Ch 15 Protocol Enhancements for Thin Streams

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

Two approaches both based on TCP

- Modifications to TCP parameters/thresholds to avoid latencies caused by congestion control and retransmission mechanisms
- Adding class distinctions to the traffic flow to identify different packet types for preferential treatment

Thin Streams and TCP

- Why wont TCP work?
  - Game packets are very small, overhead of TCP is very high in comparison
- In-order processing of packets causes additional delays
- Congestion control unnecessary as game traffic is application-limited
- Fast-retransmit ineffective as inter-arrival times (IAT) between packets is very long.

Thin Streams and UDP

- Works for some traffic types that do not need high reliability but most games other than FPS, will not use UDP because of out of order delivery and packet losses.
- Only used by some games in conjunction with a middleware layer that adds TCP like behavior to the packet stream. UDT and ENet are such examples.
- Note that for some of the streams in an online game, e.g., voice chat, UDP can be used.
- And lets not forget the Firewall issue!!!

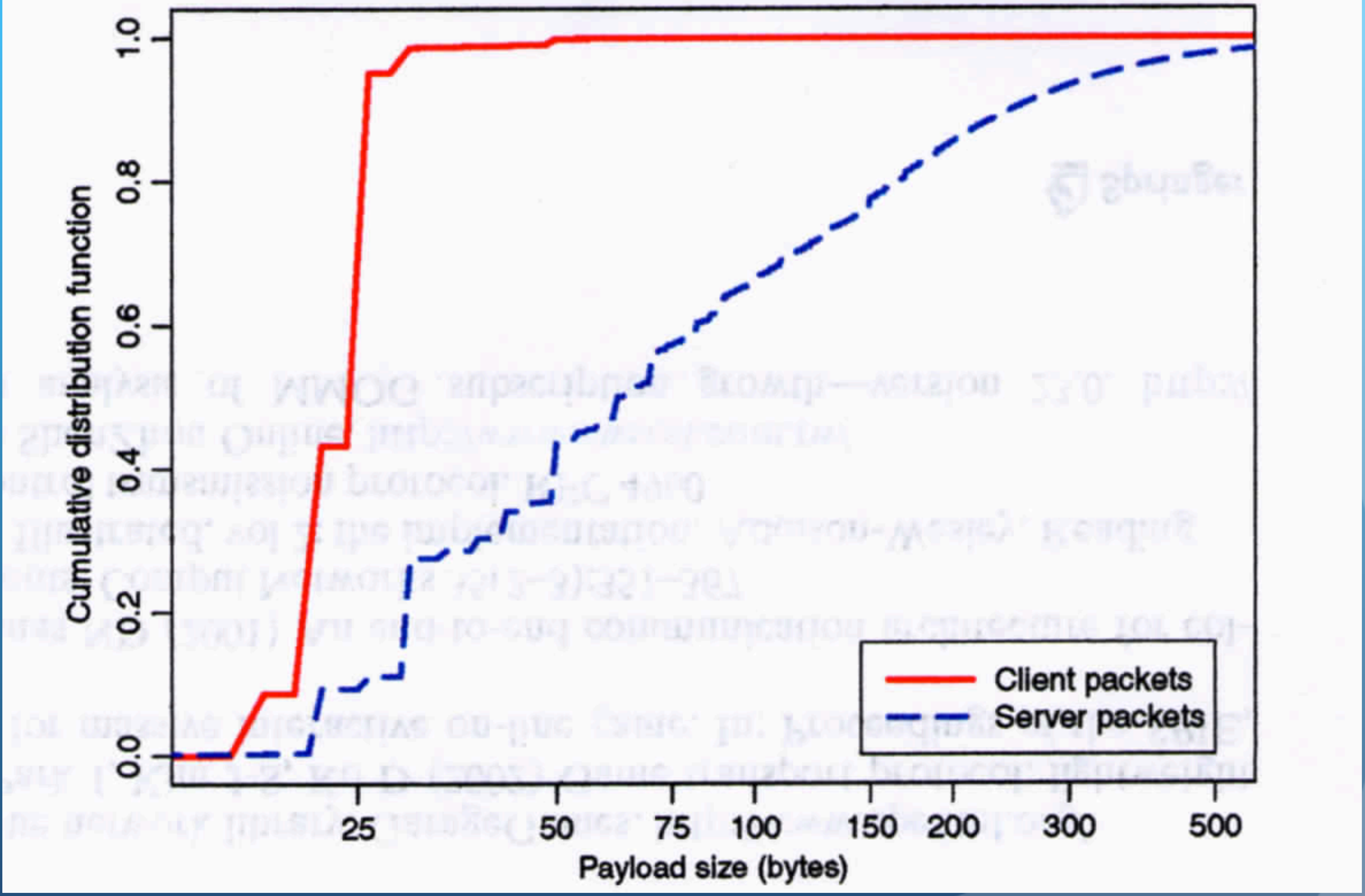
TCP behaviors and impact on Thin Streams

- TCP Overhead
- In-order delivery
- AIMD – Congestion Control
- Loss Recovery

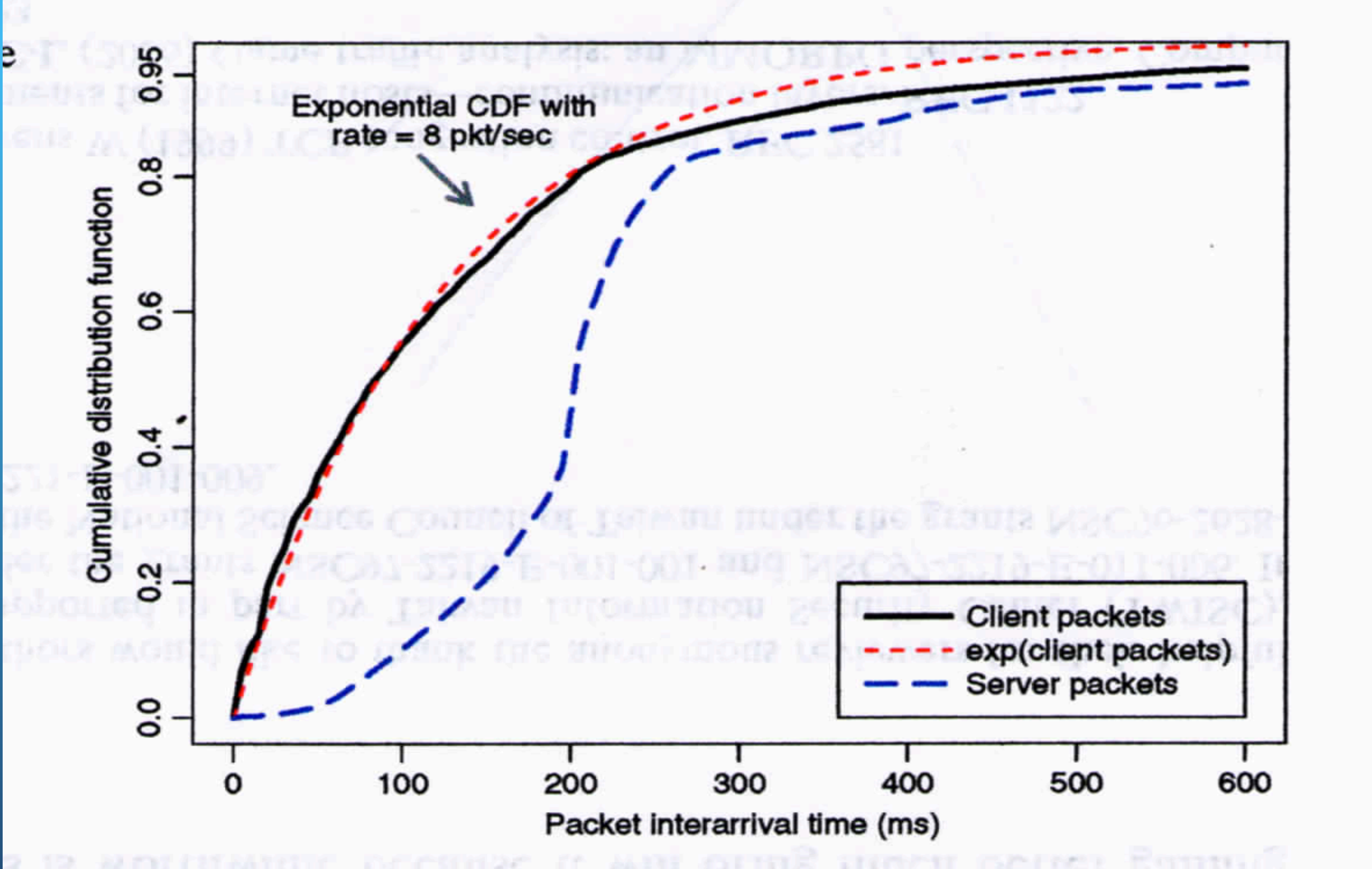
TCP Overhead

- Thin streams have very small packet sizes and very high inter packet arrivals (IAT)
- A very high percentage of the observed traffic in traces is overhead – ACKS and headers

Thin Stream Traffic – Packet Size



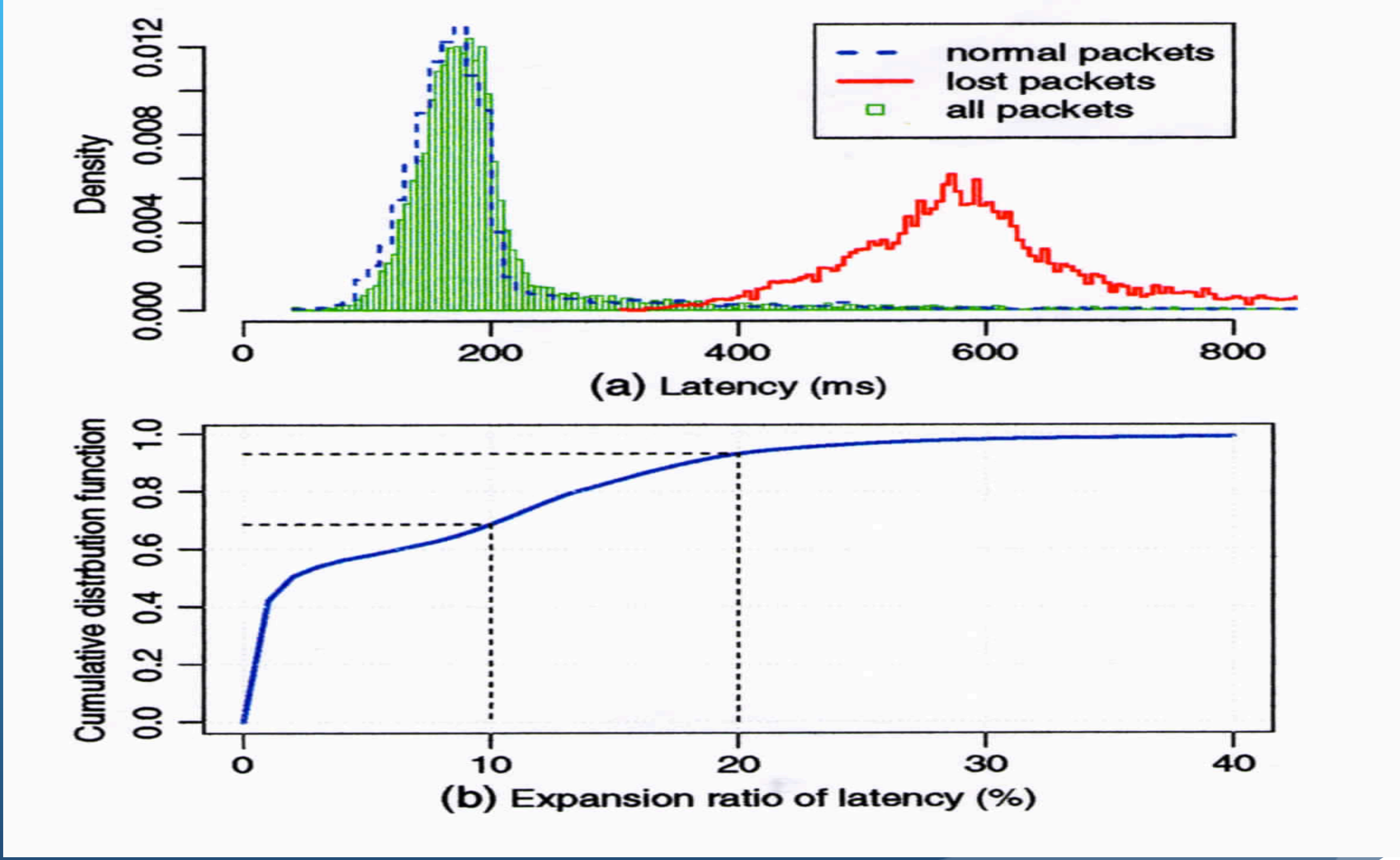
Thin Streams - IAT



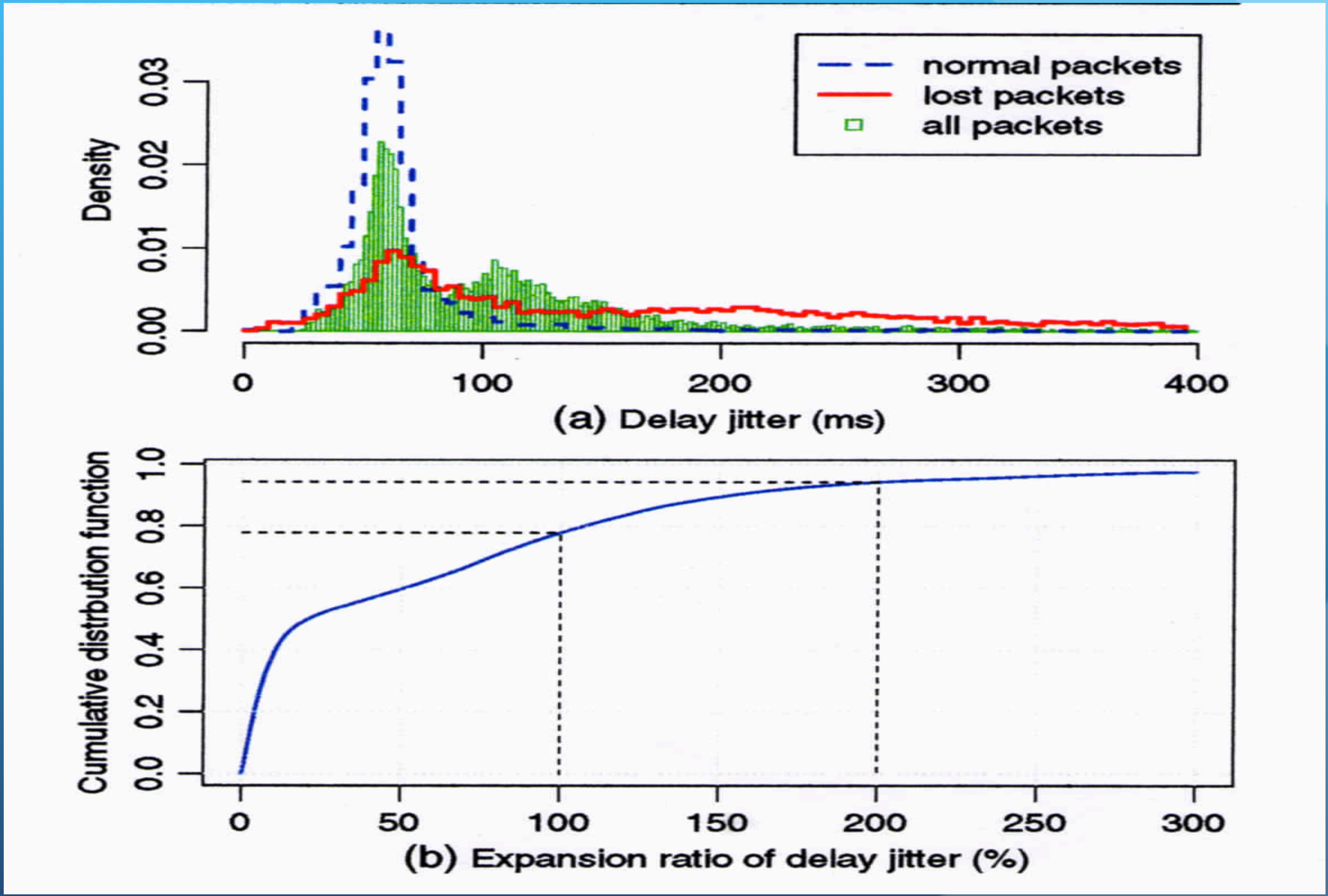
In-Order Delivery

- Packets wont be delivered to the application until all previous packets have been received and delivered.
- For games, players will often perform many fast actions in sequence. Each action is often an incremental update on on previous state. So playing packets out of order is not so bad, however in that case sometimes throwing away a packet would make sense as you only want to see the present view and not what it was a second or two ago.
- Some actions do need to be seen in succession as it could impact laying claim to some treasure for example.
- Retransmission and re-ordering due to a loss can increase latency.

Increased latency caused by Packet Losses that trigger control mechanisms



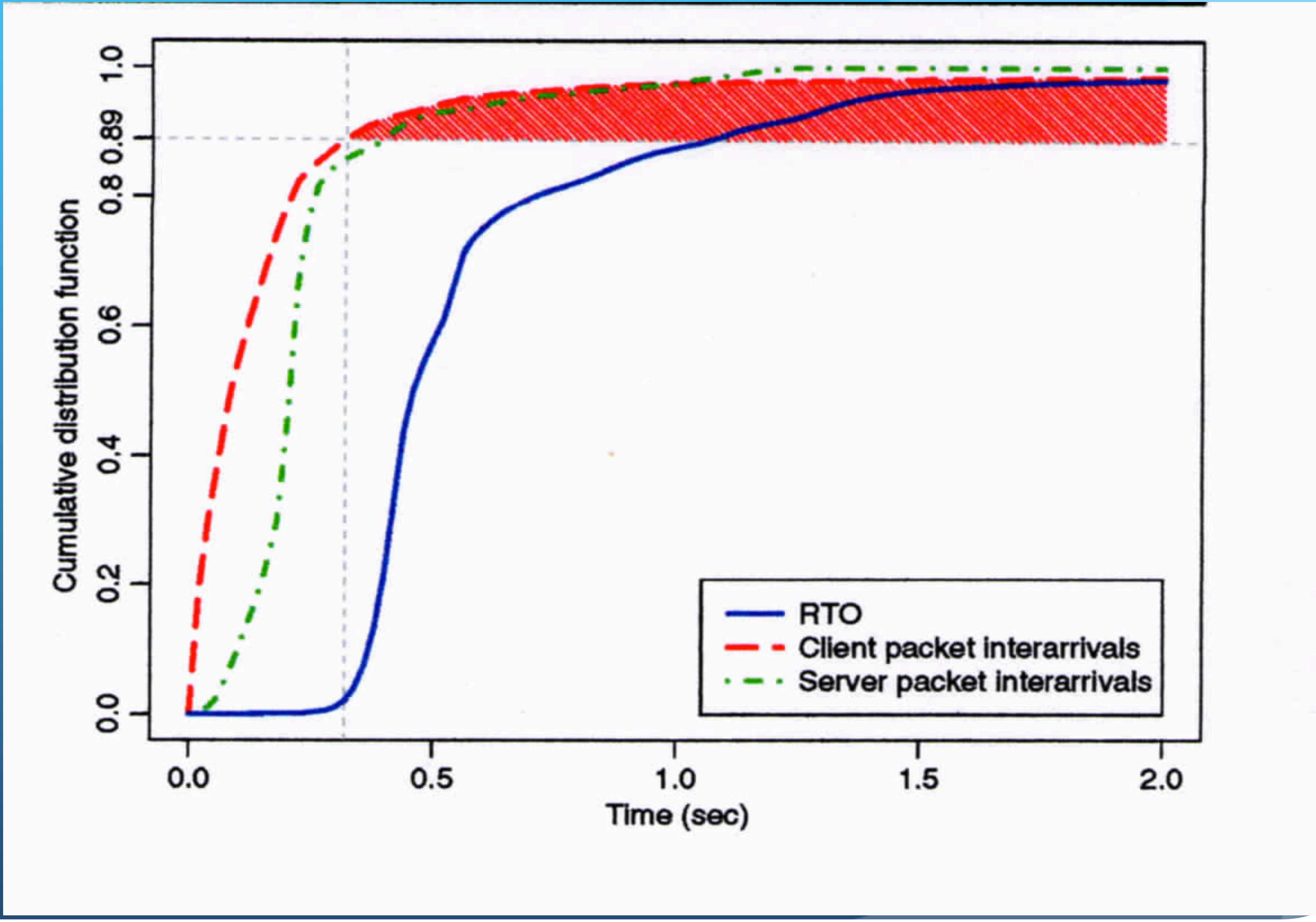
Increased jitter caused by Packet Losses that trigger control mechanisms



TCP and Congestion Control

- AIMD policy is designed for greedy traffic streams that have to be network limited.
- By contrast, thin streams are application limited.
- When there is no action on the link, i.e., the IAT is longer than the RTO-RTT, TCP sets the congestion window to 2 and keeps it there so long as this condition doesn’t change– this is called restart after idle period policy and is used to prevent an application from suddenly dumping a large burst of traffic into the pipe after a period of silence when the cwnd is still at the old value and network conditions may have changed.

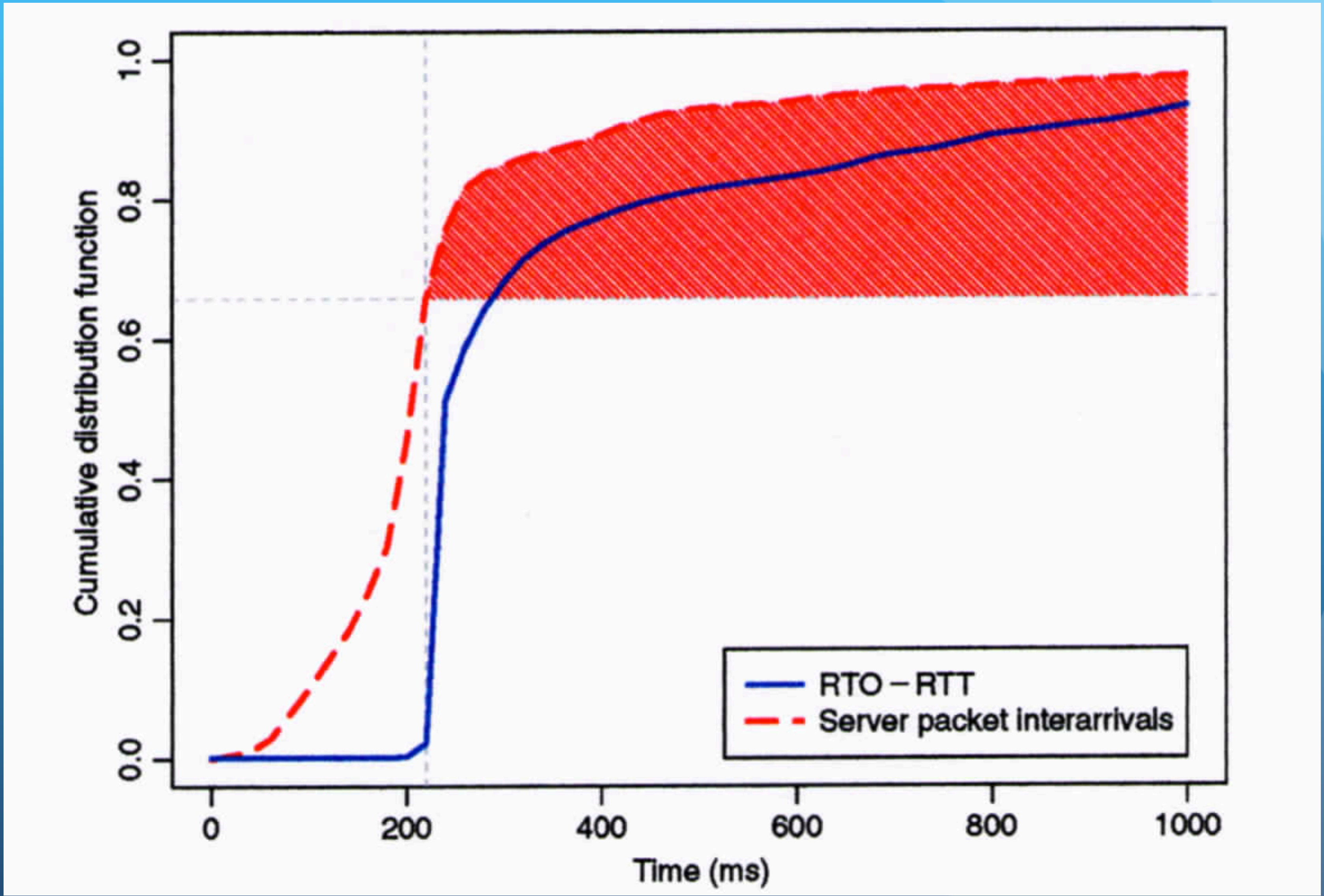
Congestion Control and Thin Streams



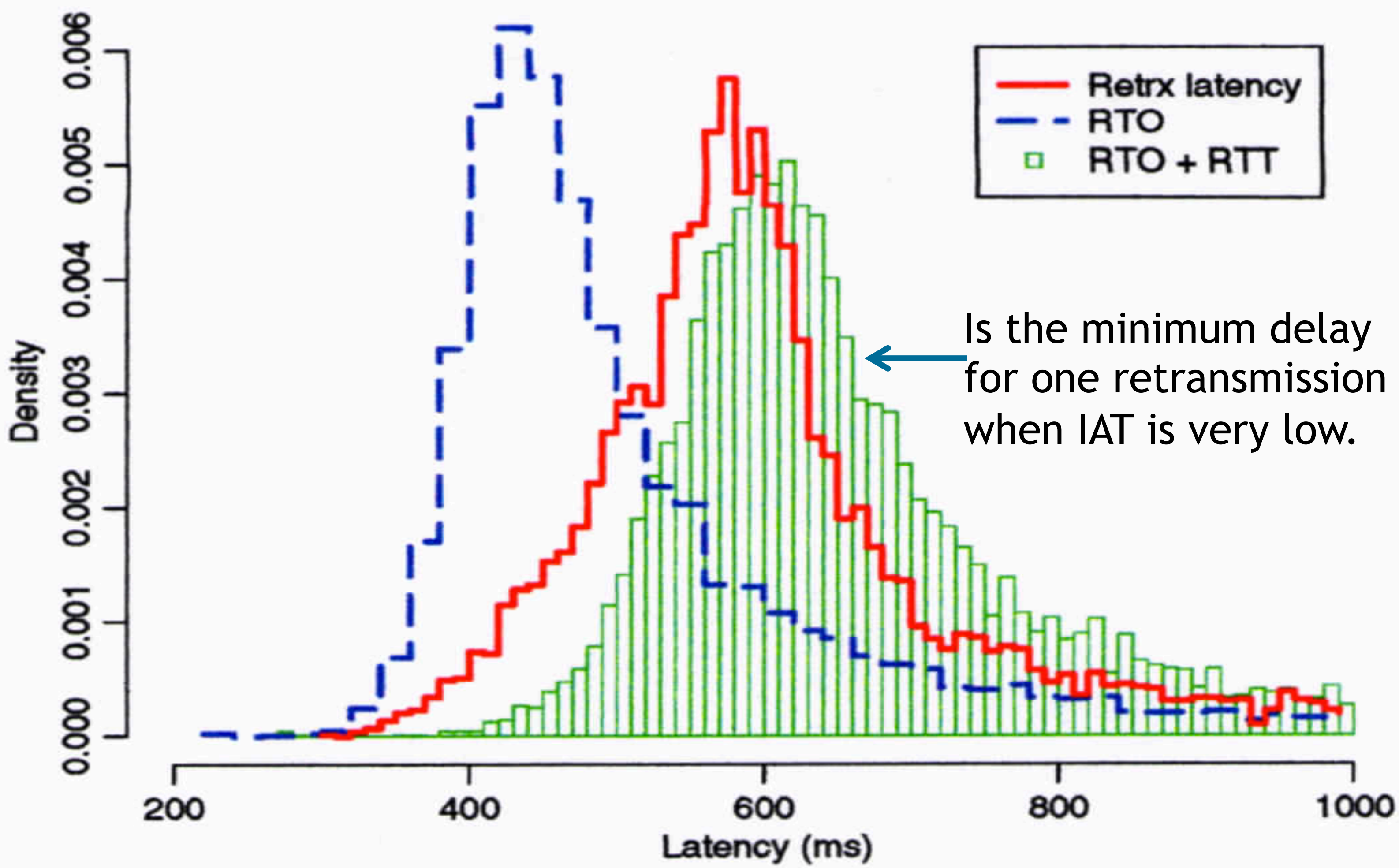
Loss Recovery

- To detect a loss in TCP:
  1. Retransmission timer expires (RTO)
  2. Fast retransmit – 3 duplicate ACKs of same packet
  3. Selective repeat – sends ACK for each received data packet.
- Because of high IAT, not enough traffic during RTO period. Means that fast retransmit will never be triggered.

IAT vs RTO-RTT



Avg. Latency of dropped packets



Is the minimum delay for one retransmission when IAT is very low.

Proposed TCP enhancements

- One paper proposes simple tweaks to some TCP calculations to bypass or change some actions if a thin stream is detected.
- The second paper is a little more elaborate, in that it breaks down the data into different types of streams and tailors the packet handling to attain the best possible transport strategies for each one.

Thin Stream Detection

- The thin-stream detection mechanism must be able to dynamically detect the current properties of a stream.
- The application should not have to be aware of the detection mechanism, nor have to feed data about its transmission rate to the network stack; it should be transparent to the application.
- Preferably should not introduce any new scheme. The chosen mechanism is based on an already existing counter of unacknowledged packets.

$$in\_transit \leq (ptt_{fr} + 1)$$

- Takes into consideration that a packet has to be lost for a fast retransmission to be triggered (1 lost packet + 3 dupACKs = 4 *in_transit*

Tweaks/Enhancements – a Wrapper

If (tcp_stream_is_thin) {

apply modifications

} *else* {

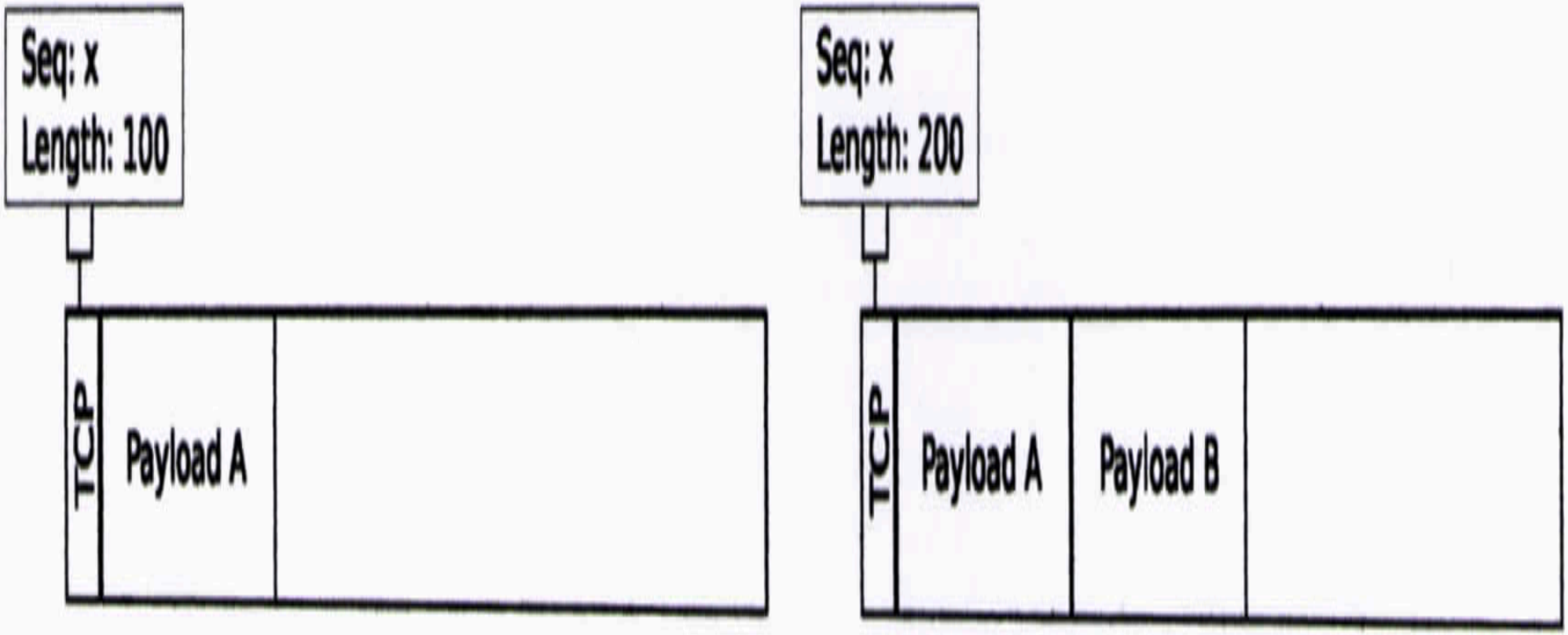
use normal TCP

}

What enhancements do we want for thin stream traffic?

- Removal of exponential backoff: To prevent an exponential increase in retransmission delay for a repeatedly lost packet, the exponential factor is removed.
- Faster Fast Retransmit: Instead of waiting for 3 duplicate acknowledgments before sending a fast retransmission, we retransmit after receiving only one.
- Redundant Data Bundling: Data is copied (bundle) from the unacknowledged packets in the send buffer into the next packet if space is available

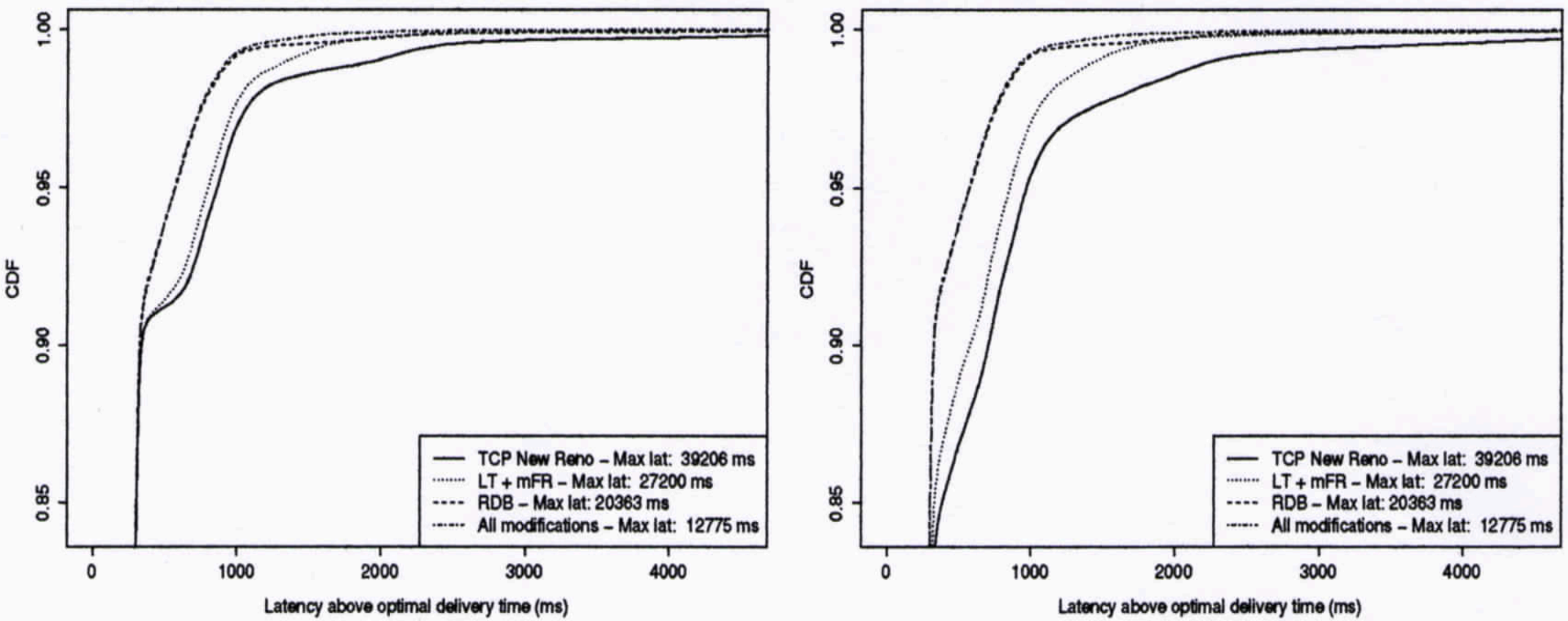
Bundling of Data



(a) First sent packet.

(b) Second packet: Bundled data.

Performance



(a) CDF of transport-layer delivery latency.

(b) CDF of application-layer delivery latency.

Using content classification

- For MMORPGs, the authors classify game messages generated by players into three types: *move*, *attack*, and *talk* messages.
  - Move messages report position updates when an avatar moves or goes to a new area. Since only the latest location in the game play matters, the server simply discards out-of-date move messages.
  - Attack messages correspond to an avatar's combat actions when it engages in fights with opponents. Such messages cannot be lost because each action will have some impact on the target. However, if several successive attack messages describe the same combat action against the same target, out-of-order arrivals of these messages can be tolerated.
  - Talk messages convey the contents of conversations between players. Must be transmitted in order and reliably.

Transport Options

- **Multi-streaming**: With this option, different types of game messages can be put into separate streams, each of which processes the messages independently.
- **Optional Ordering**: Can reduce this overhead because it allows some types of messages to be processed as soon as they are received without being buffered if their preceding messages have not arrived.
- **Optional Reliability**: With this option, messages that do not require reliable transmission can simply be ignored if they are lost in the network.

Content-based transport strategies

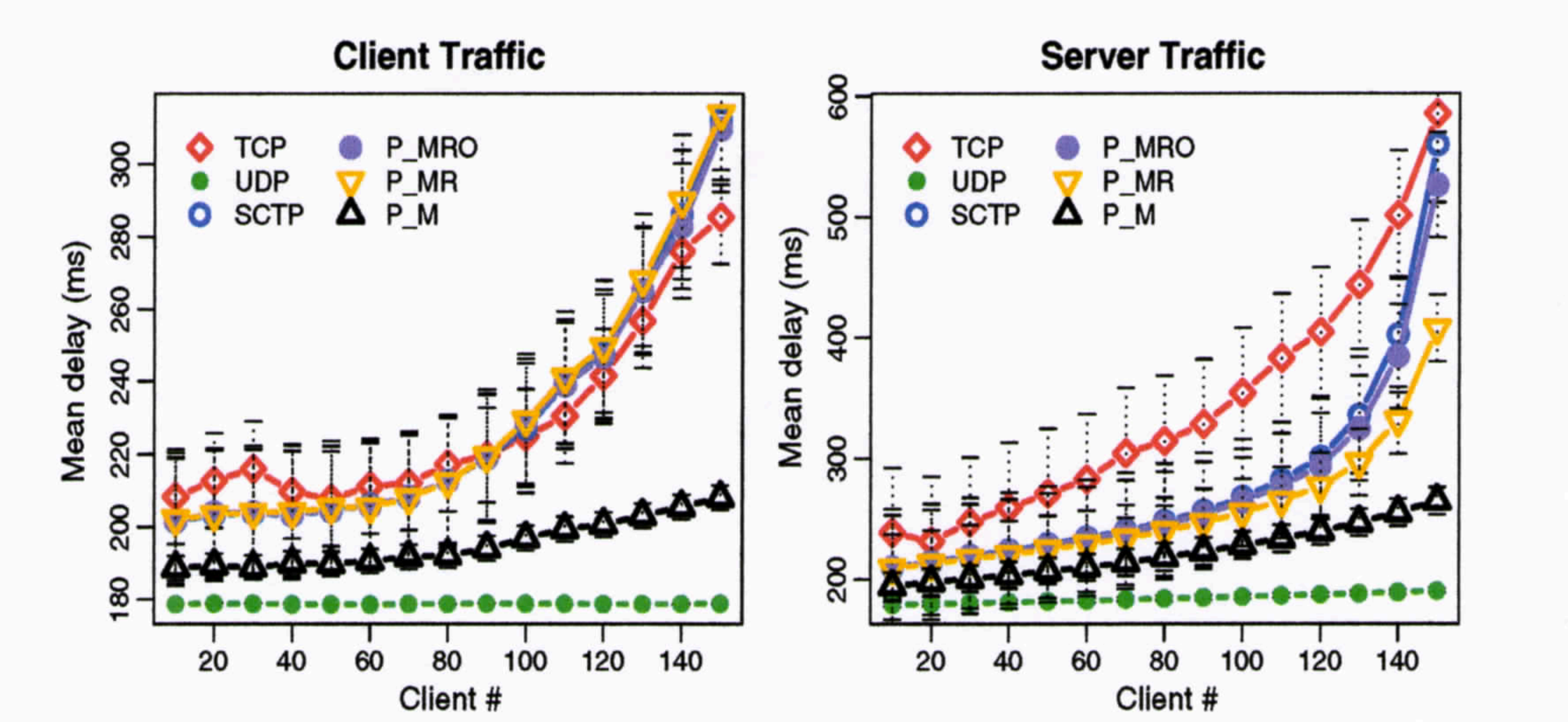
- MRO Strategy: MRO only uses *multi-streaming* (M); that is, it guarantees transmission reliability (R) as well as packet ordering (O). Under this strategy, game messages are classified into three types, namely move, attack, and talk, separate streams are used to handle each.
- MR Strategy: MR implements both *multi-streaming* and *optional ordering*. This strategy provides two kinds of streams: ordered streams and unordered streams.
- M Strategy: M combines all three options, that is, *multi-streaming*, *optional ordering*, and *optional reliability*. Under this strategy, there are three kinds of streams: ordered and reliable streams, unordered and reliable streams, and unordered and unreliable streams.

Evaluate the effect of the three content-based strategies on a live trace of Angel's Love

- P_{MRO} implements the MRO strategy, which puts move, attack, and talk messages into three separate ordered and reliable streams.
- P_{MR} is based on the MR strategy. It transmits move and attack messages via two unordered and reliable streams individually, while talk messages are put into an ordered and reliable stream.
- P_{M} employs the M strategy, which transmits move messages via an unordered and unreliable stream, attack messages via an unordered and reliable stream, and talk messages via an ordered and reliable stream.

Compare to TCP, UDP, SCTP - Latency

Used traces of *Angel's Love*, a mid-scale, TCP-based MMORPG on a test bed in a lab.



Jitter Performance

References

- ``On the challenge and design of transport protocols for MMORPGs,” Chen-Chi Wu, Kuan-Ta Chen, Chih-Ming Chen, Polly Huang, Chin-Laung Lei, Multimedia Tools and Appl. (2009) 45:7–32.
- ``TCP Enhancements for Interactive Thin-Stream Applications,” Andreas Petlund, Kristian Evensen, Carsten Griwodz, Pål Halvorsen, in *Proceedings of NOSSDAV ’08*, Braunschweig, Germany, 2008.
- ``The Fun of using TCP for an MMORPG,” Carsten Griwodz, Pål Halvorsen, in *Proceedings of NOSSDAV ’06* Newport, Rhode Island, USA, 2006.
- ``Latency Evaluation of Networking Mechanisms for Game Traffic,” Szabolcs Harcsik, Andreas Petlund, Carsten Griwodz, Pål Halvorsen, in *Proceedings of NetGames’07*, Melbourne, Australia, September 19-20, 2007.