Overhead Estimation of Selected Protocols for File Transfer

Muhammad Muhammad
Deutsches Zentrum für Luft- und Raumfahrt (DLR)
Münchenerstr. 20, 82234 Wessling, Germany
Email: Muhammad.Muhammad@dlr.de

Matteo Berioli
Deutsches Zentrum für Luft- und Raumfahrt (DLR)
Münchenerstr. 20, 82234 Wessling, Germany
Email: Matteo.Berioli@dlr.de

***Abstract*—In this paper, we derive an analytical model for the estimation of the overhead introduced in a file transfer procedure by three different reliable protocols, namely TCP, SCTP, and Saratoga. The model estimates the introduced overhead down to the IP layer, taking into account uncorrelated packet loss on the channel. The model distinguishes between the overhead on the forward link (server to client), mainly due to packet headers and retransmissions, from the overhead on the return link (client to server), which is mainly due to acknowledgments. By carrying out experimental tests, we show that the model predictions match very well with the trial results. Further, Saratoga shows the lowest overhead costs, especially for the smaller files sizes; nevertheless, on the expense of removing mechanisms like flow control and congestion control.**

I. Introduction

Transmission Control Protocol (TCP) [1] is the Transport-layer (TL) protocol extensively used by many of the Internet's most popular applications like Hypertext Transfer Protocol (HTTP) and File Transfer Protocol (FTP) . It gains its fame from being the first end-to-end TL protocol that provides reliable, ordered delivery of a stream of bytes from a program on one computer to another, over a network.

Stream Control Transmission Protocol (SCTP) [4] was originally designed to send telephony signaling [2]. As a TL protocol, it solved a number of TCP limitations while borrowing extra beneficial features from the User Datagram Protocol (UDP) [3]. SCTP provides features for high availability, increased reliability, and improved security for socket initiation.

Lastly, with the aim to transfer remote-sensing imagery from a Low Earth Orbiting (LEO) satellite constellation, Saratoga [5] was developed. Saratoga is intended for use when moving files between peers, with high throughput [6], which may have intermittent connectivity and can simply cope with highly asymmetrical links.

What unites the above mentioned protocols, regarding the topic of this work, is their ability to overcome transmission errors, occurring on the channel, leading to reliable transmission of files.

As specified in literature, file transfer is a generic term for transmitting files over a computer network or the Internet. Its applications have the client-server architecture. There are 2 types of file transfers:

1) Pull-based file transfers where the receiver initiates a file transmission request, and
2) Push-based file transfers where the sender initiates a file transmission request

When transmission capacity and time are scarce and costly, they have to be used efficiently; e.g. in satellite communications, where the return link resources are limited, or in LEO satellite communications, where the short visibility time of the satellite (almost 10 minutes) has to be used in the most efficient way to download as many data as possible from the satellite. For this reason, the overhead introduced by the protocols used for a file transfer has to be considered carefully. In particular, when considering the transmission of small-sized files that we believe they match aeronautical communication messages (e.g. Air Traffic Services (ATS) and Aeronautical Operational Control (AOC) messages).

To the best of our knowledge, no work has been done to carefully estimate the overhead introduced by the protocols used for file transfer. So, the objective of this paper is to estimate the overhead introduced by a file transfer application for the three above stated protocols (i.e. TCP, SCTP, and Saratoga).

For the aim of this work, we consider every packet header, non-data packet the COOKIE in SCTP, for example, or the non-data chunks such as control chunks, and any retransmitted packets (including payload) as an overhead.

In the next Section, we briefly introduce the protocols. In Section III, we present our analytical models that will be evaluated by obtaining experimental results using the testbed we setup in Section IV. Results are shown in Section V and Section VI concludes the paper.

II. Protocols

In the context of this work, we study three protocols that are considered fully reliable in the sense of complete delivery of the requested object from a file server to the calling endpoint, i.e. Pull-based file transfer, even when link errors exist. In this respect, we will consider packet-level errors, with given Packet Error Rate (PER).

TCP, SCTP and Saratoga share the above property. While TCP is vastly used, SCTP still needs sometime to find its place in the market to grow-up, and finally, despite the fact that

Saratoga is not yet a public standard (only an Internet Draft), it is inaction since 2004, thus it is worth to be considered.

A. TCP

For being fully reliable, TCP as a TL protocol constructs for each transmitted packet a basic header of 20 bytes. In addition to that, some header options are used. A number of these options is used only once at the negotiation period before the connection establishment phase to signal some requirements to the other communicating end, while others, such as *Timestamps* are employed during the actual data transfer.

After setting up the connection between the client and the server, a file, or many, may be transferred between the peers. A TCP server implements flow control to send packets according to its sliding window size, which is updated based on the received acknowledgments (ACK). It also implements *delayed acknowledgment*, in such a way that in a stream of full-sized segments there should be an ACK for at least every second segment.

B. SCTP

Just like TCP, SCTP is full-duplex, reliable and offers in-sequence transport messages. Moreover, it honors message boundaries like UDP. In addition to that, SCTP supports Multi-streaming and Multi-homing which are beyond our interests in this paper.

SCTP makes use of chunks as containers to send information. For that, SCTP places data messages and control information into separate chunks (data chunks and control chunks), each identified by a chunk header. SCTP chunks are bundled into SCTP packets. The SCTP packet, which is transported by the Internet Protocol (IP), consists of a packet header (common header) of 12 bytes. SCTP control chunks, when necessary, followed by SCTP data chunks, when available.

To facilitate the acknowledgment of the received DATA packets, the SCTP receiver uses a Selective Acknowledgment (SACK) chunk to inform the file sender about the missed gaps. As stated in Section 4.2 in [7], the delayed acknowledgment algorithm is applied by SCTP; an ACK (in SCTP a SACK) should be generated for at least every second packet received. Besides, in [4] it is declared that an SCTP receiver can send additional SACKs to update the sender on the status of its receiving buffer.

C. Saratoga

Saratoga is a novel file transfer protocol *proposal* still in the Standardization process at the IETF, particularity as an Internet Draft within the Transport Area Working Group (TSVW). It is a reliable protocol because it moves files without any loss thanks to the Selective Negative Acknowledgment (SNACK) mechanism.

On the other hand, Saratoga is a *command-line* Application-layer protocol built on top of UDP, so it is not properly a TL protocol. The client sends a request (command) with the desired file path/name to the server, which replies by sending the file's content (DATA). Saratoga defines five packet types, namely; *BEACON, REQUEST, METADATA, DATA* and *HOLESTOFILL* of different headers sizes.

Saratoga uses a *BEACON* to identify the sending peer. Any node can make a *REQUEST* for a file from any other endpoint. If the file exists, the file server sends a *METADATA* containing the file's properties, and then it waits for a *HOLESTOFILL* packet to begin the actual *DATA* packets transfer. Once finished, the receiver sends another *HOLESTOFILL* informing the server about the missing packets that should be retransmitted.

More to the point, a file-receiver can send *HOLESTOFILL* packets in two ways: either on the other peer's request, for instance as in the above scenario, or unsolicitedly. In this paper, we use the latter approach in order to make results comparable to the other two protocols. However, basic Saratoga uses only two *HOLESTOFILL* packets for any transaction, in order to avoid congestion on the return link and because it is designed to cope with highly asymmetrical links. Consequently, according to the first setting, the protocol will show a constant value of overhead from the receiver side in case of no errors.

III. Analytical Models

As pointed in Section II, each protocol has a different approach to follow for a file transfer operation. For instance, while we have to establish a connection in TCP or an association for SCTP before starting sending a file, in Saratoga we only need to send request for that file without any handshake or shutdown procedures, as for the other two protocols. These same protocols also behave differently on the receiving side.

The MTU of the Ethernet technology is 1500 bytes. Layer 3, when using IPv4, uses 20 bytes of header size without options. These 1500 bytes are entirely filled when sending a data packet and partially loaded for transmitting handshake/shutdown messages or control information. In this Section, we build analytical models to estimate the overhead for each protocol defined above, for the Network, Transport and Application (in case of Saratoga) layers using Ethernet frames. In Section I, we defined the overhead as being any information byte on the transmission medium, due to layer-3 and higher layers protocols, excluding the actual non-retransmitted file-data. From this, we can describe the relative overhead of a transmitted file as:

$$\mathcal{H}_X = \mathcal{H}_X(\mathrm{OH}_X, F) = \frac{\mathrm{OH}_X}{\mathrm{OH}_X + F} \tag{1}$$

where, OH_X is the absolute overhead (the price in bytes) of the $_X$ protocol (including retransmissions) and F is the total file size in bytes. Table I gives a brief overview of the parameters used in the model.

The subscript or superscript of FW and RT, in Table I, denote the load on the Forward (server to client) and Return (client to server) links, respectively. Further, all headers sizes are measured in bytes.

To calculate the payload size (MSS) in a data packet for a

TABLE I
PARAMETERS FOR ANALYTICAL MODELS

Parameters	Description
ε	Error probability
MSS	Maximum Segment Size in bytes
$N_{\text{dpkt}} = \lceil \frac{F}{\text{MSS}} \rceil$	Number of data packets
H_x	Header size of an x-type packet
H_{IP}	Header size of the IP protocol
P_x	(H_x + H_{IP}) for the x-type packet
$(P_{\text{DATA}\,X})_i$	$(H_{\text{DATA}\,X} + H_{\text{IP}})$ is the i-th data packet in the transfer for protocol X
OH_X^{FW}, OH_X^{RT}	Header sizes (prices) of the X-protocol from server and client, respectively. [absolute overhead]
$S_{\text{HANDSHUT}}^{\text{FW}}$	The handshake/shutdown header size at the forward link
$S_{\text{HANDSHUT}}^{\text{RT}}$	The handshake/shutdown header size at the return link
L_{REP}	Length of the ack. information (report) about missing packets other than header in bytes

specific protocol X, we use:

$$\text{MSS}_X = \text{MTU} - (H_{\text{DATA}\,X} + H_{\text{IP}}) \tag{2}$$

where, X can be TCP, SCTP or Saratoga.

The general models to calculate the headers sizes on both FW and RT channels for the three selected protocols is:

$$\begin{aligned}\text{OH}_X^{\text{FW}} = S_{\text{HANDSHUT}}^{\text{FW}} + \sum_{i=1}^{N_{\text{dpkt}}} (P_{\text{DATA}\,X})_i \\ +(\lceil N_{\text{dpkt}} \cdot \varepsilon \rceil \cdot (P_{\text{DATA}\,X} + \text{MSS}_X))\end{aligned} \tag{3}$$

and

$$\begin{aligned}\text{OH}_X^{\text{RT}} = S_{\text{HANDSHUT}}^{\text{RT}} + (\lfloor N_{\text{dpkt}}/y \rfloor) \cdot P_x \\ +(\lceil N_{\text{dpkt}} \cdot \varepsilon \rceil \cdot (P_x + L_{\text{REP}}))\end{aligned} \tag{4}$$

where, MSS_X is calculated from (2) accordingly, and

$$\varepsilon = \begin{cases} 0, & \text{if } N_{\text{dpkt}} < \frac{1}{\text{PER}} \\ \text{PER}, & \text{otherwise} \end{cases}$$

TABLE II
PROTOCOLS PARAMETERS

Protocol (X)	x	y	L_{REP}
TCP	ACK	2	0
SCTP	SACK	1.5	4
SAR	HOLESTOFILL	2	8

PER is the packet error rate value on the link, and SAR stands for Saratoga. Table II shows the parameters x, y and L_{REP} to be used in (3) and (4) according to the protocol X. Further, a few points need to be highlighted concerning the models:

1) In (3) we use $(P_{\text{DATA}\,X})_i$, this is because it may happen that in the last data packet sent the payload size does not add up to the MSS value.
2) For the three protocols an acknowledgment to the file sender is sent every y-DATA packets. Since in TCP the receiver acknowledges every second received packet we exploit this in the RT model by having y = 2 and thus $N_{\text{dpkt}}/2$. However, an SCTP receiver responds not only with SACK packets for determining gaps but also with SACK packets for updating the receiver buffer for at least 1 SACK per transmitted packet, as in Section II-B, and that is reflected by setting y = 1.5 in (4) for X as SCTP. As for Saratoga, we consider a voluntary SNACK is sent every two data packets and this is shown by N_{dpkt}/y, where y = 2 for a fair comparison with the other two protocols.
3) Models in (3) and (4) compute the absolute overhead. Thus, in order to calculate the relative overhead from (1) for a particular link direction (i.e. FW or RT), we always refer to:

$$\mathcal{H}_X^{\text{FW}} = \mathcal{H}_X(\text{OH}_X^{\text{FW}}, F) \tag{5}$$

and

$$\mathcal{H}_X^{\text{RT}} = \mathcal{H}_X(\text{OH}_X^{\text{RT}}, F) \tag{6}$$

A. TCP Model

TCP connection initiation is not secure, thus headers are of small sizes. Table III shows the TCP handshake and shutdown headers sizes.

TABLE III
TCP PACKETS HEADERS SIZES

Packet header	Size [Bytes]
$H_{\text{SYN}}, H_{\text{SYN-ACK}}$	40
$H_{\text{ACK}}, H_{\text{FIN}}, H_{\text{DATA}\,\text{TCP}}$	32

From Table III and [1] we conclude that:
$S_{\text{HANDSHUT}}^{\text{FW}} = P_{\text{SYN-ACK}} + P_{\text{FIN}} + P_{\text{ACK}}$,
and
$S_{\text{HANDSHUT}}^{\text{RT}} = P_{\text{SYN}} + P_{\text{FIN}} + 2 \cdot P_{\text{ACK}}$

To be accurate in evaluating our models, we take the handshake and shutdown procedures into account. We also split the load between the two links for enhanced analysis. Thus, the relative TCP overhead $\mathcal{H}_{\text{TCP}}$ estimation values for both FW and RT are retrieved from (3) and (4), then from (5) and (6), respectively by setting X to TCP. MSS_{TCP} is automatically calculated from (2). Finally, we consider L_{REP} = 0 because TCP uses simple ACKs and it performs Head-of-line blocking; so if any segment is lost, TCP will hold up delivery of consequent bytes, thus there is no gaps. Nevertheless, the SACK option will be used once gaps or correlated error losses exist.

B. SCTP Model

SCTP protocol avoids the security problem; SYN-attack, observed in TCP, by adding a cookie mechanism to the initial handshake before establishing an association. Also, it prevents the half-closed state, as in TCP, when shutting down an association. Table IV shows the SCTP handshake and shutdown chunks headers sizes. It should be noted, that SCTP defines a set of several other chunks that are beyond purpose of this work.

The values in the first three rows of Table IV are variable due the setup of the machines, such as number of IP addresses used, etc..

As in TCP, SCTP [4] defines the chunks headers for initiating and shutting-down an association as the following:
$S_{\text{HANDSHUT}}^{\text{FW}} = P_{\text{INIT-ACK}} + P_{\text{COOK-ACK}} + P_{\text{SHUT}} + P_{\text{SHUT-COMP}}$,

TABLE IV
SCTP CHUNKS HEADERS SIZES

Packet header	Size [Bytes]
H_{INIT}	60 + 12
$H_{\text{INIT-ACK}}$	316 + 12
$H_{\text{COOK-ECHO}}$	264 + 12
$H_{\text{DATA}_{\text{SCTP}}}, H_{\text{SACK}}$	16 + 12
H_{SHUT}	8 + 12
$H_{\text{COOK-ACK}}, H_{\text{SHUT-COMP}}, H_{\text{SHUT-ACK}}$	4 + 12

and
$S^{\text{RT}}_{\text{HANDSHUT}} = P_{\text{INIT}} + P_{\text{COOK-ECHO}} + P_{\text{SHUT-ACK}}$
Not forgetting to mention that each SCTP packet header H_x includes the Common header of 12 bytes, as explained in Section II-B.

The same process, as in Section III-A, can be used for computing the SCTP FW and RT channels absolute overhead $\mathcal{H}$ values, and that is done by replacing $_X$ in (3) and (4) with SCTP. Then, we refer to (5) and (6) to compute the relative overhead $\mathcal{H}_{\text{SCTP}}$.

Finally, a L_{REP} of 4 bytes only is considered because we assume that for every missed data packet, the SACK will report only one gap-start and one gap-end, each of 2 bytes, as described in [4] and [7].

C. Saratoga Model

Saratoga differs from TCP and SCTP in the way of transferring a file. Saratoga does not maintain a connection or association with the other peer. Saratoga is built on UDP, thus for every packet we have to add a UDP header size of 8 bytes. Table V shows the packets header sizes needed for a file transfer using Saratoga.

TABLE V
SARATOGA PACKETS HEADER SIZES

Packet header	Size [Bytes]
H_{BEACON}	8
H_{REQUEST}	9 + PATH
H_{METADATA}	21 + PATH
$H_{\text{DATA}_{\text{SAR}}}$	16
$H_{\text{HOLESTOFILL}}$	20

PATH is the file path on the server peer, here we consider it 0 bytes since, we assume that the server knows which file to send for every request.

Although Saratoga has no connection handshake and shutdown procedures, we use the same conventions as previously, to avoid confusion and to split data transfer from other actions, as well. Leading to:
$S^{\text{FW}}_{\text{HANDSHUT}} = P_{\text{BEACON}} + P_{\text{METADATA}},$
and
$S^{\text{RT}}_{\text{HANDSHUT}} = P_{\text{BEACON}} + P_{\text{REQUEST}}$
Following the methods as before, the estimation values for absolute Saratoga overhead on both FW and RT links are determined by setting $_X$ to SAR in (3) and (4), respectively. Then, with (5) and (6), we can calculate the relative overhead $\mathcal{H}_{\text{SAR}}$.

The L_{REP} is considered only 8 bytes because the SNACK in Saratoga uses 32 bits (4 bytes) descriptor to describe a hole-begin and another 4 bytes for a hole-end description; bearing in mind that within Saratoga a 32 bits descriptor allows us to send files as big as 4 Gbytes.

IV. Testbed

With the intention to study the overhead accumulated while transferring a file between two entities. We built a small testbed, as shown in Fig. 1, to do our experimental analysis.

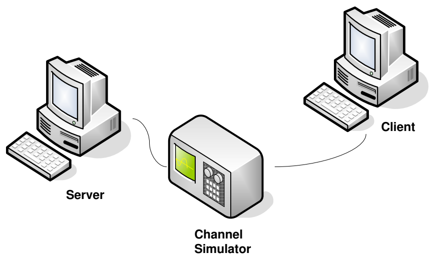


Fig. 1. Testbed used to transfer files

Between the client and the server we use a channel emulator that can add a delay for each packet, we use a constant value of 10 milliseconds (ms) as a round-trip time (RTT). Since TCP performs poorly on longer delay links, we use short RTT to keep focus on the overhead evolution for our study. In addition to that this channel simulator can drop uncorrelated IP packets randomly according to the PER given. If, for example, we set the PER to 10^{-3} then, on average 1 out of (1/PER) 1000 packets will be randomly dropped. We ran simulations with various PER values, ranging from 10^{-7} to 10^{-2}.

The client and the server are running Linux as an operating system, distribution of Opensuse 11.1 [8]. Capturing the packets for later analysis, at the server side, is done using Wireshark [9], to check the server behavior. Files of different sizes have been transfered, ranging from 10 Kbytes to 5 Mbytes.

The server and the client are running a TCP/SCTP server and a TCP/SCTP client, respectively. The client requests the file from the server which will respond by sending the required object in terms of (TCP or SCTP)/IP packets. At this point in time, we do not have a running Saratoga implementation for performing experimental assessments for it.

For the TCP we use the Linux kernel 2.6.28 build-in protocol stack with Cubic [11] congestion control algorithm. Please, keep in mind that the behavior of the congestion control mechanism is out of the scope of this work. On the other hand, for SCTP we use the Linux Kernel SCTP (lksctp-tools-1.0.10) [10] implementation.

V. Results

To validate our models, we run some tests using our testbed from Section IV. Each file, from the set of files we have at our server, is transferred to the client upon a request after the connection establishment, afterwards the connection is closed. Here, we assume that the server knows which file to send upon each request. Simultaneously, we are capturing all the packets at the server side for the analysis afterwards. The captures are split between server load and client load, respectively.

Figures 2 and 3 show the comparison of the analytical model from (5) and (6), for all protocols i.e. TCP, SCTP and Saratoga, to real tests without and with link errors, respectively.

As it can be clearly seen in both Fig. 2(a) and 2(b), where no link errors are placed on the link, the overhead decreases as the file size increases. Also, the experimental tests results (shown in discrete squares and circles) for both TCP and SCTP are well fitting with the TCP (lined) and SCTP (dashed) analytical models. In addition to that, SCTP shows higher overhead for small sized files i.e. less than 1 Mbytes, than both TCP and Saratoga model (dash-dot), this is due to the large S_{HANDSHUT} on FW channel. However, this behavior changes for medium and large sized files, where it shows lower overhead than TCP but still higher than Saratoga. On the other hand, in Fig. 2(b), SCTP shows the highest overhead due to the additional SACKs for receiver buffer size update. Finally, Saratoga shows the lowest overhead values for all files sizes on both FW and RT channels. Please, recall that for Saratoga we made y = 2 on the RT link i.e. we send a SNACK every second packet in order to make the results comparable to those of TCP and SCTP, as explained in Section II-C.

Fig. 3, demonstrates the models approximation to the tests results with PER of 10^{-2}. Fig. 3(a) and 3(b) both show that Saratoga exhibits the lowest overhead among all files tested, while SCTP exhibits the same behavior as when errors do not exist. What is interesting to focus on is the sharp edge in Fig. 3(a), in simple words; on PER = 10^{-2} we have to drop at least 1 packet from every 100. Having this in mind, this edge happens at the file that needs a minimum of 100 packet (i.e. file size $\cong$ 150 KBytes) to be transferred, and later the overhead decreases as expected.

Fig. 3(a) shows that the overhead increases for a given PER value. Complementary to this, Fig. 4(a) illustrates that for different files sizes and a specific protocol the overhead does not change until a certain PER value revealing the number of packets to be transmitted. After that, the overhead increases with the PER that is represented on the X-axis.

The results in Fig. 4(a) conform with the previous ones in such a way that as the PER increases on the FW link, the overhead of Saratoga keeps its lowest value among the other protocols while SCTP is positioned as the second lowest overhead value.

However, on the RT channel due to the receiver buffer update SACKs, SCTP shows the highest overhead cost while Saratoga keeps its lowest values. A look at Fig. 4(b), will show that as a result of small reports, held by different acknowledgments mechanisms, even with high PER values overhead keeps almost a constant value.

VI. Conclusion

In this paper we present analytical model for three protocols (TCP, SCTP and Saratoga) to estimate the overhead due to headers, control information and retransmissions produced while transferring a file from a server to a client taking the handshake and the shutdown procedures into account for more precise results. We show that the analytical model derived can reliably predict the experimental results.

Into the bargain, we can say that the intention behind Saratoga design was to focus on high link utilization while neglecting the congestion and flow control mechanisms, since it is believed that these two algorithms limit the throughput of an application as observed within the other protocols. However, Saratoga achieved its goal and as a plus it kept the lowest overhead among the three protocols for all files sizes. On the other hand, TCP and SCTP showed fine results in terms of overhead although they implement the above control mechanisms that bound their good-put. From this perspective, we showed that a protocol like Saratoga is able to guarantee reduced overhead and improved throughput compared to other file transfer protocols like SCTP and TCP. But, this is achieved at the price of removing congestion and flow control that could be worthy and less tedious for small files transfer.

References

[1] Information Science Institute - University of Southern California, *Transmission Control Protocol*, IETF RFC 793, Sep, 1981.

[2] R. Stewart, Q. Xie, K. Morneault, C. Sharp, H. Schwarzbauer, T. Taylor, I. Rytina, M. Kalla, L. Zhang, V. Paxson, *Stream Control Transmission Protocol*, IETF RFC 2690, Oct, 2000.

[3] J. Postel, *User Datagram Protocol*, IETF RFC 768, Aug, 1980.

[4] R. Stewart, Ed., *Stream Control Transmission Protocol*, IETF RFC 4960, Sep, 2007.

[5] L. Wood, J. McKim, W. Eddy, W. Ivancic, C. Jackson, *Saratoga: A Scalable File Transfer Protocol*, IETF Internet Draft draft-wood-tsvwg-saratoga-04.

[6] L. Wood, W. Eddy, W. Ivancic, J. McKim, C. Jackson, *Saratoga: a Delay-Tolerant Networking convergence layer with efficient link utilization*, International Workshop on Space and Satellite Communications (IWSSC '07), Salzburg, Austria, 13-14 September 2007.

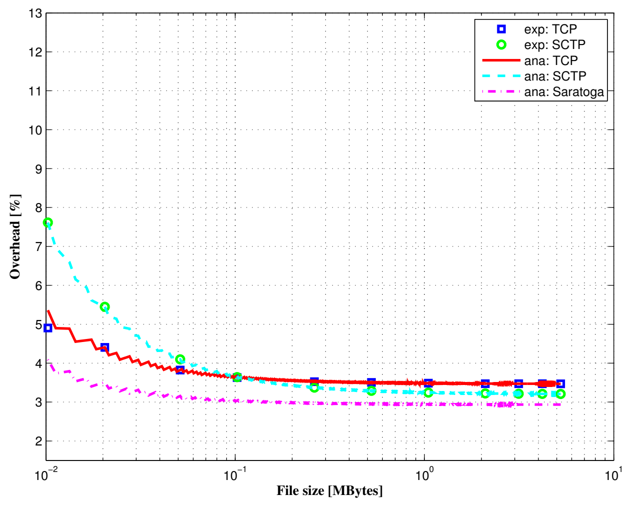
[7] M. Allman, V. Paxson, W. Stevens, *TCP Congestion Control*, IETF RFC 2581, Apr, 1999.

[8] Novell, *Opensource Operating System*, http://www.opensuse.org/.

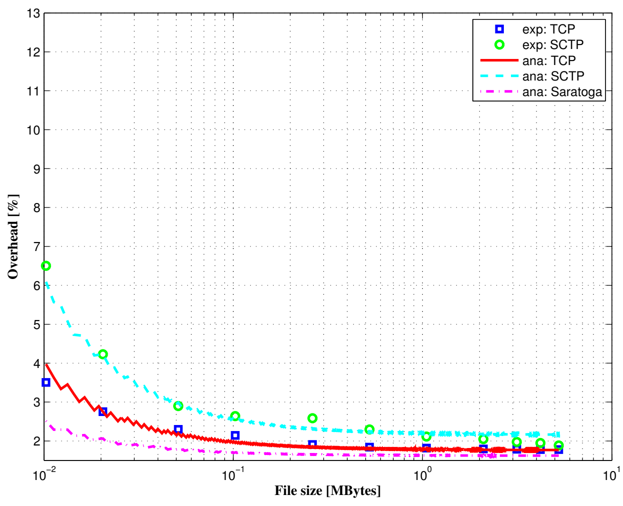
[9] Wireshark, *Network Protocol Analyzer*, http://www.wireshark.org/.

[10] IBM Corp., *Linux Kernel SCTP*, lksctp.sourceforge.net.

[11] L. Xu and I. Rhee., *CUBIC: A new TCP-Friendly high-speed TCP variant*, In Proc. Workshop on Protocols for Fast Long Distance Networks, 2005, 2005.

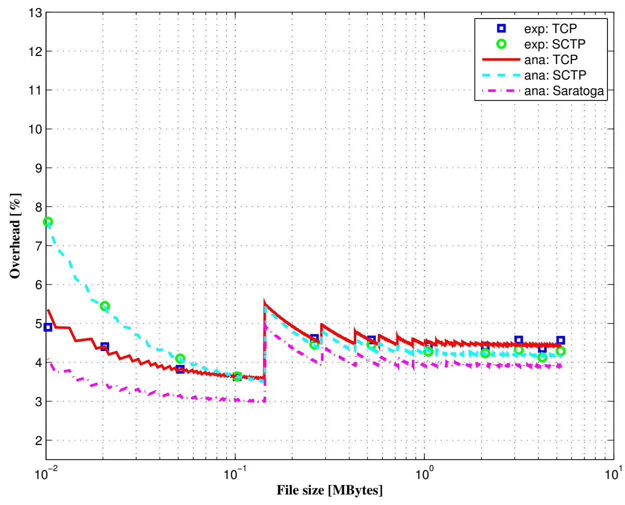


(a) FW link

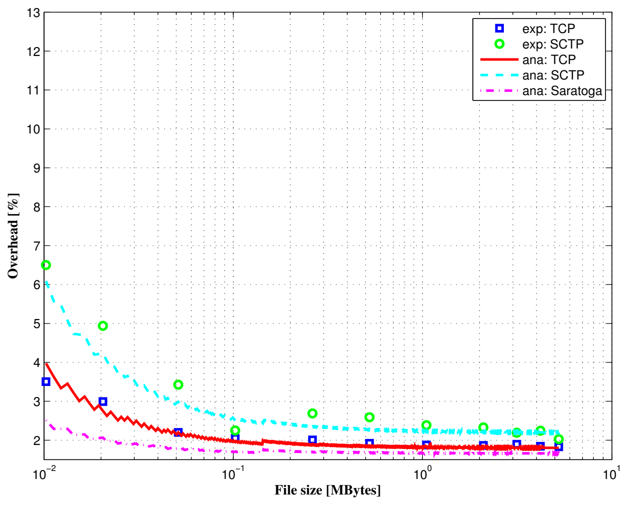


(b) RT link

Fig. 2. Testbed results vs. Analytical model approximation with No Errors

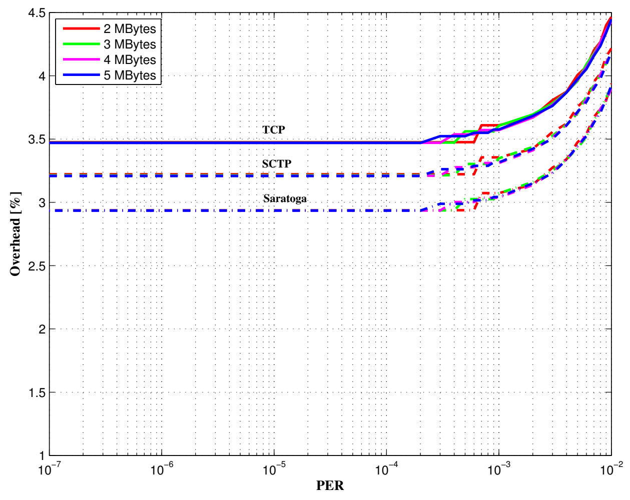


(a) FW link

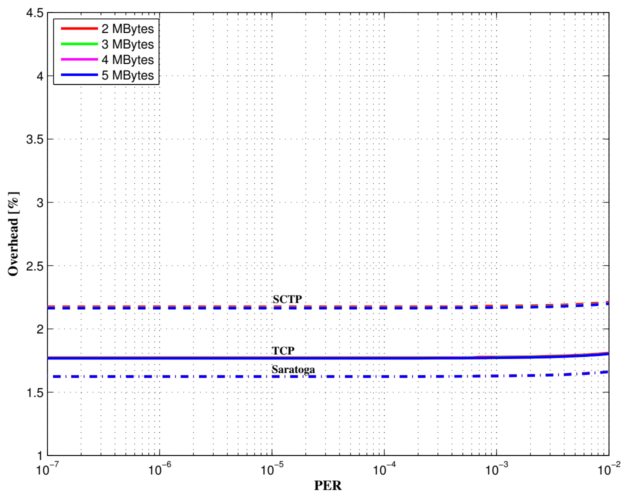


(b) RT link

Fig. 3. Testbed results vs. Analytical model approximation with PER of 10^{-2}



(a) FW link



(b) RT link

Fig. 4. Overhead increases with PER