The University Of Alabama in Huntsville Computer Science

cs670

TCP Implementation

G. W. Cox – Spring 2008

TCP Implementation 1

The University Of Alabama in Huntsville Computer Science

cs670

TCP Implementation

- ***A quick review of TCP***
- **TCP implementation options**
- **Some other implementation considerations**
- **Optimizing TCP's performance**
- **TCP over wireless networks**

G. W. Cox – Spring 2008

TCP Implementation 2

The University Of Alabama in Huntsville Computer Science

A byte-stream oriented protocol

cs670

- **TCP is designed to treat data as a generic stream of bytes.**
- **Pays no attention to message boundaries, etc.**
- **Bytes are sequence-numbered by the sender (32-bit seq number)**

G. W. Cox – Spring 2008

TCP Implementation 3

The University Of Alabama in Huntsville Computer Science

TCP transfer protocol

cs670

- **Extended form of sliding window algorithm**
  - **Review of SWA:**
    - **All segs have sequence number**
    - **Multiple segs can be in flight**
    - **Sender starts timer when transmitting each seg**
    - **Receiver sends ACK for seg when all previous segs have been received**
    - **If send timer times out before seg is ACKEd, sender re-sends the seg**

G. W. Cox – Spring 2008

TCP Implementation 4

Header

The University Of Alabama in Huntsville Computer Science

cs670

4 Bytes

Source port #		Dest port #	
Sequence #			
ACK #			
Header Length		Flags (6)	Window Size
Checksum		Urgent Pointer	
Options			

TCP header flags

The University Of Alabama in Huntsville Computer Science

cs670

- **URG – Indicates seg contains urgent data**
- **ACK – Indicates this is an ACK seg**
- **PSH – “Push”. Requests receiver to deliver data to app without buffering**
- **RST – “Reset”. NACK.**
- **SYN – Connect request / connect accepted**
- **FIN – Connection release**

Urgent data

The University Of Alabama in Huntsville Computer Science

cs670

- **A way to embed signaling in the data stream (e.g, to kill the process on the remote machine)**
- **Process:**
  - **Sending app gives signal to TCP with “Urgent” flag set**
  - **Sending TCP sets Urgent Pointer (points to end of urgent data) in seg header and immediately transmits buffer**
  - **Receiving TCP can interrupt app to transfer urgent data**

TCP flow control

The University Of Alabama in Huntsville Computer Science

cs670

- **Implemented by “Window Size” field in header**
- **Window size set by receiver to indicate how many bytes (not segs) can be sent before next ACK**
- **Returned in ACK seg**

Segment size

cs670

- Seg = 20 header bytes
  + ? Option header bytes
  +0-~64KB data
- Max seg size = 64K-20 (max IP payload)
- Actual seg size determined by sending TCP:
  – Local MTU
  – Local sending rules
  – Truncated seg for Urgent Data

The University Of Alabama in Huntsville Computer Science

G. W. Cox – Spring 2008

TCP Implementation 9

Well-known ports

cs670

- If a remote node wants to obtain a particular service from a server, how does it know which port # to make the request on?
- "Well-known ports" (ports 1-1023) are reserved for standard services:
  – E.g, 21=TCP, 80=HTTP, 110=POP3
  – For the entire list, see http://www.iana.org/assignments/port-numbers

The University Of Alabama in Huntsville Computer Science

G. W. Cox – Spring 2008

TCP Implementation 10

cs670

TCP Implementation

- A quick review of TCP
- *TCP implementation options*
- Some other implementation considerations
- Optimizing TCP's performance
- TCP over wireless networks

The University Of Alabama in Huntsville Computer Science

G. W. Cox – Spring 2008

TCP Implementation 11

TCP Implementation Options

cs670

- The TCP spec lays out the details of the protocol
- However, some policy options are left to the implementer
- Two implementations that use different options will interoperate, but performance may suffer

The University Of Alabama in Huntsville Computer Science

G. W. Cox – Spring 2008

TCP Implementation 12

TCP Imp Options: Send Policy

The University Of Alabama in Huntsville Computer Science

- **The sending TCP accepts bytes from the app and buffers them**
- **It is free to send them whenever it chooses (except for pushed data, urgent data, or a closed send window)**

TCP Imp Options: Deliver Policy

The University Of Alabama in Huntsville Computer Science

- **Receiving TCP will receive segments and buffer them, handling errors and in-order considerations.**
- **It is free to deliver the data to the app wherever it chooses (except for urgent data or pushed data)**

TCP Imp Options: Accept Policy

The University Of Alabama in Huntsville Computer Science

- **When segs arrive out of order, the receiving TCP can:**
  1. **Discard any segs that arrive out of order**
  2. **Accept any seg that has a sequence number within the receive window**
- **Note: Policy 1 is easier to implement and needs less complex buffering, but Policy 2 is better for performance**

TCP Imp Options: Retransmit Policy

The University Of Alabama in Huntsville Computer Science

- **Governs how the sender will handle a re-transmission**
  - **First-Only Policy**
  - **Batch Policy**
  - **Individual Policy**

Retransmit Policy options: First-Only Policy

cs670

- Keep a send-order queue of unACKed segs.
- Keep a single timeout timer
- When an ACK arrives, remove the ACKed seg(s) from the queue and reset the timer
- When the timer times out, re-send the seg at the head of the queue

- Simple and low-traffic, but can be slow (the timer for the second seg in the queue doesn't start until the first one times out)

Retransmit Policy Options: Batch Policy

cs670

- Same as First-Only except:
  - When the timer times out, re-send the entire queue

- Basically a go-back-n approach. Simple, but may cause needless additional traffic

Retransmit Policy Options: Individual Policy

cs670

- Keep a timeout timer for each seg in the queue
- If any timer expires re-transmit just that seg

- Complex implementation. Very traffic efficient.

Retransmit Policy Options

cs670

- Ideally, the retransmit policy would be selected to be compatible with the receiver's accept policy (e.g., if receiver uses in-order, the best match is a batch retransmit policy).
- But you can't count on that in real networks

TCP Imp Options: ACK Policy

cs670

- The receiving TCP must ACK segs that are received in order. It can do so:
  - Immediately – Immediately send an empty ACK segment for the received data seg
  - Cumulative – Hold the ACK until it can be piggybacked on outgoing data (recall that ACK applies to the seg received and all before it). Keep a timer to prevent too long a delay.
- Most installations use Cumulative because it yields lower traffic loads, but this is a good bit more complex to implement and manage.

The University Of Alabama in Huntsville Computer Science

cs670

TCP Implementation

- A quick review of TCP
- TCP implementation options
- *Some other implementation considerations*
- Optimizing TCP's performance
- TCP over wireless networks

The University Of Alabama in Huntsville Computer Science

Setting the segment size

cs670

- Max = 64KB
- Min/default = 556B
- Often restricted to 1460 data bytes:
  1460 data bytes
  + 20 TCP header
  + 20 IP header
  1500 bytes (One Ethernet payload)
- Actual Max seg size is negotiated by sender and receiver during setup

The University Of Alabama in Huntsville Computer Science

A problem with the window size field

cs670

- Window size field is 16 bits => 64KB max window
- Not enough for many purposes:
  - Example (Tanenbaum):
    - On a T3 line (44.7 Mbps), it takes 12 msec to send a 64KB window.
    - If RTT is 50 msec, sender is idle 75% of the time waiting for ACKs

The University Of Alabama in Huntsville Computer Science

A window size patch

The University Of Alabama in Huntsville Computer Science

cs670

- "Window scale" is set during negotiation*
- Sets scale factor used in interpreting the Window Size field:
  - Act_Win_Size = Win_Size_Field x 2^{Win_Scale}
- Allows window sizes up to ~2^{30} B

* RFC 1323

A potential TCP deadlock

The University Of Alabama in Huntsville Computer Science

cs670

- When the receiver wants the sender to stop sending temporarily, it will advertise a window size of 0. The sender will stop.
- Later, when the receiver can accept data again, it will advertise a larger window size. The sender will re-start.
- A deadlock occurs if the second advertisement is lost.

Deadlock avoidance

The University Of Alabama in Huntsville Computer Science

cs670

- When a sender receives a Window Size =0, it starts a "persistence timer"
- When the persistence timer times out, the sender sends a query to the receiver, and the receiver sends the current advertised window size
- If the size still = 0, the sender re-starts the persistence timer

The University Of Alabama in Huntsville Computer Science

cs670

TCP Implementation

- A quick review of TCP
- **TCP implementation options**
- **Some other implementation considerations**
- ***Optimizing TCP's performance***
- **TCP over wireless networks**

Optimizing performance

cs670

- *Deciding when to send a segment*
- **The Silly Window Syndrome**
- **Managing the Congestion Window**
- **Managing the timeout timer**

Deciding when to send a segment

cs670

- **Within the negotiated seg size limits, the sending TCP has to decide when to stop buffering data bytes and send them**
- **When bytes come in slowly from the app (e.g., a user typing), how does the sending TCP decide when to send the buffered bytes?**
  - **If you send each one separately, you are using 40 overhead bytes (20 send header and 20 ACK header) to send 1 data byte**
  - **If you wait to build a large seg, the first bytes typed may be impossibly late**

One way of helping the problem – delayed ACK

cs670

- **The receiver can delay ACKing a small seg (typ: 0.5 sec) in the hope of receiving another one that can be ACKed in the same ACK seg**
- **Fairly common approach, but not practical when fast response needed, and sender is still inefficient**

Another way: Nagle's Algorithm

cs670

- **When data comes in a byte at a time from the app,**
  - **Send the first byte immediately**
  - **Buffer succeeding bytes until the first byte is ACKed, then send them in one seg**
- **This is a good strategy when RTT is variable:**
  - **When network is lightly loaded, the impact of small segs is less. Since ACKs return quickly, more small segs are sent**
  - **When the network is congested, ACKs return slowly and more data is packed in each seg.**
- **Note: Nagle's performance may not be good enough for highly interactive applications – sometimes it is disabled**

Optimizing performance

cs670

- Deciding when to send a segment
- *The Silly Window Syndrome*
- Managing the Congestion Window
- Managing the timeout timer

A related problem: Silly Window Syndrome

cs670

- Occurs when data is sent in large blocks, but receiving app reads a byte at a time:
  - Sending TCP sends until rcv buffer full
  - Receiving TCP advertises window size = 0
  - Receiving app reads a byte
  - Receiving TCP advertises window size =1
  - Sending TCP sends a byte

A fix for the Silly Window Syndrome

cs670

- Receiver is prevented from advertising when only small buffer space is available
- It can only advertise when either:
  - It can handle the negotiated max seg size, or
  - half of the receive buffer is empty

Optimizing performance

cs670

- Deciding when to send a segment
- The Silly Window Syndrome
- *Managing the Congestion Window*
- Managing the timeout timer

Congestion control

The University Of Alabama in Huntsville Computer Science

cs670

- Recall that basic TCP considers two numbers to set the send window size:
  - Advertised window size – set by the receiver to prevent sender from overrunning the receive buffer (flow control)
  - Congestion window size – set by the sender to try to prevent aggravating network congestion (congestion control)
- Send window size is set to the minimum of the two numbers

G. W. Cox – Spring 2008

TCP Implementation 37

Managing congestion window size

The University Of Alabama in Huntsville Computer Science

cs670

- The basic Slow Start algorithm ("Additive increase, multiplicative decrease")
  - At start, set CW = 1 max seg size
  - If *n* segs ACKed, CW=CW + *n* max seg sizes
  - If timeout, CW = CW / 2
  - Once CW is reduced, it is not increased again

G. W. Cox – Spring 2008

TCP Implementation 38

A consideration

The University Of Alabama in Huntsville Computer Science

cs670

- The basic algorithm does not change CW after it is reduced.
- In complex networks, congestion will come and go.
- We'd like to have a scheme that reduces CW when congestion is high, but increases it when the congestion is relieved

G. W. Cox – Spring 2008

TCP Implementation 39

Refined Slow Start

The University Of Alabama in Huntsville Computer Science

cs670

- When timeout occurs:
  - Set threshold = CW/2
  - Reset CW to 1 and start Slow Start again
  - After CW >= threshold, only increase CW by 1 for each ACK received (regardless of how many segs the ACK is for)

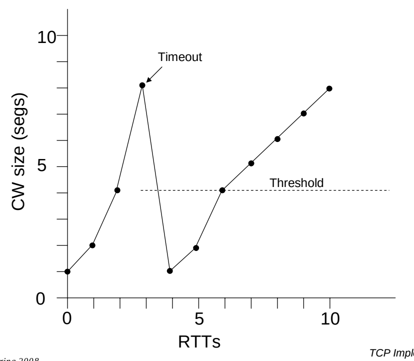
G. W. Cox – Spring 2008

TCP Implementation 40

A typical CW profile

The University Of Alabama in Huntsville Computer Science

cs670



G. W. Cox – Spring 2008

TCP Implementation 41

Optimizing performance

The University Of Alabama in Huntsville Computer Science

cs670

- **Deciding when to send a segment**
- **The Silly Window Syndrome**
- **Managing the Congestion Window**
- ***Managing the timeout timer***

G. W. Cox – Spring 2008

TCP Implementation 42

Managing the timeout timer

The University Of Alabama in Huntsville Computer Science

cs670

- **How do we set the value for the timeout timer?**
  - **Too long – performance suffers because it takes a long time to discover that a seg has been lost**
  - **Too short – many unnecessary re-transmissions, increasing network load**

G. W. Cox – Spring 2008

TCP Implementation 43

Managing the timeout timer

The University Of Alabama in Huntsville Computer Science

cs670

- **Timeout interval for a connection should be related to the RTT for that connection**
- **But on the Internet, RTT can vary wildly**
- **We would like a dynamic measure of RTT**

G. W. Cox – Spring 2008

TCP Implementation 44

Dynamic RTT measurement – simple averaging

cs670

- For each connection, TCP maintains a variable ***AvgRTT***
- When an ACK arrives, TCP calculates the RTT experienced
- AvgRTT is calculated by simple averaging
- Problem: treats long-ago behavior as importantly as recent behavior

The University Of Alabama in Huntsville Computer Science

Dynamic RTT measurement – exponential averaging

cs670

AvgRTT **= alpha x** ***AvgRTT*** **+ (1-alpha) x measured RTT**

where 0 < alpha < 1

- This formulation favors recent measurements
- Smaller alpha causes more weight on recent measurements
- Problem: works well for small variance, but does not react well to an abrupt, large change in RTT

The University Of Alabama in Huntsville Computer Science

Setting timeout from RTT

cs670

- Generally, timeout is calculated by:
  timeout = beta x ***AvgRTT***
- In the early days of the Internet, beta was set to 2
- Problem: when the variance in RTT is wide, a fixed beta may not work
- The fix: Jacobson's Algorithm tracks deviation of the measured RTT and sets beta accordingly.

The University Of Alabama in Huntsville Computer Science

Jacobson's Algorithm: The Idea

cs670

- Instead of calculating AvgRTT by averaging, take the variability of the RTT samples into account

The University Of Alabama in Huntsville Computer Science

Jacobson's Algorithm: Approach

SampleRTT = ACK_time – SEND_time

Diff = SampleRTT – Current Estimated RTT

EstimatedRTT = EstimatedRTT + δ x Diff (where $0 <= \delta <= 1$)

Dev = Dev + δ (|Diff| - Dev)

THEN:
Timeout = EstimatedRTT + p x Dev (typical p = 4)

How Jacobson's Alg acts

- **When variability is small, Timeout tends toward the long-term average of RTT**
- **When variability is large, Timeout tends toward p x (magnitude of the variation)**

A problem with timeouts

- **When network is congested, a higher fraction of attempted sends will timeout due to delay and the timed-out senders will re-send**
- **This increases the traffic that the network is trying to handle, probably increasing the congestion and lengthening delay**

A partial fix: Exponential timeout backoff

- **For each re-transmission, double the timeout**
  - **For example:**
    - **Original send – timeout = x**
    - **First re-send – timeout = 2x**
    - **Second re-send – timeout = 4x**

More problems with setting timeouts

cs670

- **Assume you send a seg, timeout, then re-send**
- **If you get an ACK, is this the ACK for the first send or the second?**
- **If you guess wrong, you will corrupt the value of *AvgRTT***

The University Of Alabama in Huntsville Computer Science

An over-all fix: Karn's Algorithm

cs670

- **Use Jacobsons' Alg to dynamically adjust RTT until a re-transmission occurs**
- **Do not use the RTT of re-transmitted segs in the *AvgRTT* calculation**
- **Use Exponential Timeout Backoff for all re-transmissions until a non-re-transmitted seg is ACKed**
- **Then, re-start Jacobson's Alg**

The University Of Alabama in Huntsville Computer Science

cs670

TCP Implementation

- A quick review of TCP
- **TCP implementation options**
- **Some other implementation considerations**
- **Optimizing TCP's performance**
- ***TCP over wireless networks***

The University Of Alabama in Huntsville Computer Science

TCP over wireless

cs670

- **The earliest versions of TCP assumed that timeouts could be either caused by congestion or by lost packets**
- **As wired network reliability improved, TCP was optimized to emphasize the congestion case (the retransmit service, which was designed for error handling, has been specialized for congestion control)**
- **But wireless links are much less reliable and lost packets can occur frequently**

The University Of Alabama in Huntsville Computer Science

An example of how TCP is not optimized for wireless

cs670

- When segs timeout, TCP re-starts Slow Start and sends more slowly (assuming this will relieve congestion)
- But when data is lost (happens often on a wireless link), the best thing to do for performance is to re-send quickly
- Furthermore, many wireless systems (e.g., 802.11) use L2 re-transmission and slowdown. L2 re-trans causes slowdown which can trigger slow-start.

TCP and disconnections

cs670

- Mobile users may experience fairly long disconnects
- If a conventional TCP connection was disconnected, TCP using normal settings would timeout and re-transmit repeatedly, doubling interval each time (up to one minute) for a max of 12 times
- If the mobile reconnected during this time, it might have to wait an entire minute before the datastream resumed.

Some things that TCP needs to tolerate to run on wireless networks

cs670

- High error rates
- Temporary disconnects during handover
- Heterogeneous wired/wireless connections (and, sender may not know the composition)
- NOTE: The fact that conventional TCP is not designed to handle these problem does not mean that it won't work on wireless systems, just that its performance is not optimized.

TCP mods are possible, but…

cs670

- Some proposals (e.g., RFC 3168) recommend returning the timeout structure to error control and adding an Explicit Congestion Notification (ECN) mechanism
- But this would require changing all TCP installations, most of which run over wired systems and operate fine as-is.

The University Of Alabama in Huntsville Computer Science

Some fixes for TCP over wireless

cs670

- **Indirect TCP (I-TCP)**
- **Snooping TCP**

The University Of Alabama in Huntsville Computer Science

Indirect TCP (I-TCP)

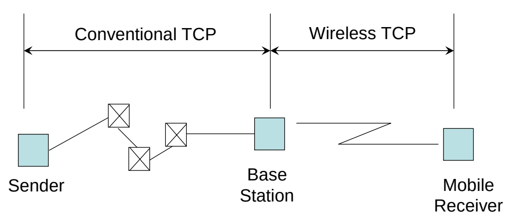
cs670

- **The idea: Combine conventional TCP on the wired network with a Wireless TCP version tuned to run on the wireless part.**

The University Of Alabama in Huntsville Computer Science

I-TCP

cs670



The University Of Alabama in Huntsville Computer Science

I-TCP: Plusses and Minuses

cs670

- **Plusses:**
  - **No changes to existing protocol**
  - **Wireless errors do not directly affect wired system**
- **Minuses:**
  - **Breaks the paradigm of TCP operation – an ACK does not mean that the receiver got the seg**
  - **The base station buffers data for the mobile and ACKs it as it arrives. When a handover occurs, there is no way to get the sender to re-send the buffered data – the base station must forward it.**
  - **To tolerate disconnections, must have very large buffers – when disconnect is followed by handover, even more data to forward.**

Snooping TCP

The University Of Alabama in Huntsville Computer Science

- **Uses a single end-to-end TCP connection (so the paradigm is not broken)**
- **Base station "snoops" on the connection**

"Snooping"

The University Of Alabama in Huntsville Computer Science

- **The base station buffers and forwards all packets destined to the mobile (without ACKing the sender)**
- **Monitors the connection from the mobile to detect ACKs and gaps in transmitted data**
  - **Missing ACKs: Performs local re-transmits based on a short timeout timer**
  - **Missing data: sends NACK to mobile for re-transmit**

Snooping TCP –Plusses and Minuses

The University Of Alabama in Huntsville Computer Science

- **Plusses:**
  - **Preserves the end-to-end TCP paradigm**
  - **No changes in the wired part of the network**
  - **Handovers to new base stations happen automatically (any buffered data will simply time out at the sender and be re-transmitted)**
- **Minuses:**
  - **Problems in the wireless segment can affect the wired segment (e.g., lost wireless packets can trigger Slow Start)**
  - **End-to-end encryption at L3 (e.g, IPSec) foils snooping**
  - **Does not help in the event of a disconnect (there are no ACKs to snoop)**