Ch 14 Understanding Transport Protocols

Magda El Zarki
Prof. of CS
Univ. of CA, Irvine
Email:elzarki@uci.edu
http://www.ics.uci.edu/~magda

Overview

- The most common end-to-end transport protocols today are:
  - Transmission Control Protocol (TCP)
  - User Datagram Protocol (UDP)
- TCP is the prime choice for applications that need
  - reliability and in-order delivery of data
  - provides congestion control and emphasizes fairness and sharing of resources
- UDP is common choice for
  - time-dependent applications with no need for reliability
  - exercises no control over flows and as such is blocked by some firewalls

New Protocols and Services

- Protocols that seek to extend the range of services and versatility of the transport layer:
  - Stream Control Transmission Protocol (SCTP) – developed to transport SIP
  - Datagram Congestion Control Protocol (DCCP) – TCP like congestion control, no re-ordering or reliability
  - Game Transport Protocol – very similar to TCP, with some minor modifications and QOS bits added for traffic classes.
- Application level frameworks that use UDP for low latency but provide the reliability and other functionality lacking in UDP are:
  - Enet – goal to provide a flexible, minimalist framework to add functionality to UDP for apps that need low latency and some of the features that TCP has to offer such as reliability.
  - UDP-based data transfer (UDT), specifically designed for high speed nets

Thin Streams

- Characterized by
  - Small packet sizes
  - Low packet inter-arrival times
- Need low end to end latency and some (for a subset of the packets) reliability.
- TCP and most of its variants not suitable due to retransmission latencies.
- Use UDP but no reliability for any of the data and firewall issue forcing the apps to fall back on TCP.

Thin Stream Traffic C/Cs

application	payload size (bytes)			packet interarrival time (ms)				percentiles		avg bandwidth used	
	avg	min	max	avg	med	min	max	1%	99%	(pps)	(bps)
~~Casa (sensor network)~~	~~175~~	~~93~~	~~572~~	~~7287~~	~~307~~	~~305~~	~~29898~~	~~305~~	~~29898~~	~~0.137~~	~~269~~
Windows remote desktop	111	8	1417	318	159	1	12254	2	3892	3.145	4497
VNC (from client)	8	1	106	34	8	< 1	5451	< 1	517	29.412	17K
VNC (from server)	827	2	1448	38	< 1	< 1	3557	< 1	571	26.316	187K
Skype (2 users) (UDP)	111	11	316	30	24	< 1	20015	18	44	33.333	37K
Skype (2 users) (TCP)	236	14	1267	34	40	< 1	1671	4	80	29.412	69K
SSH text session	48	16	752	323	159	< 1	76610	32	3616	3.096	2825
Anarchy Online	98	8	1333	632	449	7	17032	83	4195	1.582	2168
World of Warcraft	26	6	1228	314	133	< 1	14855	< 1	3785	3.185	2046
Age of Conan	80	5	1460	86	57	< 1	1375	24	386	11.628	12K
BZFlag	30	4	1448	24	< 1	< 1	540	< 1	151	41.667	31K
Halo 3 - high intensity (UDP)	247	32	1264	36	33	< 1	1403	32	182	27.778	60K
Halo 3 - mod. intensity (UDP)	270	32	280	67	66	32	716	64	69	14.925	36K
World in Conflict (from server)	365	4	1361	104	100	< 1	315	< 1	300	9.615	31K
World in Conflict (from client)	4	4	113	105	100	16	1022	44	299	9.524	4443
YouTube stream	1446	112	1448	9	< 1	< 1	1335	< 1	127	111.111	1278K
HTTP download	1447	64	1448	< 1	< 1	< 1	186	< 1	8	> 1000	14M
FTP download	1447	40	1448	< 1	< 1	< 1	339	< 1	< 1	> 1000	82M

Problem Statement

- Reliable transport of thin streams with low latency requirement

- TCP with no congestion control -> 

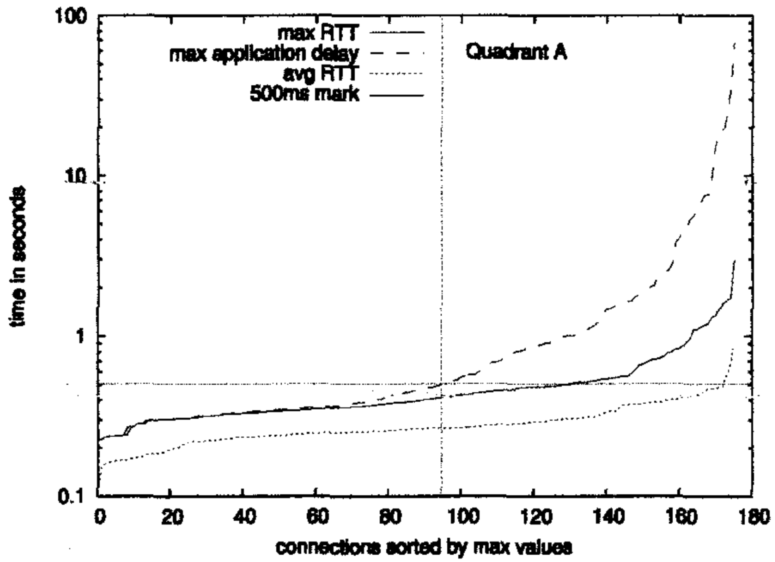
What we know:

- Thin streams are very often a product of time-dependent and/or interactive applications.
- Retransmission mechanisms and congestion control mechanisms have been developed to maximize throughput, and may therefore cause higher retransmission latency when the transported stream is thin – not greedy.

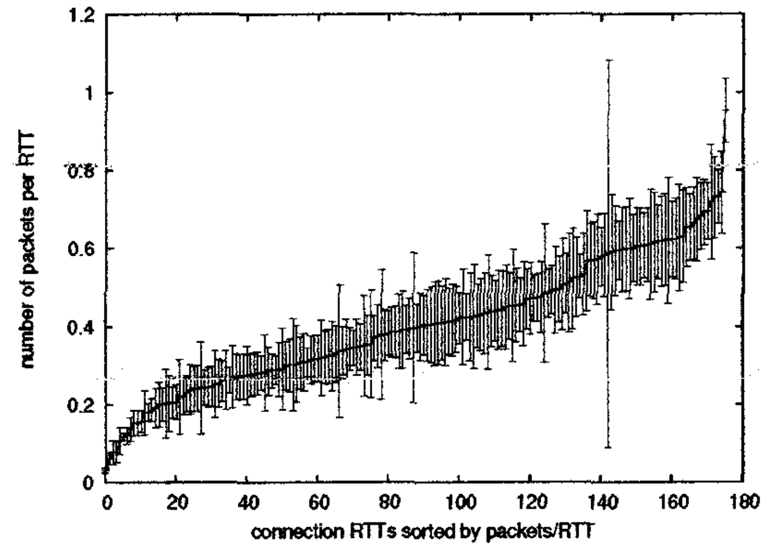
Goal:

- Adapt existing retransmission and congestion control mechanisms to achieve lower latency for thin streams without jeopardizing performance for greedy streams.
- Take advantage of the thin stream properties to achieve lower delivery latencies for the corresponding applications.
- Make modifications to improve thin-stream latency in such a way that unmodified receivers may benefit from them too.

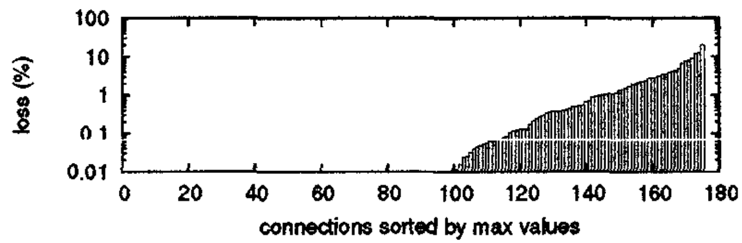
Latency Analysis of a Thin Stream (Anarchy Online Game)



(a) RTT versus maximum application delay.



(b) Packets per RTT with standard deviation



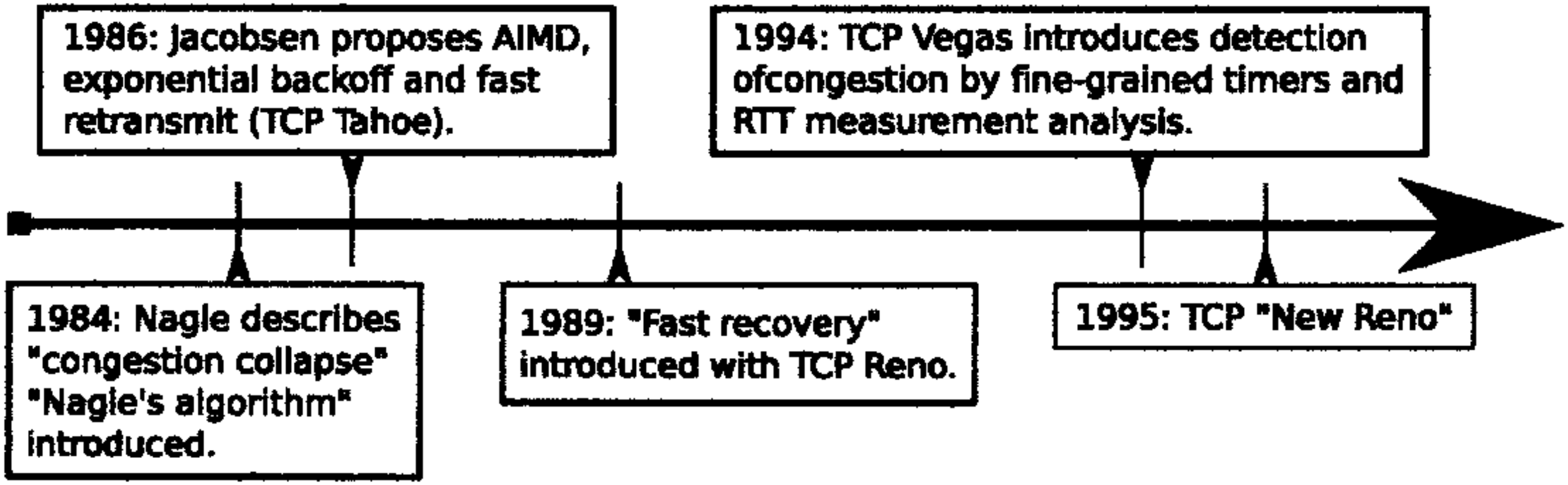
(c) Per-stream loss rate.

Choosing a transport protocol

- Use established transport protocols (like TCP) that provide a range of desirable services, but that can be modified to meet the low latency requirement.
- Use unreliable protocols (like UDP or DCCP) and implement reliability and in order delivery on the application layer. Problem with firewalls will not go away!
- Design new reliable protocol that is tailored for the needs of time-dependent applications - not a popular approach with commercial developers.
- Use of quality of service (QoS) options –not widely adopted by network providers

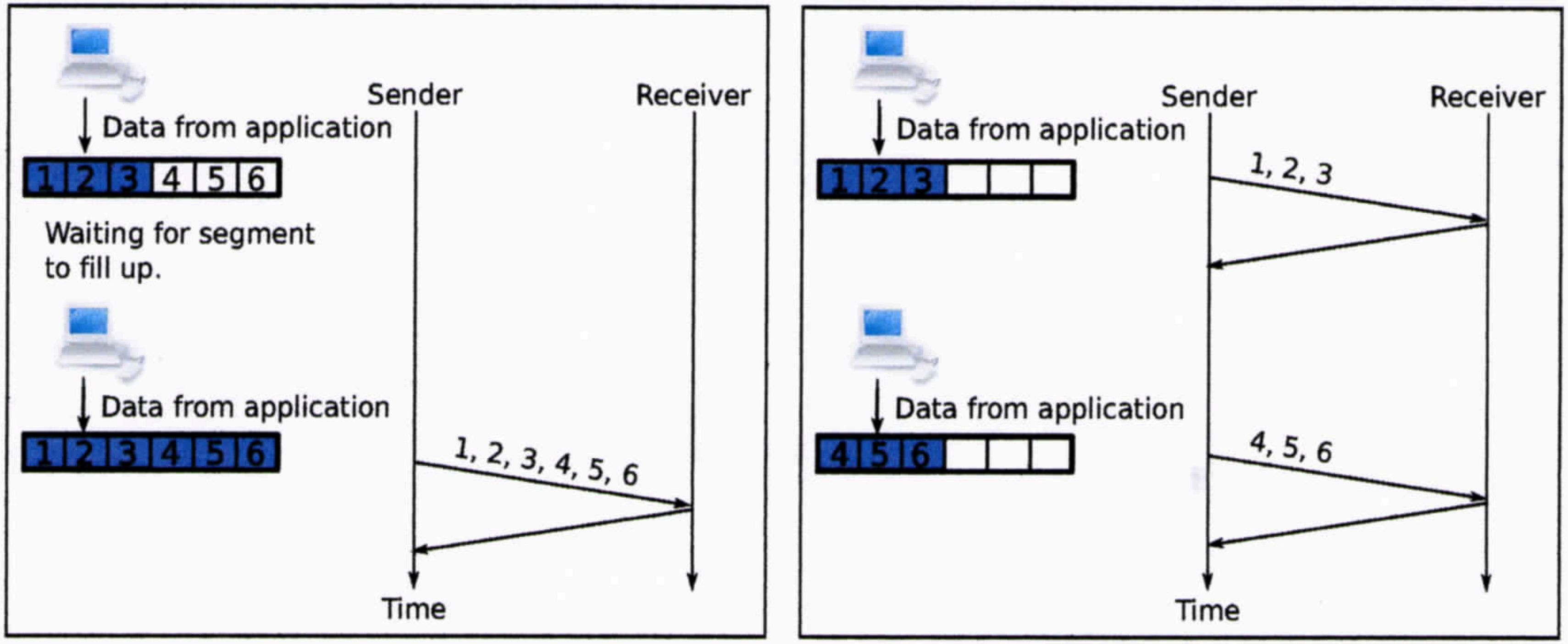
TCP Developments

- Timeline of TCP



Nagle's Algorithm – not suited for Thin Streams

- Aim to conserve bandwidth. Data only delayed if there are unACKed segments for the connection

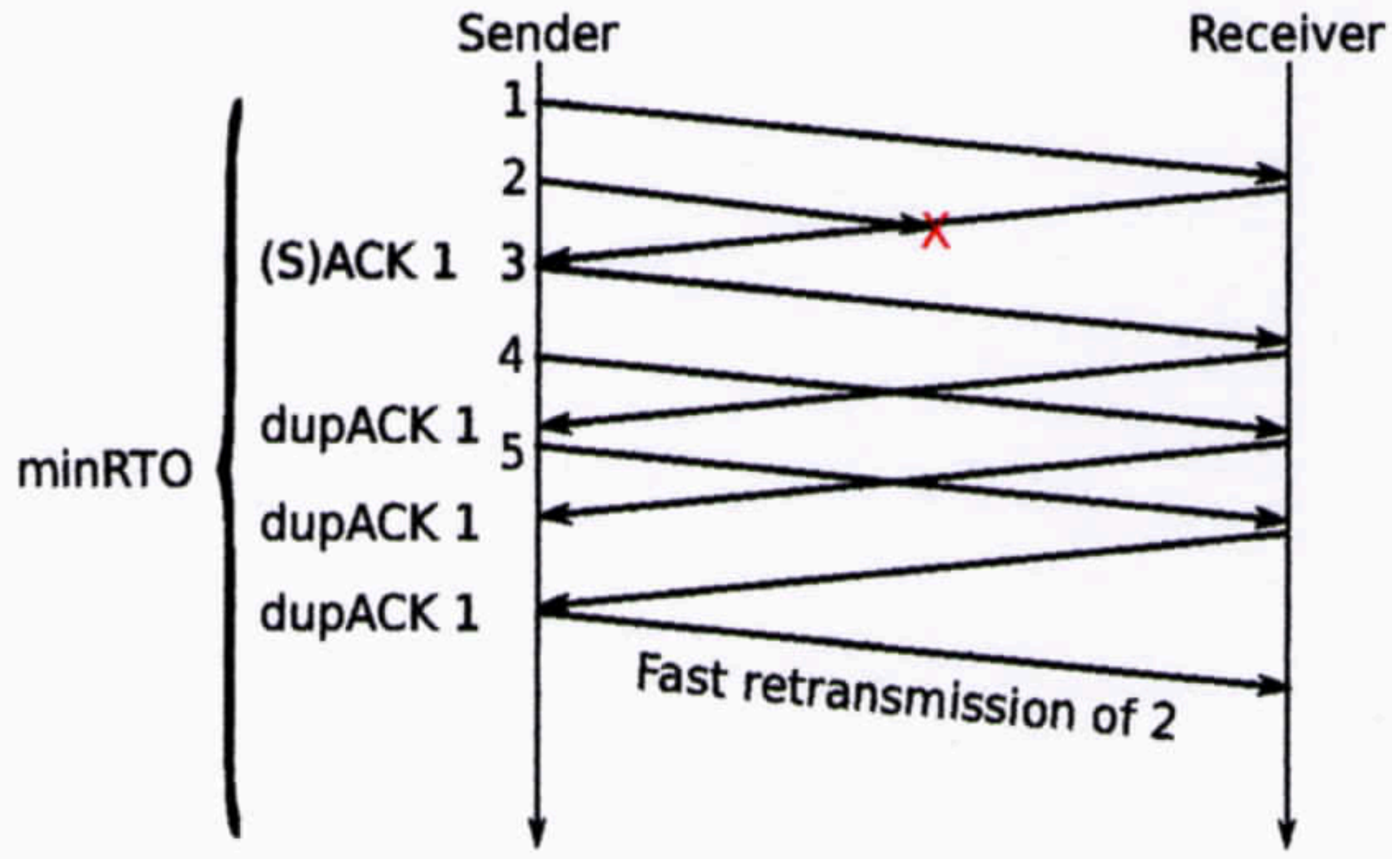


(a) With Nagle's algorithm. (b) Without Nagle's algorithm.

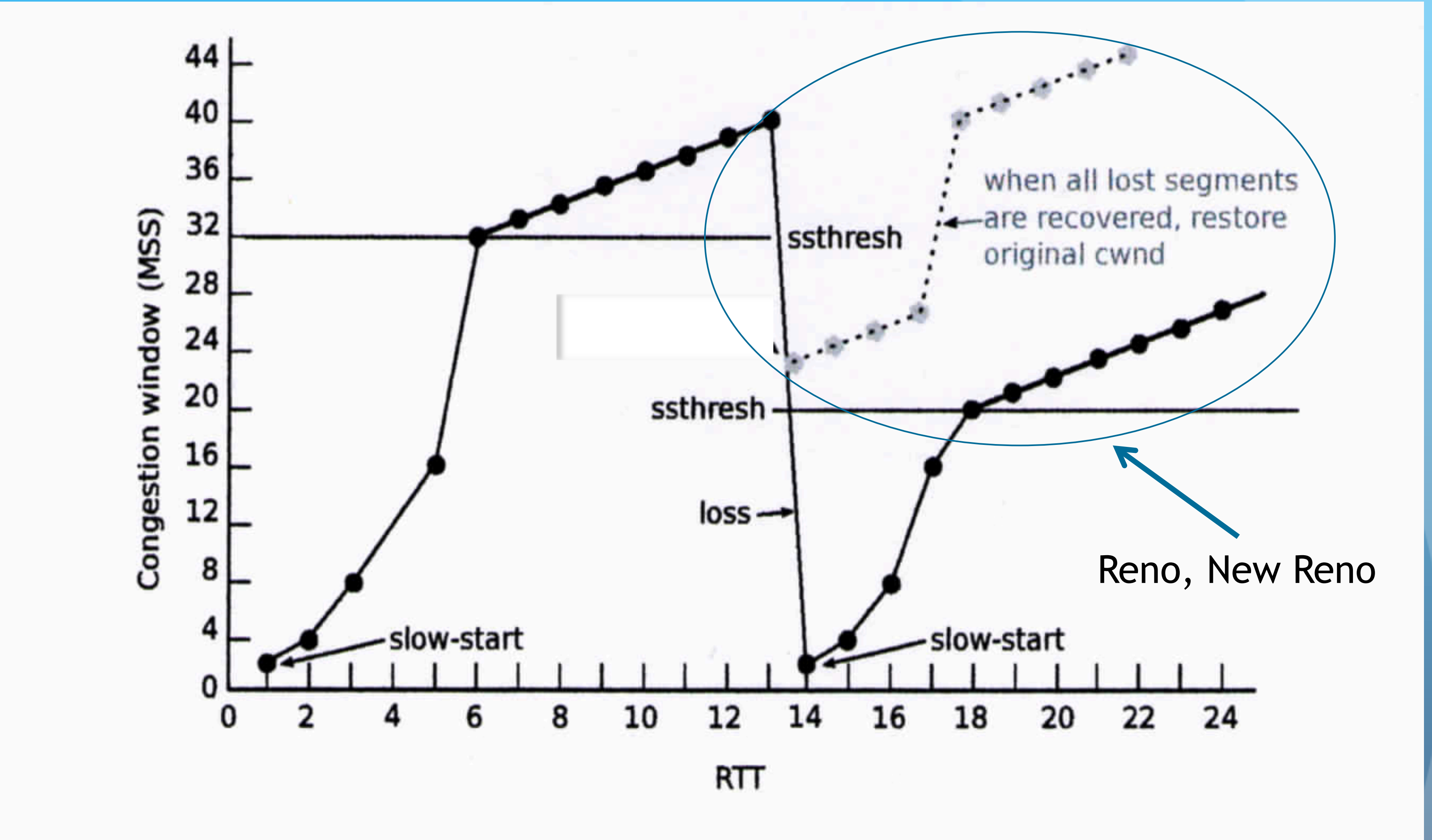
Congestion Control

- Slow start, congestion avoidance (additive increase, multiplicative decrease AIMD)
- Exponential Backoff – increase the retransmission timer
- Fast Restransmit – don't wait for timer, retransmit after 3 duplicate ACKs, set ssthresh1 to half the congestion window size, and initiate slow start
- TCP Reno – same as above but don't go into slow start. Continue as before until all segments recovered then jump to window size set before going into Fast Recovery
- TCP New Reno – same as above but allows retransmissions of segments that are still unacknowledged by a partial ACK – fills the holes in a sequence of outstanding packets with losses.

Fast Recovery and RTO



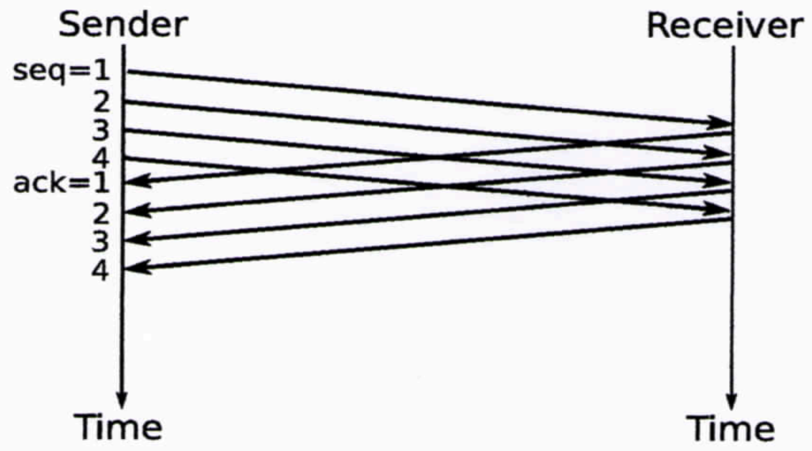
AIMD, Slow Start and Fast recovery+



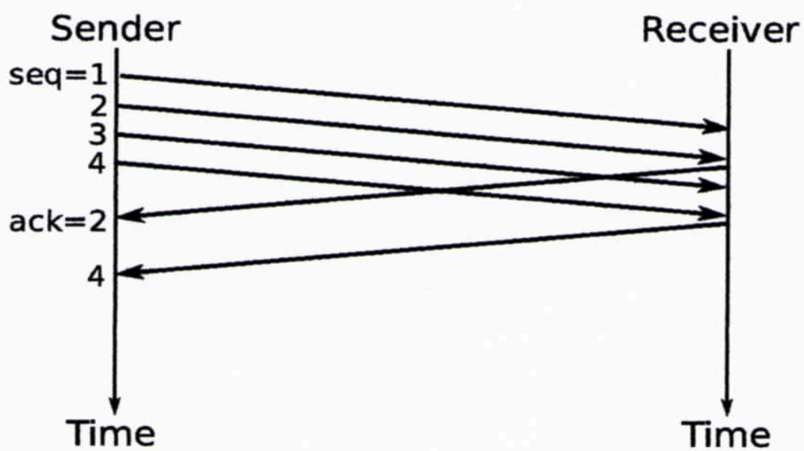
More TCP Mechanisms

- SACK – Selective ACK. Seq. no. of received segments listed in option field. When used with New Reno, improves latency.
- Delayed ACK – Wait for a short duration to piggyback ACK on a data packet being sent out. Also results in larger group ACKs (more data arrived during the wait interval) but it messes up RTO calculations as the RTT is now inflated by the delay.

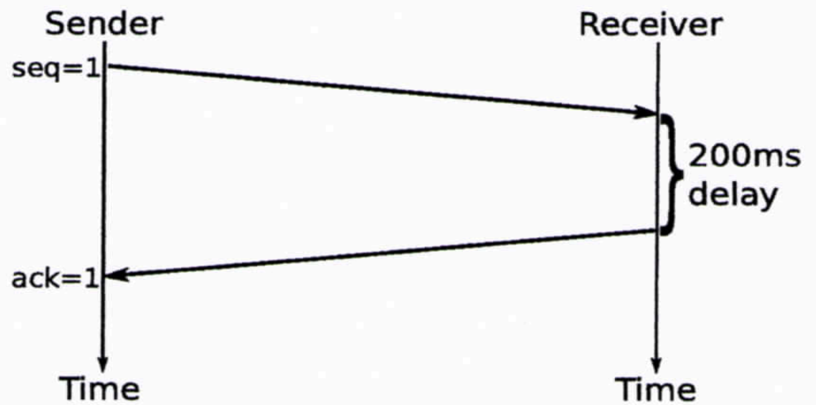
Delayed ACK



(a) Without delayed ACKs. Every received data segment is ACKed.



(b) With delayed ACKs. Bandwidth is saved on the upstream path.



(c) With delayed ACKs. If no further segments arrive, the ACK is triggered by a timer.

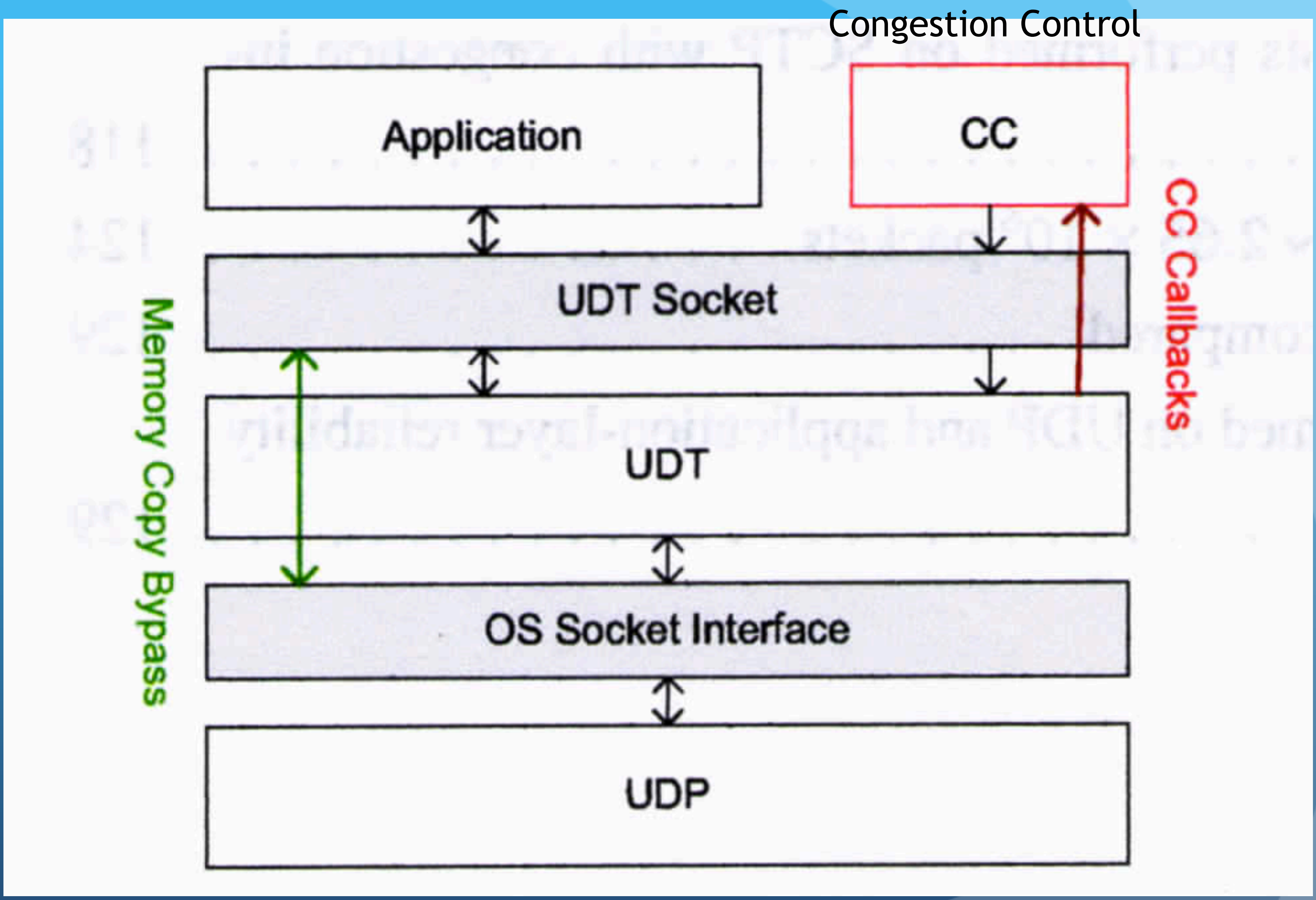
UDP and Application Level Reliability

- Two approaches:
  - A simple library of low level network functions and basic services – e.g., ENet
  - A comprehensive library giving many options – e.g., UDT

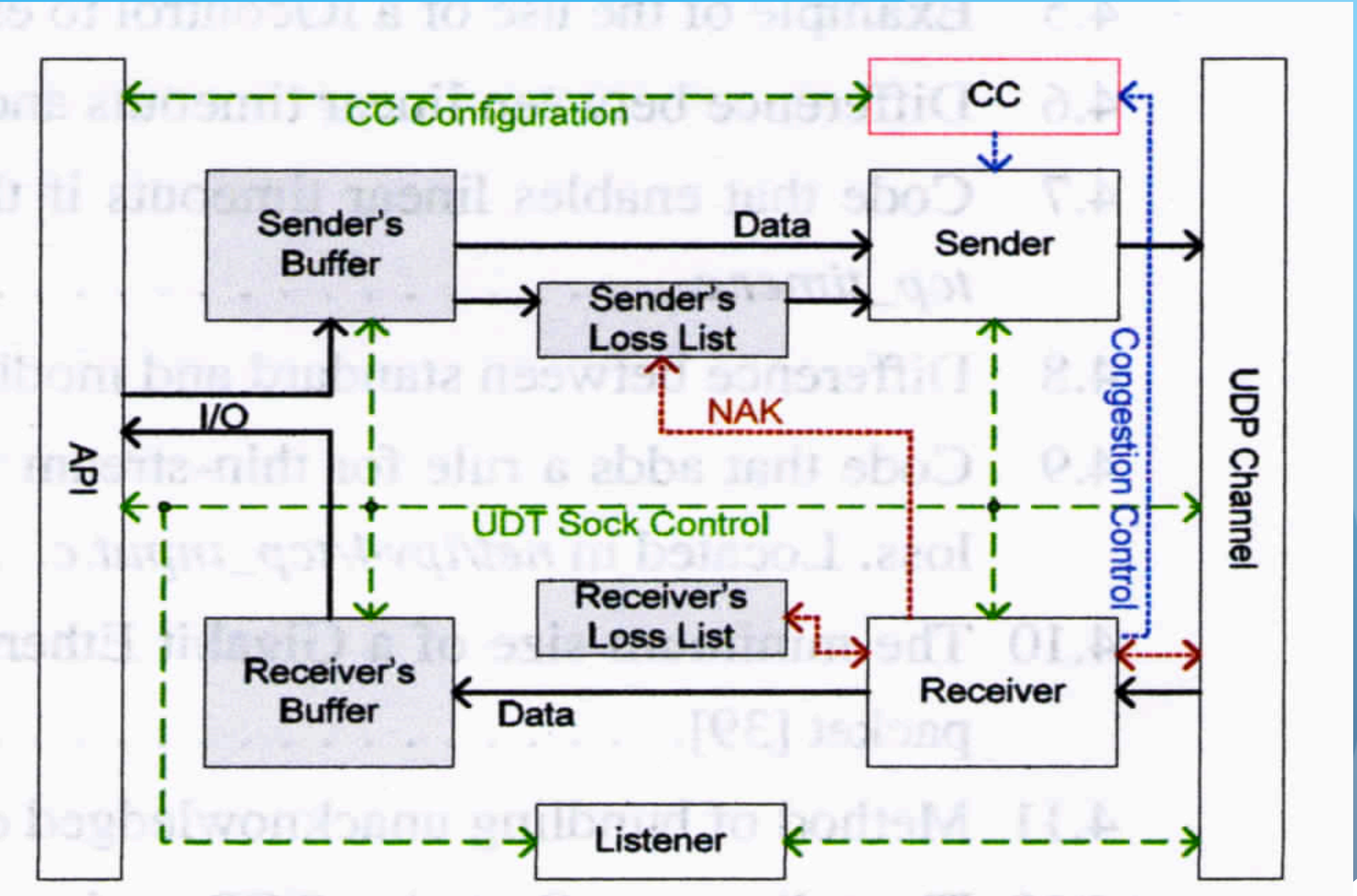
UDT – UDP based Transfer

- It is built on the top of UDP with reliability control and congestion control. Designed for high speed links.
- The congestion control algorithm is the major internal functionality to enable UDT to effectively utilize high bandwidth links.
- Also implemented a set of APIs to support easy application implementation, including both reliable data streaming and partial reliable messaging.

UDT Architecture



UDT Flow


API
CC
CC Configuration
Sender's Buffer
Data
Sender
Sender's Loss List
NAK
I/O
UDT Sock Control
Congestion Control
UDP Channel
Receiver's Loss List
Receiver's Buffer
Receiver
Data
Listener

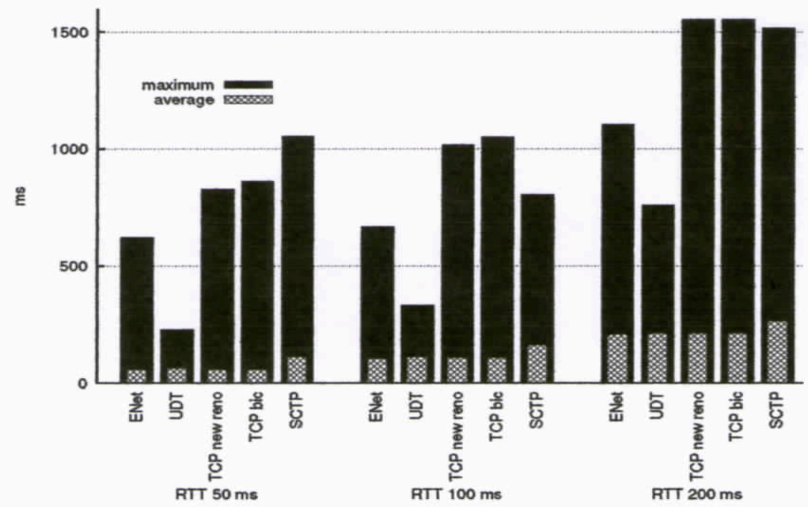
UDT Operation

- A two way handshake is used for connection set up. A client sends a request with sequence numbers, window and message size.
- The server ACKs the request and sends its own parameters to the client.
- Data transfer starts once client has received the ACK.
- It uses timer-based selective acknowledgment, which generates an acknowledgment at a **fixed** interval. If there are new continuously received data packets, this saves BW.
- At very low bandwidth, UDT acts like protocols that acknowledge every data packet.
- Negative acknowledgment (NAK) is used to explicitly feed back packet loss. NAK is generated once a loss is detected so that the sender can react to congestion as quickly as possible.

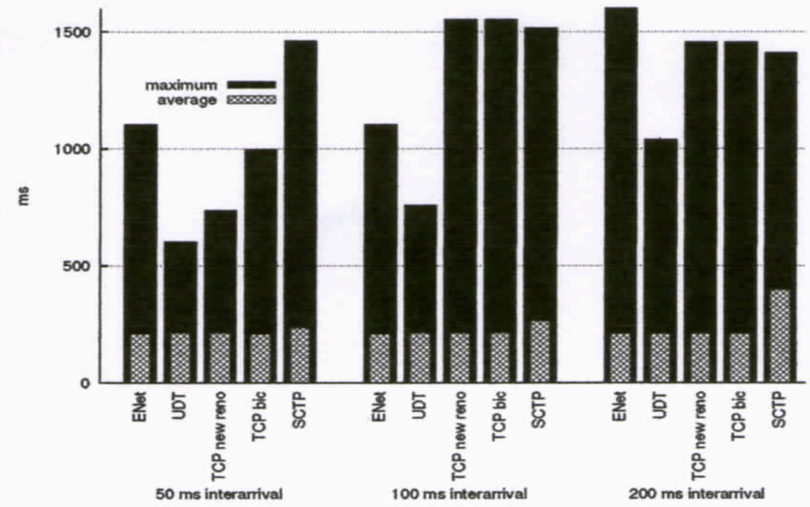
ENet

- Designed for online gaming support. It was developed for the Cube game engine and was later used by other networked games.
- ENet provides a relatively thin, simple and robust network communication layer on top of UDP that supports optional, reliable, in-order delivery of packets
- The services include a connection interface for communicating with the remote host.
- Delivery can be configured to be stream oriented or message oriented.
- The state of the connection is monitored by pinging the target, and network conditions such as RTT and packet loss are recorded.
- Retransmissions are triggered using timeouts based on the RTT, much like the TCP mechanisms.
- The congestion control implements exponential backoff like TCP.
- ENet also applies bundling of queued data if the maximum packet size is not reached.

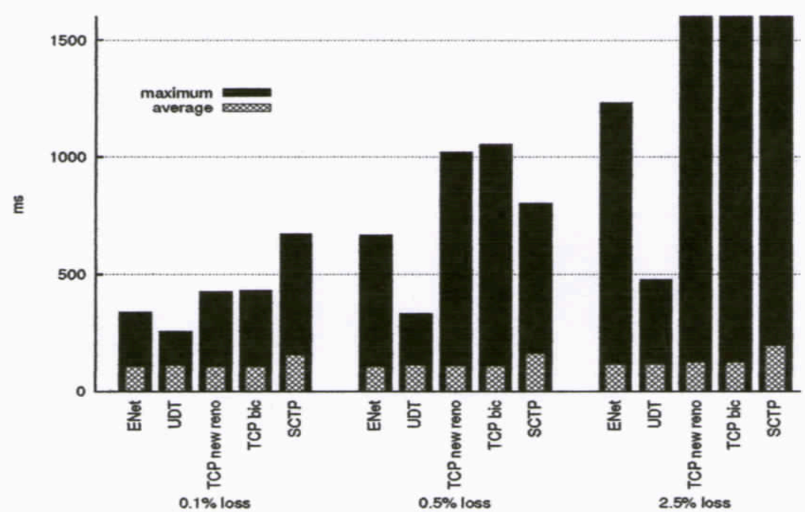
Comparison of UDP and TCP based schemes



(a) Latency vs. RTT. Loss=0.5%. Packet IAT=100 ms.



(b) Latency vs. packet IAT. Loss=0.5%. RTT=200 ms.



(c) Latency vs. loss rate. RTT=100 ms. IAT=100 ms.

Challenges of Thin Streams

- Thin-streams suffer from high latencies when using reliable transport protocols.
- Implementations of reliability and in-order delivery on top of UDP are modeled on the principles from TCP.
- The foremost tool used by TCP to recover without triggering a timeout is the *fast retransmit* mechanism.
- This is also the key to understanding the high latencies that can be observed for thin streams.
  - Thin streams often have no more than one packet in flight per RTT. As a fast retransmit needs three dupACKS to be triggered, this seldom (or never) happens for such streams. The effect is that recovery for thin streams is limited almost entirely to timeouts.
  - A retransmission by timeout triggers exponential backoff, thus delaying further retransmission attempts. Subsequent lost retransmissions increases the delay until we can observe extreme values, e.g., 67secs delay for 6 retransmissions (taken from a trace of Anarchy Online)

References

- A. Petlund, Improving latency for interactive, thin-stream applications over reliable transport, PhD thesis, Simula Research Laboratory / University of Oslo, Unipub, Kristian Ottosens hus, Pb. 33 Blindern, 0313 Oslo, 2009.
- Yunhong Gu and Robert L. Grossman, UDT: UDP-based Data Transfer for High-Speed Wide Area Networks, Computer Networks (Elsevier). Volume 51, Issue 7. May 2007.