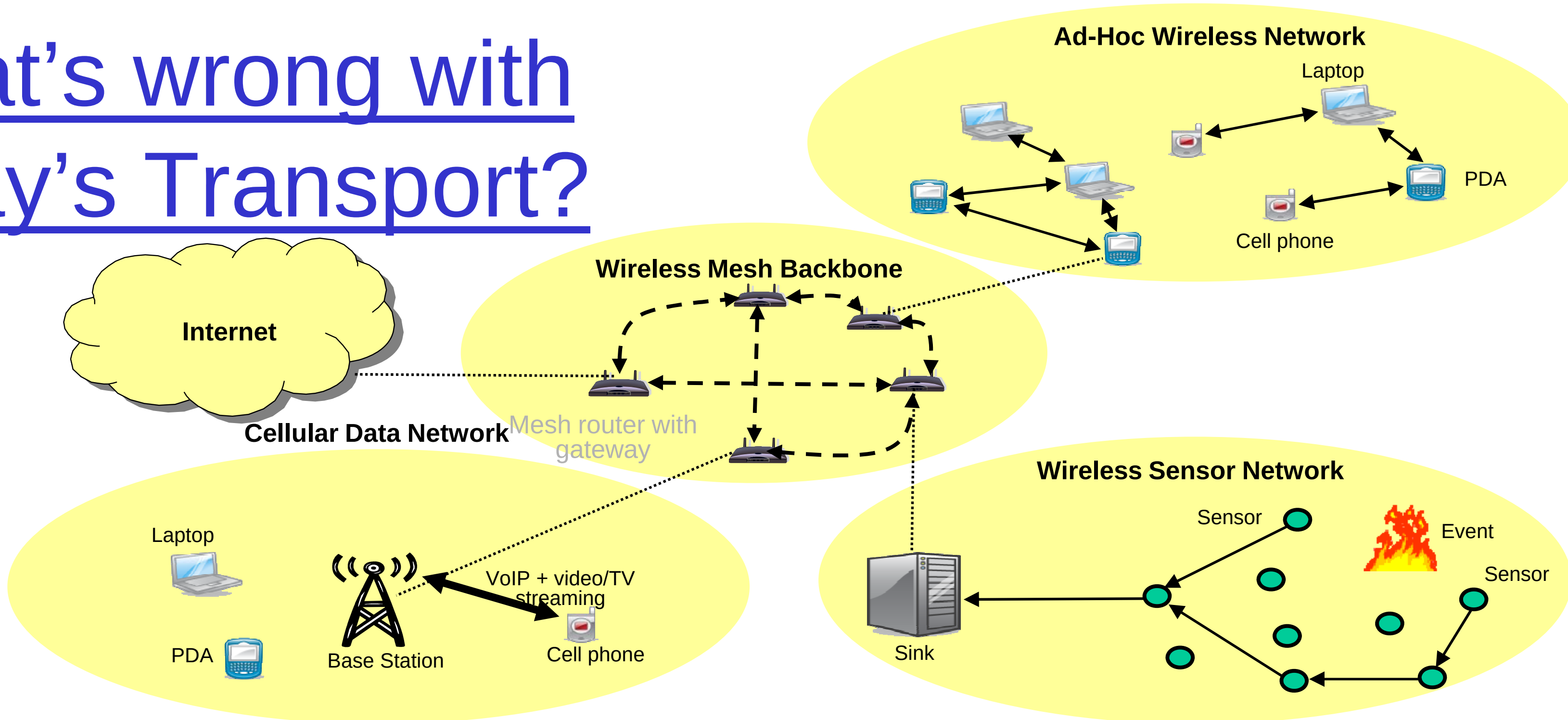
Revisiting a Soft-State Approach to Managing Reliable Transport Connections

Gonca Gursun, **Ibrahim Matta**, Karim Mattar

Computer Science

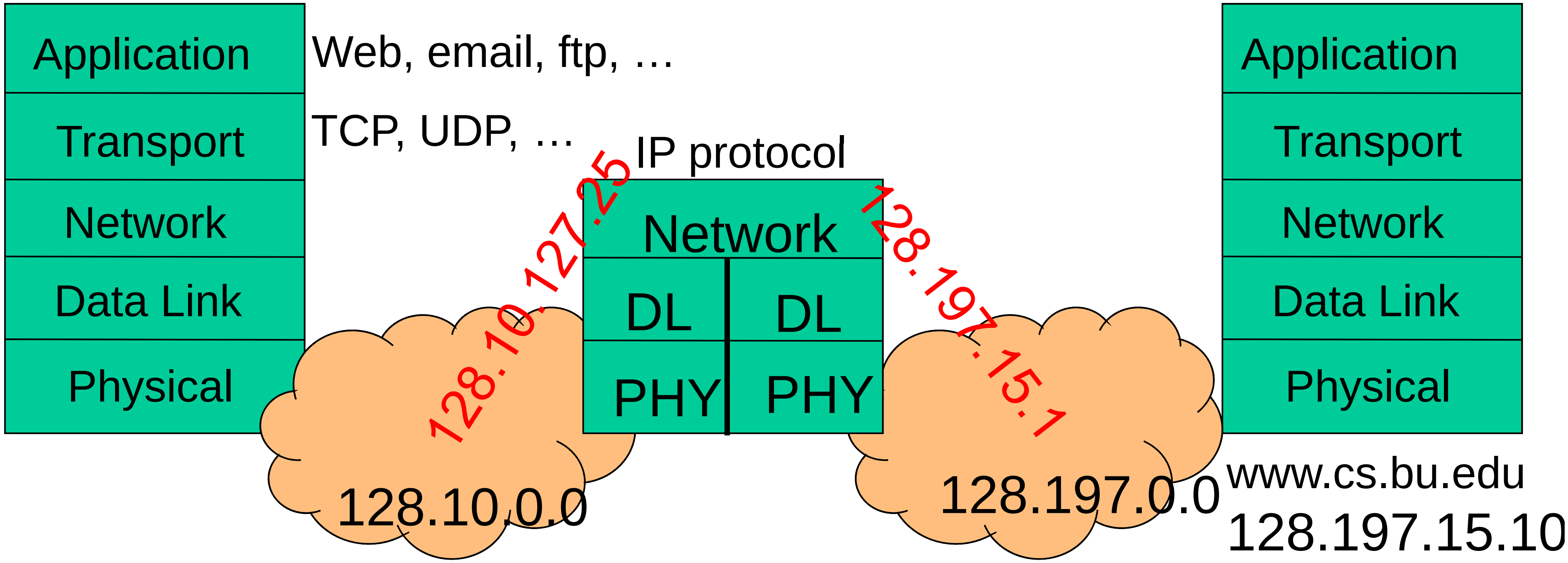
Boston U.

What's wrong with today's Transport?



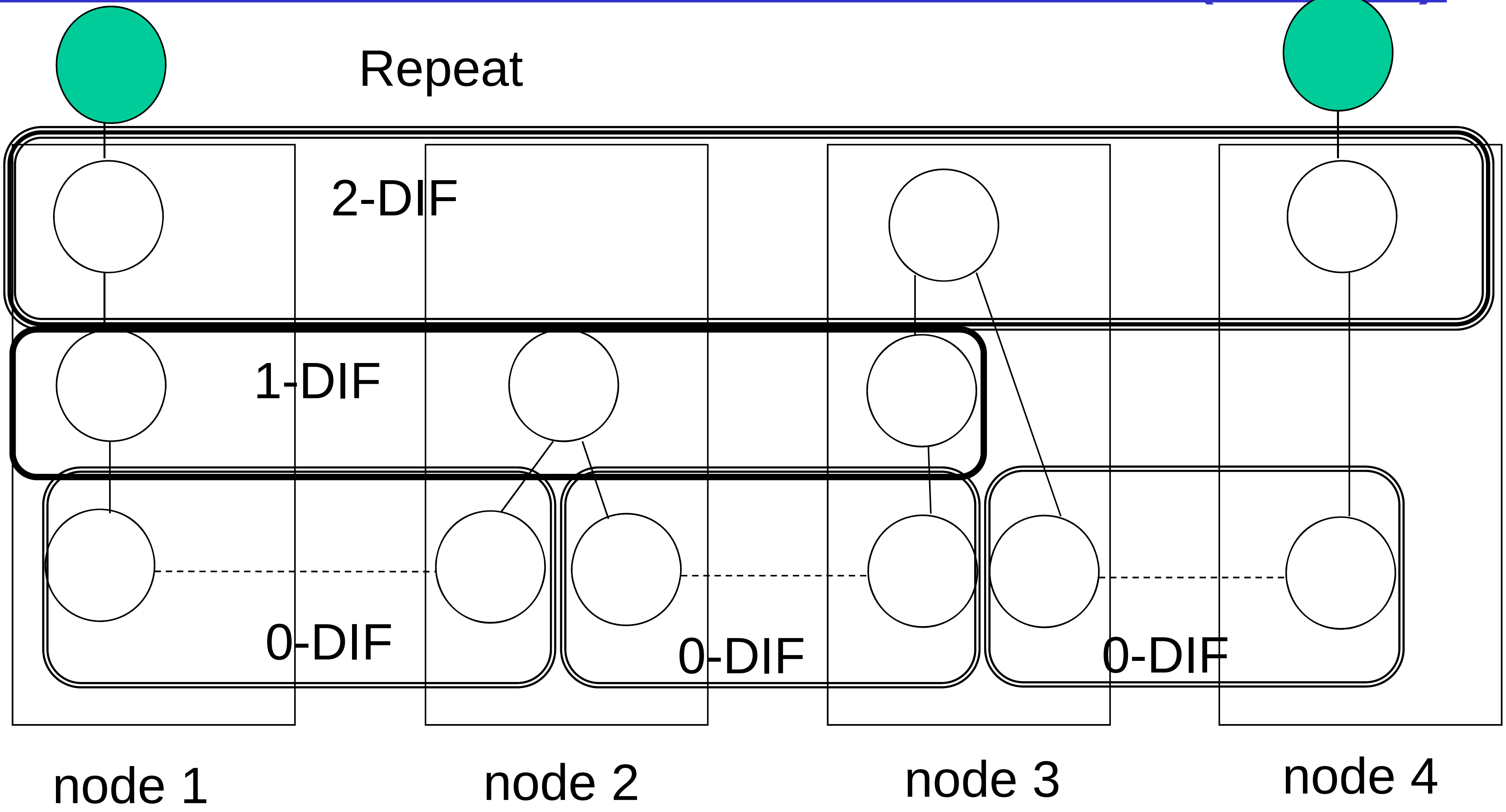
- The new brave world
  - Larger scale, more diverse technologies
  - New services: content-driven, context-aware, mobile, socially-driven, secure, profitable, …
- Custom point-solutions: No or little "science"
- Lots of problems: bad performance, hard to manage, hard to adopt, …

Internet's view: one big, flat, open net



- There's no building block
- The "hour-glass" model imposed a least common denominator

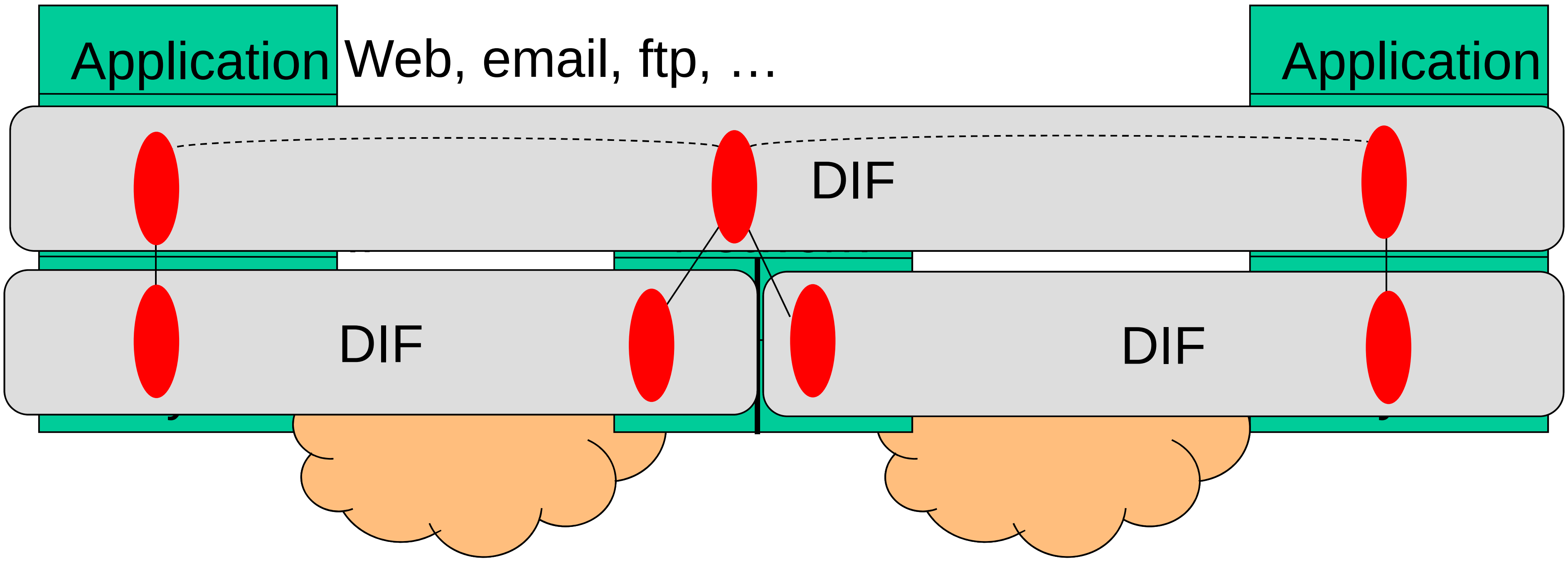
Recursive InterNet Architecture (RINA)



DIF = Distributed IPC Facility (locus of shared state=scope)

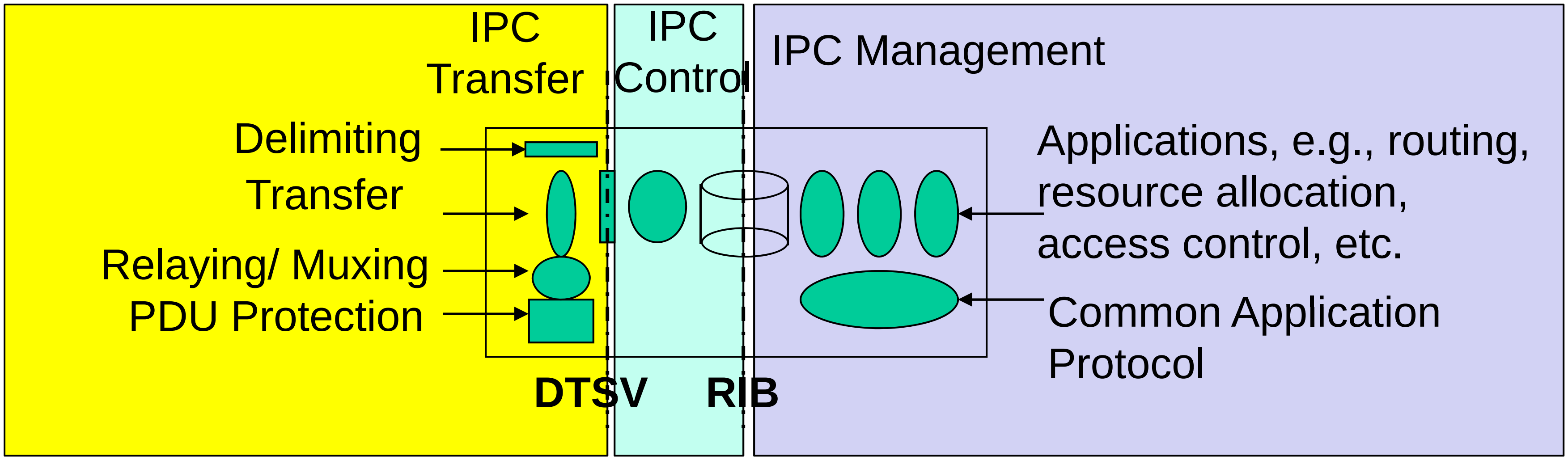
Policies are tailored to scope of DIF

RINA allows scoping of services



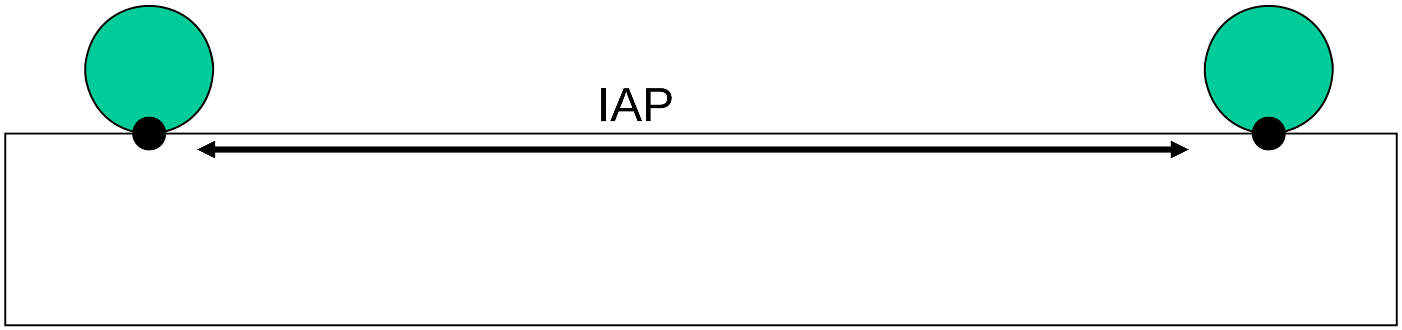
- The DIF is the building block and can be composed
- Good we split TCP, but we split TCP in the wrong direction!
- E2E (end-to-end principle) is not relevant
  - Each DIF layer provides (transport) service / QoS over its scope

What Goes into a DIF?



- Processing at 3 timescales, decoupled by either a Data Transfer State Vector or a Resource Information Base
  - IPC Transfer actually moves the data
  - IPC Control (optional) for error, flow control, etc.
  - IPC Management for routing, resource allocation, locating applications, access control, monitoring lower layer, etc.

Only one Data Transfer Protocol



- RINA decouples port allocation and access control from data transfer
- Allocating conn ID (ports) is done by management, IPC Access Protocol (IAP), in a hard-state (HS) fashion
- Once allocated, Data Transfer can start, ala Delta-t [Watson'81]
  - Flows without data transfer control are UDP-like. Flows without reliability requirement do not ACK. Different policies support different requirements
- Delta-t is a soft-state (SS) protocol
- If there is a long idle period, conn state is discarded, but ports remain

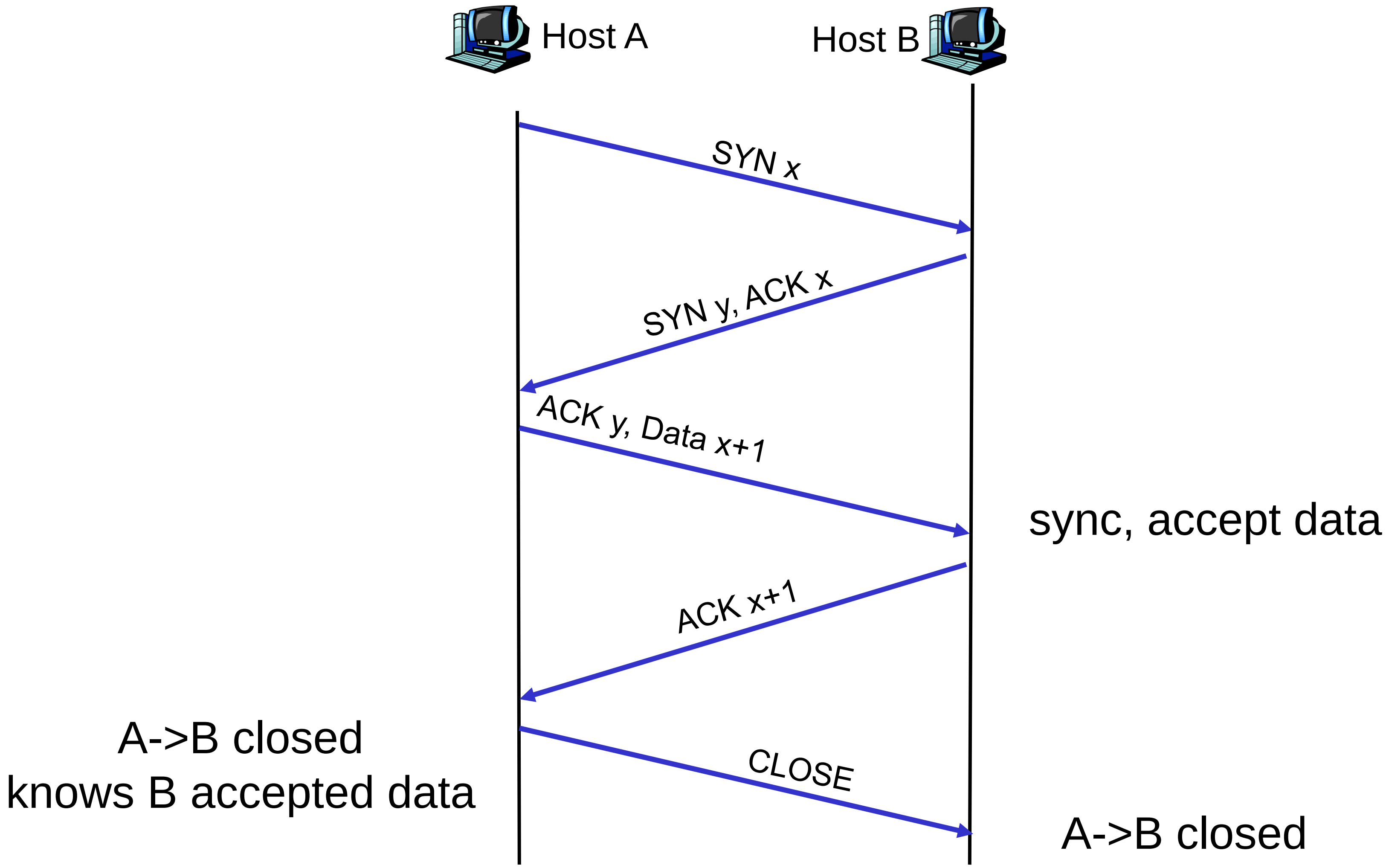
Why not TCP?

- Hard-state must be explicitly discarded
- But we don't need it to be [Watson '81]
- Watson proves that if 3 timers are bounded:
    - Maximum Packet Lifetime (MPL)
    - Maximum time for retries (G)
    - Maximum time before ACK (UAT)
  - That no explicit state synchronization, i.e., hard-state, is necessary
    - SYNs, FINs are unnecessary
- In fact, TCP uses all these timers and more
- TCP is really hybrid HS+SS

This paper …

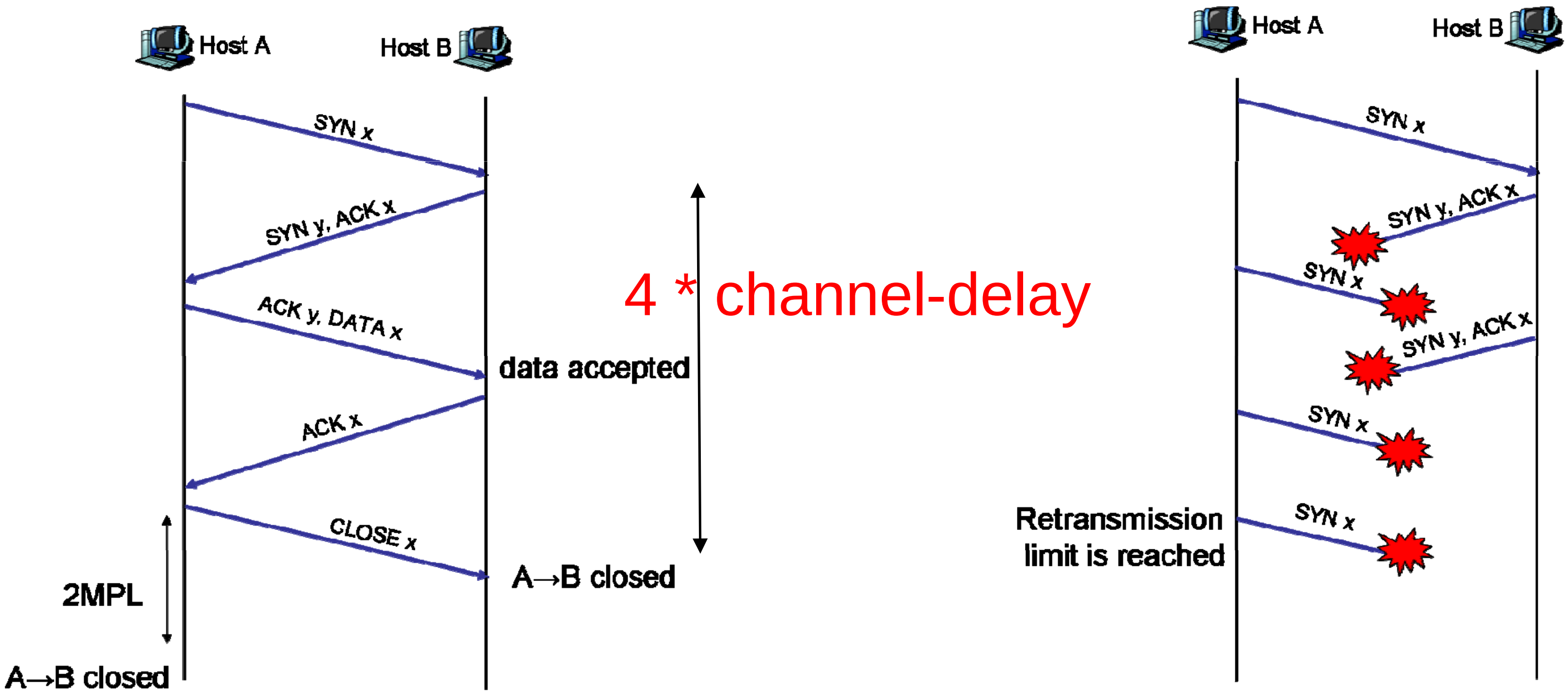
- Revisit connection management for reliability, i.e. to ensure no data loss and no data duplication
- Previous studies focused on correctness
- Here we focus on performance and robustness
- We consider worst-case single-message conversation
  - No flow / congestion control
- We compare four approaches:
  - Two-packet exchange (DATA + ACK)
  - Three-packet (… + CLOSE)
  - Five-packet (ala TCP)
  - Delta-t

Reliable One-Message Delivery using five-packet handshaking

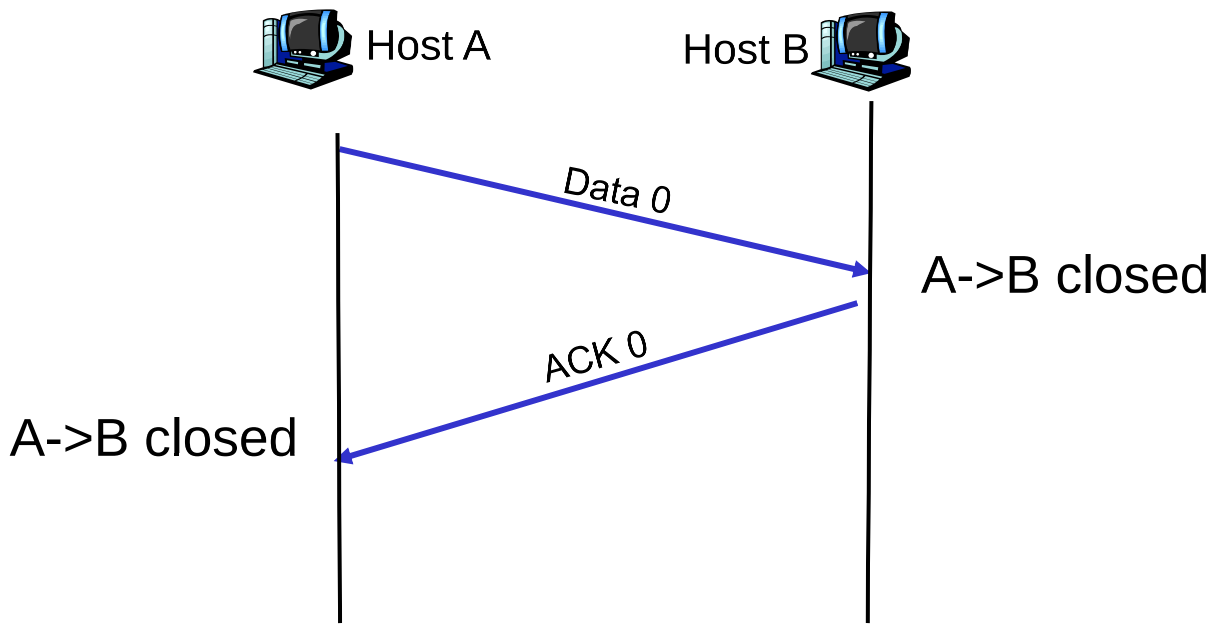


Five-Packet Protocol (ala TCP)

- Explicit handshaking: SYN and SYN+ACK messages
- For single-message communication, TCP uses five-packet protocol + timers (HS+SS)
- Vulnerability: Aborted connections ☹

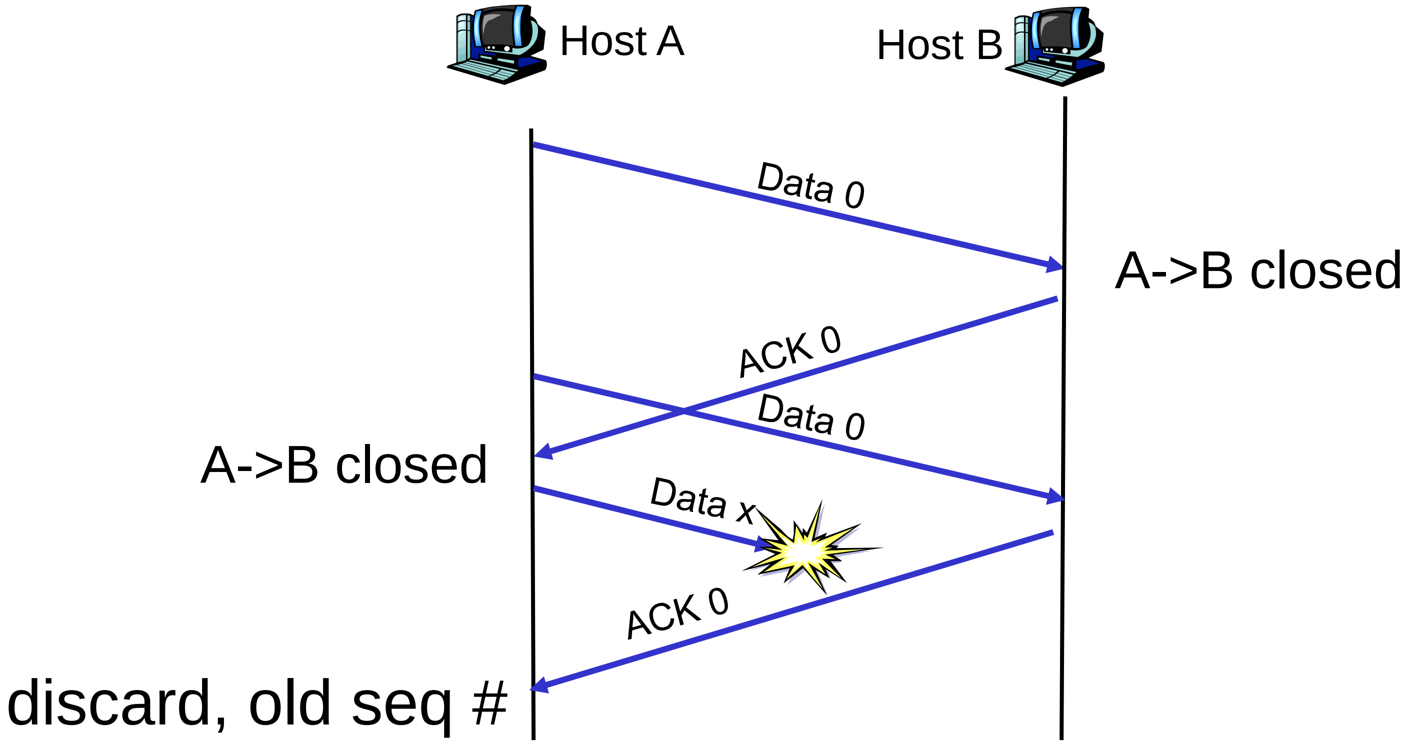


Two-packet exchange [Belsnes 76]



- Premature timeout results in duplicate
- Duplicate ACK may ACK a lost “new Data 0”

Two-packet exchange [Belsnes 76]



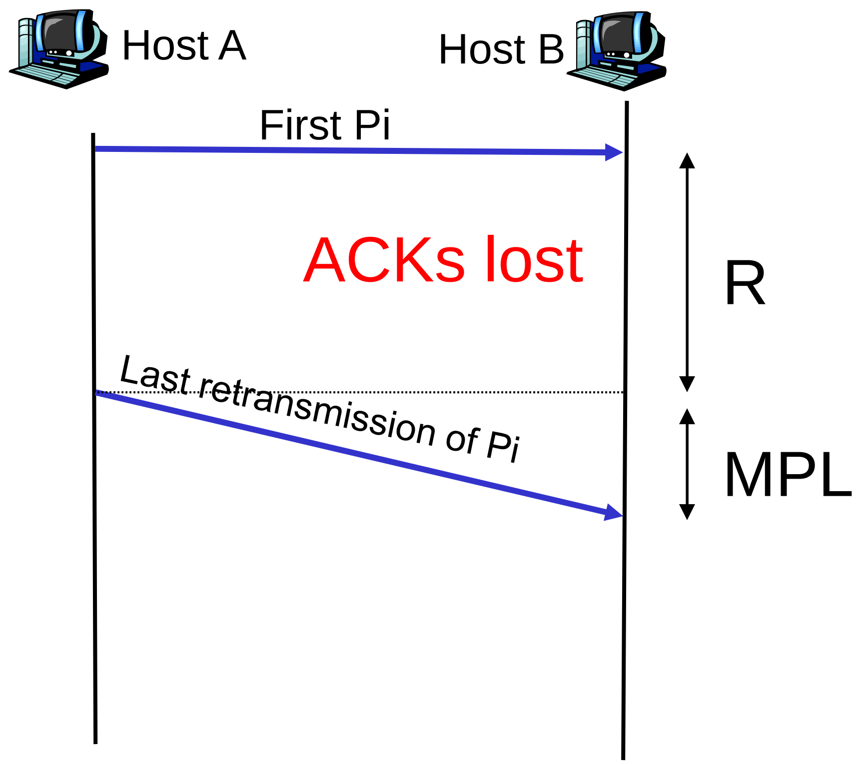
- Solution to lost data:
  use a new seq # that does NOT wrap
  around for at least 2 * MPL (Max Packet Lifetime)
- Duplicates still possible if ACK is lost,
  even with RTO > 2 * MPL

Delta-t [Watson 78]

- Two-packet exchange suffices if we can leave it to applications to detect duplicates

- Delta-t solves the duplicate problem of two-packet using appropriate timers for keeping conn. state

Delta-t: Conn. Open [Watson 78]

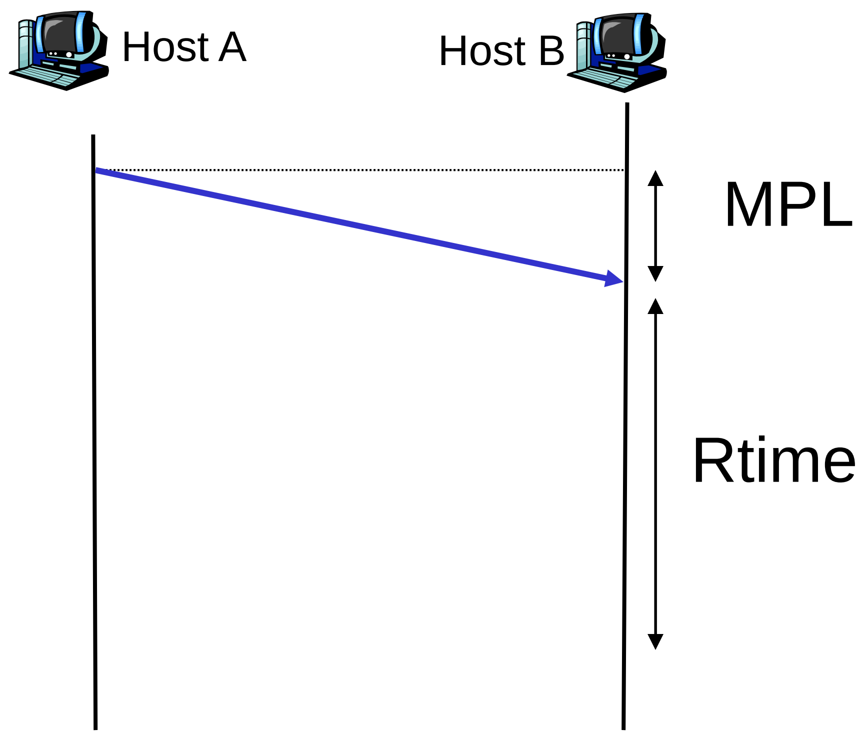


- Delta-t receiver does not delete state for at least

  Rtime = R+MPL

  enough for duplicates to die out
- R = max time for retransmission attempts
- Rtime reset at every reception of new in-seq packet

Delta-t: Conn. Close [Watson 78]

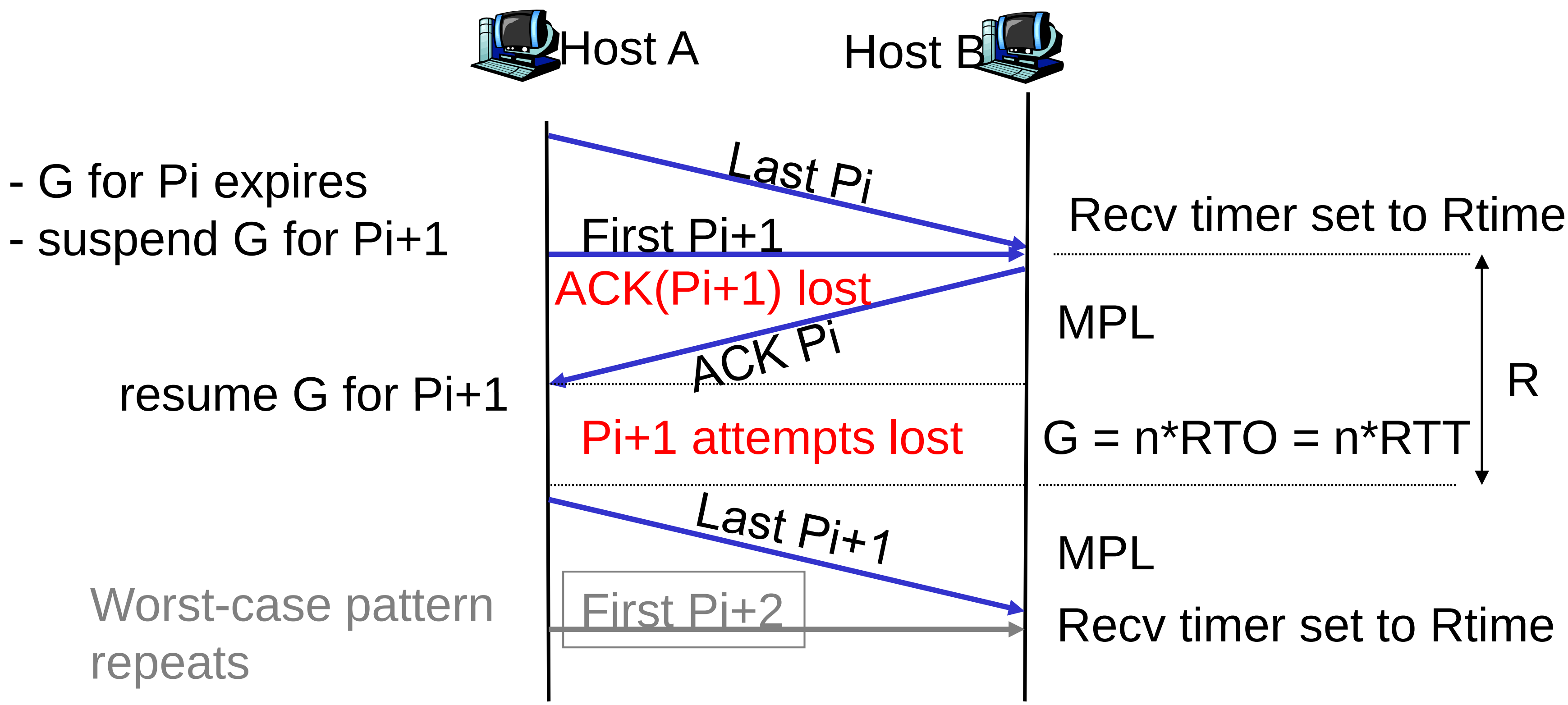


- Delta-t sender does not delete state for at least

  Stime = Rtime+MPL

  enough to ensure sender does not delete state before receiver
- Stime reset at every transmission

Delta-t: Timers [Watson 78]



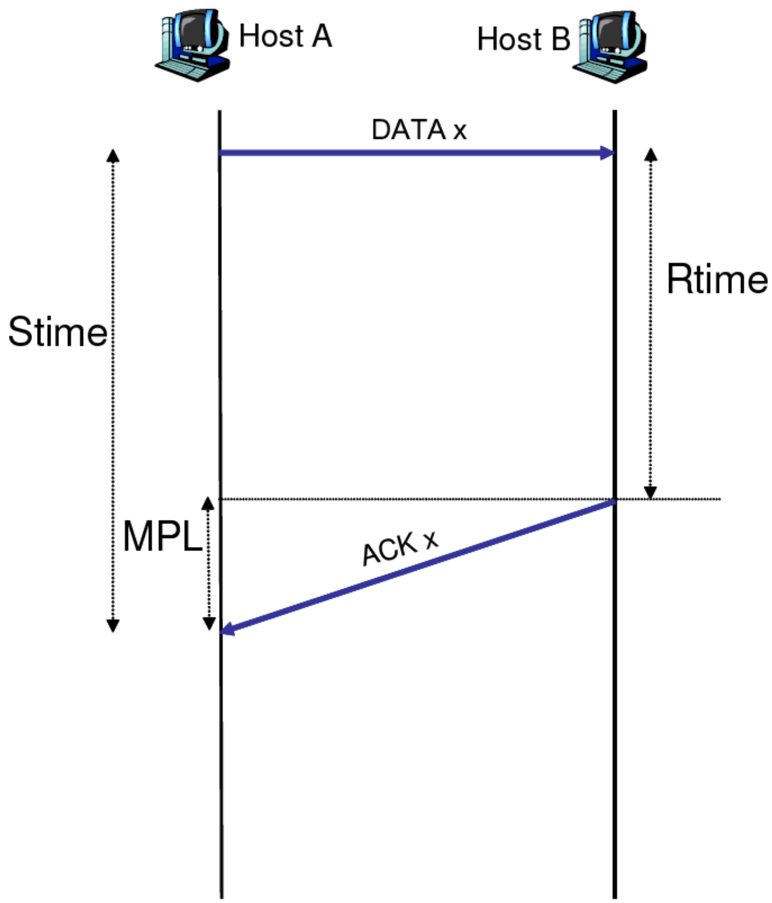
- Rtime >= R + MPL = (MPL + G) + MPL ~ 2MPL, if MPL>>G
- Stime >= Rtime+MPL ~ 3MPL

* Figure ignores UAT

Moral of the Story

- We need timers anyway
- We need to know something about MPL anyway
- We may need to reliably send a single message, or a stream of messages
- We should just use Delta-t anyway ☺
- No need to worry about init seq # since conn. ID / state is not released (re-used) until all its packets have died out

Delta-t Protocol (Watson 81)



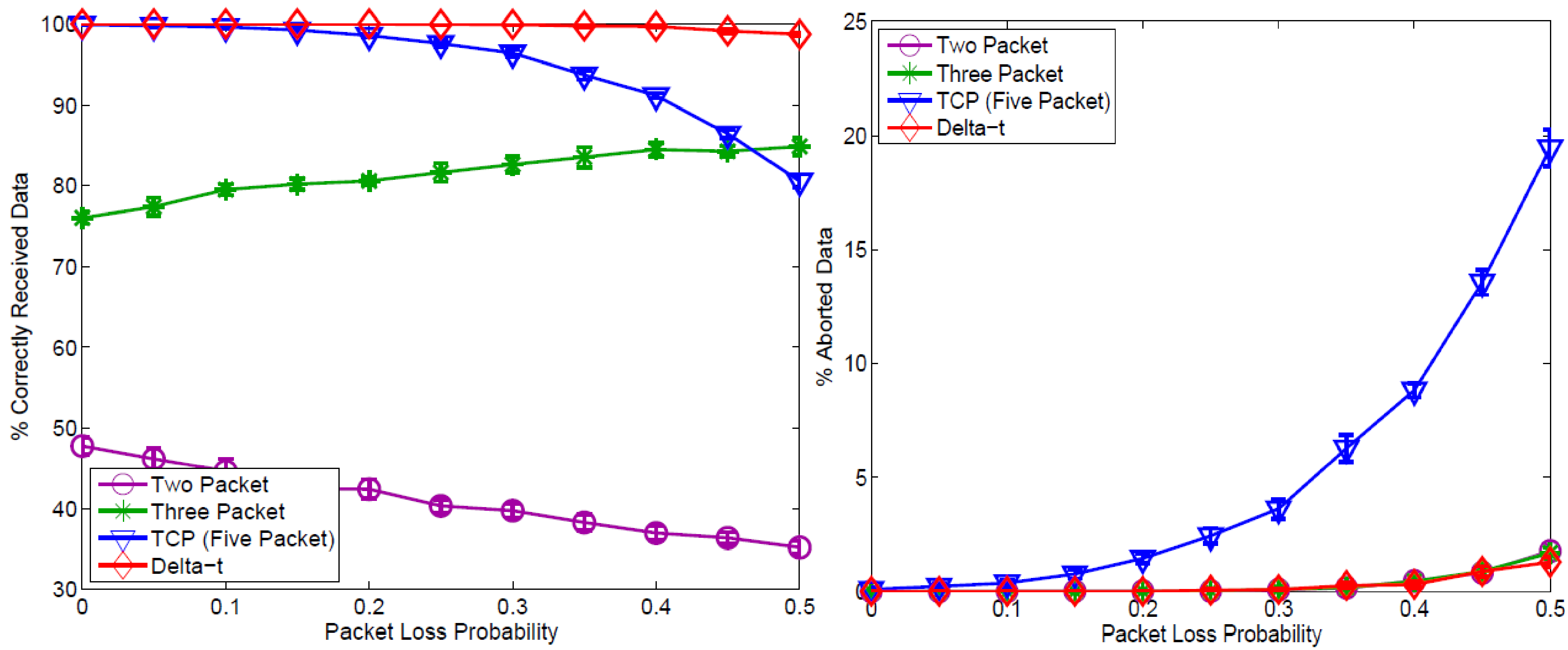
- A pure SS approach
- Two-Packet Protocol (Belsnes '76) with timers
  - Assumes all connections exist all the time
  - TCBs are simply caches of state of ones with recent activity
- G = n x RTO
- Rtime = 2MPL + G + UAT
- Stime = 3MPL + G + UAT

Rtime ~ 2 MPL > 4 channel-delay

- **Memory requirement is not a concern**
  - o only few MB needed at Delta-t receiver (server) in a typical setting
- **We should revisit MPL: should be seconds rather than minutes!**

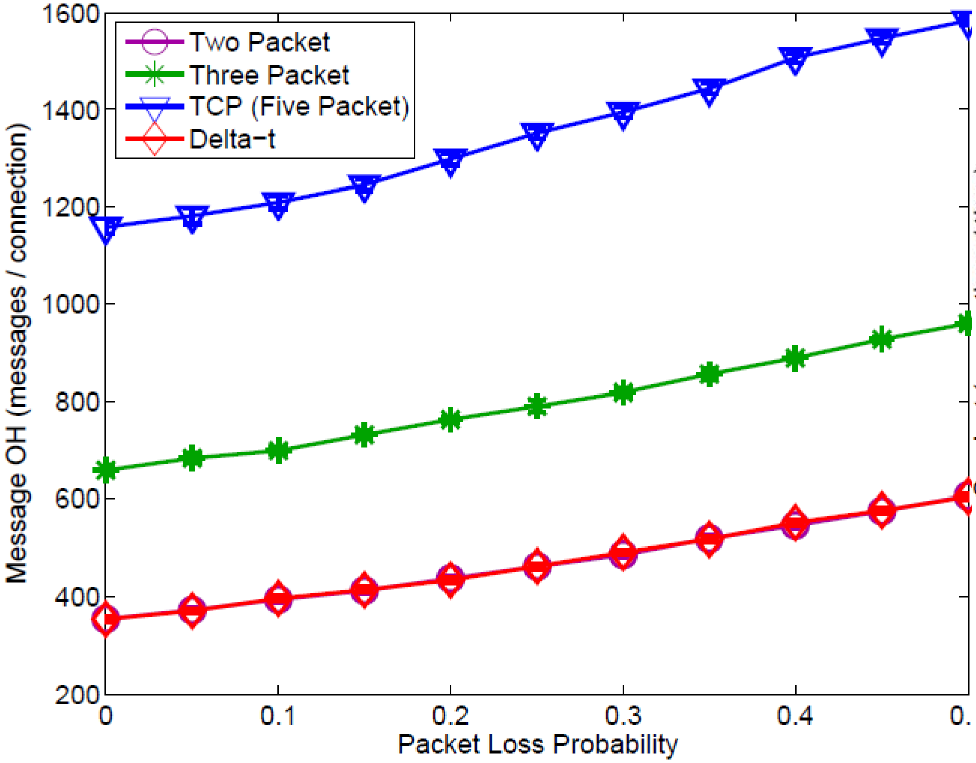
Simulation Results: Correctness

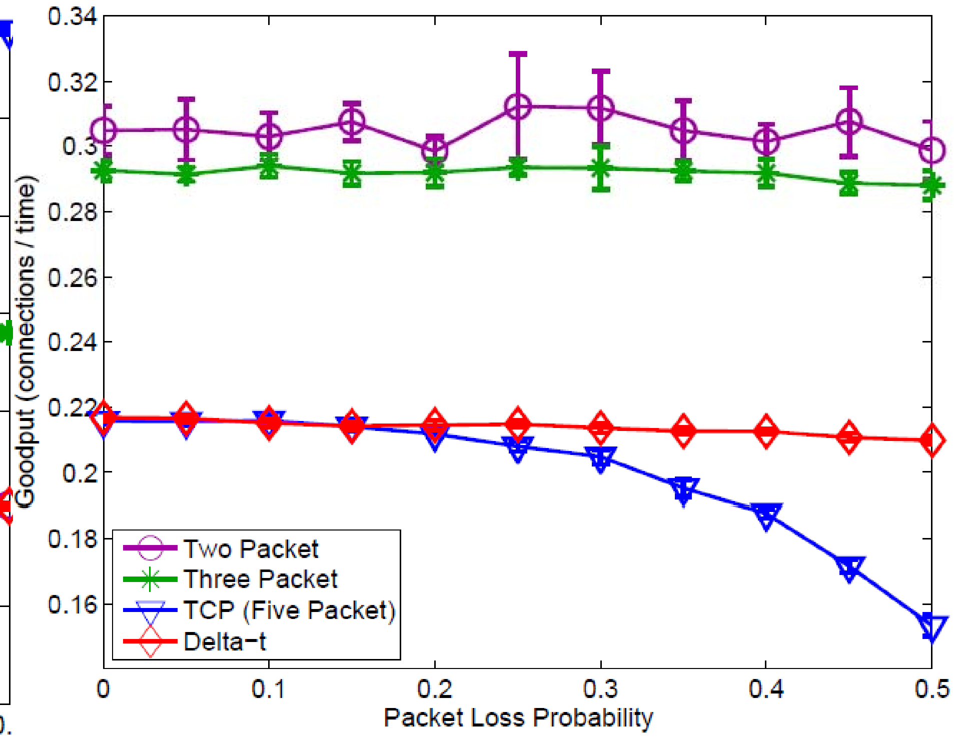
- Two-state channel-delay model, random initial sequence numbers



- SS (Delta-t) is more robust to bad net conditions

Simulation Results: Performance





- SS (Delta-t) has higher goodput and lower message overhead than HS+SS (TCP)

Conclusion

- SS is more robust to high packet losses and channel delay variations
  - No explicit handshaking messages for opening and closing connections
- SS can more easily establish its connections while delivering data reliably
- In our RINA architecture, port allocation and access control is decoupled from data transfer
  - Data transfer is done in an SS fashion
  - Port allocation and access control is HS
  - More @ http://csr.bu.edu/rina

RINA A Better Network Structure