Designing DCCP: Congestion Control Without Reliability

Eddie Kohler — UCLA

Mark Handley — UCL

Sally Floyd — ICIR

SIGCOMM 2006

12 September 2006

“Case study of a badly designed protocol”

Eddie Kohler
UCLA

Mark Handley
UCL

Sally Floyd
ICIR

SIGCOMM 2006
12 September 2006

Hot Topics in Networking: Sequence Numbers!!!

Eddie Kohler
UCLA

Mark Handley
UCL

Sally Floyd
ICIR

SIGCOMM 2006
12 September 2006

Transport protocols

- Transport: generalization of application needs
- TCP: reliable congestion-controlled byte stream
- SCTP: reliable congestion-controlled packet stream**s**
- UDP: unreliable any-rate datagrams
- Missing protocol: congestion-controlled *unreliable* delivery

  Can do above UDP, but difficult for applications

The applications

- Long-lived, high-bandwidth unreliable flows on the public Internet
  - Should use congestion control for safety, fairness
- Streaming/interactive media, games, IP telephony . . .
- **Prefer timeliness to reliability**
  - Old data isn't useful, delays new data; why waste bandwidth on it?
- A latent desire for good network citizenship

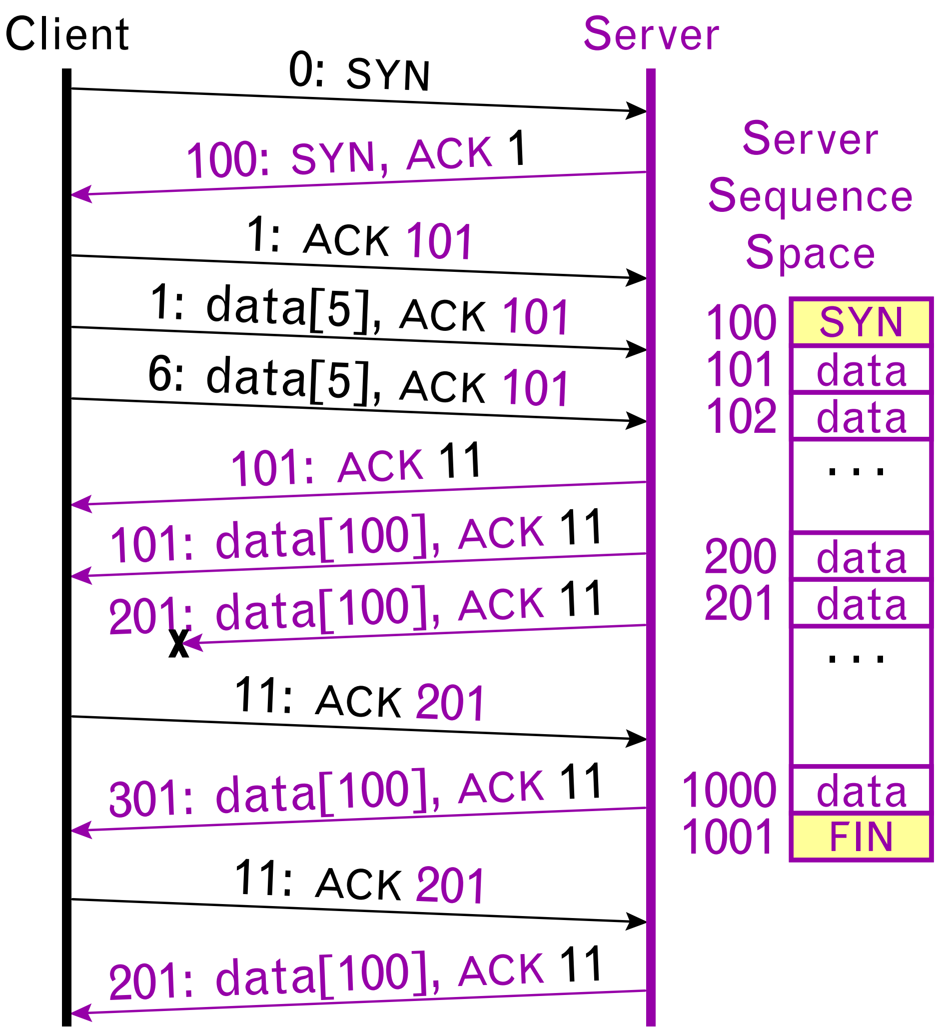
Datagram Congestion Control Protocol

- A transport protocol for congestion-controlled flows of unreliable datagrams
  - Goal: API as simple as UDP
  - Support new congestion control algorithms, as applications require them (initially TCP-like, TFRC)
  - Hope to ease safe deployment of new applications
- Proposed Standard RFC March 2006

Design challenges

- Expected DCCP design to go smoothly
  - Expected to reuse TCP mechanisms
- It did not (and we did not)
- Case study of feature interactions in protocol design
  - Mostly required by the current messy Internet
  - Some due to our choices

TCP



- Byte sequence numbers
  - SYN and FIN in sequence space
- Cumulative acknowledgement
- Windowed flow control
  - Congestion control an outgrowth
- Retransmissions

Toward DCCP sequence numbers

- UDP datagram-oriented API

    Datagram sequence numbers

- What about nondata-grams?

    Like TCP, want connection state in band

    Connection setup and teardown (SYN and FIN)

    Acknowledgements

    Connection feature negotiation

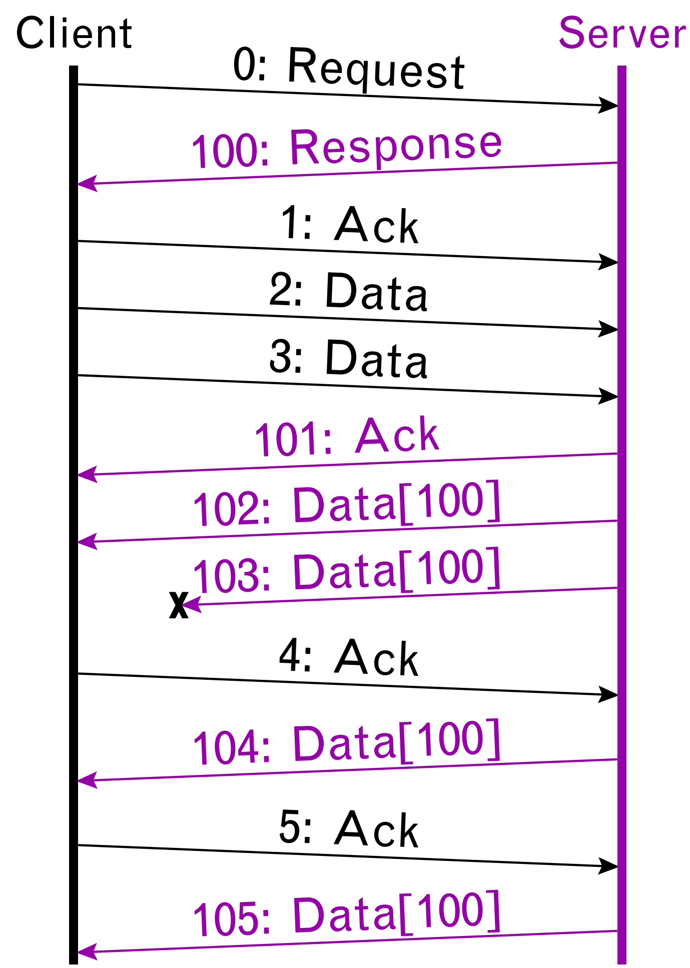
DCCP sequence numbers

- **Every packet** occupies sequence space

+ Uniform ack mechanism for features and options
+ Unambiguous ack relationship
+ Detect ack loss (enables ack congestion control)

– Lost packet: data or ack? Complicates CC

– . . .



Toward DCCP acknowledgement numbers

- TCP: cumulative acknowledgement
  - First sequence number not received
- DCCP is unreliable
  - Once lost, never found
  - Application-level retransmissions not part of protocol for minimality (easy to layer on top)

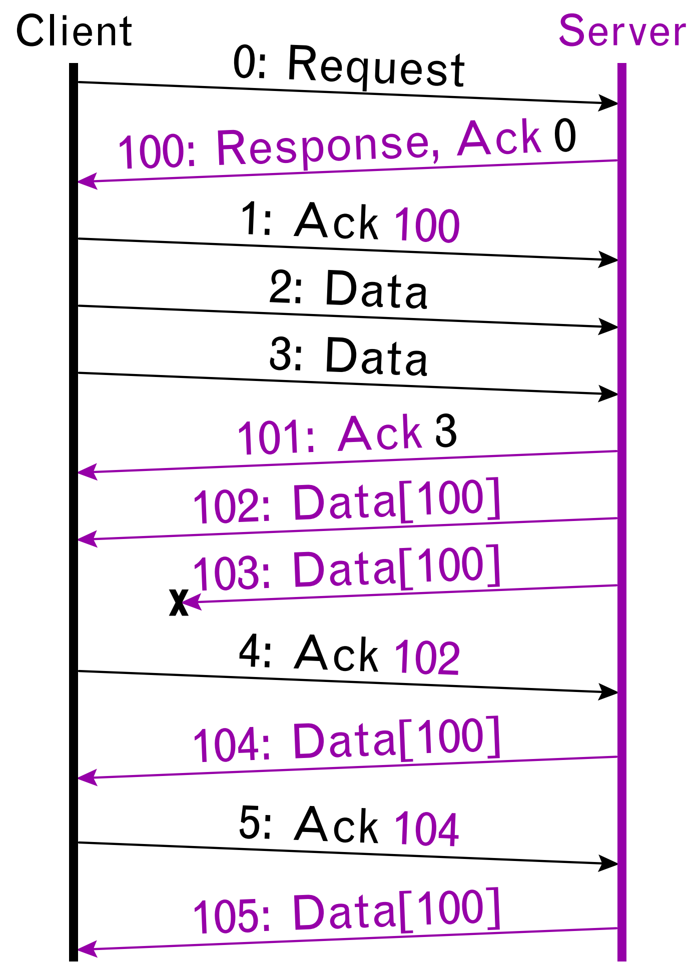
DCCP acknowledgement numbers

- Acknowledge **latest packet received**

+ Inevitable absent flow control

+ Supports different ack formats

– . . .



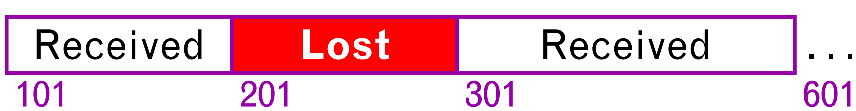
TCP acks

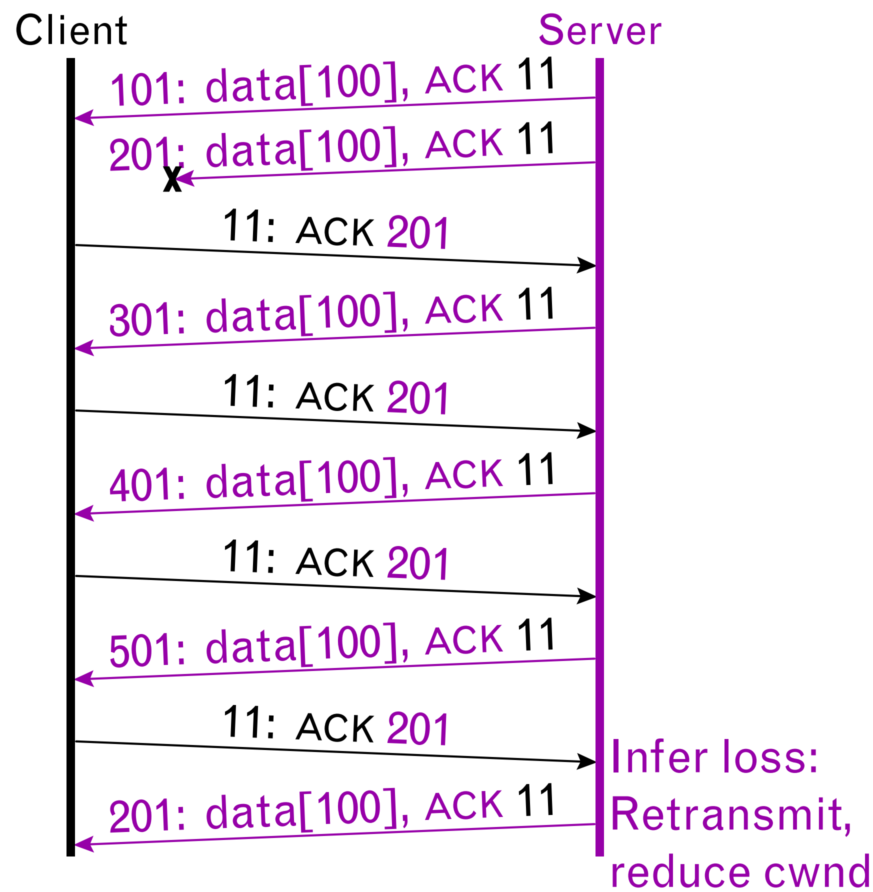
- Acks report losses

    Sender infers packet losses from receiver's timeouts, duplicate acks, SACK information

    Groups losses into loss events, reduces congestion window accordingly

    ⇒ Infer SACK scoreboard





TFRC acks

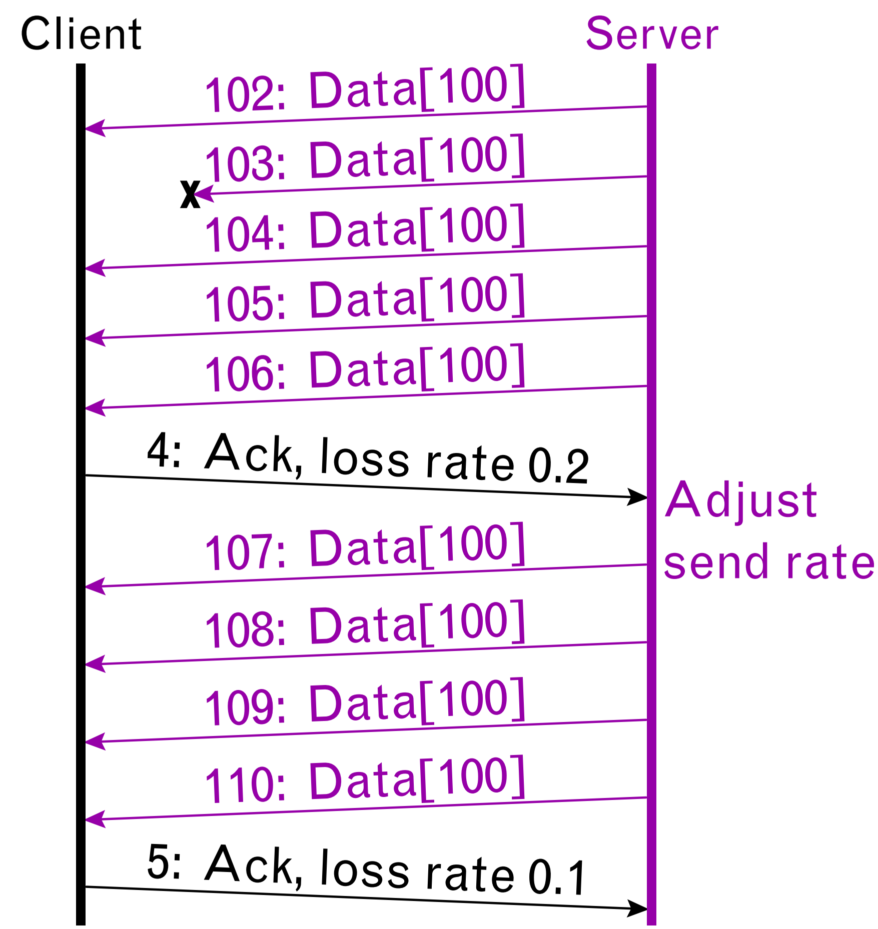
- TFRC: rate-based

    Receiver calculates a loss rate based on recent loss events

    Reports loss rate ∼ once per RTT

    Sender uses loss rate to calculate a TCP-friendly send rate

    No scoreboard needed



DCCP supports either style

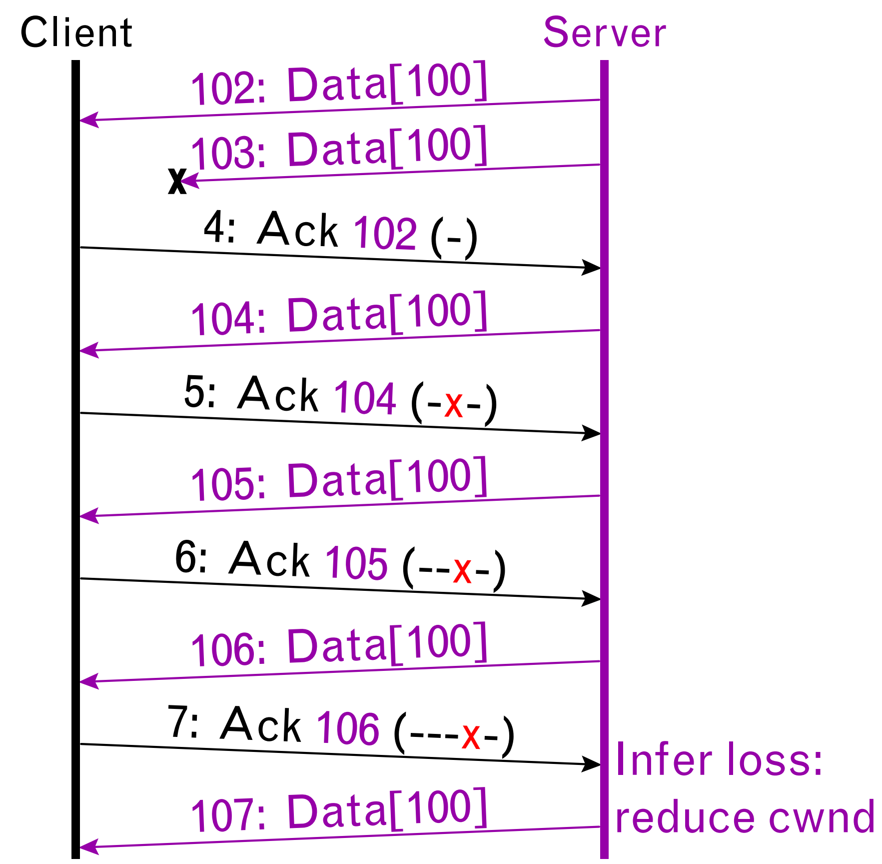
- TCP-like: Ack Vector →

  Run-length-encoded scoreboard

  Runs backwards from latest packet received

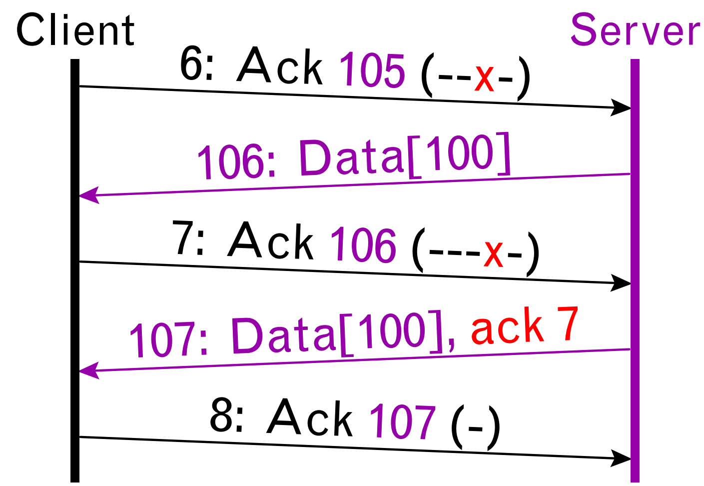
- TFRC: Loss Event Rate

  Sent once per RTT



Acknowledgement state and acks of acks

- Acks must be reliable, though protocol is not
  - Ack Vector state grows without bound!
- Must occasionally acknowledge an acknowledgement

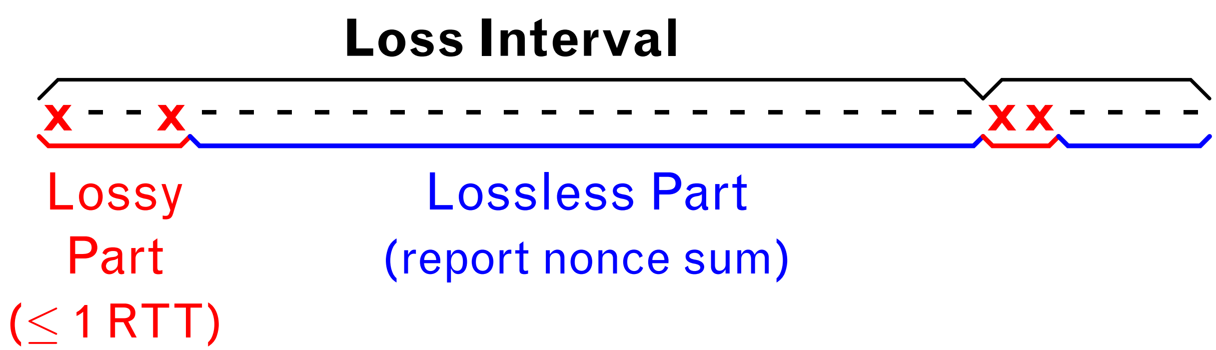


- TCP requires cumulative ack, but this takes bounded state
  - TCP SACK can grow, but only a hint

Detecting receiver misbehavior

- Receiver has incentive to pretend losses didn't happen [SCWA99]
- Critical problem for an unreliable protocol!
- Solution: echo per-packet nonce [ECN]
- TFRC's loss event rate cannot be checked!

  New Loss Intervals option: report lengths of loss intervals and relevant nonce sums



- Simpler than TCP

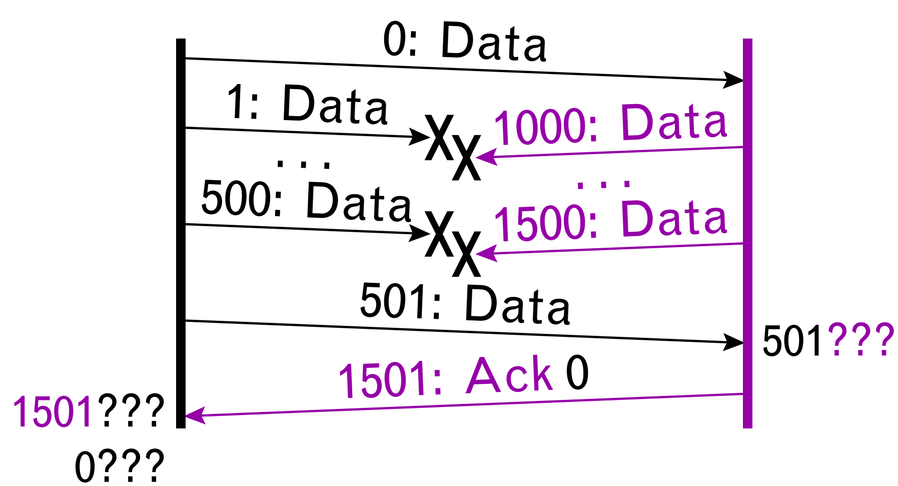
Connection synchronization

- Network failure or bad luck: many packets lost in succession
- TCP: network probes are pure acks or retransmissions

  Always use expected sequence numbers
- DCCP: each probe gets a new sequence number!

  Endpoints in odd sequence space when connectivity returns

  How to get back in sync?



Synchronization attempts

- Pure acks à la TCP?

    Then can't distinguish out-of-sync traffic from sync attempt

- Options?

    Don't want to parse options on out-of-sync packets

- Flow control?

    Artificial limitation for timely applications

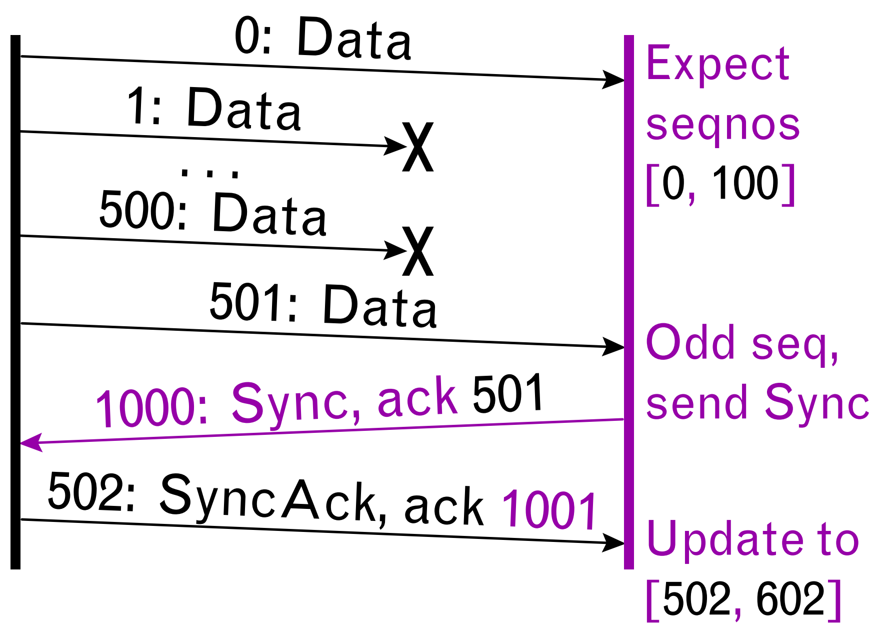
    And wouldn't help: pure acks use sequence space

Synchronization design

- Special Sync and SyncAck packets recover synchronization

  Challenge (Sync)/response (SyncAck)

- Sequence number checks on Sync and SyncAck more lenient



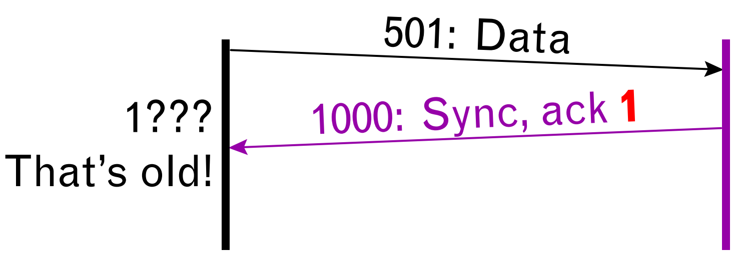
Synchronization subtleties

- An out-of-sequence packet arrives with seqno *A*
- Send Sync with new seqno and with ackno *A*

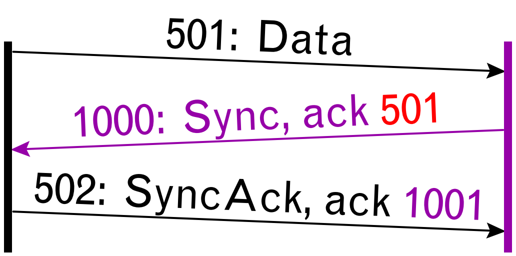
  Does not imply that *A* was processed!

  Using the expected seqno would confuse sender: can't differentiate actual Sync from old Sync or attack

Bad



Good

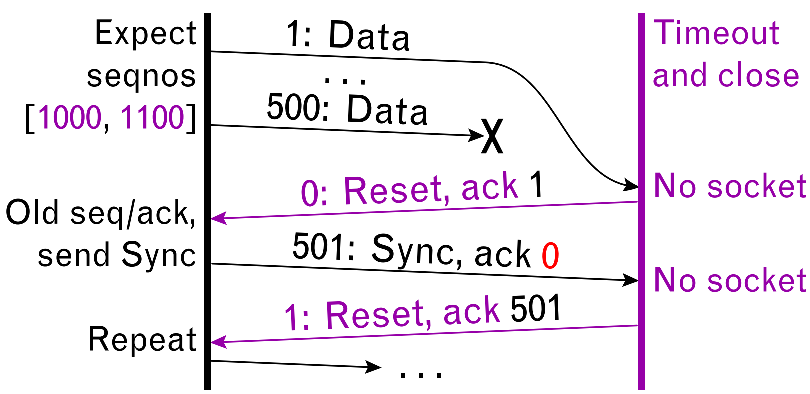


Formal modeling

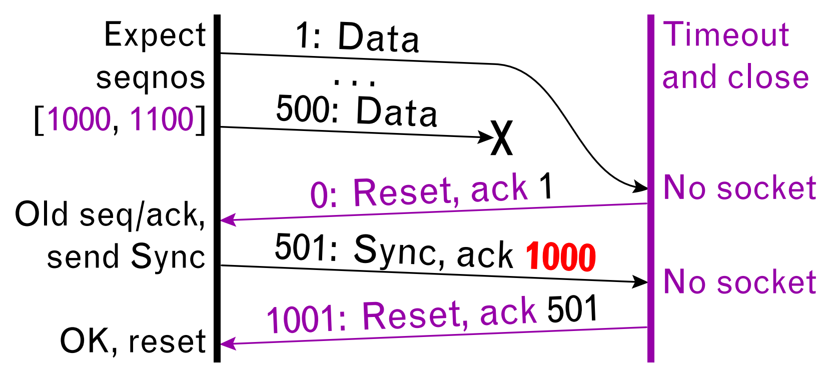
- “Hi, our verifier takes forever on your protocol.” —Somsak Vanit-Anunchai
- Previous algorithm works correctly for all packets *except Reset*
- After Reset, half-open connection

  Give closed end enough information to shut down open end

Bad



Good



Simplicity?

- Did DCCP choose efficiency over simplicity?
- Simplicity means many things. We wanted minimal mechanism.

  Rather than solve a problem many times, prefer a parsimonious yet general mechanism that can solve several problems at once.

  Example: sequence numbers. Many aspects of this design still seem successful—ack formats, explicit synchronization
- Sometimes, the accusation fits

  Data packets have no acknos

  Saves header space (e.g. 8B telephony payload)

  But must Sync some Resets, Reset synchronization . . .

Conclusions

- Still space for new transport protocols
- Appreciate unified mechanisms of TCP

    Flow control $\Leftrightarrow$ synchronization

    Cumulative ack $\Leftrightarrow$ stateless ack $\Leftrightarrow$ unidirectional communication

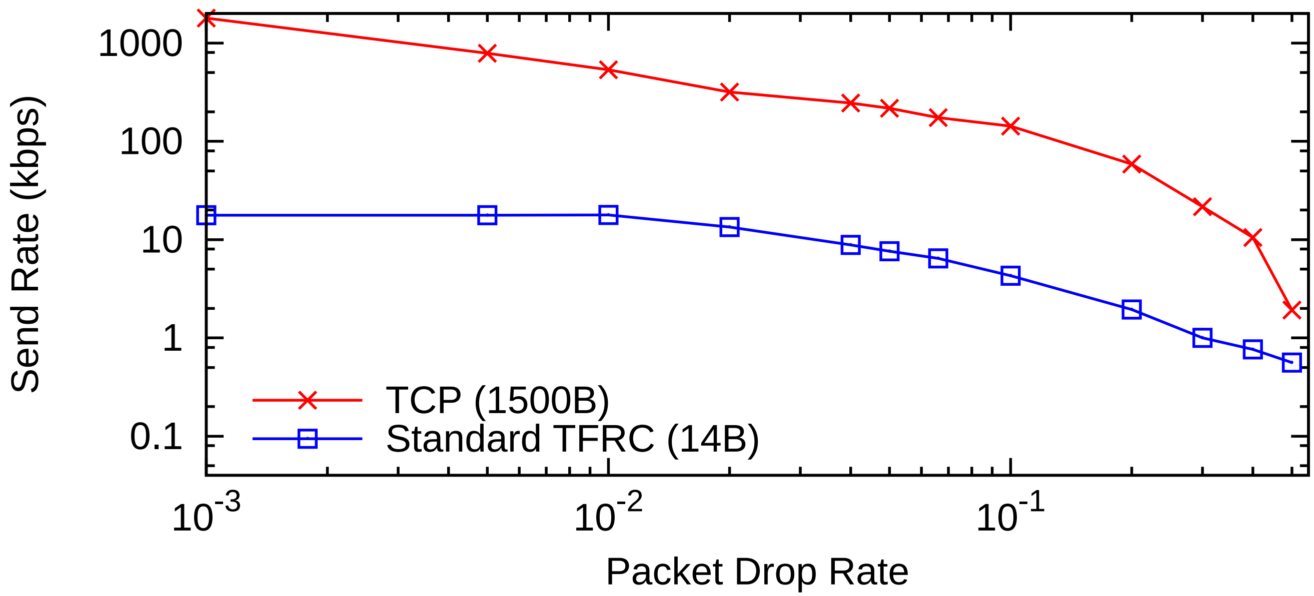
- Appreciate where TCP's unified face masks dangers below

    Robustness against attack

    Ack ambiguity

Congestion control for unreliable applications

- Example issue: packet size
- Delay-sensitive applications send many small packets
  - Necessary to meet latency constraints
- But TFRC's throughput equation matches the bandwidth of a TCP flow with the *same* size packets



Small-Packet TFRC

- Solution: compensate for packet size
- TFRC-SP's throughput equation matches the bandwidth of a TCP flow with *1500-byte* packets
    - Can send multiple pkt/RTT even in the face of persistent losses
    - Caveats: Min Interval, bottleneck types