Lab 5

Fast, Reliable File Transfer

Team Polaris

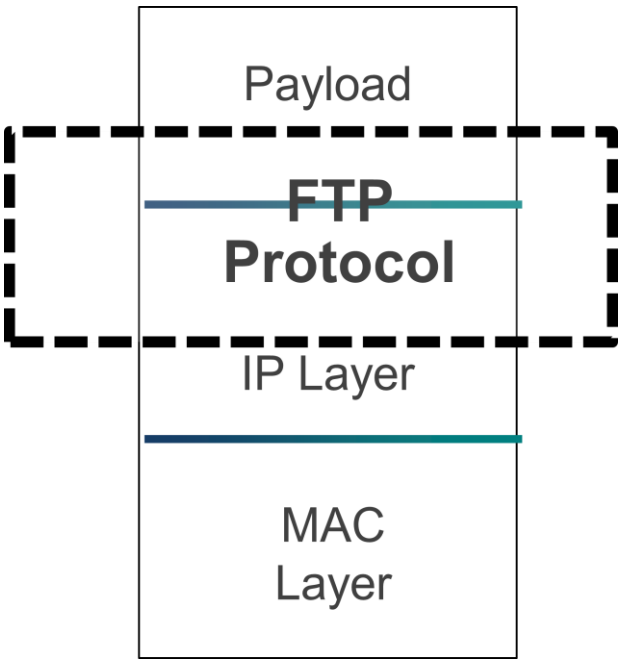
Rohith Narasimhamurthy
Thai Shankar Shanmugha Sundaram
Ullas Simhan Mandayam Nyayachavadi

Transfer Utility Implementation

- The reliable file transfer is achieved through a server-client model, with the server/sender initially breaking down the file into chunks which goes into the packet as the payload.
- Each segment of the file that is sent to the receiver is sequenced.
- The transfer utility does not use TCP/UDP sockets but instead builds the packets and frames using PCAP library. This reduces up to 20 bytes of header overhead and gives control over IP fields.
- We reuse the IP Identification field to maintain sequence numbers and calculate size using IP length field. ID field as per rfc, allows re-use even through routers. Thus we achieve reduction in overhead of up to 20 bytes in the payload for every packet.
- The channel is probed to get an estimate of loss/delay and the protocol dynamically adjusts and switches to the best way to send ACK back.
- The server maintains the state of ACK'd/not-ACK'd packets using a hash map.
  Retransmissions are done for all unacknowledged packets by checking if hash exists or not for individual packets. When ACKs are received, entries are made in the hash map to convey the receipt.
- To indicate to the file transfer has been completed, the sender sends a unique sequence number and "NULL" written to the payload.

We tried different acknowledgement algorithms such as flooding multiple acknowledgements into the channel, probing the channel and progressive acknowledgements. We found that probing the channel worked the best over wide range of lossy/delay scenarios.

Protocol Stack



Our Protocol works on the IP stack and the payload. Sequence numbers and length of each segment has been moved to IP ID and length fields.

ANALYSIS

All the observed results in terms of

1. Throughput vs delay

2. Throughput vs loss

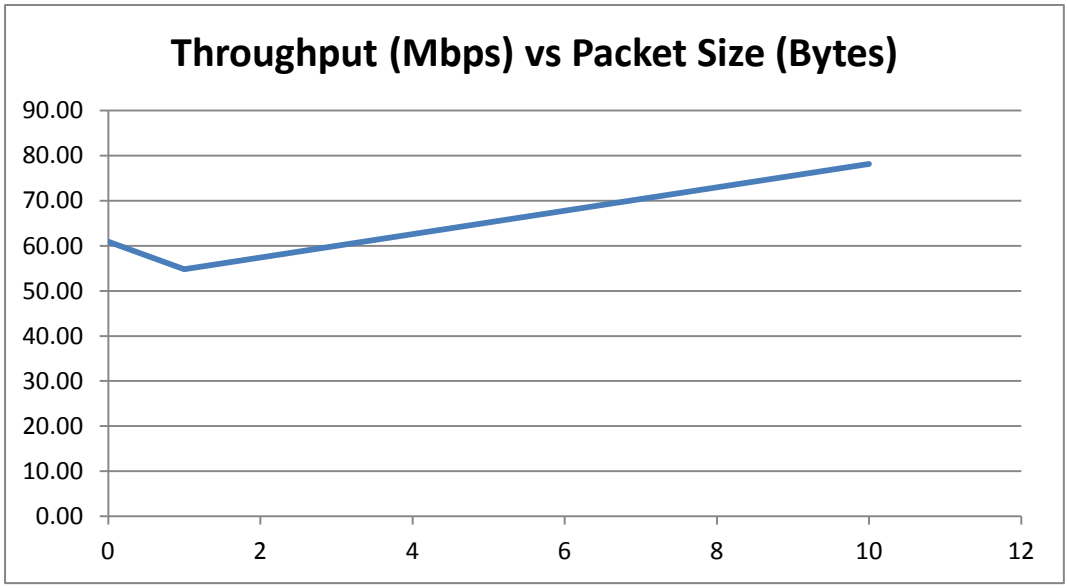
3. Throughput vs packet size / filesize are tabulated below:

Results of Flooding

Throughput vs Packet Size Analysis

For File Size : 1073741824

Packet Size (Bytes)	Transfer Time (seconds)	Throughput (Mbps)
1000	242.88	70.73
1200	230.69	74.47
1452	**228.70**	**75.12**

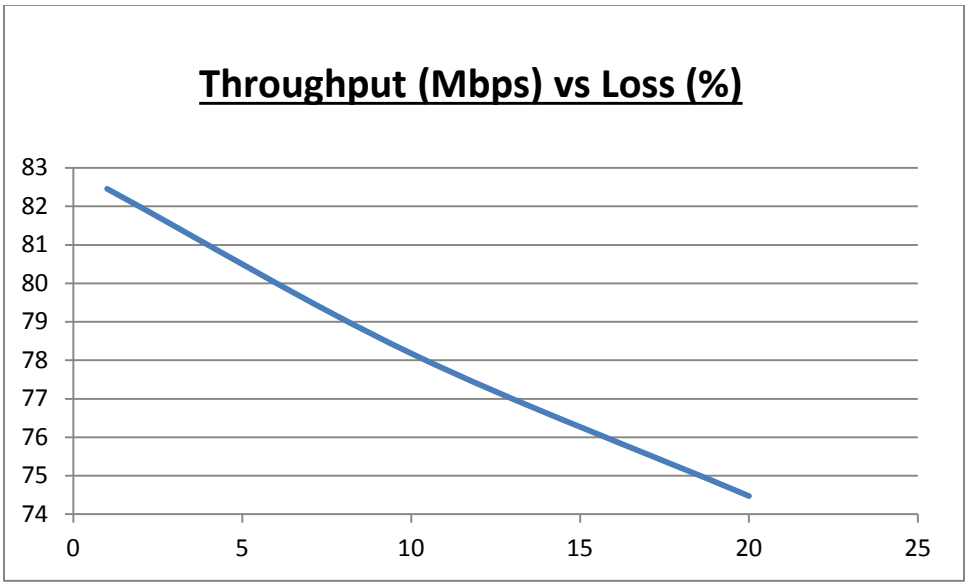


Increasing the packet size at a fixed loss rate increased the throughput. Hence we moved the overheads from the payload to the IP fields

Throughput (Mbps) vs Loss (%)

For File Size : 1073741824, Packet Size: 1200 (Bytes)

Loss (%)	Transfer Time (seconds)	Throughput (Mbps)
1	**208.35**	**82.45**
10	219.75	78.18
20	230.69	74.47

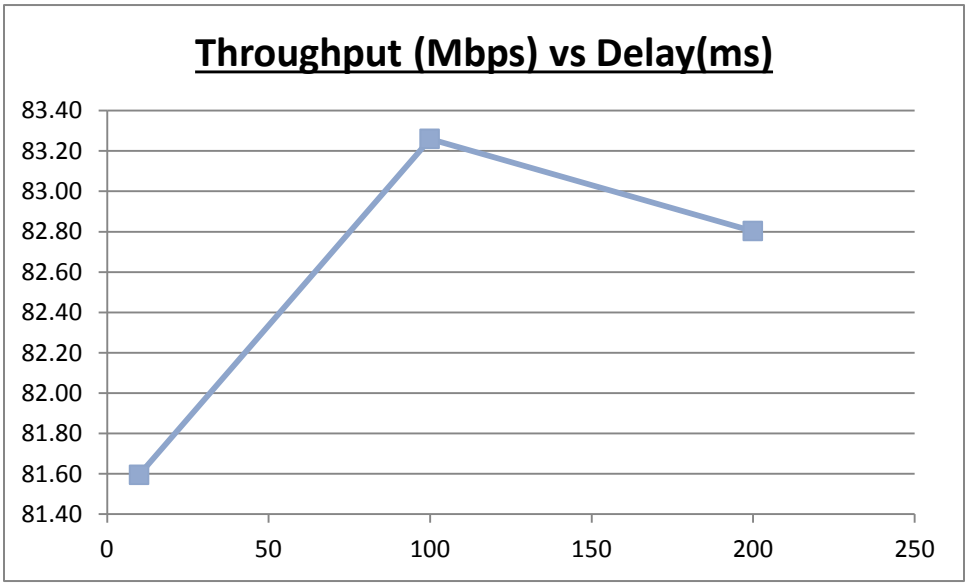


Flooding is inefficient when loss in the link is low. It takes up the bandwidth for communication the same information over and over. Our algorithm probes the channel by sending 50 packets and observes the throughput to decide on the number of acknowledgements to be flooded subsequently.

Throughput vs Delay Analysis

For File Size : 1073741824, Packet Size: 1452 (Bytes)

One Way Delay (ms)	Transfer Time (seconds)	Throughput (Mbps)
10	210.55	70.73
100	206.34	74.47
200	**207.48**	**75.12**

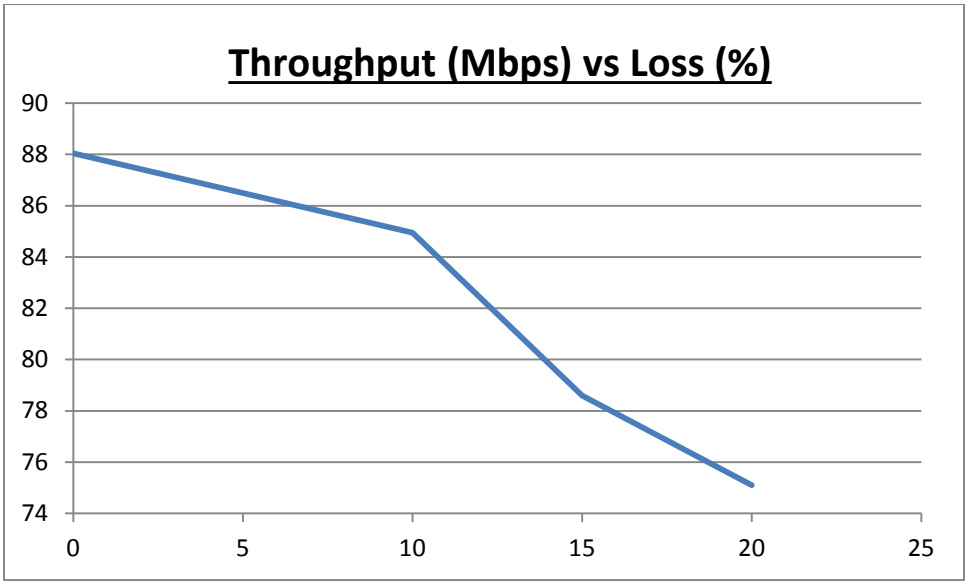


Results of Probing

Throughput (Mbps) vs Loss (%)

For File Size : 1073741824, Packet Size: 1452 (Bytes)

Loss (%)	Transfer Time (seconds)	Throughput (Mbps)
0	**195.12**	**88.04**
10	202.23	84.95
15	218.58	78.59
20	228.75	75.10

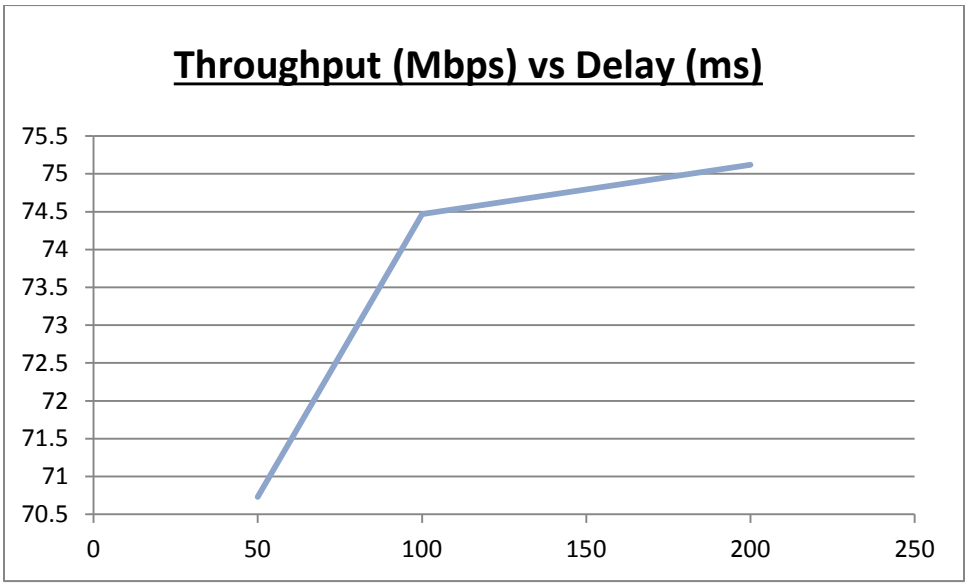


By probing the link before flooding the acknowledgements, we were able to increase the throughput from 78Mbps without probing to 84Mbps with probing (both measurements with 10%)

Throughput vs Delay Analysis

For File Size : 1073741824, Packet Size: 1452 (Bytes), Loss 20%

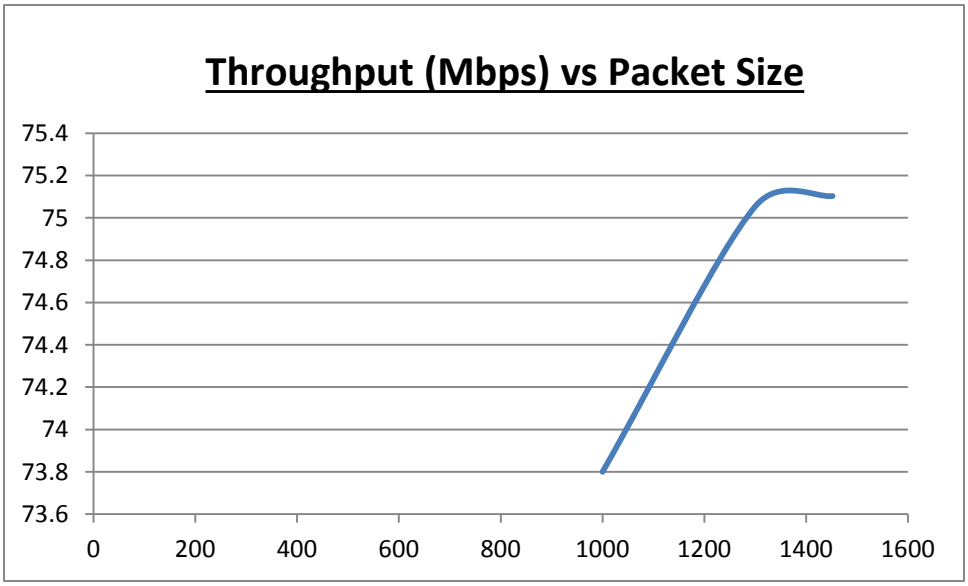
One Way Delay (ms)	Transfer Time (seconds)	Throughput (Mbps)
50	230.92	70.73
100	227.66	74.47
200	**228.75**	**75.12**



Throughput vs Packet Size Analysis

For File Size : 1073741824

Packet Size (Bytes)	Transfer Time (seconds)	Throughput (Mbps)
1000	232.79	75.76
1300	228.90	75.05
1452	**228.75**	**75.10**



CHALLENGES FACED

- Pcap_handle close was not flushing out packets when transmitting the file the 2nd time. This used to happen at random times and the received file hash failed sometimes because of some extra writes. This is an inherent problem with lib_pcap library and support for this is only available on win_pcap version.
- With the hint given by TA, we flushed out the buffers when transmitting the file back and opened a new handle again and got the correct hash.

ENHANCEMENTS

- Setting core affinities on a Multi-core processor to parallelize different parts of the program