Introduction to Communication Networks
The University of Kansas EECS 563
End-to-End Transport

James P.G. Sterbenz

Department of Electrical Engineering & Computer Science
Information Technology & Telecommunications Research Center
The University of Kansas

jgs@eeecs.ku.edu

http://www.ittc.ku.edu/~jgs/courses/intronets
End-to-End Transport

Outline

TL.1  Transport services and interfaces
TL.2  End-to-end functions and mechanisms
TL.3  Internet transport protocols
End-to-End Transport
Services and Interfaces

TL.1 Transport services and interfaces
TL.2 End-to-end functions and mechanisms
TL.3 Internet transport protocols
Transport Layer
Hybrid Layer/Plane Cube

Layer 4: end-to-end data transfer in data and control planes
Transport Layer

Motivation

Why do we need a transport protocol?
Transport Layer

Ideal Network Model

- Ideal network model
  - zero end-to-end delay
  - unlimited end-to-end bandwidth
  - perfect end-to-end transmission (no errors)

But what?
Transport Layer

Ideal Network Model and Motivation

- **Real networks are not ideal**
- Need an end-to-end protocol to
  - handle delay between communicating systems
  - control transmission rate for bandwidth-limited paths
  - perform error recovery
Transport Layer
Transport Association Definition

- **Transport association** between *end systems*
  - *transport connection* in the case of connection state
  - *transport flow* in the case of soft state or stateless
  - may be point-to-point or multipoint
• Transport protocol
  – is responsible for end-to-end transfer of a TPDU
  – note to Bell-heads: *not* layer 2 “transport network”
Transport Layer
Service and Interfaces

- Transport layer (L4) service to application layer (L7)
- Transfer PDU (protocol data unit) E2E (end-to-end)
  - sender: encapsulate ADU into TPDU and transmit
  - receiver: receive TPDU and decapsulate into ADU
- Multiplexing to application on end system
- Optional reliability:
  - error checking/correction/retransmission
  - need depends on application
  - may be done application-to-application (A2A)
- Flow control between end systems
  - assistance for network congestion control
Transport Layer
Service and Interfaces

- Transport layer *uses* service of network layer (L3)
  - network layer establishes a path through the network
- Transport mechanisms depend on L3 service
  - datagrams vs. connections
  - best effort vs. QoS
  - error characteristics
  - strength and symmetry of connectivity

*What are the Internet assumptions & service model?*
Transport Layer
Service and Interfaces

- Transport layer *uses* service of network layer (L3)
  - network layer establishes a path through the network
- Transport mechanisms depend on L3 service
  - *datagrams* vs. connections
  - *best effort* vs. QoS
  - error characteristics: low error rate, ordered delivery
  - strong and symmetric connectivity

Internet assumptions & service model
Transport Layer
Service and Interfaces

- **Transport PDU** encapsulates *application data unit*
  - ADU: application data unit
  - TPDU: transport protocol data unit (*segment* if TCP or UDP)
  - TPDU = header + ADU + opt. trailer (protocol dependent)
Transport layer functionality only in end system

- host software or
- sometimes offloaded to network interface (NIC)
Transport Layer
End-to-End vs. Hop-by-Hop

- Transport layer is E2E analog of HBH link layer
  - link layer (L2) transfers frames HBH
  - network layer (L3) determines the *path* of per-link hops
End-to-End vs. Hop-by-Hop
End-to-End Arguments

• The end-to-end arguments (1st half)

• Some functions can be correctly and completely implemented *only* at the endpoints of a communication association

• Providing these functions as features (only) in the network is not possible

paraphrased from [Saltzer, Reed, Clark 1981]
End-to-End vs. Hop-by-Hop

End-to-End Arguments

- Hop-by-Hop functions do not compose end-to-end
  why?
End-to-End vs. Hop-by-Hop

End-to-End Arguments

- Hop-by-Hop functions do not compose end-to-end
  - between HBH boundaries, function $f$ is defeated ($g$)
    - e.g. error control: errors may occur within switches
    - e.g. encryption: cleartext within switches may be snooped
- These functions must be done E2E
  - doing them HBH is redundant, and may lower performance
End-to-End vs. Hop-by-Hop

End-to-End Arguments

- The end-to-end arguments (2nd half)
  - performance enhancement corollary

- Functions should be *duplicated* hop-by-hop if there is an overall (end-to-end) performance benefit

paraphrased from [Saltzer, Reed, Clark 1981]
End-to-End vs. Hop-by-Hop

Hop-by-Hop Performance Enhancement

- E2E Argument (1st half) says what *must* be E2E
- HBH Performance enhancement (2nd half)
  - functions *should* duplicated HBH if overall E2E benefit
- Analysis is required to determine cost/benefit
  - added functionality in net may add overhead not offset
End-to-End vs. Hop-by-Hop Performance Enhancement Example

- E2E vs. HBH error control for reliable communication
  - E2E argument says error control *must* be done E2E
    - e.g. E2E ARQ (error check code and retransmit if necessary)
  - but *should* HBH error control *also* be done?

```
100 m 15 000 km 100 m
wireless LAN fiber WAN fiber LAN
Univ. Kansas HK PolyU
```
End-to-End vs. Hop-by-Hop Performance Enhancement Example

- E2E vs. HBH error control for reliable communication
  - E2E argument says error control *must* be done E2E
    - e.g. E2E ARQ (error check code and retransmit if necessary)
  - but *should* HBH error control *also* be done?

- Effect of high loss rate on wireless link
  - \(~250\) ms RTT retransmission for every corrupted packet
End-to-End vs. Hop-by-Hop Performance Enhancement Example

- **E2E vs. HBH error control for reliable communication**
  - E2E argument says error control *must* be done E2E
    - e.g. E2E ARQ (error check code and retransmit if necessary)
  - but *should* HBH error control also be done? *Yes!*

- **Effect of high loss rate on wireless link**
  - ~250 ms RTT retransmission for every corrupted packet
- **Error control on wireless link reduces to ~1μs RTT**
  - shorter control loop results in dramatically lower E2E delay
End-to-End vs. Hop-by-Hop Security

- Security and information assurance *must* be E2E
  - information in the clear inside network nodes *not* secure

- Justification for HBH security mechanisms
  - link security may be good enough for *some*
    - wireless link encryption for WEP (wire equivalent protection)
      - note: 802.11 WEP not strong enough
  - subnetwork or edge-to-edge security for VPNs
    - assures enterprise security across open network...
      - ...but not individual flow security
End-to-End Transport
Functions and Mechanisms

TL.1  Transport services and interfaces
TL.2  End-to-end functions and mechanisms
TL.3  Internet transport protocols
E2E Framing
Structure and Fields

• Delimits TPDU
  – physical layer is coded bit stream
  – receiver needs to delimit frames

• Header: how to process the TPDU
  – protocol processing options
  – demultiplexing fields: destination application identifier
    • typically called *port* (Internet protocol suite terminology)
  – error check (CRC or checksum) for reliability
E2E Multiplexing
Concepts

- Multiple applications use a particular TP
- Applications multiplexed into sending TP
  - need to identify which application

*why?*
E2E Multiplexing
Concepts

- Multiple applications use a particular TP
- Applications multiplexed into sending TP
  - need to identify which application
  - using identifier typically called a *port*
- TP demultiplexes TPDU (segment)
  - sending to correct application
  - using port number
E2E Functions and Mechanisms
Transport Protocol Service Categories

- Transfer mode
  - connectionless datagram
  - connection-oriented
  - transaction
  - continuous media streaming

- Reliability
  - loss tolerance

- Delivery order
  - ordered
  - unordered

- Traffic characteristics
E2E Transfer Modes

Alternatives

- End-to-end transfer modes
  - datagram
  - connection
  - stream
  - transaction
E2E Transfer Modes

Datagram

- **Independent PDUs**
  - each has fully self-describing header
  - no end-to-end flow state needed to forward
E2E Transfer Modes

Connection

- Explicit connection setup
  - 3-way handshake
  
  SETUP / CONNECT / ACK

*why?*
E2E Transfer Modes

Connection

- Explicit connection setup
  - 3-way handshake
    SETUP / CONNECT / ACK
  - both side have been acknowledged
- Data flow
  - PDUs need connection or flow identifier
    - used by nodes to look up state
- Release of resources and state
  - explicit RELEASE
  - time out state
E2E Transfer Modes
Connection State Machine

- **Connection State Machine**
  - send/receive paths

- **States**
  - **idle**
  - **establishing**
    - install state
  - **initialising**
    - state convergence
  - **steady state**
  - **closing**
    - release state

- Events:
  - originate SETUP request
  - receive SETUP from net
  - issue SETUP request
  - issue SETUP
  - issue CONNECT
  - issue CONNECT
  - issue RELEASE
  - issue RELEASE
  - release SETUP from app
  - release RELEASE from net
  - release RELEASE from app
  - release RELEASE
E2E Transfer Modes

Media Stream

- Various mechanisms to start stream
  - explicit client request
  - server push
  - may or may not establish connection state

- Data flow
  - synchronisation and control
    - embedded or
    - out-of-band

More about this in Lecture MS
E2E Transfer Modes

Transaction

- Transaction request
  - may or may not establish connection state
  - explicit release of connection state optional
- Data returned
- Overlap to reduce latency
  - request/response with control

- Recall: this is *not* how HTTP over TCP works
E2E Error Control

Types of Errors

*What are the types of errors?*
E2E Error Control

Types of Errors

- Bit errors
  *types?
- Packet errors
  *types?
E2E Error Control

Types of Errors

• Bit errors
• Packet errors
  – packet loss
  – fragment loss
  – burst loss
  – packet misordering
  – packet insertion
  – packet duplication
E2E Error Control

Causes of Bit Errors

- Bit errors

*causes?*
E2E Error Control

Causes of Bit Errors

• Bit errors
  – flaky hardware
  – wireless channels
    • noise and interference
  – optical channels
    • long-haul fiber links engineered on margin with FEC
    • control of amplifier and regenerator cost

More in Lecture PL
E2E Error Control

Causes of Packet Loss

• Packet loss

causes?
E2E Error Control

Causes of Packet Loss

- Packet loss
  - network buffer overflow or intentional drop
    more in Lecture TQ
  - transport protocol packet discard when bit error
E2E Error Control
Causes of Packet Loss

• Packet loss
  – network buffer overflow or intentional drop
  \[ \textit{more in Lecture TQ} \]
  – transport protocol packet discard when bit error

• Packet \textit{fragment loss}
  \[ \textit{causes?} \]
  \[ \textit{impact?} \]
E2E Error Control
Causes of Packet Loss

• Packet loss
  – network buffer overflow or intentional drop
    more in Lecture TQ
  – transport protocol packet discard when bit error

• Packet fragment loss
  – loss of a piece of a fragmented packet
  – effect: error multiplication
    • loss of a fragment in net generally results in entire packet loss
  – rare in Internet since router fragmentation rarely enabled
E2E Error Control
Causes of Packet Burst Loss

• Packet *burst* loss

*causes?*
E2E Error Control

Causes of Packet Burst Loss

- Packet burst loss
  - network buffer overflow during congestion
    - with high rate flow having adjacent (non-interleaved) packets
      
      more in lecture TQ

  - wireless channel fades
      
      more in lecture PL
E2E Error Control

Causes of Packet Misordering

- Packet *misordering causes?*
E2E Error Control
Causes of Packet Misordering

- Packet misordering
  - multiple paths through switch or network
  - non-deterministic paths through switch or network
  - path reconfiguration
    - after failure
    - due to mobility
    - due to load balancing

more in lecture NL
E2E Error Control

Causes of Packet Insertion

• Packet *insertion*
  
  *meaning?*
  
  *causes?*
E2E Error Control

Causes of Packet Insertion

- Packet insertion
  - undetectable header error
    - packet appears that was destined elsewhere
  - long packet life
    - packet with same destination and sequence number still in net
E2E Error Control

Causes of Packet Duplication

- Packet *duplication*
  *causes?*
E2E Error Control

Causes of Packet Duplication

- Packet duplication
  - retransmission but original still arrives
E2E Error Control
ARQ Closed Loop Retransmission

- Automatic repeat request (ARQ)
  - TPDUs are retransmitted for reliable transfer
- Acknowledgments are used to request retransmission

alternatives?
E2E Error Control
ARQ Closed Loop Retransmission

- Automatic repeat request (ARQ)
  - TPDUs are retransmitted for reliable transfer

- Acknowledgments are used to request retransmission
  - ACK: positive acknowledgement
  - NAK (or NACK): negative acknowledgement

*tradeoff?*
E2E Error Control
ARQ Closed Loop Retransmission

• Automatic repeat request (ARQ)
  – TPDUs are retransmitted for reliable transfer

• Acknowledgments are used to request retransmission
  – ACK: positive acknowledgement
  – NAK (or NACK): negative acknowledgement
  – tradeoff between error rate and predictability

Simplest ARQ mechanism?
E2E Error Control

Closed Loop Retransmission: Stop-and-Wait

- Each TPDU is acknowledged
E2E Error Control
Closed Loop Retransmission: Stop-and-Wait

• Each TPDU is acknowledged
E2E Error Control

Closed Loop Retransmission: Stop-and-Wait

- Each TPDU is acknowledged
  - subsequent TPDU waits on previous ACK

*disadvantages?*
E2E Error Control
Closed Loop Retransmission: Stop-and-Wait

• Each TPDU is acknowledged
  – subsequent TPDU waits on previous ACK
    • significant delay and underutilisation
    • 1 RTT per TPDU
    • serious penalty in long-delay environment
E2E Error Control
Closed Loop Retransmission: Stop-and-Wait

- Each TPDU is acknowledged
  - subsequent TPDU waits on previous ACK
    - significant delay and underutilisation
    - 1 RTT per TPDU
    - serious penalty in long-delay environment
  - sender timer $t_{ack}$ fires if ACK not received

issues?
E2E Error Control
Closed Loop Retransmission: Stop-and-Wait

- Each TPDU is acknowledged
  - subsequent TPDU waits on previous ACK
    - significant delay and underutilisation
    - 1 RTT per TPDU
    - serious penalty in long-delay environment
  - sender timer $t_{ack}$ fires if ACK not received
    - too conservative: issues?
    - too aggressive: issues?
E2E Error Control
Closed Loop Retransmission: Stop-and-Wait

- Each TPDU is acknowledged
  - subsequent TPDU waits on previous ACK
    - significant delay and underutilisation
    - 1 RTT per TPDU
    - serious penalty in long-delay environment
  - sender timer $t_{ack}$ fires if ACK not received
    - too conservative: unnecessary delay
    - too aggressive: spurious retransmissions
E2E Error Control
Closed Loop Retransmission: Stop-and-Wait

- Each TPDU is acknowledged
  - subsequent TPDU waits on previous ACK
    - significant delay and underutilisation
    - 1 RTT *per TPDU*
    - serious penalty in long-delay environment
  - sender timer $t_{ack}$ fires if ACK not received
    - too conservative: unnecessary delay
    - too aggressive: spurious retransmissions

*How can we do better?*
E2E Error Control
Closed Loop Retransmission: Go-Back-\( n \)

- Go-back-\( n \): pipeline transmissions
E2E Error Control
Closed Loop Retransmission: Go-Back-$n$

- Go-back-$n$: *pipeline* transmissions
  - multiple TPDUs simultaneously in flight
E2E Error Control

Closed Loop Retransmission: Go-Back-\(n\)

- Go-back-\(n\): *pipeline* transmissions
  + multiple TPDUs simultaneously in flight
- TPDUs are *sequentially* acknowledged
E2E Error Control
Closed Loop Retransmission: Go-Back-$n$

- Go-back-$n$: *pipeline* transmissions
  + multiple TPDUs simultaneously in flight
- TPDUs are *sequentially* acknowledged

*how is TPDU loss detected?*
E2E Error Control
Closed Loop Retransmission: Go-Back-$n$

- Go-back-$n$: *pipeline* transmissions
  + multiple TPDUs simultaneously in flight
- TPDUs are *sequentially* acknowledged
  - previous ACK retransmitted for
    - subsequent TPDUs after loss
    - out of sequence TPDU
E2E Error Control

Closed Loop Retransmission: Go-Back-\( n \)

- Go-back-\( n \): *pipeline* transmissions
  + multiple TPDUs simultaneously in flight
- TPDUs are *sequentially* acknowledged
  - previous ACK retransmitted for
    - subsequent TPDUs after loss
    - out of sequence TPDU
  - sender timer fires if ACK not received

*implication?*
E2E Error Control
Closed Loop Retransmission: Go-Back-\(n\)

- **Go-back-\(n\):** *pipeline* transmissions
  - multiple TPDUs simultaneously in flight
- TPDUs are *sequentially* acknowledged
  - previous ACK retransmitted for
    - subsequent TPDUs after loss
    - out of sequence TPDU
  - sender timer fires if ACK not received
    - reset transmission beginning at lost TPDU
E2E Error Control
Closed Loop Retransmission: Go-Back-$n$

- Go-back-$n$: *pipeline* transmissions
  - multiple TPDUs simultaneously in flight
- TPDUs are *sequentially* acknowledged
  - previous ACK retransmitted for
    - subsequent TPDUs after loss
    - out of sequence TPDU
  - sender timer fires if ACK not received
    - reset transmission beginning at lost TPDU
E2E Error Control
Closed Loop Retransmission: Go-Back-$n$

- Go-back-$n$: pipeline transmissions
  + multiple TPDUs simultaneously in flight
- TPDUs are sequentially acknowledged
  - previous ACK retransmitted for
    - subsequent TPDUs after loss
    - out of sequence TPDU
  - sender timer fires if ACK not received
    - reset transmission beginning at lost TPDUs
E2E Error Control
Closed Loop Retransmission: Go-Back-

- **Go-back-**: *pipeline* transmissions
  + multiple TPDUs simultaneously in flight
- **TPDUs are** *sequentially* acknowledged
  - previous ACK retransmitted for
    - subsequent TPDUs after loss
    - out of sequence TPDU
  - sender timer fires if ACK not received
    - reset transmission beginning at lost TPDU
E2E Error Control
Closed Loop Retransmission: Go-Back-\(n\)

- Go-back-\(n\): *pipeline* transmissions
  - multiple TPDUs simultaneously in flight
- TPDUs are *sequentially* acknowledged
  - previous ACK retransmitted for
    - subsequent TPDUs after loss
    - out of sequence TPDU
  - sender timer fires if ACK not received
    - reset transmission beginning at lost TPDU
E2E Error Control
Closed Loop Retransmission: Go-Back-\(n\)

- **Go-back-\(n\):** *pipeline* transmissions
  + multiple TPDUs simultaneously in flight
- **TPDUs are** *sequentially* acknowledged
  - previous ACK retransmitted for
    - subsequent TPDUs after loss
    - out of sequence TPDU
  - sender timer fires if ACK not received
    - reset transmission beginning at lost TPDU

*disadvantages?*
E2E Error Control
Closed Loop Retransmission: Go-Back-$n$

• Go-back-$n$: *pipeline* transmissions
  + multiple TPDUs simultaneously in flight
• TPDUs are *sequentially* acknowledged
  o previous ACK retransmitted for
    • subsequent TPDUs after loss
    • out of sequence TPDU
  o sender timer fires if ACK not received
    • reset transmission beginning at lost TPDU
  – significant loss penalty for high bw-$\times$-delay
    • go back and retransmit *all* since loss
    • many unneeded retransmissions
    • significant additional delay
E2E Error Control
Closed Loop Retransmission: Go-Back-\(n\)

- Optimisation possible for go-back-\(n\)?
E2E Error Control

Closed Loop Retransmission: Go-Back-\( n \)

- Optimisation go-back-\( n \)

  *timer the only way to detect loss?*
E2E Error Control

Closed Loop Retransmission: Fast Retransmit

- **Fast retransmit**
  - optimisation for go-back-$n$
E2E Error Control

Closed Loop Retransmission: Fast Retransmit

- Fast retransmit
  - optimisation for go-back-$n$
E2E Error Control
Closed Loop Retransmission: Fast Retransmit

- Fast retransmit
  - optimisation for go-back-$n$
E2E Error Control
Closed Loop Retransmission: Fast Retransmit

- Fast retransmit
  - optimisation for go-back-$n$
E2E Error Control

Closed Loop Retransmission: Fast Retransmit

- Fast retransmit
  - optimisation for go-back-$n$
- Assume several duplicate ACKs are loss
E2E Error Control
Closed Loop Retransmission: Fast Retransmit

- Fast retransmit
  - optimisation for go-back-$n$
- Assume several duplicate ACKs are loss
  - even if timer hasn’t yet fired
E2E Error Control

Closed Loop Retransmission: Fast Retransmit

- Fast retransmit
  - optimisation for go-back-$n$
- Assume several duplicate ACKs are loss
  - even if timer hasn’t yet fired
  + recovers from loss more quickly
E2E Error Control
Closed Loop Retransmission: Fast Retransmit

- Fast retransmit
  - optimisation for go-back-$n$
- Assume several duplicate ACKs are loss
  - even if timer hasn’t yet fired
  - recovers from loss more quickly
E2E Error Control

Closed Loop Retransmission: Fast Retransmit

- Fast retransmit
  - optimisation for go-back-$n$
- Assume several duplicate ACKs are loss
  - even if timer hasn’t yet fired
  - recovers from loss more quickly
E2E Error Control
Closed Loop Retransmission: Fast Retransmit

- Fast retransmit
  - optimisation for go-back-$n$
- Assume several duplicate ACKs are loss
  - even if timer hasn’t yet fired
  + recovers from loss more quickly
E2E Error Control
Closed Loop Retransmission: Fast Retransmit

- Fast retransmit
  - optimisation for go-back-$n$
- Assume several duplicate ACKs are loss
  - even if timer hasn’t yet fired
  - recovers from loss more quickly
E2E Error Control
Closed Loop Retransmission: Fast Retransmit

- Fast retransmit
  - optimisation for go-back-$n$
- Assume several duplicate ACKs are loss
  - even if timer hasn’t yet fired
  - recovers from loss more quickly
  - still suffers from go-back-$n$ inefficiencies
E2E Error Control
Closed Loop Retransmission: Go-Back-$n$

- Go-back-$n$
  - fast retransmit and delayed ACK help slightly
  - but still significant delay penalty for losses

*Alternative?*
E2E Error Control

Closed Loop Retransmission: Selective Repeat

- Go-back-$n$
  - fast retransmit and delayed ACK help slightly
  - significant delay penalty for losses

- Alternative
  - don’t go back: selective repeat
E2E Error Control
Closed Loop Retransmission: Selective Repeat

- **Selective repeat**
  - *all* TPDUs acknowledged
E2E Error Control

Closed Loop Retransmission: Selective Repeat

- Selective repeat
  - all TPDUs acknowledged
E2E Error Control
Closed Loop Retransmission: Selective Repeat

- Selective repeat
  - *all* TPDUs acknowledged
E2E Error Control
Closed Loop Retransmission: Selective Repeat

- Selective repeat
  - all TPDUs acknowledged
E2E Error Control

Closed Loop Retransmission: Selective Repeat

- Selective repeat
  - all TPDUs acknowledged
- Missequenced TPDUs
  - acknowledged and held at receiver
E2E Error Control
Closed Loop Retransmission: Selective Repeat

- Selective repeat
  - all TPDUs acknowledged
- Missequenced TPDUs
  - TPDUs held *and reordered* at receiver
    - increases receiver complexity
E2E Error Control
Closed Loop Retransmission: Selective Repeat

- Selective repeat
  - *all* TPDUs acknowledged
- Missequenced TPDUs
  - TPDUs held and reordered at receiver
    - increases receiver complexity
- Lost TPDUs
  - selectively retransmitted
E2E Error Control

Closed Loop Retransmission: Selective Repeat

- Selective repeat
  - all TPDUs acknowledged

- Missequenced TPDUs
  - TPDUs held and reordered at receiver
    - increases receiver complexity

- Lost TPDUs
  - selectively retransmitted
    + no go-back-$n$ latency penalty
    - requires more receiver buffer space
E2E Error Control
Closed Loop Retransmission: Selective Repeat

- Selective repeat
  - all TPDUs acknowledged
- Missequenced TPDUs
  - TPDUs held and reordered at receiver
    - increases receiver complexity
- Lost TPDUs
  - selectively retransmitted
  + no go-back-$n$ latency penalty
  - requires more receiver buffer space
E2E Error Control
Closed Loop Retransmission: Selective Repeat

- Selective repeat
  - *all* TPDUs acknowledged

- Missequenced TPDUs
  - TPDUs held and reordered at receiver
    - increases receiver complexity

- Lost TPDUs
  - selectively retransmitted
    + no go-back-$n$ latency penalty
    - requires more receiver buffer space
E2E Error Control

Closed Loop Retransmission: Selective Repeat

- Selective repeat
  - *all* TPDUs acknowledged

- Missequenced TPDUs
  - TPDUs held and reordered at receiver
    - increases receiver complexity

- Lost TPDUs
  - selectively retransmitted
    + no go-back-\(n\) latency penalty
    - requires more receiver buffer space

- A2A delay and bandwidth reduced
E2E Error Control
Closed Loop Retransmission: Comparison
E2E Transmission Control

Definitions: Flow Control

- Flow control
  - control transmission to avoid overwhelming receiver
  - end-to-end control
E2E Transmission Control
Definitions: Congestion Control

- **Flow control**
  - control transmission to avoid overwhelming *receiver*
  - end-to-end control

- **Congestion control**
  - control transmission to avoid overwhelming *network* paths
E2E Transmission Control
Definitions: Implicit vs. Explicit

- Flow control
  - control transmission to avoid overwhelming receiver
  - end-to-end control

- Congestion control
  - control transmission to avoid overwhelming network paths
  - network control with end-to-end assistance (throttling)

- Types
  - explicit relies on congestion information from nodes
    - allows decoupling of error, flow, and congestion mechanisms
  - implicit assumes loss is congestion
    - bad assumption in wireless networks (see [Krishnan 2004])
E2E Transmission Control
Closed-Loop Flow and Congestion Control

• Flow control
  – sender–receiver
• Congestion control
  – sender–network
• Packet scheduling  *Lecture MT*
E2E Transmission Control
Closed-Loop Flow Control

• Feedback from *receiver* to limit transmission rate
  – end-to-end *only*

• Window to limit amount of unacknowledged data
  – static window
    • based on initial negotiation or assumption
  – dynamic window
    • renegotiated based on receiver buffer availability
    • may be combined with congestion control
E2E Transmission Control

Closed-Loop Flow Control

- Initial negotiation
  - amount of unacknowledged data can be sent to receiver
    - maintained as *receive window* at sender
    - based on assumption or value returned at connection setup
    - static window does not change over life of connection
E2E Transmission Control

Closed-Loop Flow Control

- Initial negotiation
- Steady state
  - dynamic window: receive window varies over time
  - receiver adjusts with time-varying available buffer space
E2E Transmission Control
Closed-Loop Congestion Control

- Feedback from *network* to limit transmission rate
- Window to limit amount of unacknowledged data
  - dynamic window
    - adjusted based on network state
E2E Transmission Control
Closed-Loop Congestion Control

- Initial negotiation
  - amount of unacknowledged data can be sent into network
    - maintained as *congestion window* at sender
    - initial conservative default

接收窗口

© James P.G. Sterbenz
E2E Transmission Control
Closed-Loop Congestion Control

- Initial negotiation
  - amount of unacknowledged data can be sent into network
- Data transfer
  - congestion window adjusted to adapt to congestion
Congestion Control

Ideal Network

- Ideal network

  *throughput characteristics?*
  *delay characteristics?*
Congestion Control

Ideal Network

- **Ideal network:**
  - throughput: carried load = offered load (45° line)
  - zero delay (flat line)

*Why isn’t this possible?*
Congestion Control
Real Network

• Delay can’t be zero: speed-of-light
  – delay through network channels
  – delay along paths in network nodes

• Throughput can’t be infinite
  – channel capacity limits bandwidth
  – switching rate of components limits bit rate
Congestion Control

Ideal Network

- Desired network: 
  what happens to carried load?
• Desired network:
  – carried load = offered load up to share of capacity

*what happens to delay?*
Congestion Control

Causes of Congestion

- Congestion when offered load $\rightarrow$ net path capacity
- Best effort
  - occurs whenever load is high
- Probabilistic guarantees
  - occurs when load exceeds resource reservations
- Absolute guarantees
  - can’t happen (in theory)
Congestion Control

Infinite Buffers / No Retransmission

- Congestion when offered load $\rightarrow$ link capacity
- Infinite buffers
  - queue length builds to infinity
  - $d \rightarrow \infty$

*discuss why in Lecture MT*
Finite Buffers with Retransmission

• Finite buffers cause packet loss
  – retransmissions contribute to offered load
    • assuming reliable protocol
  – throughput curve levels off
Congestion Control

Multihop Network

- Multiple hops
  - downstream packet loss
  - upstream transmission wasted
Congestion Control
Consequences of Congestion

- Delay *increases*
  - due to packet queuing in network nodes
  - due to retransmissions when packets overflow buffers
    - finite buffers must drop packets when full

- Throughput
  - levels off gradually (with real traffic)
  - then *decreases*
    - due to retransmissions when packets dropped
    - particularly over multiple hops
  - congestion collapse
    - “cliff” of the throughput curve
Congestion control: operation at the cliff

Congestion avoidance: operation at the knee
  - detect and correct impending congestion before the cliff
Congestion Control

Local Action

- Local action at point of congestion
- Drops packets: discard policy
  - tail drop simplest: drop packets that overflow buffers
  - more intelligent policies possible
    - need flow or connection state in switches
    - discriminate *which* flows are causing congestion and penalise
Congestion Control
End-to-End Action

- End-to-end action
- Throttle source
  - reduce transmission rates
  - prevent unnecessary retransmissions
- Types
  - explicit
  - implicit
E2E Transmission Control
Closed-Loop Congestion Control

- Feedback from *network* to limit rate
- Required unless open loop with hard reservations
- Window to limit amount of unacknowledged data
  - dynamic window
  - conservation of packets in the network
  - self clocking: ACKs determine the transmission of new data
E2E Transmission Control
Closed-Loop Control: Initialisation

- Initial small window sent
  - assume 1 TPDU
E2E Transmission Control
Closed-Loop Control: Initialisation

- Initial small window sent
  - assume 1 TPDU
  - TPDU acknowledged
E2E Transmission Control
Closed-Loop Control: Initialisation

- Initial small window sent
  - assume 1 TPDU
  - TPDU acknowledged
- Window doubled
  - TPDU + one more sent
E2E Transmission Control
Closed-Loop Control: Initialisation

- Initial small window sent
  - assume 1 TPDU
  - TPDU acknowledged

- Window doubled
  - TPDU + one more sent
  - TPDUs acknowledged
E2E Transmission Control
Closed-Loop Control: Initialisation

- Initial small window sent
  - assume 1 TPDU
  - TPDU acknowledged
- Window doubled
  - TPDU + one more sent
  - TPDUs acknowledged
- Window doubled again
  - TPDUs + one more/ACK sent
E2E Transmission Control

Closed-Loop Control: Initialisation

- Slow start initialisation
  - increase window until path loaded
  - double window / RTT
    - exponential growth
    - not really “slow”

**Critical parameters?**
E2E Transmission Control

Closed-Loop Control: Initialisation

- Slow start initialisation
  - increase window until path loaded
  - double window / RTT

- Critical parameters
  - initial window size
  - rate of increase

Tradeoffs?
E2E Transmission Control
Closed-Loop Control: Initialisation

- Slow start initialisation
  - increase window until path loaded

- Critical parameters
  - initial window size
  - rate of increase

- Tradeoffs
  - conservative on high bw-x-delay:
    - multiple round trips
    - never get to full rate for transactions
  - aggressive:
    - induced congestions
E2E Congestion Control
Closed-Loop Control

- Initialisation phase: slow start

Steady-state phase?
E2E Congestion Control
Closed-Loop Control: Steady State

• AIMD steady state
  – additive-increase slowly increases rate
    • increment window
  – multiplicative-decrease quickly throttles with congestion
    • divide window when packet lost
E2E Congestion Control
AIMD Fairness and Stability

- AIMD
  - $R =$ available bandwidth
  - initial bandwidth share $I$
  - AI: both increase
    - until loss beyond $R$
  - MD: both decrease
    - half way to origin
    - halve window size
  - trajectory toward ideal
    - intersection of $R$ and equal

[based on Kurose fig. 3.53]
E2E Congestion Control
Closed-Loop Control

- Initialisation phase: slow start
- Steady-state phase: AIMD
E2E Congestion Control
Closed-Loop Implicit Control

- Implicit control
  - missing ACK assumed to be congestion

*good assumption?*
E2E Congestion Control
Closed-Loop Implicit Control

• Implicit control
  – missing ACK assumed to be congestion
  – reasonable when all losses are due to congestion
    • fiber optic channels connected by reliable switches
**E2E Congestion Control**

**Closed-Loop Implicit Control**

- Implicit control
  - missing ACK assumed to be congestion
  - reasonable when all losses are due to congestion
    - fiber optic channels connected by reliable switches
  - performs poorly when significant losses in channel

*why?*
E2E Congestion Control
Closed-Loop Implicit Control

- Implicit control
  - missing ACK assumed to be congestion
  - reasonable when all losses are due to congestion
    - fiber optic channels connected by reliable switches
  - performs poorly when significant losses in channel
    - mobile wireless links
    - under-provisioned CDMA (including optical CDMA)
E2E Congestion Control
Closed-Loop Implicit Control

- Implicit control
  - missing ACK assumed to be congestion
  - reasonable when all losses are due to congestion
    - fiber optic channels connected by reliable switches
  - performs poorly when significant losses in channel
    - mobile wireless links
    - under-provisioned CDMA (including optical CDMA)

*Alternatives?*
E2E Congestion Control
Closed-Loop Explicit Control

- Explicit control
  - explicit congestion notification (ECN)
    - throttle with ECN message
  - some decoupling of error and congestion control
    - throttle before packet loss
    - not sufficient for lossy wireless link

EECS 882
E2E Congestion Control
Closed-Loop Explicit Control

- Explicit control
  - explicit congestion notification (ECN)
    - throttle with ECN message
  - some decoupling of error and congestion control
    - throttle before packet loss
    - not sufficient for lossy wireless link
  - not sufficient for discrimination of corrupted packets
    - ELN (explicit loss notification) necessary

more in lecture TQ
Transport Layer
Internet Transport Protocols

TL.1 Transport Services and interfaces
TL.2 End-to-end functions and mechanisms
TL.3 Internet transport protocols
Internet Transport Protocols

History

- ARPANET / Internet transport protocol history
  - NCP (network control program) original ARPANET protocol
  - replaced with TCP in 1977
  - TCP split from IP in 1978
  - UDP added in 1980
  - RTP RFC published 1996
Internet Transport Protocols
Overview

- **UDP**: user datagram protocol [RFC 0768 / STD 0006]
  - basic end-to-end multiplexing and error checking
  - no error, flow, or congestion control

- **RTP**: real-time protocol [RFC 3550 / STD 0064]
  - support for end-to-end real-time synchronisation
  - typical implementations run over UDP

- **TCP**: transmission control protocol [RFC 0793 / STD 0007]
  - connection-oriented reliable byte-stream protocol
  - full error, flow, and congestion control
# Internet Transport Protocols

## Important Examples

<table>
<thead>
<tr>
<th>Protocol</th>
<th>Name</th>
<th>Function</th>
<th>Status</th>
<th>Ref</th>
</tr>
</thead>
<tbody>
<tr>
<td>TCP</td>
<td>transmission control protocol</td>
<td>reliable data transfer with congestion control</td>
<td>standard</td>
<td>RFC 0793 STD 0007</td>
</tr>
<tr>
<td>UDP</td>
<td>user datagram protocol</td>
<td>socket access to unreliable IP datagrams</td>
<td>standard</td>
<td>RFC 0768 STD 0006</td>
</tr>
<tr>
<td>RTP</td>
<td>real-time protocol</td>
<td>real-time synchronisation (typically over UDP)</td>
<td>standards track</td>
<td>RFC 3550 STD 0064</td>
</tr>
<tr>
<td>T/TCP</td>
<td>TCP for transactions</td>
<td>low connection setup overhead for transactions</td>
<td>experimental</td>
<td>RFC 1644</td>
</tr>
<tr>
<td>RDP</td>
<td>reliable data protocol</td>
<td>reliable data transfer with no congestion control</td>
<td>experimental</td>
<td>RFC 0908</td>
</tr>
<tr>
<td>SCTP</td>
<td>stream control transmission protocol</td>
<td>signalling proposed for wireless</td>
<td>proposed standard</td>
<td>RFC 2960</td>
</tr>
</tbody>
</table>
Internet Transport Protocols

TL.1 Transport services and interfaces
TL.2 End-to-end functions and mechanisms
TL.3 Internet transport protocols
  TL.3.1 UDP
  TL.3.2 RTP introduction
  TL.3.3 TCP
TL.4 Multipoint communication and reliable multicast
User Datagram Protocol
Overview and Transfer Mode

- UDP: user datagram protocol [RFC 0768 / STD 0006]
- Basic access to Internet datagram transfer mode
  - no connection establishment
    - each UDP segment handled independently
    - no connection setup penalty
    - no connection state to maintain
  - basic end-to-end multiplexing
  - no reliability
  - no flow or congestion control
User Datagram Protocol
Segment Format and Multiplexing

- Relatively small header
  - source and destination port
  - length [B] including header
  - checksum

- Multiplexing: UDP socket
  - dest (IP addr, port#)

- Demux
  - receiver passes to port#
    - source addr, port irrelevant
  - negotiated or well-known
  - may use source port to reply

<table>
<thead>
<tr>
<th>source port#</th>
<th>destination port #</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>checksum</td>
</tr>
</tbody>
</table>

application payload
(= length – 8B)
User Datagram Protocol

Multiplexing

- Interface to IP: protocol mux/demux (UDP=17)
- Interface to application: port mux/demux
User Datagram Protocol
Error Detection

- Recall E2E arguments: error check *must* be done E2E
- Error checking: *checksum*
  - sequence of 16-bit integers
  - binary addition
    - carry overflows *wrapped*;
    - *1’s complement* of sum

\[
1110011001100110 + 1101010101010101 = 1101011010101011
\]

<table>
<thead>
<tr>
<th>source port#</th>
<th>destination port #</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>checksum</td>
</tr>
</tbody>
</table>

application payload
(= length – 8B)
User Datagram Protocol

Error Detection

- Recall E2E arguments: error check *must* be done E2E
- Error checking: checksum
  - sequence of 16-bit integers
  - binary addition
    - carry overflows wrapped;
    - 1’s complement of sum

<table>
<thead>
<tr>
<th>source port#</th>
<th>destination port #</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>checksum</td>
</tr>
</tbody>
</table>

application payload
(= length – 8B)
User Datagram Protocol

Error Detection

- Recall E2E arguments: error check *must* be done E2E
- Error checking: checksum
  - sequence of 16-bit integers
  - binary addition
    - carry overflows wrapped;
    - 1’s complement of sum

```
  1110011001100110
+1101010101010101
  1110111011011011
  1011101110111100
```

<table>
<thead>
<tr>
<th>source port#</th>
<th>destination port #</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>checksum</td>
</tr>
</tbody>
</table>

application payload

(= length – 8B)
User Datagram Protocol
Error Detection

- Recall E2E arguments: error check *must* be done E2E
- Error checking: checksum
  - sequence of 16-bit integers
  - binary addition
    - carry overflows wrapped;
    - 1’s complement of sum

\[
\begin{array}{c}
1110110011001100 \\
+1101010101010101 \\
1101110111011101 \\
1011101110111100 \\
0100010001000011
\end{array}
\]

<table>
<thead>
<tr>
<th>source port#</th>
<th>destination port #</th>
<th>length</th>
<th>checksum</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

application payload
(= length – 8B)
User Datagram Protocol

Error Detection

- Recall E2E arguments: error check must be done E2E
- Error checking: checksum
  - sequence of 16-bit integers
  - binary addition
    - carry overflows wrapped;
    - 1’s complement of sum
      \[
      \begin{array}{c}
      1110011001100110 \\
      +1101010101010101 \\
      \hline
      1011101110111011
      \end{array}
      \]
  - sender computes+inserts
  - receiver compute+compares
  - weak protection [RFC 3385]
- No retransmission at L4
User Datagram Protocol
Flow Control and Congestion Control

- No flow control
  - sender blasts; may overwhelm receiver
- No congestion control
  - sender may congest network
  - no mechanism for sender to back off
User Datagram Protocol

Typical Applications

- Typical application: multimedia streaming

  *why?*
User Datagram Protocol

Typical Applications

- Typical application: multimedia streaming
  - loss tolerant
  - rate sensitive (do not adapt well to TCP congestion control)
  - research community considering alternatives
User Datagram Protocol

Typical Applications

• Typical application: multimedia streaming
  – loss tolerant
  – rate sensitive (do not adapt well to TCP congestion control)
  – research community considering alternatives

• Other applications: network control protocols
  – e.g. DNS
Real-Time Protocol
Overview and Transfer Mode

- RTP: real-time protocol [RFC 3550 / STD 0064]
- Streaming of data with real-time properties
  - uses UDP for basic transport
    - no connection establishment
    - basic end-to-end multiplexing
    - no reliability
    - no flow or congestion control
  - adds real-time support fields
    - sequence number
    - timestamp
    - source identifiers

More in Lecture MS
Real-Time Protocol
Segment Format

- Encapsulated in UDP
- RTP header
  - version = 02
  - P: padding bytes at end
  - X: extension header
  - CC: CSRC count (4b)
  - PT: payload type (7b)
  - sequence # (16b)
  - timestamp (32b)
    - resolution app dependent
  - source identifiers
    - SSRC
    - CSRC (mixed in)

<table>
<thead>
<tr>
<th>source port#</th>
<th>destination port #</th>
</tr>
</thead>
<tbody>
<tr>
<td>length</td>
<td>checksum</td>
</tr>
<tr>
<td>02</td>
<td>P X</td>
</tr>
<tr>
<td></td>
<td>CC</td>
</tr>
<tr>
<td></td>
<td>PT</td>
</tr>
<tr>
<td></td>
<td>sequence #</td>
</tr>
<tr>
<td></td>
<td>timestamp</td>
</tr>
<tr>
<td></td>
<td>SSRC: synchronisation source id</td>
</tr>
<tr>
<td></td>
<td>CSRC list: contributing source ids ...</td>
</tr>
<tr>
<td></td>
<td>application payload</td>
</tr>
<tr>
<td></td>
<td>optional padding</td>
</tr>
<tr>
<td></td>
<td>#B pad</td>
</tr>
</tbody>
</table>
Transmission Control Protocol Overview

• TCP: transmission control protocol
  RFC 0793 / STD 0007
  + RFC 1122 implementation requirements
  + RFC 1323 high performance extensions
  + RFC 2018 SACK (selective acknowledgements)
  + RFC 2581 congestion control
  + RFC 3168 ECN (explicit congestion notification)

• IANA defines name/number space for some fields
  – Internet Assigned Numbers Authority: www.iana.org/numbers.html
  – protocol types for IP (TCP = 06)
  – option kinds
  – well known port numbers
Transmission Control Protocol
Transfer Mode and Characteristics

- Connection-oriented transfer mode
  - latency of connection setup
  - connection state must be maintained on each end system

- Point-to-point full-duplex
  - reliable byte stream: no message boundaries
  - pipelined transfer (not just stop-and-wait)

- Flow control
  - will not overwhelm receiver

- Congestion control
  - attempt to avoid congesting network
  - implicit and optional ECN
Transmission Control Protocol

Segment Format: Overview

- Relatively large header
  - fixed length & position fields
  - variable length options
  - header length \( 4b \) in Bytes
    - \( = 40 + |\text{options}| \) B
- Variable length payload
- Options
  - may not be present
    (details later)
- Checksum on entire segment
  - same algorithm as UDP

<table>
<thead>
<tr>
<th>source port #</th>
<th>dest port #</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequence number</td>
<td></td>
</tr>
<tr>
<td>ack number</td>
<td></td>
</tr>
<tr>
<td>HL flags</td>
<td>receive window</td>
</tr>
<tr>
<td>checksum</td>
<td>urg data pointer</td>
</tr>
</tbody>
</table>

[options (variable length)]

application
payload
(variable length)
**Transmission Control Protocol**

**Segment Format: Overview**

<table>
<thead>
<tr>
<th>Control Flags</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>CWR</td>
<td>cong. win. reduced</td>
</tr>
<tr>
<td>ECE</td>
<td>ECN echo</td>
</tr>
<tr>
<td>URG</td>
<td>urgent data</td>
</tr>
<tr>
<td>ACK</td>
<td>acknowledgement</td>
</tr>
<tr>
<td>PSH</td>
<td>push data</td>
</tr>
<tr>
<td>RST</td>
<td>reset connection</td>
</tr>
<tr>
<td>SYN</td>
<td>connection setup</td>
</tr>
<tr>
<td>FIN</td>
<td>connection teardown</td>
</tr>
</tbody>
</table>

*details later*

- **Control flags (1 bit each)**
- CWR: congestion window reduced
- ECE: ECN echo
- URG: urgent data
- ACK: acknowledgement
- PSH: push data
- RST: reset connection
- SYN: connection setup
- FIN: connection teardown

<table>
<thead>
<tr>
<th>Field</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>source port #</td>
<td>dest port #</td>
</tr>
<tr>
<td>sequence number</td>
<td>ack number</td>
</tr>
<tr>
<td>receive window</td>
<td>checksum</td>
</tr>
<tr>
<td>urg data pointer</td>
<td>options (variable length)</td>
</tr>
<tr>
<td>application</td>
<td>payload</td>
</tr>
<tr>
<td></td>
<td>(variable length)</td>
</tr>
</tbody>
</table>
Transmission Control Protocol
Segment Format: No Options

- Header with no options
  - 20B long

<table>
<thead>
<tr>
<th>Field</th>
<th>Length</th>
</tr>
</thead>
<tbody>
<tr>
<td>source port #</td>
<td>16 bits</td>
</tr>
<tr>
<td>dest port #</td>
<td>16 bits</td>
</tr>
<tr>
<td>sequence number</td>
<td>32 bits</td>
</tr>
<tr>
<td>ack number</td>
<td>32 bits</td>
</tr>
<tr>
<td>flags</td>
<td>4 bits</td>
</tr>
<tr>
<td>receive window</td>
<td>32 bits</td>
</tr>
<tr>
<td>checksum</td>
<td>32 bits</td>
</tr>
<tr>
<td>urg data pointer</td>
<td>32 bits</td>
</tr>
</tbody>
</table>

application
(payload
(variable length)
Transmission Control Protocol
Segment Format: No Options Nor Data

- Entire segment
  - 20B long
- Example:
  - ACKs
  - significant fraction of Internet packets 40B

*why?*
Transmission Control Protocol
Segment Format: Options

- **Options**
  - standardised extensions
  - experimental use

- **Used for**
  - control: negotiation in SYN
  - data: alternate processing

- **Format (max 40B total)**
  - kind_{1B} (assigned by IANA)
  - length_{1B}
  - option fields (variable)

<table>
<thead>
<tr>
<th>source port #</th>
<th>dest port #</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequence number</td>
<td></td>
</tr>
<tr>
<td>ack number</td>
<td></td>
</tr>
<tr>
<td>HL</td>
<td>flags</td>
</tr>
<tr>
<td>receive window</td>
<td></td>
</tr>
<tr>
<td>checksum</td>
<td>urg data pointer</td>
</tr>
<tr>
<td>options (variable length)</td>
<td></td>
</tr>
<tr>
<td>application</td>
<td></td>
</tr>
<tr>
<td>payload</td>
<td></td>
</tr>
<tr>
<td>(variable length)</td>
<td></td>
</tr>
</tbody>
</table>
Transmission Control Protocol

Selected Options

- Selected TCP options
  
  http://www.iana.org/assignments/tcp-parameters
  
  00  end of option list
  01  no operation (for padding)
  02  MSS (max. segment size)
  03  window scale factor
  04  SACK permitted
  05  SACK blocks
  08  timestamp
Transmission Control Protocol

Segment Format: Multiplexing

- **Socket**: 5-tuple
  - \( \langle \text{sp}, \text{sa}, \text{dp}, \text{da}, \text{port} \rangle \)

- **Sender multiplexing**
  - assign **source port #**
  - network: **IP address**

- **Receiver demultiplexing**
  - network: **IP address**
  - IP: **protocol #**
    - **UDP** = 17
    - **TCP** = 06
  - **TCP**: **destination port #** determines receiving app

<table>
<thead>
<tr>
<th>source port #</th>
<th>dest port #</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequence number</td>
<td></td>
</tr>
<tr>
<td>ack number</td>
<td></td>
</tr>
<tr>
<td>HL flags</td>
<td>receive window</td>
</tr>
<tr>
<td>checksum</td>
<td>urg data pointer</td>
</tr>
<tr>
<td>options (variable length)</td>
<td></td>
</tr>
</tbody>
</table>

application
payload
(variable length)
Transmission Control Protocol

Multiplexing

- Interface to IP: protocol mux/demux (TCP = 06)
- Interface to application: port mux/demux
Transmission Control Protocol

Connection Management: Segment Format

- Flags define segment type
  - **ACK segment**
    - ACK for SYN or FIN
    - SYNACK may be *piggybacked*
    - ACK for data
  - **RST (reset) segment**
    - e.g. bad port received
  - **SYN (synchronise) segment**
    - connection setup
  - **FIN (finish) segment**
    - connection release

<table>
<thead>
<tr>
<th>source port #</th>
<th>dest port #</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequence number</td>
<td>ACK number</td>
</tr>
<tr>
<td>receive window</td>
<td>checksum</td>
</tr>
<tr>
<td>urg data pointer</td>
<td>options (variable length)</td>
</tr>
<tr>
<td>application payload (variable length)</td>
<td></td>
</tr>
</tbody>
</table>
Transmission Control Protocol
Connection Management: Setup

- Three way handshake
  - "client" and "server"
  1. SYN (init seq#) →
     - randomised for security
  2. ← SYNACK (init seq#)
     - randomised for security
     - after buffer allocation
     - SYN flood vulnerability
  3. ACK →
     - buffer allocation
     - segment may include data
     - client may send more
  4. server may return data
Transmission Control Protocol
Connection Management: Teardown

- Two two-way handshakes
  - each independent *half-close*
  - may occur simultaneously
    - *full close*
  - ACK can’t be piggybacked

- Client closes socket
  1. FIN →
  2. ← ACK
  - timed wait before close

- Server closes socket
  1. ← FIN
  2. ACK →
Transmission Control Protocol
Actual Connection State Machine

[Sewell] http://www.cl.cam.ac.uk/users/pes20/Netsem
Transmission Control Protocol
Segment Format: Sequence and Window

- **Sequence #**
  - forward data transfer
  - 1st byte # in segment

- **ACK number**
  - reverse acknowledgements
  - seq # of next byte expected
  - if ACK flag set
  - data segment may **piggyback** ACK

- **Receive window**
  - # unACKed bytes
Transmission Control Protocol

Unidirectional Data Transfer

- Data is sequence of bytes
  - number by byte #
  - (not segment #)
Transmission Control Protocol
Unidirectional Data Transfer

• Data is sequence of bytes
  – number by byte # (not segment #)
  – each segment ≤ MSS B
• Sequence numbers
  – byte # in byte stream of 1st byte in segment
Transmission Control Protocol
Unidirectional Data Transfer

- Data is sequence of bytes
  - number by byte # (not segment #)
  - each segment ≤ MSS B
- Sequence numbers
  - byte # in byte stream of 1st byte in segment
- ACK numbers
  - seq # of next byte expected
Transmission Control Protocol
Unidirectional Data Transfer

- Data is sequence of bytes
  - number by byte # (not segment #)
  - each segment ≤ MSS B
- Sequence numbers
  - byte # in byte stream of 1st byte in segment
- ACK numbers
  - seq # of next byte expected
Transmission Control Protocol
Unidirectional Data Transfer

• Data is sequence of bytes
  – number by byte # (not segment #)
  – each segment ≤ MSS B

• Sequence numbers
  – byte # in byte stream of 1st byte in segment

• ACK numbers
  – seq # of next byte expected
  – cumulative ACKs
Transmission Control Protocol
Unidirectional Data Transfer

- Data is sequence of bytes
  - number by byte # (not segment #)
  - each segment ≤ MSS
- Sequence numbers
  - byte # in byte stream of 1st byte in segment
- ACK numbers
  - seq # of next byte expected
  - cumulative ACKs
Transmission Control Protocol
Unidirectional Data Transfer

• Data is sequence of bytes
  – number by byte #
    (not segment #)
  – each segment ≤ MSS B

• Sequence numbers
  – byte # in byte stream
    of 1st byte in segment

• ACK numbers
  – seq # of next byte expected
  – cumulative ACKs

• Data is sequence of bytes
  – number by byte #
    (not segment #)
  – each segment ≤ MSS B

• Sequence numbers
  – byte # in byte stream
    of 1st byte in segment

• ACK numbers
  – seq # of next byte expected
  – cumulative ACKs
Transmission Control Protocol
Unidirectional Data Transfer

- Data is sequence of bytes
  - number by byte # (not segment #)
  - each segment ≤ MSS B

- Sequence numbers
  - byte # in byte stream of 1st byte in segment

- ACK numbers
  - seq # of next byte expected
  - cumulative ACKs

- Out of order arrival
  - implementation specific
Transmission Control Protocol
Bidirectional Data Transfer

- Data both directions
  - ACKs returned both ways
Transmission Control Protocol
Bidirectional Data Transfer

- Data both directions
  - ACKs returned both ways
  - may be *piggybacked*

Transmission Control Protocol
Bidirectional Data Transfer

- Data both directions
  - ACKs returned both ways
  - may be *piggybacked*
Transmission Control Protocol
Bidirectional Data Transfer

• Data both directions
  – ACKs returned both ways
  – may be *piggybacked*
  – *delayed ACK* waits briefly
Transmission Control Protocol
Bidirectional Data Transfer

• Data both directions
  – ACKs returned both ways
  – may be *piggybacked*
  – *delayed ACK* waits briefly

• Bidirectional stream
  – data segments
  – piggybacked ACKs
Transmission Control Protocol
Window-Based Feedback Control

- Windowing used for a combined:
  - error control
  - flow control
  - congestion control

- Window limits amount of unACKed data transmitted
Transmission Control Protocol
Error Control: Original Go-Back-\(N\)

- TCP provides reliable byte stream
- *Cumulative* ACKs using window: go-back-\(n\)
  - byte number rather than segment number
- Corrupted data
  - checksum fails
- Packet loss in network
  - timeout events
  - triple duplicate ACKS (fast retransmit)
Transmission Control Protocol

Error Control: SACK

• TCP provides reliable byte stream
  – cumulative ACKs perform poorly if loss rate not very low
  – cumulative ACKs limit ability to reorder

• Selective ACKs [RFC2018]
  – selective repeat ARQ mechanism
    • used in conjunction with cumulative ACK mechanisms
  – SACK byte range in TCP header option

• Retransmissions triggered by:
  – holes in SACK ranges (SACK is positive ACK)
Transmission Control Protocol

Segment Format: SACK Options

- SACK permitted negotiation
  - kind=4 in SYN
- SACK ranges
  - kind = 05
    - after kind=01 NOP pads
  - length = 8n+2 B for n SACK blocks
    - max of 3 SACK blocks
      - assuming another option
  - each block:
    - seq num of 1st byte
    - seq num of last byte +1
  - ACK semantics unchanged

<table>
<thead>
<tr>
<th>source port #</th>
<th>dest port #</th>
</tr>
</thead>
<tbody>
<tr>
<td>seq num</td>
<td></td>
</tr>
<tr>
<td>ack number</td>
<td></td>
</tr>
<tr>
<td>flags</td>
<td>receive window</td>
</tr>
<tr>
<td>checksum</td>
<td>urg data pointer</td>
</tr>
<tr>
<td>01 01 05 len</td>
<td>seq num of 1st byte</td>
</tr>
<tr>
<td>seq num after last byte</td>
<td></td>
</tr>
<tr>
<td>application payload</td>
<td></td>
</tr>
<tr>
<td>(variable length)</td>
<td></td>
</tr>
</tbody>
</table>
Transmission Control Protocol
Error Control: SACK Operation

- Capability announced with SYN
  - SACK permitted option 5
- Cumulative ACKs as before
  - ACK byte in last segment
  - with no missing prior segments
- Selective ACKs
  - sent only when non-contiguous segments received
  - indicate byte range received
  - holes between SACK blocks indicate missing byte ranges
Transmission Control Protocol

Timeout

*How to set TCP timeout value?*
Transmission Control Protocol

**Timeout**

- How to set TCP timeout value?
  - longer than RTT, but RTT varies
  - too short: premature timeout; unnecessary retransmissions
  - too long: slow reaction to segment loss

Implication?
Transmission Control Protocol

Timeout

• How to set TCP timeout value?
  – longer than RTT, but RTT varies
  – too short: premature timeout; unnecessary retransmissions
  – too long: slow reaction to segment loss

• Need a good estimate of RTT
Transmission Control Protocol
Round Trip Time Estimate

• RTT Estimate
  – sample RTT<sub>samp</sub>: measured time from segment transmission until ACK receipt
  – ignore retransmissions

• Estimated RTT<sub>est</sub> is smoothed
  – average several recent measurements
  – EWMA: exponentially weighted moving average
    • influence of past sample decreases exponentially
  – RTT<sub>est</sub> = (1−<i>α</i>) RTT<sub>est</sub> + <i>α</i> RTT<sub>samp</sub>
    • typical value: <i>α</i> = 0.125
Transmission Control Protocol

RTT Estimate Example

RTT: gaia.cs.umass.edu to fantasia.eurecom.fr

![Graph showing RTT values with time (seconds) on the x-axis and RTT (milliseconds) on the y-axis. The graph includes two lines: Sample RTT and Estimated RTT.](image-url)
Transmission Control Protocol

Fast Retransmit

- Time-out period often too long
  - long delay before resending lost packet
- Detect lost segments via duplicate ACKs
  - sender often sends many segments back-to-back
  - if lost there will likely be many duplicate ACKs
- If sender receives 3 duplicate ACKs
  - 4th identical ACK
  - assume that segment after ACKed data was lost
Transmission Control Protocol

Flow Control

- Sender matches transmission rate to receiver
  - receive window dynamically adjusted
- Receiver advertised its window
  - transmitted in each TCP segment header
- Sender limits unACKed data to receive window size
Transmission Control Protocol

Congestion Control

- Slow start to probe capacity
- Implicit congestion control
  - lack of expected ACK assumed to be congestion-based loss
  - sender halves congestion window (AIMD)
- Recovery
  - then retransmits (go-back-$n$ or SACK block)
  - begins increase of congestion window (AIMD)
    - 1 MSS (max. segment size) every RTT
Transmission Control Protocol

Congestion Control

- Initialisation phase: slow start
- Steady-state phase: AIMD
Transmission Control Protocol
Congestion Control Optimisations

- Fast recovery
  - 1/2 congestion window after 3dupACK rather than slow start
- RED: random early detection (discard) [Floyd 1993]
  - discard packets when router queue threshold exceeded
  - throttle TCP source earlier before congestion occurs
    more in lecture TQ
- ECN: explicit congestion notification [Floyd 1994]
  - use IP ECN to trigger multiplicative decrease
Transmission Control Protocol
Explicit Congestion Notification

- Network nodes set congestion indication
  - hist. ICMP source quench
  - ECN bits in IP header
- TCP reacts
  - uses flags for signalling

```
<table>
<thead>
<tr>
<th>source port #</th>
<th>dest port #</th>
</tr>
</thead>
<tbody>
<tr>
<td>sequence number</td>
<td></td>
</tr>
<tr>
<td>ACK number</td>
<td></td>
</tr>
<tr>
<td>receive window</td>
<td>checksum</td>
</tr>
<tr>
<td></td>
<td>urg data pointer</td>
</tr>
<tr>
<td>options (variable length)</td>
<td></td>
</tr>
<tr>
<td>application payload</td>
<td>(variable length)</td>
</tr>
<tr>
<td></td>
<td></td>
</tr>
</tbody>
</table>
```
Transmission Control Protocol
Congestion Control Implementations

• Tahoe
  – slow start and congestion avoidance algorithms
  – fast retransmit after triple duplicate ACKs
  – slow start after any loss (3dupACK or RTO)

• Reno – widely implemented
  – Tahoe +
  – fast recovery (MD after 3dupACK rather than SS)

• NewReno [Floyd 1999]
  – Reno +
  – partial ACK recovery
Transmission Control Protocol
Congestion Control Implementations

• BIC (binary-increase TCP) [Xu 2004]
  – binary search of window size during steady state
  – CUBIC variant is now default in Linux kernel > 2.6.19

• CTCP (compound TCP) [Tan 2005]
  – maintains dual AIMD and Vegas-style delay windows
  – default in MS-Windows since Vista
Transmission Control Protocol
High-Bandwidth-×-Delay Optimisations

- High bandwidth-×-delay paths (long fat pipes) *problem?*
Transmission Control Protocol
High-Bandwidth-×-Delay Optimisations

- High bandwidth-×-delay paths (long fat pipes)
  - problem: large number of bits in flight

*Implications to TCP?*
Transmission Control Protocol
High-Bandwidth-×-Delay Optimisations

- High bandwidth-×-delay paths (long fat pipes)
  - problem: large number of bits in flight
- Implications to TCP
  - max window size is 64KB (*why?*) (*problem?*)
  - packet loss has significant impact on *goodput*
  - only one RTT measurement per window
  - sequence numbers limited to 32 bit (*problem?*)
Transmission Control Protocol
High-Bandwidth-×-Delay Optimisations

• High bandwidth-×-delay paths (long fat pipes)
  – problem: large number of bits in flight

• Modifications [RFC 1323]
  – window scale option (kind=3)
  – SACK [RFC 2018]
  – timestamp option (kind=8) enables RTT per segment
  – PAWS uses timestamp to detect duplicate seq. numbers
    (protection against wrapped sequence numbers)
Transport Layer

Further Reading and Additional References

Transport Layer

Acknowledgements

Some material in these foils comes from the textbook supplementary materials:

- Kurose & Ross, *Computer Networking: A Top-Down Approach Featuring the Internet*
- Sterbenz & Touch, *High-Speed Networking: A Systematic Approach to High-Bandwidth Low-Latency Communication*

http://hsn-book.sterbenz.org