Internet Engineering

241-461

Robert Elz

kre@munnari.OZ.AU
kre@coe.psu.ac.th

http://fivedots.coe.psu.ac.th/~kre

Network Requirements

- Reliable Communications
  - Nothing lost
  - Nothing added
  - Nothing out of order
  - Not arriving too quickly
- Clean Termination
  - Last data always arrives
  - End indication follows last data
- Transport Layer
  - TCP
    - Transmission Control Protocol

Reliable ?

- Reliable means
  - It must work
  - We can rely upon it working
  - No need to test
    - service guaranteed
- NO
- Reliable means
  - It works
  - Or we are told it failed

TCP

- RFC 793
  - IETF/ISOC Request For Comments
  - Internet Standards
- Transport Protocol
  - Arranges communications between applications
    - Between peers
      - client & server
      - or equal nodes
  - Recovers from errors
    - Lost data in network
    - Reordered data
    - Duplicated data
  - Delivers original data
    - In order
    - Without errors
  - Or reports Error

Underlying Network

- To come later, but
  - ethernets deal with packets
  - applications (often) deal with files
    - One file ==> many packets
- data stream
    - file or whatever it is
  - must be broken into pieces
  - each sent individually
  - reassembled at receiver
- TCP does this
  - pieces called segments

Handling Loss

- Some segments might be lost in network
  - never arrive at destination
- How does TCP know?
  - TCP works for any data
  - It cannot look at image
    - "Look ma - person with no face"
- TCP sender adds identifiers to segments
  - TCP receiver looks for incoming identifiers
  - Notices any missing
- Identifiers?
  - 1 2 3 4 5 ...

Identify what?

- TCP sends segments
  - So, one identifier for each segment ?
- How big is a segment?
  - Depends upon network underneath
  - Packet might pass through several networks
    - different maximum sizes
- TCP identifies octets
    - octet == byte
  - One identifier for each octet
- Sequence Number

Sequence Numbers

- One number for each octet in data
  - Numbers monotonically increasing
    - They get bigger
      - never smaller
      - not even equal
    - Increase by 1 for each octet
- Data stream might be large
  - Many giga-bytes
- Sequence number will get big
  - If start at 1
  - Will end at big number
    - when transfer completes
- Two problems
  - sequence numbers take more space than data
    - waste lots of network bandwidth
  - we have no idea how big maximum will be

Sequence Numbers (2)

- Two problems
  - sequence numbers take more space than data
    - waste lots of network bandwidth
- Solution
  - Assume data in segment is a unit
    - all arrives
    - in same order
    - or none arrives
  - In segment
    - include sequence number of first octet
      - explicitly
    - calculate sequence numbers for others
      - using addition

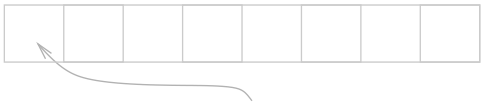
Sequence Numbers (2)

- Two problems
  - sequence numbers take more space than data
    - waste lots of network bandwidth
- Solution
  - Assume data in segment is a unit
    - all arrives
    - in same order
    - or none arrives
  - In segment
    - include sequence number of first octet
      - explicitly
    - calculate sequence numbers for others
      - using addition

Sequence Number: 1

Sequence Numbers (2)

- Two problems
  - sequence numbers take more space than data
    - waste lots of network bandwidth
- Solution
  - Assume data in segment is a unit
    - all arrives
    - in same order
    - or none arrives
  - In segment
    - include sequence number of first octet
      - explicitly
    - calculate sequence numbers for others
      - using addition



Sequence Number: 1

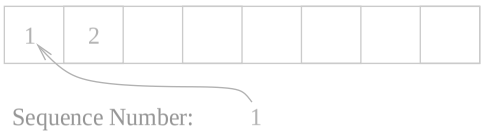
Sequence Numbers (2)

- Two problems
  - sequence numbers take more space than data
    - waste lots of network bandwidth
- Solution
  - Assume data in segment is a unit
    - all arrives
    - in same order
    - or none arrives
  - In segment
    - include sequence number of first octet
      - explicitly
    - calculate sequence numbers for others
      - using addition

Sequence Number: 1

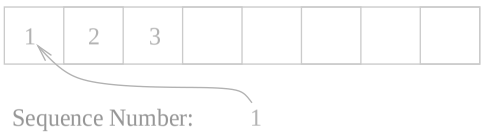
Sequence Numbers (2)

- Two problems
  - sequence numbers take more space than data
    - waste lots of network bandwidth
- Solution
  - Assume data in segment is a unit
    - all arrives
    - in same order
    - or none arrives
  - In segment
    - include sequence number of first octet
      - explicitly
    - calculate sequence numbers for others
      - using addition



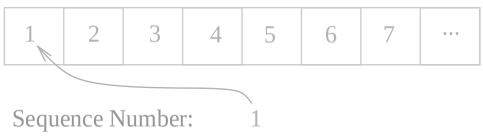
Sequence Numbers (2)

- Two problems
  - sequence numbers take more space than data
    - waste lots of network bandwidth
- Solution
  - Assume data in segment is a unit
    - all arrives
    - in same order
    - or none arrives
  - In segment
    - include sequence number of first octet
      - explicitly
    - calculate sequence numbers for others
      - using addition



Sequence Numbers (2)

- Two problems
  - sequence numbers take more space than data
    - waste lots of network bandwidth
- Solution
  - Assume data in segment is a unit
    - all arrives
    - in same order
    - or none arrives
  - In segment
    - include sequence number of first octet
      - explicitly
    - calculate sequence numbers for others
      - using addition



Sequence Numbers (2)

- Two problems
  - sequence numbers take more space than data
    - waste lots of network bandwidth
  - we have no idea how big maximum will be
- Solution
  - Define sequence number space as a circle
    - 1 2 3 4 5 6 7 0 1 2 3 4 5 6 7 0 1 2 ...
  - Sequence continues indefinitely
    - Just like time on a clock
- Issue
  - How big is the circle?

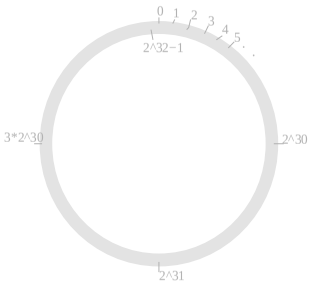
TCP Sequence Numbers

- Issue
  - How big is the circle?
  - How many different identifiers are needed?
- Must have:
  - different ID for every octet alive in network
    - actually alive
    - possibly alive
- Consider
  - 1 Gbit/sec network
    - ~100,000,000 bytes/second
  - If data might be alive 1 second
    - need at least 100,000,000 sequence numbers

TCP Sequence Numbers (2)

- 2^{32} was selected
  - considered big enough
  - 4,294,967,296 sequence numbers
- Allows packets > 40 seconds @ 1Gb/sec
  - or > 4 seconds @ 10Gb/sec
- 32 bit number
  - convenient for computers to manipulate

TCP Sequence Number Space



- Sequence numbers run from 0 to $2^{32} - 1$
- 0, 1, 2, 3, ..., $2^{32} - 2$, $2^{32} - 1$, 0, 1, 2, ...

Using Sequence Number

- Needed for reliability
  - Discovering lost segments
  - Discovering reordered segments
  - Discovering duplicated segments
- How ?

Reordered Segments

- Sequence numbers transmitted in order
  - 100 101 102 103 ... 200 201 202 ... 300 301 302 ...
- Sequence numbers arriving out of numeric order
  - 100 101 102 103 ... 300 301 302 ... 200 201 202 ...
    - segments reordered
- Receiver can easily sort sequence numbers
  - return them to sender order
- Require sender to transmit only in numeric order

Duplicated Segments

- Sequence numbers transmitted
  - 100 101 102 103 ... 200 201 202 ... 300 301 302 ...
- Sequence numbers received
  - 100 101 102 103 ... 200 201 202 ... 100 101 102 103 ... 300 ...
- Receiver can just ignore second copy
  - remember which numbers have been received
  - not receive them again
    - until sequence number space circles around

Lost Segments

- Sequence numbers transmitted
  - 100 101 102 103 ... 200 201 202 ... 300 301 302 ...
- Sequence numbers received
  - 100 101 102 103 ... 300 301 302 ...
- Receiver can notice missing sequence numbers
- Provided sender MUST leave no gaps
  - must transmit every sequence number, in order
- But what to do?
  - Receiver cannot invent missing data
- And how do we know just not reordered?

Lost Segments (2)

- Receiver could tell sender
  - I did not receive 200 201 202 ...
- But consider:
  - Sequence numbers received
  - 100 101 102 103 ... 200 201 202 ...
- How does receiver know anything is missing ?
  - All numbers received
    - in correct order
  - Unless receiver knows last number expected
    - How could it know that?

Lost Segments (3)

- Sender knows what was transmitted
  - So make sender responsible for discovering loss
- How does sender know what was received ?
  - Have receiver inform sender
    - I did receive 100 101 102 ... 300 301 302 ...
- When sender sees missing numbers
  - Send them again
- Requires:
  - sender must keep a copy of transmitted data
  - until receiver has indicated arrival

Acknowledging

- Receiver can say
  - I did receive 100 101 102 ... 300 301 302 ...
- Then later
  - I did receive 200 201 202 ... 400 401 402 ...
- Then later
  - I did receive 500, 501, 502, ...
- But what if one of those messages is lost ??
  - data receiver must send it again
  - but how does it know message was lost ??

Acknowledging

- Easier to say
  - I did receive 100 101 102 ... 200 201 202 ...
  - 301 302 303 ... 400 401 402 ... 500 501 502 ...
- But consider after millions of octets received
  - acknowledgment message would be very large
- Easier to say
  - I received everything up to 599
- or
  - The first one I am waiting for is 600
- TCP does it the last way (next expected is...)