Protocol Implementation Challenges

Today: Silliness

Observation

- A Protocol defines how to say things on the wire
  - “here’s a bunch of bytes”
  - “please send me some more bytes”
- A protocol *Implementation* defines what things get said when
  - “of all that I can send, how much should I send?”
  - “how long should I wait before asking for more data?”
  - “should I resend some data that I’ve already sent”
- In other words:
  - The protocol gets it right
  - The implementation gets it efficient.
- Implementations evolve over time as we get smarter

TCP: A Protocol For All Seasons

- Consider these uses for TCP:
  - One-way bulk data transfer
  - Telnet
  - General request/response (RPC)
    - One request at a time
  - Pipelined RPC
    - Many requests at a time
    - Eg, Web Server

Some Silliness

- Silly Receiver Windows
  - Receiver's buffer is full.
  - Reads one byte
  - Advertises a one byte window
  - Sender sends one byte
  - Rinse, Lather, Repeat
- Why do we care?
  - SEND:
    - (21 byte TCP 'x'+20 byte IP)
    - 4000% overhead!
  - Plus Ack

Clark: Avoiding Silly Receiver Window

- Change to IMPLEMENTATION
  - Not Protocol
- Receiver should not advertise a new window until
  - It can handle a large packet (MSS), OR
  - Recv buffer is half full
- Means *delaying acks*
  - But don't delay indefinitely
  - Spec say: "must ack at least every 2*MSS bytes, and no later than 500ms after segment receipt"
- Allows receiver to drain slowly without "bothering" the network or the sender.

Silly Senders

- Consider Telnet/ssh:
  - KEY STROKE:
    - CLIENT: TYPE ‘x’, (21 byte TCP ‘x’+20 byte IP)
    - **SERVER**: ***(20 byte TCP ACK + 20 byte IP), READ x, (20 byte TCP Window Size+20 byte IP), ECHO x***
  - *KEY ECHO:*
    - **SERVER:** ***(21 byte TCP ‘x’ + 20 byte IP)***
    - CLIENT: READ x, PRINT x, get next keystroke.
- More generally, Silly Sender Windows (“the small packet problem”)
  - Sender sends 1 byte.
    - (21 byte TCP ‘x’+20 byte IP)
    - 4000% overhead!
  - Receiver acknowledges.
  - Receiver reads 1 byte.
  - Receiver advertises a new window.
  - Rinse, Lather, Repeat

Nagle:
Foil Silly Senders

- Concatenate Send Buffers
  - When sender is has less than a full sized segment:
    1. Send what's available (eg, maybe just one byte)
    2. Buffer until last sent byte acknowledged
    3. On acknowledgement, send all buffered characters.
    4. Go to 2.

    Also, can send if have MSS bytes ready, or window half full)
  - If network is slow, segments carry a lot of data (good bandwidth).
  - If network is fast, segments are acked quickly after they are produced (good latency)
  - Not always appropriate:
    - Eg, erratic "mouse movements"

No Silliness Nowhere: Nagle+Clark

- Goal: Sender shouldn't send small segments and receiver should not ask for them.
  - Receiver avoids advertising small TCP windows and delays acks
  - Sender delays transmission of partially filled segments until all previously transmitted data has been accepted.
- Each is simple and makes a lot of sense
  - Prevent the network from becoming congested with small packets.
- What happens when you combine them?

Temporary Deadlock

- Nagle prevents sender from transmitting *more data* until it receives an outstanding **ACK**
- Delayed ACK keeps the receiver from generating an **ACK** until it gets *more data*.
- Timeout breaks the deadlock
  - But 200-500ms for an acknowledgement can really hurt some transfers

Example of Interaction

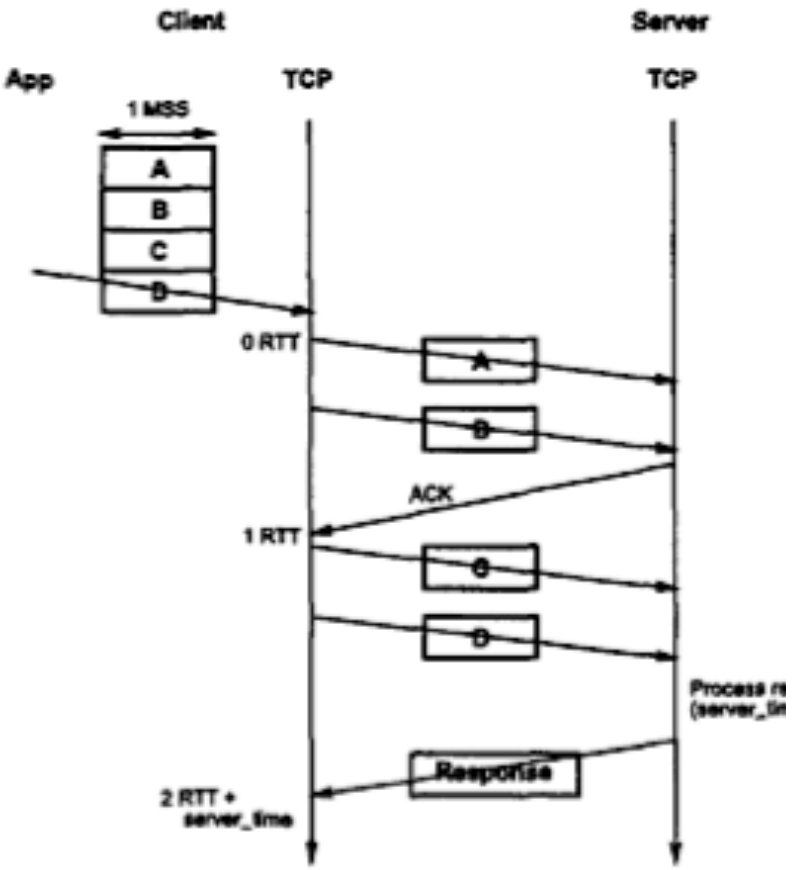

Figure 1: Example of successful transfer

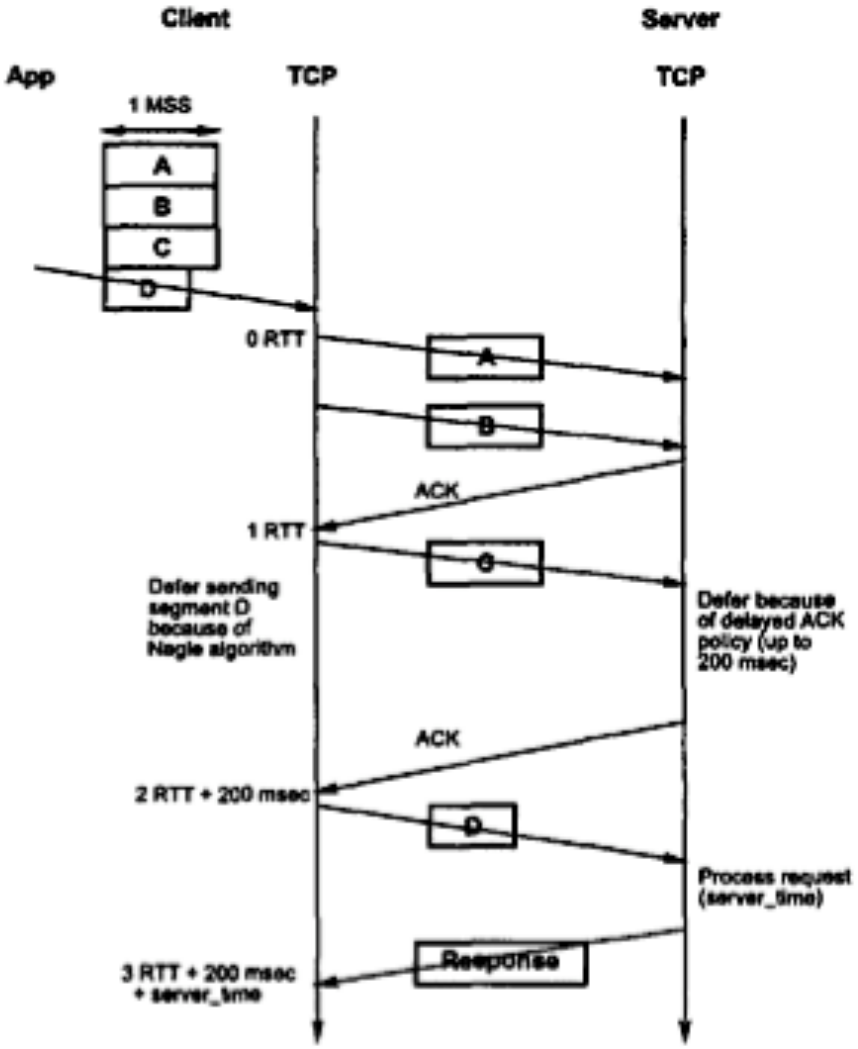

Figure 2: Example of Nagle-delayed transfer

Summary

- TCP the protocol vs TCP the implementation
- Effects can occur in any window-based protocol
- Beware subtle interactions