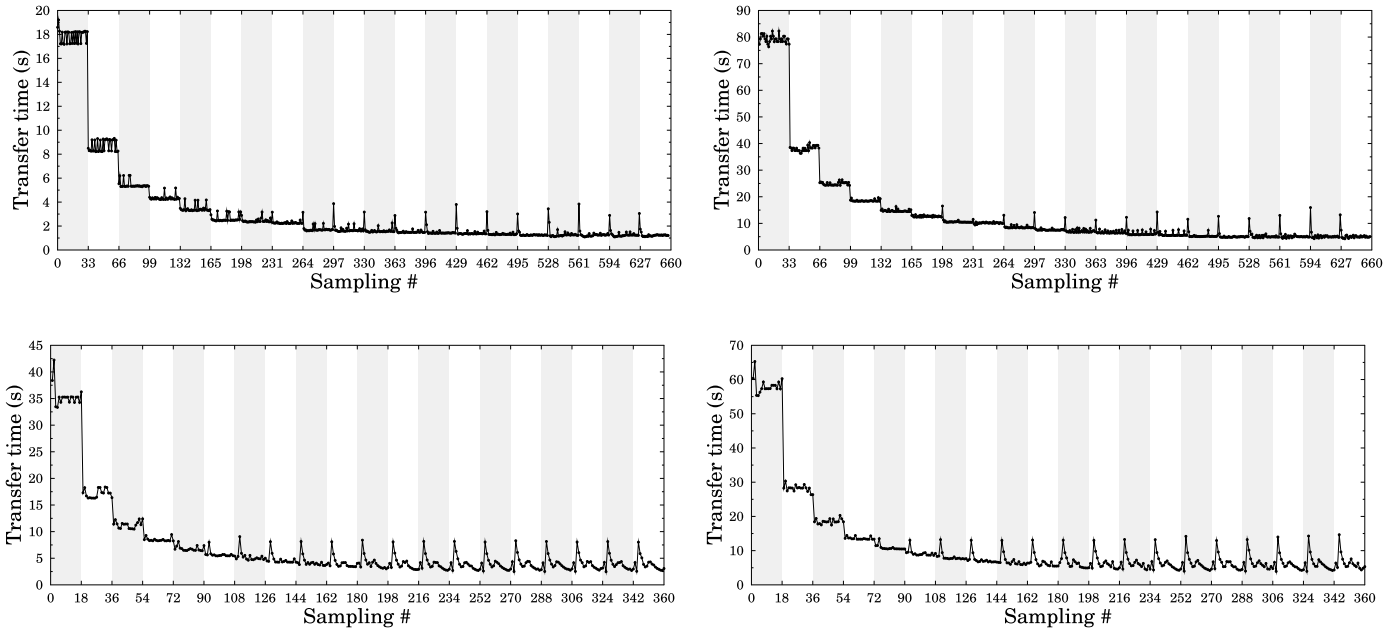

Fig. 11. Transfer time of files transmitted and sampled with FMFT tool. The 20 stripes of the background correspond to different values of bandwidth parameter k. Number of measurements for a given value of k is m. *(Up-left)*: Client$_1$, file$_1$, *(Up-right)*: Client$_1$, file$_2$, *(Down-left)*: Client$_2$, file$_1$, *(Down-right)*: Client$_2$, file$_2$.

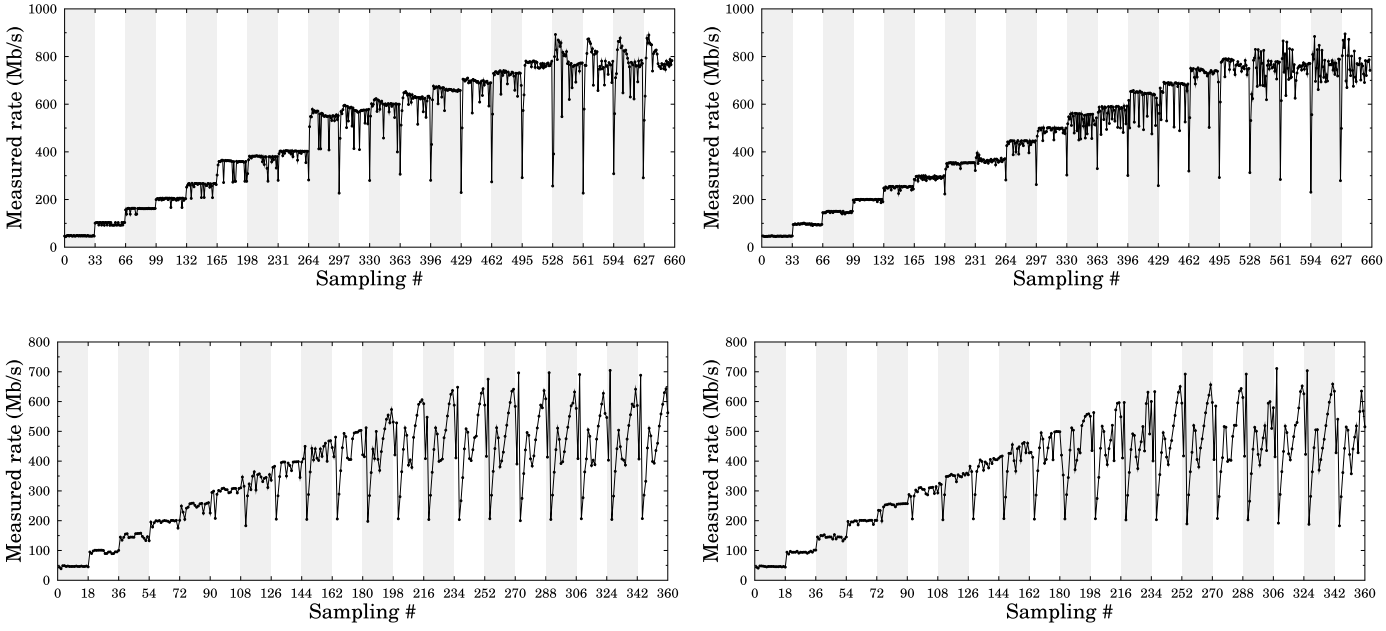

Fig. 12. Measured data transfer rate. The 20 stripes of the background corespond to different values of bandwidth parameter k. Number of measurements for a given value of k is m. *Up-left*: Client$_1$, file$_1$; *Up-right*: Client$_1$, file$_2$; *Down-left*: Client$_2$, file$_1$; *Down-right*: Client$_2$, file$_2$.

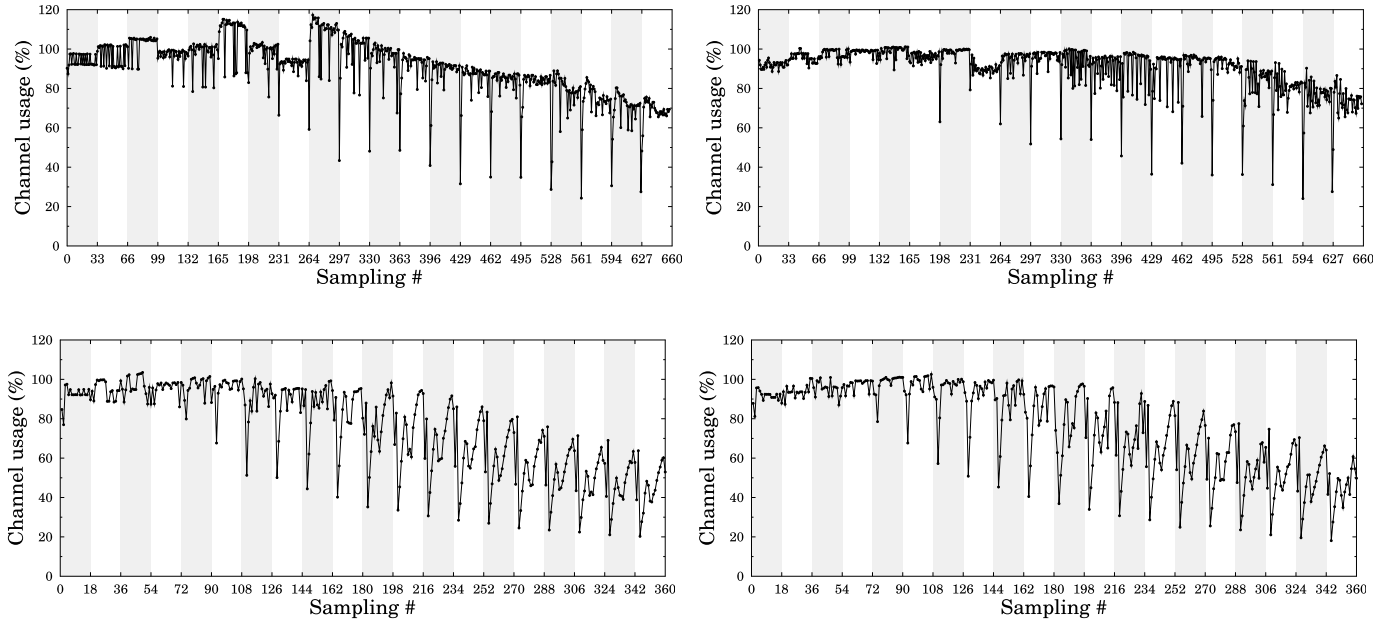

Fig. 13. Measured channel load. The 20 stripes of the background correspond to different values of bandwidth parameter k. Number of measurements for a given value of k is m. *Up-left*: Client$_1$, file$_1$; *Up-right*: Client$_1$, file$_2$; *Down-left*: Client$_2$, file$_1$; *Down-right*: Client$_2$, file$_2$.

Reno TCP (New) Reno congestion control algorithm

TCP Reno is ancient TPC congestion control algorithm which was obsoleted by TCP New Reno which still lives in the Linux kernel. Reno performs well when packet losses are rare was only able to detect single packet losses. New Reno can also detect multiple packet losses.

Scalable - Scalable TCP congestion control algorithm

It is a simple change to the traditional TCP congestion control algorithm. Scalable TCP makes a simple change to the congestion window change function aiming faster reach of the available bandwidth on a high-speed network.

Vegas - TCP Vegas congestion control algorithm

TCP Vegas is a modification of TCP Reno congestion control. It can detect packet losses but also it can detect congestion before packet losses occur, so it tries to be proactive instead of reactive. It has a modified slow-start, in that it increases TCP window size exponentially after every RTT.

Veno - TCP Veno congestion control algorithm

TCP Veno is a modification of TCP Reno optimized for wireless networks [21]. It changes the multiplicative decrease of TCP Reno to adaptively adjust to the perceived network congestion level. The Veno

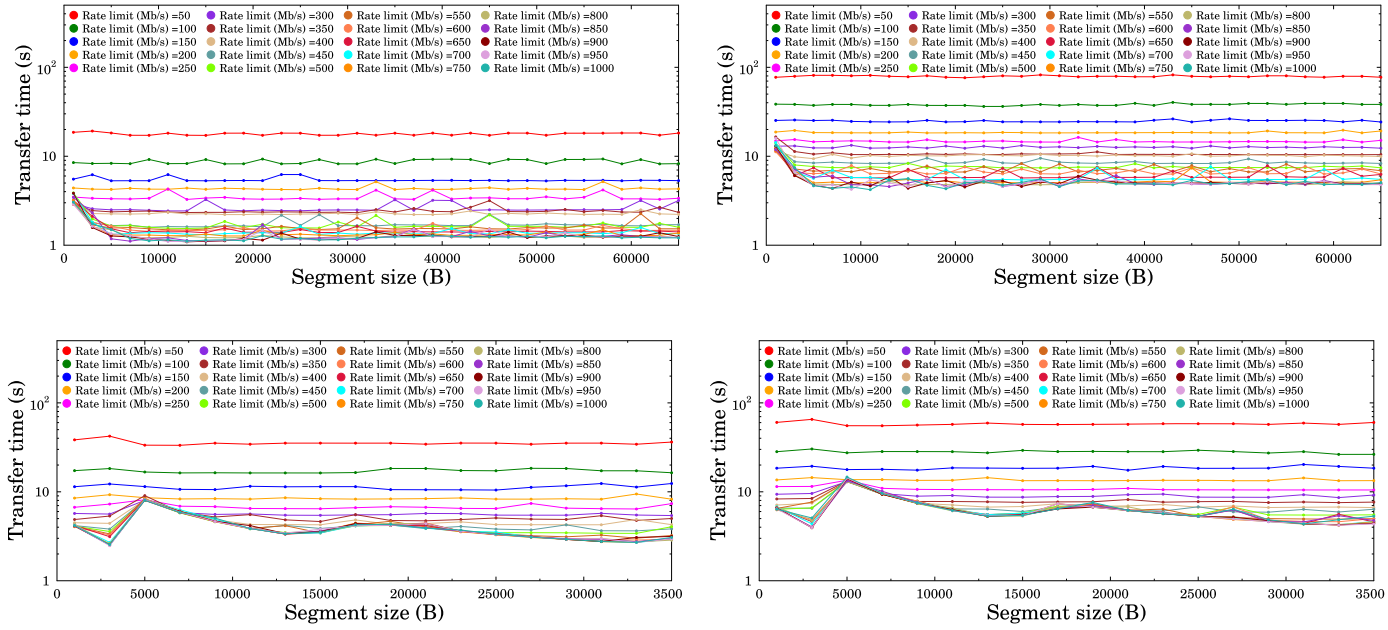

Fig. 14. Comparison of file transfer times. *Up-left*: Client$_1$, file$_1$; *Up-right*: Client$_1$, file$_2$; *Down-left*: Client$_2$, file$_1$; *Down-right*: Client$_2$, file$_2$.

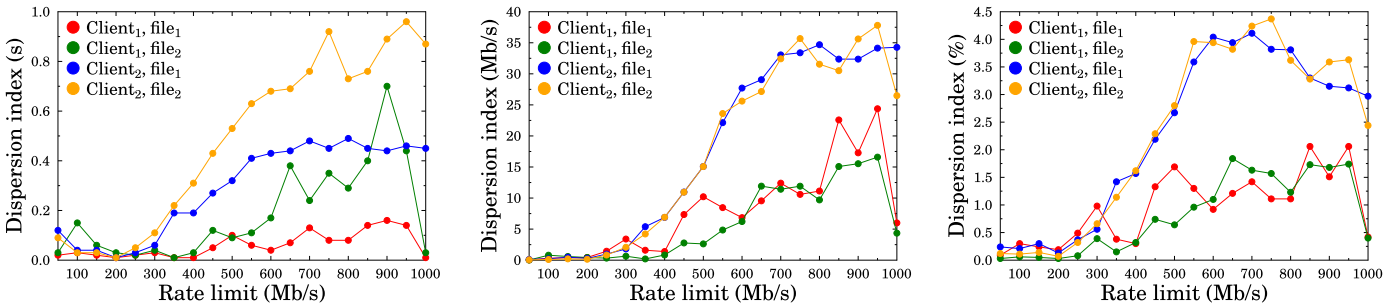

Fig. 15. Dependence of the dispersion indexes on the preset bandwidth *(Left*: Dispersion index of measured transfer time; *Middle*: Dispersion index of measured transfer rate; *Right*: Dispersion index of measured channel load.

algorithm estimates the connection state and in case of packet loss it classifies the case whether it happens because of network congestion or random packet loss.

Westwood - TCP Westwood+ congestion control algorithm

The TCP Westwood congestion control algorithm is a sender side modification of the TCP protocol [22]. It is based on end-to-end bandwidth estimation with distinguishing the cause of packet loss if it occurs. Especially effective can be its fast recovery mechanism in case of mixed wired and wireless network environment.

Yeah - YeAH-TCP: Yet Another Highspeed TCP congestion control algorithm

This congestion control algorithm belongs to the ones which give solution for high-speed networks where the earlier TCP variants have not adapted quickly to the high available bandwidth [23]. YeAH-TCP does it very efficiently. It can use the high bandwidth but still maintain friendliness. It is friendly at least as Reno and it is Reno-friendly.

3.2. *Overview of the connectionless services*

In contrary to connection oriented ones, in the case of connectionless

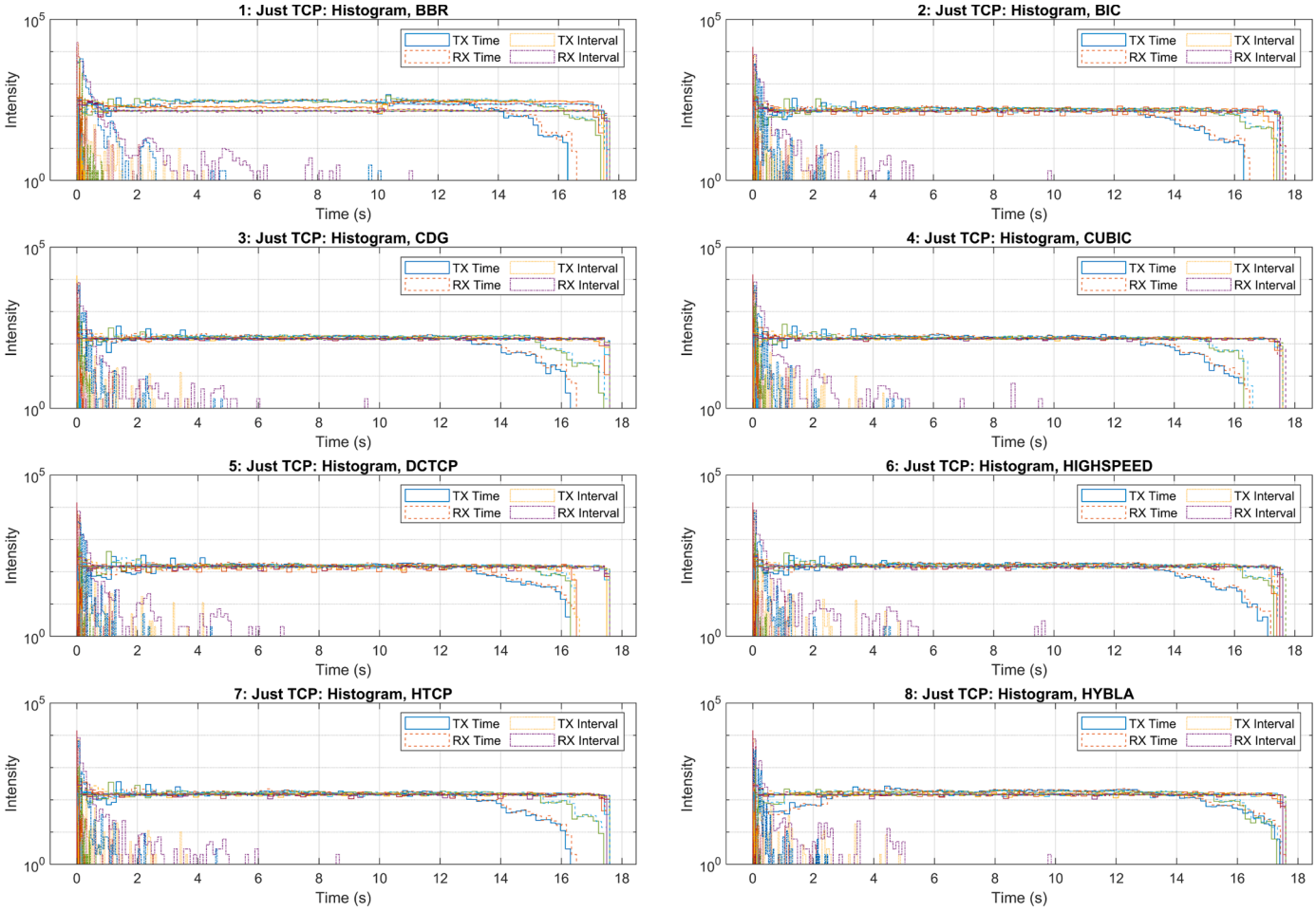

Fig. A.1. Histogram of transmit (Tx Time), receive (Rx Time), transmit interval (Tx interval) and receive interval (Rx Interval) (Only TCP parallel sessions - TCP mechanisms: BBR, BIC, CDG, CUBIC, DCTCP, HIGHSPEED, HTCP, HYBLA). Transmit and receive times of the TCP segments are roughly uniform distributed for each congestion control mechanism. Transmit and receive time intervals of the TCP segments have exponential distribution.

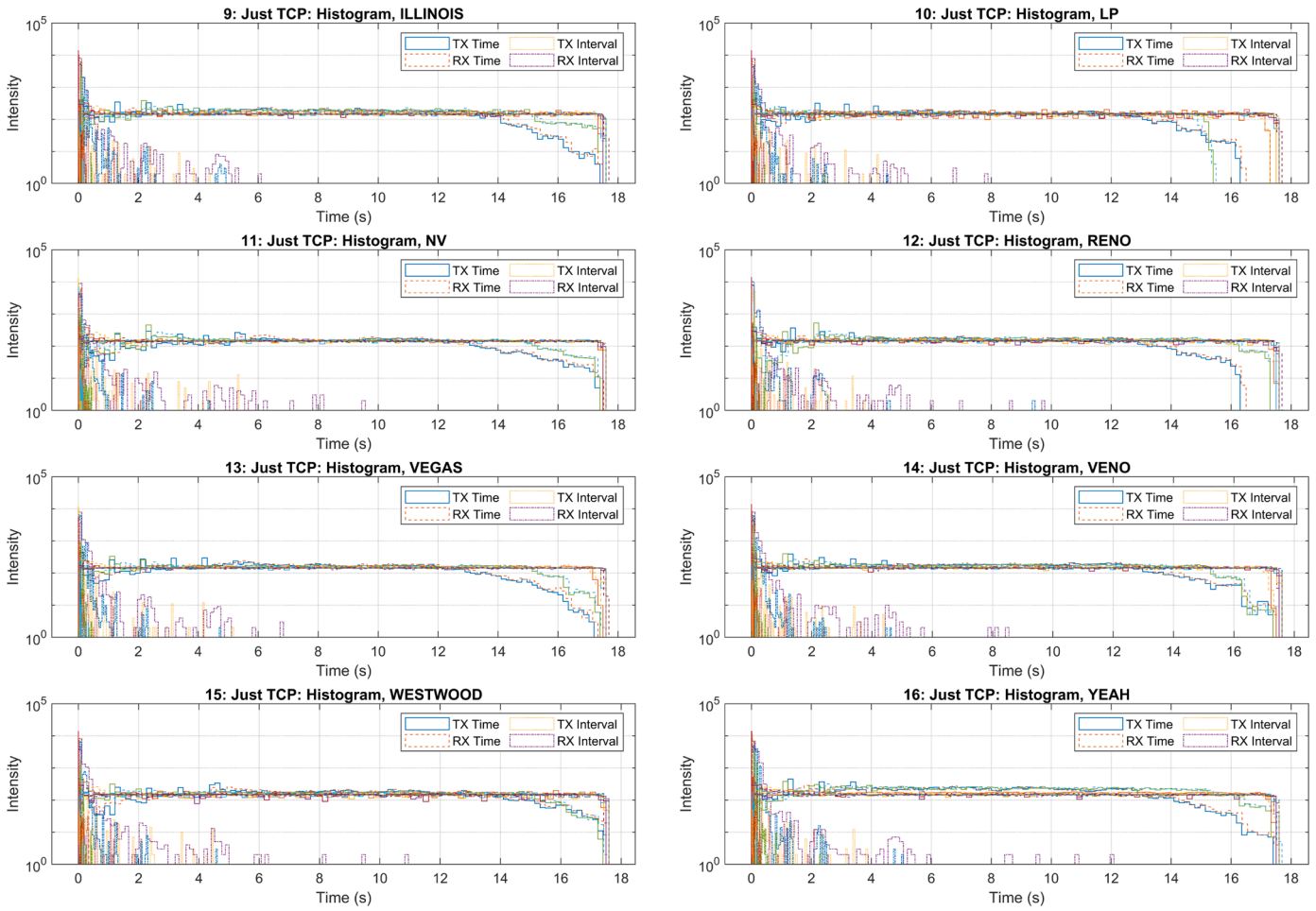

Fig. A.2. Histogram of transmit (Tx Time), receive (Rx Time), transmit interval (Tx interval) and receive interval (Rx Interval) (Only TCP parallel sessions - TCP mechanisms: ILLINOIS, LP, NV, RENO, VEGAS, VENO, WESTWOOD, YEAH). Transmit and receive times of the TCP segments are roughly uniform distributed for each congestion control mechanism. Transmit and receive time intervals of the TCP segments have exponential distribution.

protocols data units sent between the communicating entities have no relation to each other. The name comes from the fact that these services do not build a connection before the communication starts. This usually results in a very lightweight communication solution [24]. Connectionless communication protocols appear on several levels of the OSI reference model including protocols like some services of the LLC sublayer of the Data Link layer or the IP protocol in the Network layer however from the point of our investigations the Transport layer User Datagram Protocol (UDP) [25] is the interesting one. Operating as an *L*4 data transmission protocol UDP segment headers contain only port information about the sender and the receiver followed by fields useful to provide data integrity.

Since sent segments have no information about each other congestion control mechanisms like those that the TCP protocol implementations have are missing from this protocol. This usually results in a much less fair utilization of the link bandwidth. Intuitively this means that one strong UDP session can force multiple TCP sessions to the background. On the other hand however, without tracking the segments UPD protocol on its own cannot guarantee reliable data transfer. As a result, if no application layer segment management is added, some segments will be lost. Knowing this, the amount of lost data segments is a key property we have studied in the case of this protocol.

4. Efficiency analysis of the parallel sessions in transport layer

Chapter two described the general problem of the multisession file transmission mechanisms. Usage of multipath TCP on single physical LAN interface card of the client computer with multiple core processor degrades the performance of the overall communication. Asymmetric usage of the IP based best effort network services inside of client node in favour of UDP sessions makes usable greater amount of network resources for fast file transfer mechanisms based on UDP. This hypothesis is analyzed in the next subsections using special measurement network scenario. Parallel file transfer sessions were running to evaluate cross effect of simultaneous data transfer processes. It should be mentioned that the packet capturing tools were running on the same nodes as the fast file transfer processes using low but non negligible amount of computation capacity of the test nodes. This implies that the results found in favour of UDP should provide greater performance in practice than the results shown in our figures.

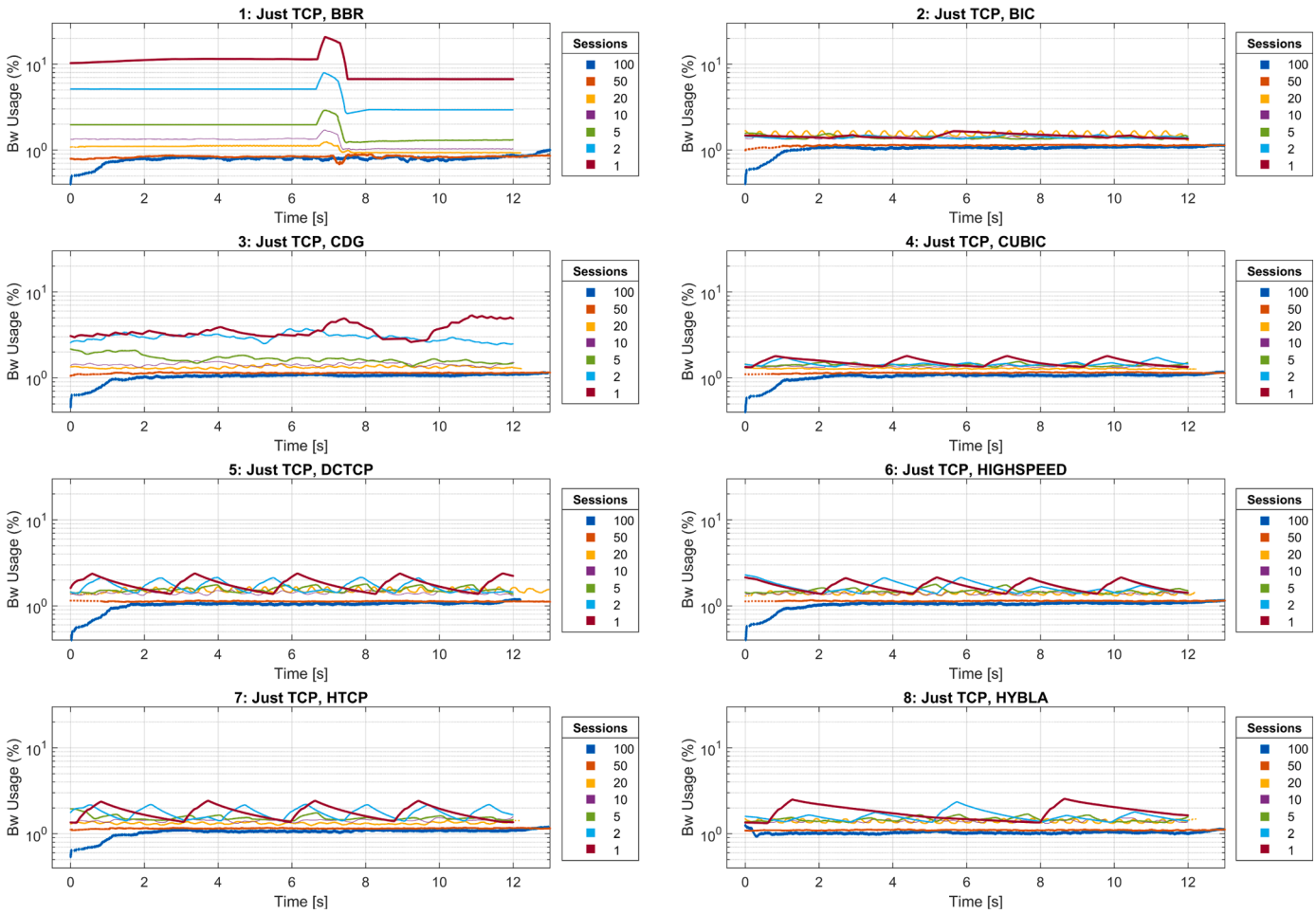

Fig. A.3. Bandwidth usage (Only TCP parallel sessions - TCP mechanisms: BBR, BIC, CDG, CUBIC, DCTCP, HIGHSPEED, HTCP, HYBLA). The higher is the number of parallel TCP sessions, the lower bandwidth remains for each TCP session and the lower is the standard deviation. Dynamicity of each congestion control mechanism determine the behaviour of parallel sessions.

4.1. Measurement scenario of the parallel communication sessions (Tool 1)

In order to be as close to the actual real use-case as possible, a physical measurement environment was engineered and built between two end nodes running servers. The system consists of a dedicated IEEE 802.3 (Ethernet) interface between the servers with two routers. Fig. 1 shows the relationship between the parts. $R1$ and $R2$ routers are used to provide LAN/MAN network connectivity and to have better control over the Ethernet line. These intermediate nodes are reached through separate interfaces ($C1$ and $C2$) to modify the configurations and allow customized measurements.

Both servers are connected to the Internet, as well. These connections provide direct interaction with routers $R1$ and $R2$ without disturbing any measurement running on the dedicated line. The servers are connected with gigabit Ethernet interfaces to the routers and the dedicated link between the routers is Ethernet, too. The transmission ratio of this link is set to maximum 10 Mb/s to limit the amount of output data generated in each measurement case and form a bottleneck of the dumbbell network.

Our framework has been tested and validated before running the performance measurements. During several tests we have also evaluated and fine-tuned the parameters to be used for the performance evaluation.

We measured TCP and UDP throughput on our test system. It was necessary to run many parallel sessions, try out TCP congestion control algorithms and run mixed TCP and UDP tests. We also needed to monitor every packet sent and arrived on the measurement interface for further evaluation. We used *iperf*3 to generate the data flow being a widely used network throughput measurement software. *iperf*3 can generate both TCP and UDP streams in multiple parallel sessions, it can control UDP bitrate and use TCP congestion control algorithms.

To be able to analyze the network behaviour during the measurement we needed to capture and store each IP packet. We also needed to measure the amount of time passed between the packet being sent from server $S1$ and arrived to server $S2$. This task is provided by a script sniffing on the Ethernet line during the communication both on server $S1$ (sender) and $S2$ (receiver) and catches each packet with its timestamp. At the end of each measurement test case the two data set

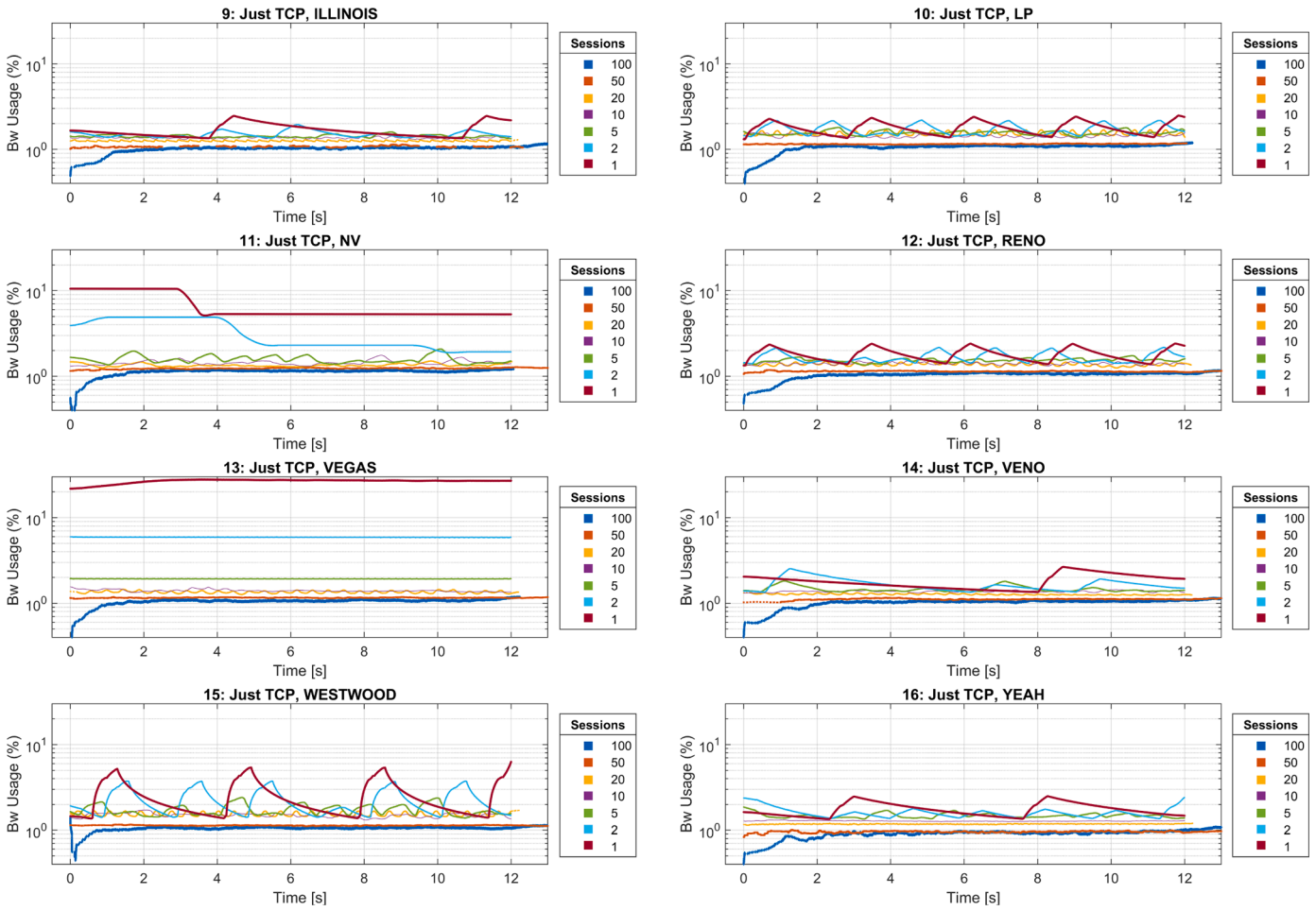

Fig. A.4. Bandwidth usage (Only TCP parallel sessions - TCP mechanisms: ILLINOIS, LP, NV, RENO, VEGAS, VENO, WESTWOOD, YEAH). The higher is the number of parallel TCP sessions, the lower bandwidth remains for each TCP session and the lower is the standard deviation. Dynamicity of each congestion control mechanism determine the behaviour of parallel sessions.

sampled at both servers were joined. The output data set then contains every package sent with its sending and arrival timestamps. This means that we needed to pair up each packet captured on $S1$ with its counterpart on $S2$. The pairing of the packets with the necessary time stamps require unique identification method. Because neither the IP header nor the TCP or UDP headers contain unique identification field, we had to implement this functionality in the payload of each packet. The base implementation of *iperf*3 sends random generated data as the payload of each package. We modified this behaviour so each package contains a 32 bit number at the start of the payload, and we used this number as an identifier.

To sniff on the Ethernet line we chose *tcpdump*. It catches every packet which goes through the local interface of the server. It can parse the binary data of the packet, and returns the header information and the pure packet payload data. *tcpdump* assigns the timestamp to the packet at the capturing moment. We used this timestamp to measure the delay of each packet. We also got the session port number, the packet size and the generated 32 bit data from the payload of each packet.

After each measurement session the software collects the output data from both servers, then joins those using the generated packet ID. The final output data is in a CSV file format. Rules of the data structure organization are following:

1. The first number is the timestamp when the packet is sent from server $S1$ in epoch seconds with microsecond accuracy (Column 1).
2. The second number is the timestamp when the packet arrives to server $S2$ in the same format. The value is ”-1” when the packet did not arrive to server $S2$. Such event can occur in UDP sessions because this protocol does not guaranty that each packet will be successfully delivered (Column 2) in congestion situation.
3. The third number is the session port number used to separate the captured data from parallel sessions (Column 3).
4. The next number is the size of the IP packet in bytes (Column 4), and
5. The last one is the generated identification number for each individual sent and received packet (Column 5).

Six records of the measured data set in a CSV file is given in the Table 1.

To ensure accuracy of the delay time between consecutive packets and delivery time of each individual packet it was necessary to synchronize the two server clocks. We installed a time server on Server $S2$ and before each measurement case the application syncs the clock of server $S1$ to server $S2$. The final software architecture is shown on Fig. 1.

There were executed different simultaneous traffics between the servers S1 and S2: i) Homogeneous TCP sessions; ii) Homogeneous UDP

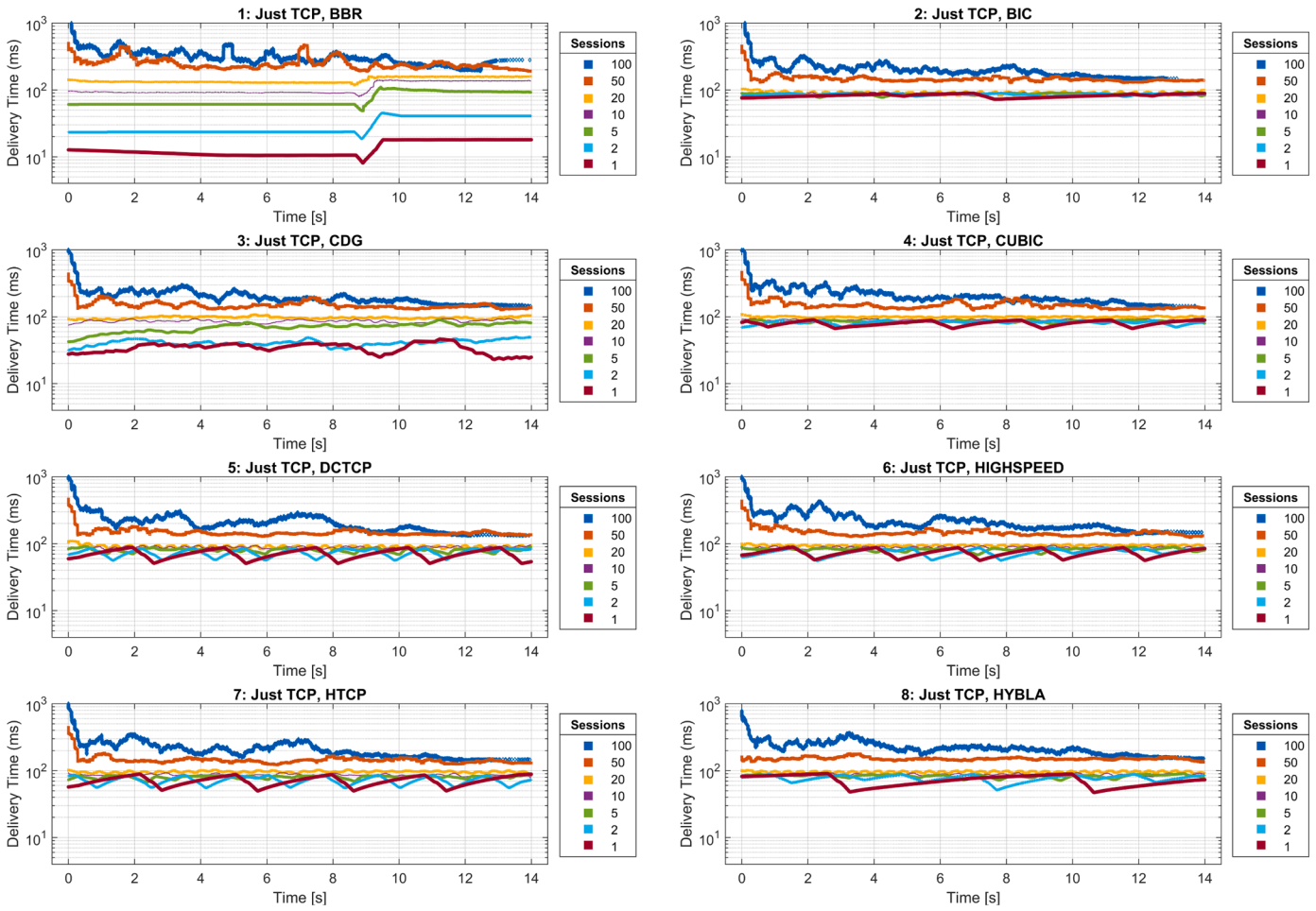

Fig. A.5. Delivery time (Only TCP parallel sessions - TCP mechanisms: BBR, BIC, CDG, CUBIC, DCTCP, HIGHSPEED, HTCP, HYBLA). The higher is the number of sessions, the higher time is required to deliver TCP segments. For high number of parallel TCP sessions the delivery time decreases in time.

sessions; iii) Heterogeneous TCP and UDP with equal number of flows; iv) Heterogeneous TCP and UDP with unequal number of flows. The number of homogeneous sessions were: 1, 2, 5, 10, 20, 50, 100. The number of heterogeneous sessions TCP/UDP sessions were k/k (k = 1, 2, 5, 10, 20, 50, 100) for equal and k/(40-k), (k = 0, 1, 2, 5, 10, 15, 20, 25, 30, 35, 38, 39,40) for unequal number of flows, respectively. Sixteen different congestion control mechanisms were applied when TCP was used as transport layer service: BBR, BIC, CDG, CUBIC, DCTCP, HIGHSPEED, HTCP, HYBLA, ILLINOIS, LP, NV, RENO, VEGAS, VENO, WESTWOOD and YEAH.

4.2. *Efficiency aspects of the homogeneous parallel TCP sessions*

Based on the transmitting and receiving timestamps, the delivery time τ of frames is analyzed as a function of time. This time series analysis showed the differences of the flow control mechanisms of various TCP versions. Some representative examples are illustrated in Fig. 2 (left). The behavioural differences of TCP version are visible, saw tooth curves are different in shape, amplitude and frequency. That is why it is important to know the effect of these differences in case of a more complex network session scenario.

Not just the TCP versions differ, but also each session in case of the same control flow mechanism. Fig. 2 (right) shows the huge variety of the average performance of individual session and the significant fluctuation of them during the measurement. As one can see the interacting behaviour of this complex parallel system cannot be predicted by the behaviour of only one session.

The channel utilization of the aggregated traffic between the routers are always high, even in case of 100 parallel TCP sessions it is above 99.46%. However the version of TCP has influence on this value, it is negligible. Consequently the collective behaviour of parallel sessions of any TCP version ensure high performance, nevertheless the individual performance of sessions strongly fluctuates.

Due to the heavy load and the bottleneck of the system router $R1$ sometimes discards incoming packets from server $S1$. This leads to retransmission of the related TCP segments, resulting in more traffic. The number of re-sent segments as a function of the number of parallel sessions obeys to power-law in case of most TCP version. Significantly the BBR congestion control algorithm of the TCP has the highest ratio of retransmission. (see Fig. 3)

More graphical details with short explanations of the homogeneous parallel TCP sessions can be found in the Annex (see Figs. A.1, A.2, A.3,

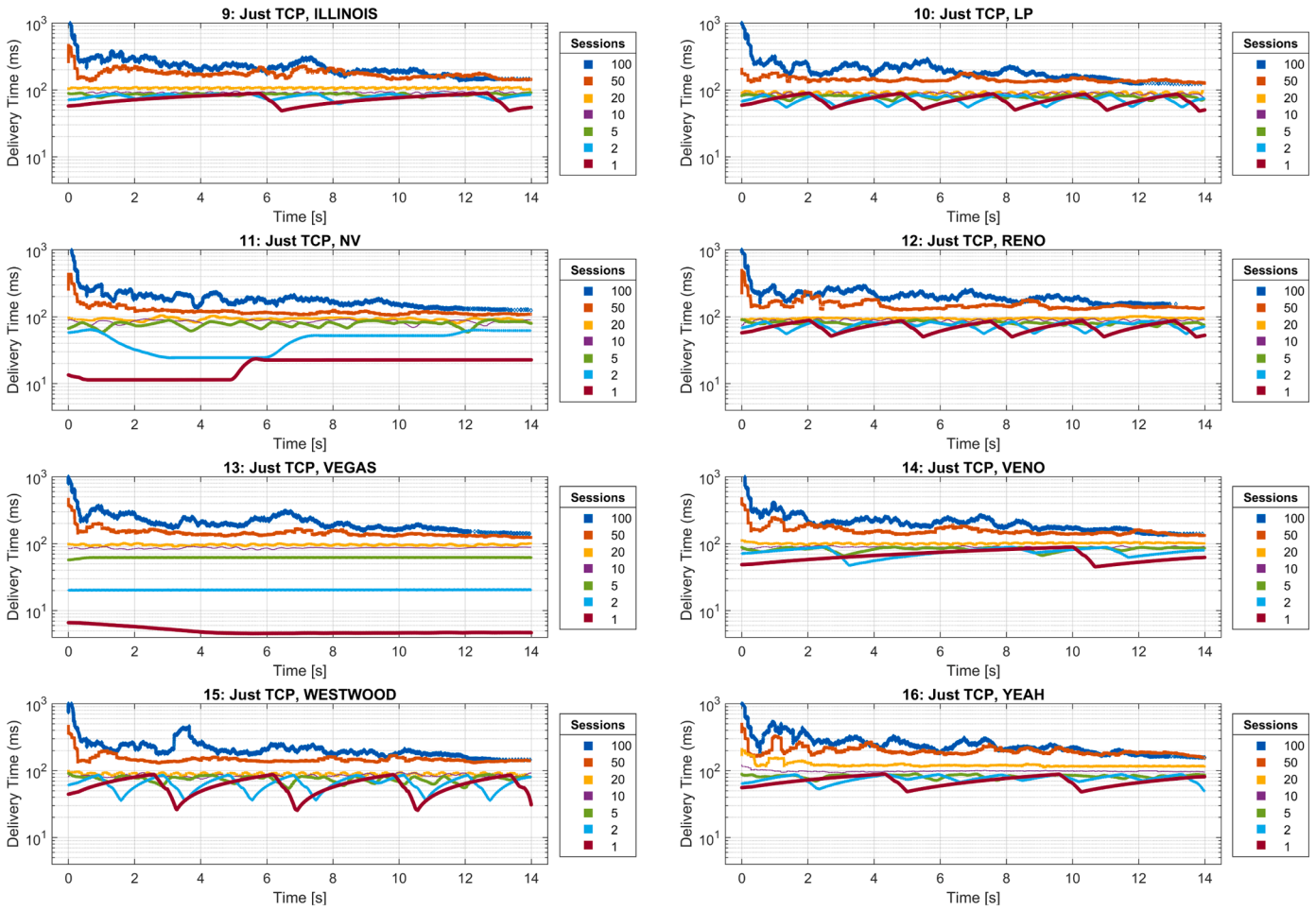

Fig. A.6. Delivery time (Only TCP parallel sessions - TCP mechanisms: ILLINOIS, LP, NV, RENO, VEGAS, VENO, WESTWOOD, YEAH). The higher is the number of sessions, the higher time is required to deliver TCP segments. For high number of parallel TCP sessions the delivery time decreases in time.

A.4, A.5, and A.6).

4.3. *Efficiency aspects of the homogeneous parallel UDP sessions*

As it was pointed out in Section 3.2 in the case of homogeneous UDP protocol sessions one of the most interesting property of the sessions is data loss. We ran simulations for 1, 2, 5, 10, 20, 50 and 100 homogeneous UDP sessions where the maximal speed of each session was limited to 1 Mb/s. As one can see on Fig. 4 when the number of sessions is less than 10 not surprisingly we get linear growth of the aggregated speed and no lost segments for the sessions. However, both *(left)* and *(right)* graphs on Fig. 4 show that even though in case of 10 sessions the amount of data would just fit to the bandwidth we lose some segments and the speed stays on a lower level than the possible theoretical maximum. This speed stays constant for more than 10 segments while the ratio of dropped data increases.

Examining the speed of the sessions separately for different session numbers one can find that the lost data is not originated equally from the different sessions. Instead of that some sessions show relatively high speed while others hardly send anything during the measurements. It is also interesting to note that even in case of relatively high number of sessions a small number of behaviours are followed by all the sessions instead of behaving independently. As an illustrative picture see Fig. 5 *(left)* where it is shown the speed of 10 separate UDP sessions and their aggregated speed as a function of time.

Here the sessions group to three sets between 0 and 1 Mb/s. The most interesting part of course of the above investigations is that for the case of 10 or more sessions the total capacity of the link is not used. Particularly, for 10 pieces of 1 Mb/s sessions the aggregated speed is not 10 Mb/s but only about 9.5 Mb/s. This maximal speed of the link in case of homogeneous UDP sessions seems to be robust, since for all cases where we have more than 10 sessions the same result was found. In order to find out whether this behaviour is somehow related to the session number we compared equal aggregated speed cases - i.e. cases where we had different number of sessions, while their aggregated transmit speed were the same. As plotted on Fig. 5 (right) we found that for lower than 10 Mb/s aggregated transmit speed this ratio is the simple sum of the transmit speeds of the sessions, while in cases where we had a greater or equal to 10 Mb/s aggregated transmit speed this ratio was about to take a constant value somewhere close to 9.5 Mb/s or even less in cases when some sessions died out as a result of always congesting packets.

As some basic properties of the sessions we also examined the waiting times and the delays as functions of time. Our measurements showed that the dependence on the session number we seen in the case

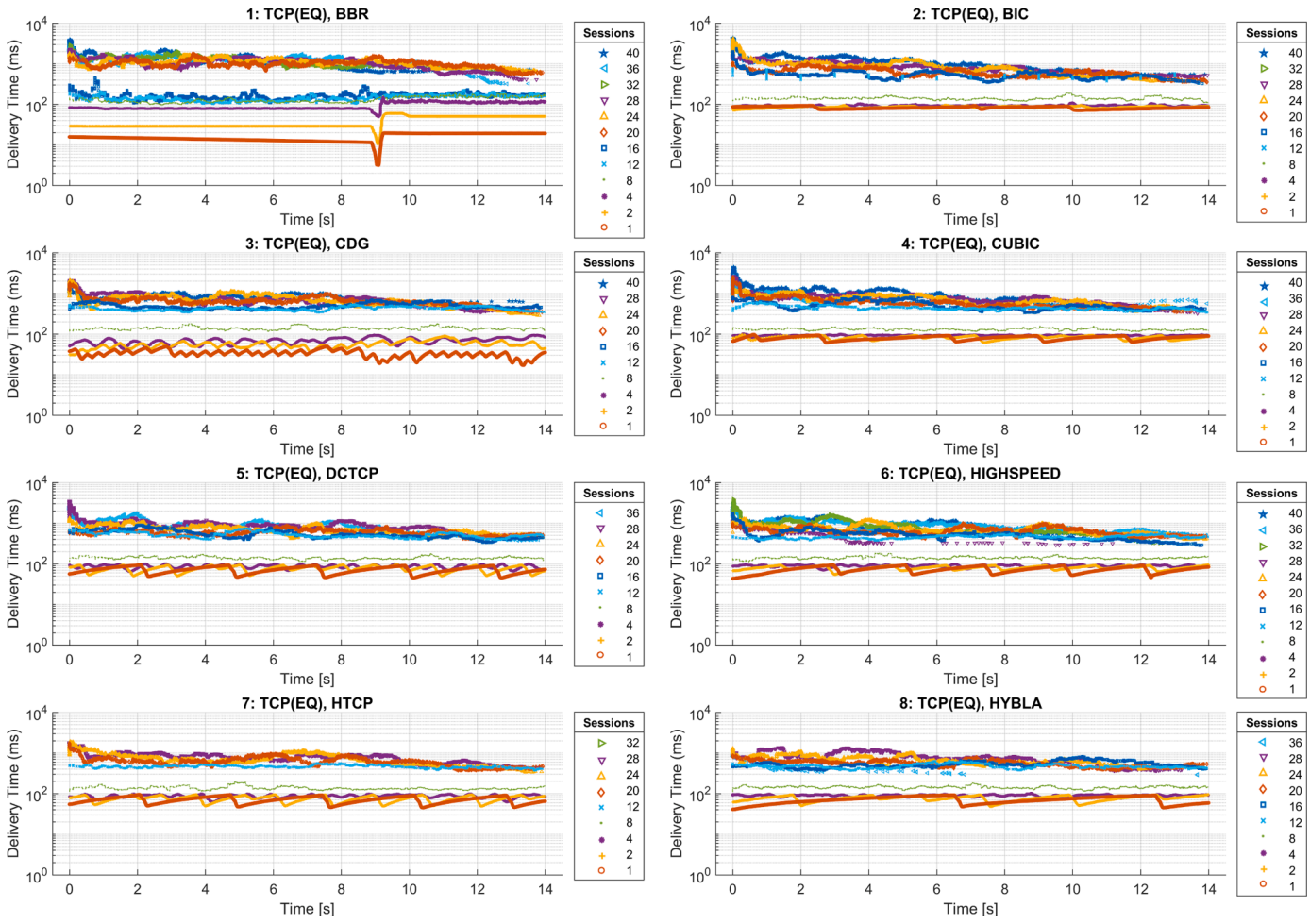

Fig. A.7. Delivery time of TCP segments (Equal No. of TCP and UDP sessions - TCP mechanisms: BBR, BIC, CDG, CUBIC, DCTCP, HIGHSPEED, HTCP, HYBLA). The higher is the number of parallel TCP and UDP sessions, the higher becomes the delivery time of TCP segments for each TCP session. For high number of parallel TCP and UDP sessions the delivery time of TCP segments decreases in time independently of the congestion control mechanism. Somme congestion control mechanisms do not permit any number of parallel sessions (e.g. HTCP, NV, VENO, WESTWOOD).

of the previous properties do not appear for the waiting time. More precisely the waiting times (the time elapsed between two sent segments) proven to be around 0.011s in all cases for all sessions. As an illustrative example we plotted the case of 50 sessions on Fig. 6 (left). For the delays we saw some fluctuations for lower than 10 number of while from 10 sessions for all the session numbers we found that the delays are about 0.1s. Fig. 6 (right) shows the average delays as a function of the session number.

4.4. Efficiency aspects of the heterogeneous parallel TCP and UDP sessions

When the channel is used by the same amount of TCP and UDP sessions in parallel, the aggregated utilization of the 10 Mb/s channel is very close to the 100% independently of the number of sessions. We found that the ratio of the TCP and UDP traffic as a number of sessions has two regimes: i) When the UDP sessions alone are not able to fill the channel totally, the number of delivered TCP segments is changing. More sessions lead to less delivered TCP segments. ii) After the crossover around 10 TCP and 10 UDP sessions, the number of UDP drops is linearly increasing, while the number of successfully delivered TCP segments is constant. Thus in case of several competing TCP and UDP sessions, UDP cannot suppress completely the TCP, at least the 6% of the channel is loaded by TCP traffic independently of the heavy UDP traffic (see Fig. 7).

More graphical details with short explanations of the heterogeneous parallel TCP+UDP sessions can be found in the Annex. Dependence of the TCP and UDP segments delivery time on the number of parallel UDP sessions is given in Figs. A.7, A.8, A.9 and A.10, respectively.

When the number of parallel sessions is constant, just the ratio of the two protocols is changing and crossover can be observed again just its location is different. Numerous UDP sessions dominate over the few TCP sessions naturally, but when the number of TCP sessions is higher than the number of UDP ones significant TCP traffic can be found in the channel as it is shown in Fig. 8. Generally, the channel utilization is high but one can observe a minimum just before the pure TCP case (at 38 TCP and 2 UDP sessions).

5. FMFT (Fast Manager of File Transfer) (Tool 2)

A new server - client model software pair was developed based on Xinan Liu's work (see Fig. 9) [26]. This software tool is based on parallel data forwarding and control sessions.

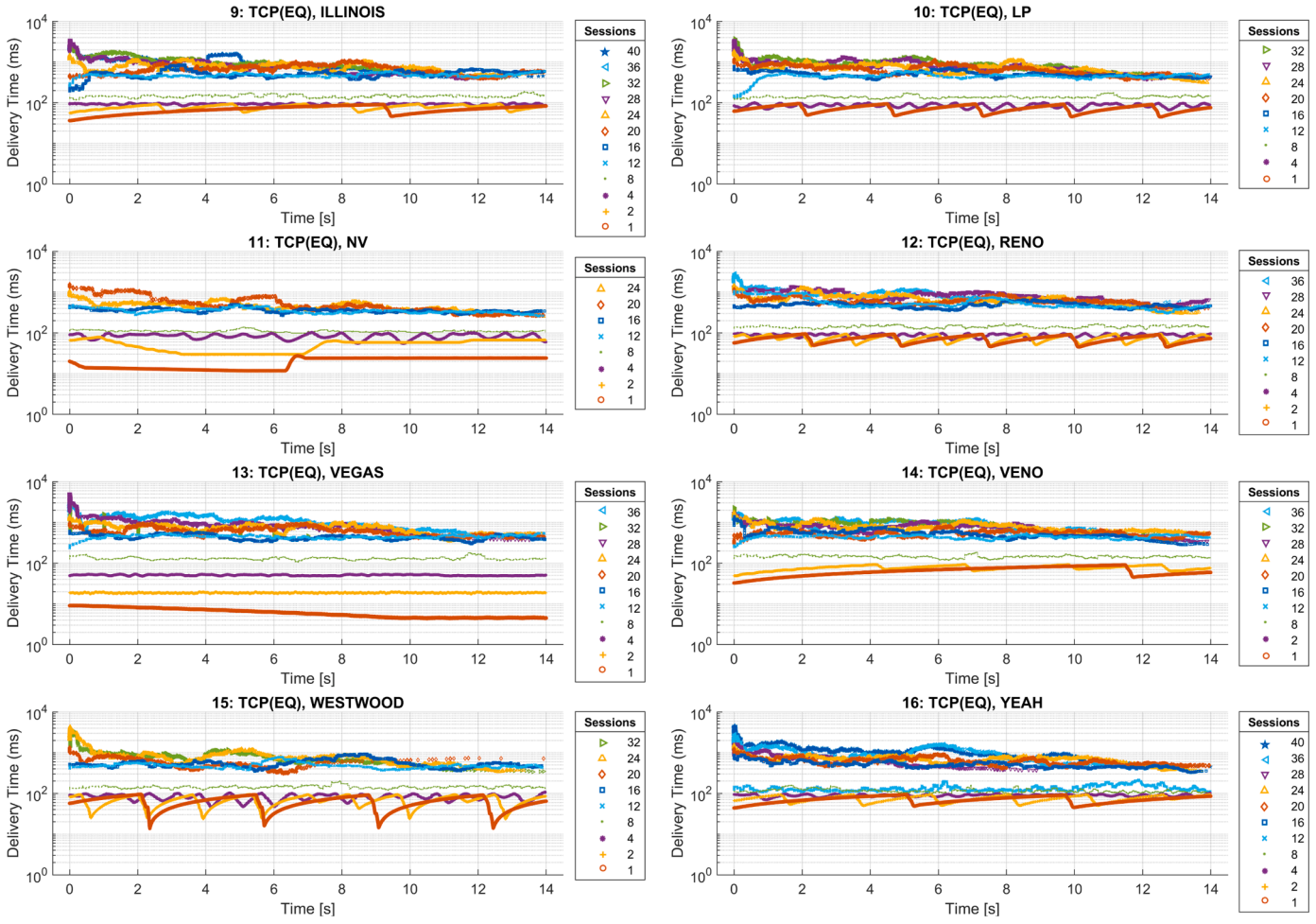

Fig. A.8. Delivery time of TCP segments (Equal No. of TCP and UDP sessions - TCP mechanisms: ILLINOIS, LP, NV, RENO, VEGAS, VENO, WESTWOOD, YEAH). The higher is the number of parallel TCP and UDP sessions, the higher becomes the delivery time of TCP segments for each TCP session. For high number of parallel TCP and UDP sessions the delivery time of TCP segments decreases in time independently of the congestion control mechanism. Somme congestion control mechanisms do not permit any number of parallel sessions (e.g. HTCP, NV, VENO, WESTWOOD).

5.1. Architecture and service description of FMFT

As a part of our software engineering tasks we have improved the logging subsystem. The new software is command line driven and the transfer rate can be limited. Moreover, the type how the transfer rate is controlled can be set to bursted or equally distributed among a time slot of one second. Basically a TCP control channel is used for UDP based file transfer. A boolean array is utilized for the received packages.

The channel opening fixed length first chunk (describing the required channel parameters) is repeatedly sent until the first acknowledgement package is received. Afterwards, there is a first round of trying to send all the packages in consecutive order. Later the new resend round starts again from the first element.

In this way the maximum time slot is given for the sent packages to arrive at the destination. The chunks can arrive in mixed order, the software stores them in the target file at the appropriate places. Resending phases happen in full loops containing less and less undelivered elements. A Windows GUI application was developed for testing purposes as it is shown in Fig. 10.

Here we can easily setup the parameters. The GUI has control elements for our client, as follows:

- `Browse` button, for selecting the folder holding data files. Choosing the folder can be done by using a standard dialog panel.
- `Server` combo box, it holds the predefined target server machine addresses for file transfers.
- `Files` list, it contains the files of the selected folder.
- `Copy` button, initiate the transfer of the selected data file.
- `Burst mode` checkbox, it sets how the transfer rate is controlled. Bursted or equally distributed segment transfer modes can be chosen.
- `Overwrite` checkbox, it enables overwriting the file on the server size. It makes testing easier by enabling to use the same test file multiple times.
- `Small log` checkbox, when just the basic information as transfer time is enough for the measurements.
- `Port number` control, sets the port of communication.
- `Chunk size` control, sets the desired chunk length sent at once. Test of UDP segments transmission is executed on control channel at end of each chunk.
- `Transfer rate` control, it can limit the usage of the bandwidth, thus allow other network nodes to have a reserved part of the channel.

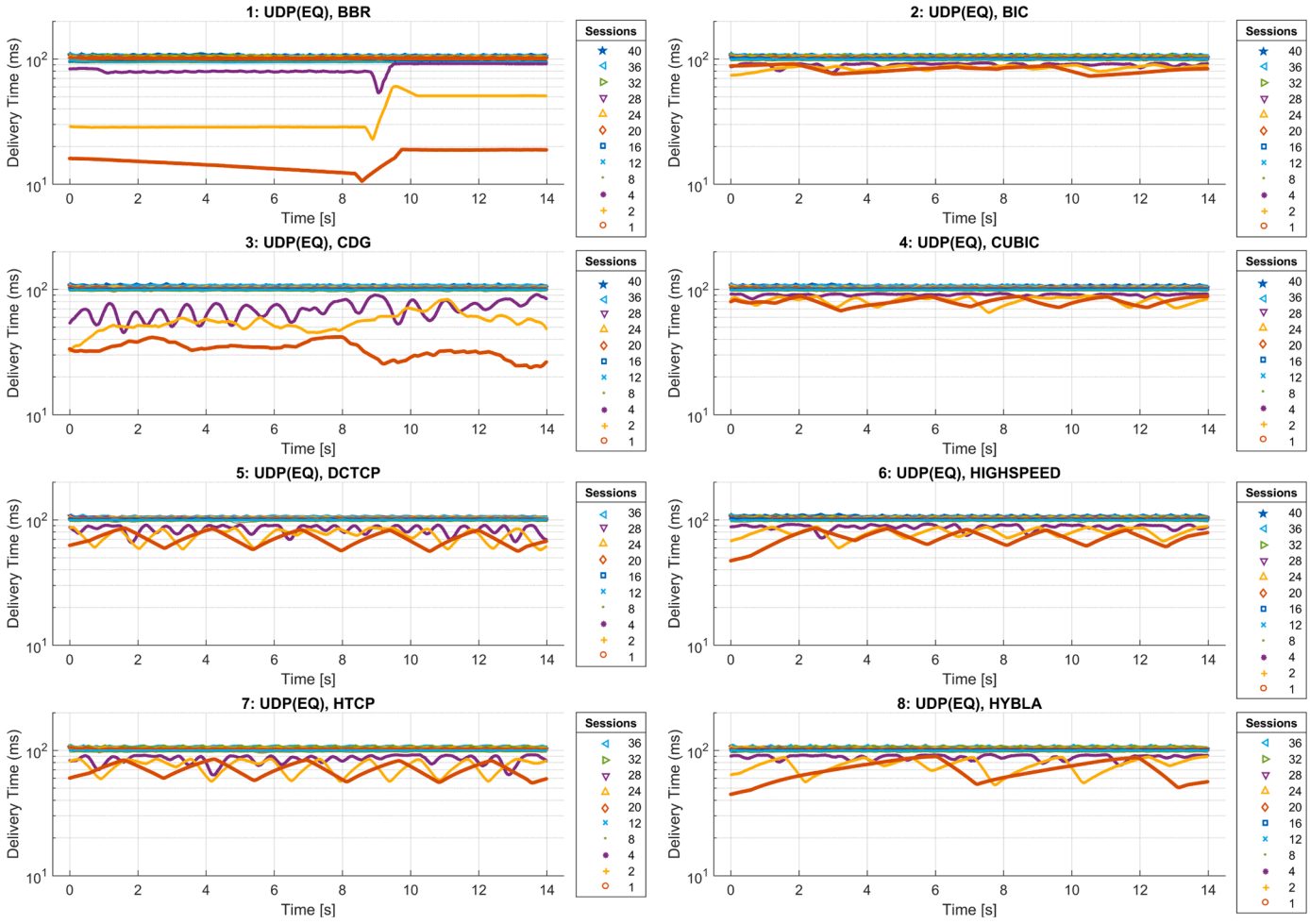

Fig. A.9. Delivery time of UDP segments (Equal No. of TCP and UDP sessions - TCP mechanisms: BBR, BIC, CDG, CUBIC, DCTCP, HIGHSPEED, HTCP, HYBLA). The higher is the number of parallel TCP and UDP sessions, the higher becomes the delivery time of UDP segments for each UDP session. The delivery time of UDP segments is 3...10 times faster than of TCP segments. Depending on the congestion control algorithm type (e.g. DCTCP, LP, NV, WESTWOOD) of the parallel TCP sessions higher number of the parallel UDP sessions fail to transmit segments. When the channel becomes congested due of parallel sessions the UDP sessions start to fluctuate the delivery time of segments.

- `Start chunk size` control, for testing purposes it sets the smallest chunk size to test.
- `Chunk size step` control, defines the incremental step of the chunk size during a sweep copy test.
- `End chunk size` control, this is the maximum allowed chunk size for transferring the data. Lower or equal size is used in a sweep copy series.
- `Sweep copy` button, initiates a series of consecutive transfers of the same file using varying parameters. The target file is always overwritten if it already exists.

The valid ranges of the slider controls are enlisted in Table 2.

In this way the application is well suited for making sweep tests across parameter value ranges. Series of measurements can be carried out easily. The results are saved in log files. The send and receive events have time stamps. The inner resolution of the time measurement is nanosecond. Albeit, in the full log only microseconds are stored.

A small log item typically looks like this:

```
Host: 10.10.10.10
Port number: 10000
File: test.dat
Chunk size: 992
Overwrite: true
[2019.05.14 23:18:43.085] Sending started.
File length: 206566075
Number of chunks: 208232
[2019.05.14 23:18:50.287] File sent in 7.18 seconds.
Transfer rate: 230.2 Mbit/s.
```

The structure of a full log is the following (note, that packages in the range 193848-193855 arrived in the end in mixed order, after a resending loop):

```
Host: 10.10.10.10
Port number: 10000
File: test.dat
Chunk size: 992
Overwrite: true
[2019.05.14 23:19:02.924] Sending started.
File length: 206566075
Number of chunks: 208232
3277 S 0
3444 S 0
3604 S 0...
40203 S 0
40280 S 0
53470 S 0
50127 R 0
53606 S 1
53686 S 2
53758 S 3...
55067 S 20
55158 S 21
55230 S 22
55240 R 1
55302 R 2
55308 S 23
55360 R 3...
10483439 S 193853
10483491 S 193854
10483539 S 193855
10483584 R 193848
10483602 R 193849
10483617 R 193850
10483817 R 193851
10483834 R 193852
10483850 R 193853
10483865 R 193854
10484038 R 193855
[2019.05.14 23:19:13.414] File sent in 10.49 seconds.
Transfer rate: 157.5 Mbit/s.
```

5.2. Evaluation of parallel communication sessions with FMFT tool

We have used two desktop machines as clients for testing. This two machine configurations were applied: Test machine 1 ($Client_1$: Windows 10 Enterprise 64-bit, Intel Core i7 4771 @ 3.50GHz, Haswell 22nm Technology, 24 GB RAM DDR3 @ 799MHz ASRock H87M Pro4 Motherboard, Intel Gigabit NIC); Test machine 2 ($Client_2$: Windows 8.1, Intel Core i3 2120 @ 3.30GHz, Sandy Bridge 32nm Technology, 8 GB RAM DDR3 @ 665MHz, Gigabyte Technology Co. H61M-S2PV REV 2.2 Motherboard, Realtek PCIe GBE NIC). Both clients were connected to a common server running Linux operating system. Each client executed transfers of two files with different transfer rate preset in FMFT tool to capture data segment transmission details. File transfer tests were executed in non-overlapping time intervals. The preset data rates were $Bw[Mbps] = k{\cdot}50, k = 1, \ldots, 20$. The number of measurements for each value of k depends on the file size and segments size values. Possible segment size in number of bytes is given by $Ss[B] = 1000 + (m-1){\cdot}2000$ formula. The file sizes and segment sizes used in the measurement scenario are included in Table 3. The UDP segment size was incremented by 2,000 bytes starting from 1,000 bytes for both clients. The maximum UDP segment size for $Client_1$ was 65,000 B. Because of small capacity of $Client_2$ the maximum size of UDP segments were just 35,000 bytes.

Measured transfer time of the files is represented in Fig. 11. As the preset bandwidth of the communication increases, the transfer time for each of the four files decreases. It should be mentioned that the variance of transfer time increases with parameter k. Each set of measurements with fixed value of bandwidth starts very slowly, implying spike at each strip of the graph. However, quick decrease of the transmit time can be observed inside of strip region. This feature of the UDP traffic produced with FMFT emulates slow start feature of the TCP protocol.

Transfer rate of the files measured is given in Fig. 12. As the preset bandwidth of the communication increases, the average measured transfer rate for each of the four files increases. However the transfer rate at the beginning of each set of measurements for preset bandwidth is relatively small then the data rate increases significantly. For big file sizes exist some segment sizes with relatively low communication performance. This phenomenon is more intensive for client with higher computation capacity.

Measured channel load of the file transfer is given in Fig. 13. It is a special feature of the FMFT tool to use higher bandwidth than was preset previously. It can be observed that neither low nor high bandwidth preset values are advantageous. Several local maximums appear and even the global maximum channel usage may be reached with more than one preset parameter tuples. Similar feature can be detected for lower capacity of the client, as well. We must be careful with the preset bandwidth parameter because during the transmission of large files all the network resources of the physical LAN channel can be consumed.

Comparison of data transfer times is given in Fig. 14. Increasing the preset bandwidth decreases the transfer time for any file size and client processing capacity. Windows 8.1 operating systems behaves unusually for different segment sizes when the data transfer rate limit is preset high for relatively low capacity client (See *Down-left* and *Down-right*).

Dispersion index of the measured transfer time, measured data transfer rate, and measured channel load are represented in Fig. 15. As we mentioned earlier, the dispersion of the measured values increases when the preset transfer rate limit increases.

6. Conclusions and future work

Cross effect of the simultaneous communication flows in transport layer was analyzed in this paper. Sixteen different congestion control algorithms were used in a dumbbell traffic measurement scenario. Homogeneous TCP, homogeneous UDP and heterogeneous TCP/UDP flows were transmitting simultaneous data flows in the same communication path. It was found that TCP mechanism cannot use the maximum channel bandwidth because of the dynamic adaptable congestion and flow control algorithm. Different congestion control algorithms have different behaviour in LAN network environment with high number of simultaneous communication sessions and intensive traffic rate. UDP delivers segment stream much faster than TCP but needs efficient supervising mechanism of the transport layer PDUs to provide controlled transmission time of a big file. Limit needs to be set for the maximum transmission rate to protect the channel from the intensive UDP datagram retransmissions. More analyses are proposed to evaluate parallel communication sessions in WAN environment, too.

The FMFT software tool is written in *Java*. To further improve the performance of the tool, reimplementation in C++ would increase the overall speed and timing at reduced stress on the CPU hopefully. It is planned to find the optimal number of UDP channels for the highest preset data transfer capacity in the future. FMFT shall be improved to handle SCP connections as well and make comprehensive throughout tests. Beyond testing, the software is planned to be enhanced to use it for regular high speed reliable file transfer for big files on busy connections. Thanks to the chosen Qt framework, the GUI can be compiled for various platforms, including mobile platforms. Moreover, using the WASM target it will also be available through a web browser.

CRediT authorship contribution statement

Zoltán Gál: Conceptualization, Methodology, Supervision, Writing - review & editing. **Gergely Kocsis:** Visualization, Validation, Writing - original draft. **Tibor Tajti:** Data curation, Investigation, Validation, Writing - original draft. **Robert Tornai:** Software, Validation, Writing -

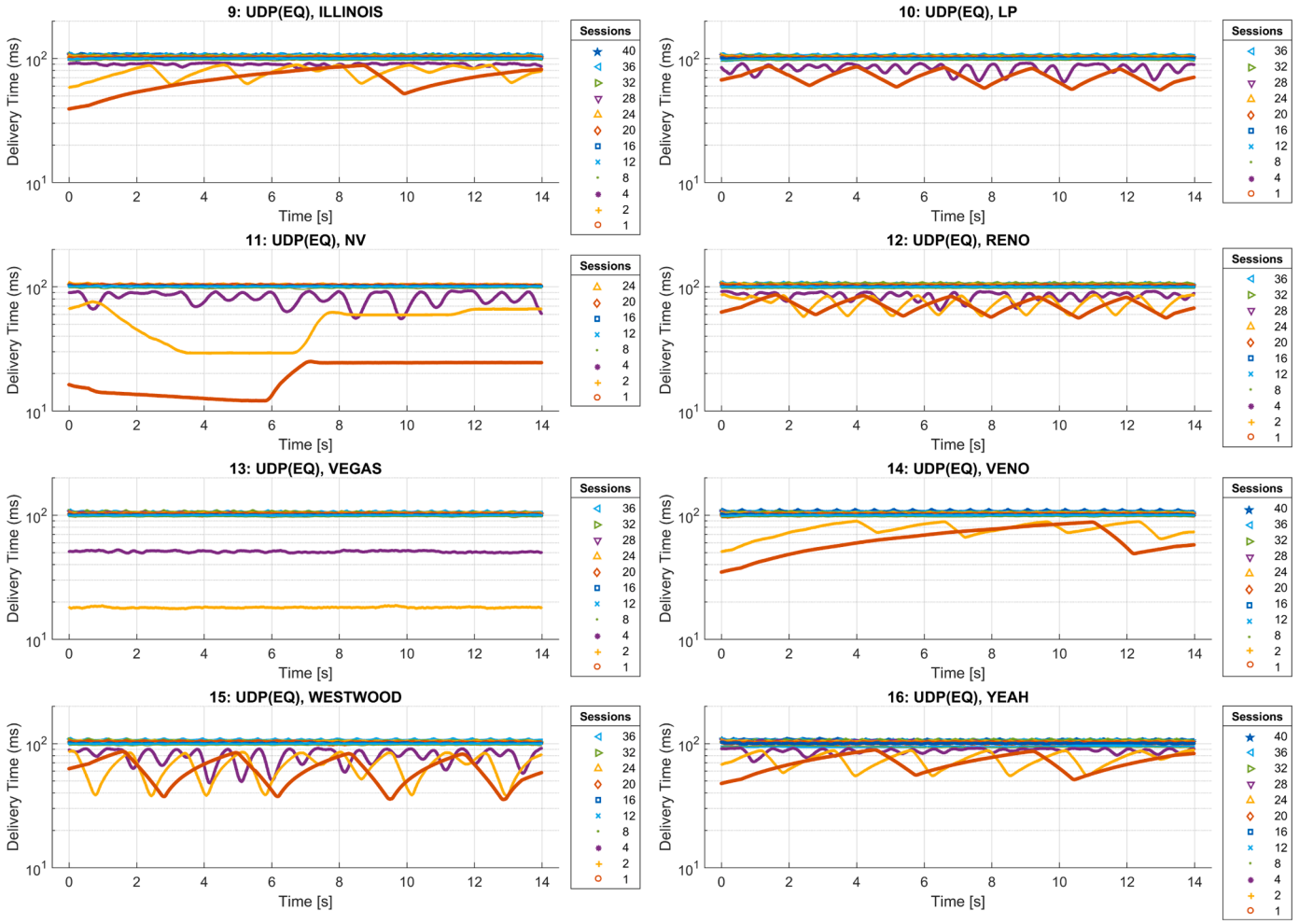

Fig. A.10. Delivery time of UDP segments (Equal No. of TCP and UDP sessions - TCP mechanisms: ILLINOIS, LP, NV, RENO, VEGAS, VENO, WESTWOOD, YEAH). The higher is the number of parallel TCP and UDP sessions, the higher becomes the delivery time of UDP segments for each UDP session. The delivery time of UDP segments is 3...10 times faster than of TCP segments. Depending on the congestion control algorithm type (e.g. DCTCP, LP, NV, WESTWOOD) of the parallel TCP sessions higher number of the parallel UDP sessions fail to transmit segments. When the channel becomes congested due of parallel sessions the UDP sessions start to fluctuate the delivery time of segments.

original draft.

Declaration of Competing Interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Acknowledgments

This paper was supported by two projects of the University of Debrecen, Debrecen, Hungary: Big Data project with code FIKP-20428-3/2018/FEKUTSTRAT. This work was also supported by the construction EFOP-3.6.3-VEKOP-16-2017-00002. The project was supported by the European Union, co-financed by the European Social Fund and EFOP-3.6.3-VEKOP-16-2017-00002, respectively. Thanks to QoS-HPC-IoT Laboratory for technological assistance.

Appendix A. Annex: Figures of parallel data traffics

Supplementary material

Supplementary material associated with this article can be found, in the online version, at 10.1016/j.advengsoft.2021.103010

References

[1] Elgendy N, Elragal A. Big data analytics: a literature review paper. In: Perner P, editor. Advances in data mining. applications and theoretical aspects. ICDM 2014. Lecture Notes in Computer Science. vol. 8557. Cham: Springer; 2014.
[2] Lee R, Luo T, Huai Y, Wang F, He Y, Zhang X. Ysmart: Yet another SQL-to-mapreduce translator. IEEE International conference on distributed computing systems (ICDCS). 2011. p. 25–36.
[3] Gál Z, Varga I, Tajti T, Kocsis G, Langmajer Z, Kosa M, Panovics J. Performance evaluation of massively parallel communication sessions. In: Iványi P, Topping BHV, editors. Proceedings of the sixth international conference on parallel, distributed, GPU and cloud computing for engineering. Stirlingshire, UK: Civil-Comp Press; 2019. p. 34. https://doi.org/10.4203/ccp.112.34.
[4] Bagnulo M. Threat analysis for TCP extensions for multipath operation with multiple addresses. RFC 6181. INTERNET STANDARD; 2011.
[5] Ford A. Architectural guidelines for multipath TCP development. RFC 6182. INTERNET STANDARD; 2011.
[6] Huston G.. TCP and BBR, RIPE 76 meeting. 2018. https://ripe76.ripe.net/presentations/10-2018-05-15-bbr.pdf(last visited 08.11.2019).
[7] Cardwell N, Cheng Y, Gunn CS, Yeganeh SH, Jacobson V. BBR: congestion-based congestion control - measuring bottleneck bandwidth and round-trip propagation time. ACMQueue Netw 2016;14:5.
[8] Allman M, Paxson V, Blanton E. TCP congestion control. RFC 5681. 2009.

[9] Floyd S. Congestion control principle. ser RFC2914. Internet Engineering TaskForce (IETF); 2000.
[10] Xu L, Harfoush K, Rhee I. Binary increase congestion control for fast, long distance networks. IEEE INFOCOM. 2004.
[11] Hayes DA, Armitage G. Revisiting TCP congestion control using delay gradients. IFIP Networking. Springer; 2011. p. 328–41.
[12] CUBIC T. A transport protocol for improving the performance of TCP in long distance high bandwidth cyber-physical systems. IEEE International conference on communications workshops (ICC Workshops). 2018.
[13] Alizadeh M, Greenberg A, Maltz DA, Padhye J, Patel P, Prabhakar B, Sengupta S, Sridharan M. Data center TCP (DCTCP). Proc. ACM SIGCOMM, New Delhi. Data Center Networks session; 2010.
[14] Alizadeh M, Javanmard A, Prabhakar B. Analysis of DCTCP: stability, convergence, and fairness. Proc ACM SIGMETRICS, San Jose. 2011.
[15] Floyd S. Highspeed TCP for large congestion windows. RFC. INTERNET STANDARD; 2003.
[16] Floyd S, Ratnasamy S, Shenker S. Modifying TCP's congestion control for high speeds. Technical note. 2002.
[17] Shorten RN, Leith DJ. H-TCP: TCP for high-speed and long-distance networks. Proc PFLDnet, Argonne. 2004.
[18] Caini C, Firrincieli R. TCP-hybla: a TCP enhancement for heterogeneous networks. Int J Satell Commun 2004.
[19] Liu S, Basar T, Srikant R. TCP-illinois: a loss and delay-based congestion control algorithm for high-speed networks. ScienceDirect Perform Eval 2008;65:417–40.
[20] Kuzmanovic A, Knightly EW. TCP-LP: a distributed algorithm for low priority data transfer. IEEE INFOCOM. 2003.
[21] Fu CP, Liew SC. TCP veno: TCP enhancement for transmission over wireless access networks. IEEE J Sel Areas Commun 2003.
[22] Mascolo S, Casetti CE, Gerla M, Sanadidi MY, Wang R. TCP westwood: Bandwidth estimation for enhanced transport over wireless links. MobiCom; 2001.
[23] Baiocchi A, Castellani AP, Vacirca F. YeAH-TCP: Yet another highspeed TCP. CiteSeerX; 2008.
[24] Meister BW, Janson PA, Svobodova L. Connection-oriented versus connectionless protocols: a performance study. IEEE Trans Comput 1985:1164–73.C34/12
[25] Postel J. ISI, user datagram protocol. RFC 768. INTERNET STANDARD; 1980.
[26] Liu X.. 2019. https://github.com/xinan/ReliableFileTransferProtocol/tree/master/src(last visited 08.11).