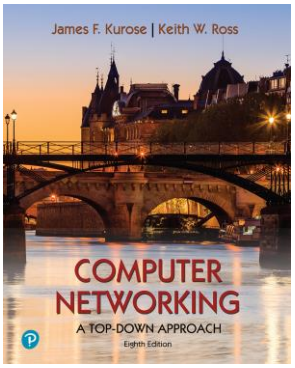
Chapter 3
Transport Layer

A note on the use of these PowerPoint slides:
We're making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a *lot* of work on our part. In return for use, we only ask the following:

- If you use these slides (e.g., in a class) that you mention their source (after all, we'd like people to use our book!)
- If you post any slides on a www site, that you note that they are adapted from (or perhaps identical to) our slides, and note our copyright of this material.

For a revision history, see the slide note for this page.

Thanks and enjoy! JFK/KWR



Computer Networking: A Top-Down Approach
8th edition
Jim Kurose, Keith Ross
Pearson, 2020

Slide 1 - Chapter 3 Transport Layer
Book reference (Kurose & Ross, Chapter 3, p. 182): short quote: "logical communication between application processes".
Open this slide with the big picture: the transport layer sits between the application layer and the network layer, and it gives application processes the impression of end-to-end communication. The first idea students should retain is this: IP carries datagrams between hosts, while the transport layer organizes communication between processes.
Do not read the roadmap as a plain list. Tie each item to a problem: multiplexing delivers data to the correct application, reliable transfer handles loss and corruption, flow control prevents overwhelming the receiver, and congestion control prevents overwhelming the network.
Visible slide keywords: Computer Networking: A Top-Down Approach, 8, th, edition, Jim Kurose, Keith Ross, Pearson, 2020, Chapter 3, Transport Layer, A note on the use of these PowerPoint slides:, We', re making these slides freely available to all (faculty, students, readers). They're in PowerPoint form so you see the animations; and can add, modify, and delete slides (including this one) and slide content to suit your needs. They obviously represent a, lot, of work on our part. In return for use, we only ask the following:, If you use these slides (e.g., in a class) that you mention their

source (after all, we’, d like people to use our book!), If you post any slides on a www site, that you note that they are adap.
Quick question: if a browser, a DNS client, and a messaging app are running at the same time, how does incoming data reach the correct program?
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

Transport layer: overview

Our goal:

- understand principles behind transport layer services:
  - multiplexing, demultiplexing
  - reliable data transfer
  - flow control
  - congestion control
- learn about Internet transport layer protocols:
  - UDP: connectionless transport
  - TCP: connection-oriented reliable transport
  - TCP congestion control

Slide 2 - Transport layer: overview
Book reference (Kurose & Ross, Chapter 3, p. 182): short quote: "logical communication between application processes".
Open this slide with the big picture: the transport layer sits between the application layer and the network layer, and it gives application processes the impression of end-to-end communication. The first idea students should retain is this: IP carries datagrams between hosts, while the transport layer organizes communication between processes.
Do not read the roadmap as a plain list. Tie each item to a problem: multiplexing delivers data to the correct application, reliable transfer handles loss and corruption, flow control prevents overwhelming the receiver, and congestion control prevents overwhelming the network.
Visible slide keywords: Transport layer: overview, Our goal:, understand principles behind transport layer services:, multiplexing, demultiplexing, reliable data transfer, flow control, congestion control, learn about Internet transport layer protocols:, UDP: connectionless transport, TCP: connection-oriented reliable transport, TCP congestion control, Transport Layer: 3-, 2.
Quick question: if a browser, a DNS client, and a messaging app are running at the same time, how does incoming data reach the correct program?

Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

Transport layer: roadmap

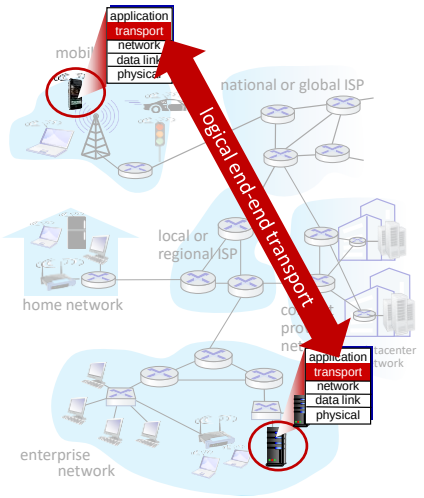
- Transport-layer services
- Multiplexing and demultiplexing
- Connectionless transport: UDP
- Principles of reliable data transfer
- Connection-oriented transport: TCP
- Principles of congestion control
- TCP congestion control
- Evolution of transport-layer functionality

Slide 3 - Transport layer: roadmap
Book reference (Kurose & Ross, Chapter 3, p. 182): short quote: "logical communication between application processes".
Open this slide with the big picture: the transport layer sits between the application layer and the network layer, and it gives application processes the impression of end-to-end communication. The first idea students should retain is this: IP carries datagrams between hosts, while the transport layer organizes communication between processes.
Do not read the roadmap as a plain list. Tie each item to a problem: multiplexing delivers data to the correct application, reliable transfer handles loss and corruption, flow control prevents overwhelming the receiver, and congestion control prevents overwhelming the network.
Visible slide keywords: Transport layer: roadmap, Transport-layer services, Multiplexing and demultiplexing, Connectionless transport: UDP, Principles of reliable data transfer, Connection-oriented transport: TCP, Principles of congestion control, TCP congestion control, Evolution of transport-layer functionality, Transport Layer: 3-, 3.
Quick question: if a browser, a DNS client, and a messaging app are running at the same time, how does incoming data reach the correct program?

Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

Transport services and protocols

- provide *logical communication* between application processes running on different hosts
- transport protocols actions in end systems:
  - sender: breaks application messages into *segments*, passes to network layer
  - receiver: reassembles segments into messages, passes to application layer
- two transport protocols available to Internet applications
  - TCP, UDP



Slide 4 - Transport services and protocols
Book reference (Kurose & Ross, Chapter 3, p. 182): short quote: "logical communication between application processes".
Define the transport-layer service directly: application processes communicate without handling the details of physical paths, routers, and link technologies. Routers do not run the end-to-end TCP or UDP logic; that logic is implemented in end systems.
Translate "logical communication" into a concrete idea: the application behaves as if it were talking directly to the remote process, even though the actual data crosses many network devices.
Visible slide keywords: Transport services and protocols, provide, logical communication, between application processes running on different hosts, mobile network, home network, enterprise, network, national or global ISP, local or regional ISP, datacenter, network, content, provider, network, application, transport, network, data link, physical, application, transport, network, data link, physical, logical end-end transport, transport protocols actions in end systems:, sender: breaks application messages into, segments, , passes to network layer, receiver: reassembles segments into messages, passes to application layer, two transport protocols available to Internet applications, TCP, UDP, Transport Layer: 3-, 4.

Check question: is the transport layer implemented inside routers or end systems?
Expected answer: end systems.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

Transport vs. network layer services and protocols

household analogy:

12 kids in Ann's house sending letters to 12 kids in Bill's house:

- hosts = houses
- processes = kids
- app messages = letters in envelopes

Slide 5 - Transport vs. network layer services and protocols
Book reference (Kurose & Ross, Chapter 3, p. 184): short quote: "network-layer protocol provides logical communication between hosts".
Explain the household analogy slowly: houses are hosts, children are processes, letters are application messages, Ann and Bill are the transport protocol, and the postal service is the network layer. This gives students an intuitive way to separate host-to-host delivery from process-to-process delivery.
Emphasize the distinction: the network layer aims to deliver a packet to the right machine; the transport layer delivers the data to the right application process on that machine. The transport layer relies on the network layer, but it adds ports, sockets, error checking, and, in some protocols, reliability.
Visible slide keywords: Transport vs. network layer services and protocols, household analogy:, 12 kids in Ann', s house sending letters to 12 kids in Bill's house:, hosts = houses, processes = kids, app messages = letters in envelopes, transport protocol = Ann and Bill who, demux, to in-house siblings, network-layer protocol = postal service, Transport Layer: 3-, 5.
Ask the class: if the postal service only delivers to the house, what information is needed inside the house to reach the correct child? In networking, the counterpart is the port.

Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

Transport vs. network layer services and protocols

- network layer: logical communication between *hosts*
- transport layer*:* logical communication between *processes*
  - relies on, enhances, network layer services

household analogy:

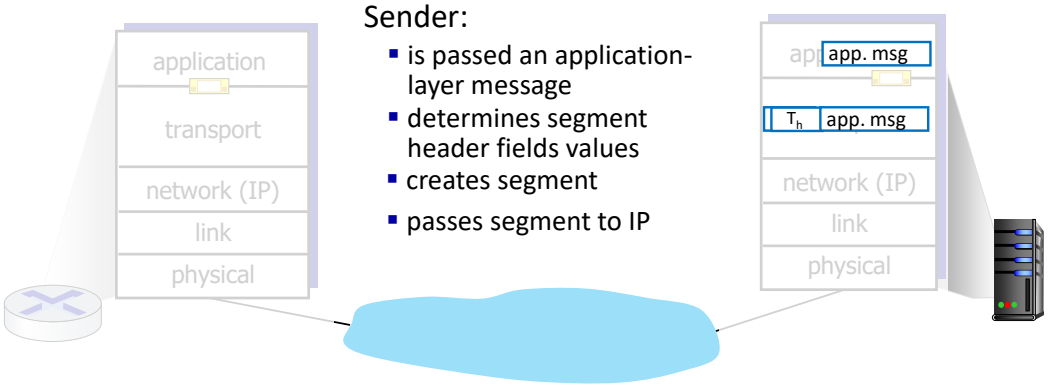
12 kids in Ann's house sending letters to 12 kids in Bill's house:

- hosts = houses
- processes = kids
- app messages = letters in envelopes

Slide 6 - Transport vs. network layer services and protocols
Book reference (Kurose & Ross, Chapter 3, p. 184): short quote: "network-layer protocol provides logical communication between hosts".
Explain the household analogy slowly: houses are hosts, children are processes, letters are application messages, Ann and Bill are the transport protocol, and the postal service is the network layer. This gives students an intuitive way to separate host-to-host delivery from process-to-process delivery.
Emphasize the distinction: the network layer aims to deliver a packet to the right machine; the transport layer delivers the data to the right application process on that machine. The transport layer relies on the network layer, but it adds ports, sockets, error checking, and, in some protocols, reliability.
Visible slide keywords: Transport vs. network layer services and protocols, network layer:, logical communication between, hosts, transport layer, :, logical communication between, processes, relies on, enhances, network layer services, household analogy:, 12 kids in Ann', s house sending letters to 12 kids in Bill's house:, hosts = houses, processes = kids, app messages = letters in envelopes, transport protocol = Ann and Bill who, demux, to in-house siblings, network-layer protocol = postal service, Transport Layer: 3-, 6.
Ask the class: if the postal service only delivers to the house, what information is

needed inside the house to reach the correct child? In networking, the counterpart is the port.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

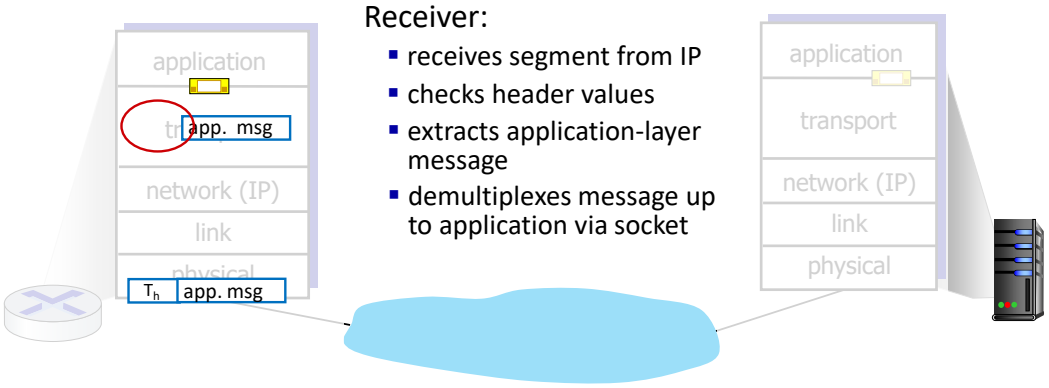
Transport Layer Actions



Slide 7 - Transport Layer Actions
Book reference (Kurose & Ross, Chapter 3, p. 182): short quote: "transport-layer packets, known as transport-layer segments".
Walk through the sender and receiver actions step by step. At the sender, the application message is accepted, a transport header is added, and the resulting segment is passed to IP. At the receiver, the segment arrives from IP, header values are checked, the payload is extracted, and the message is delivered through the correct socket.
Explain why header fields exist: carrying data is not enough. Source and destination ports, checksums, sequence numbers, and acknowledgments are control information that define protocol behavior. This slide prepares students for the UDP and TCP segment formats.
Visible slide keywords: physical, link, network (IP), application, physical, link, network (IP), application, transport, Transport Layer Actions, Sender:, app. msg, is passed an application-layer message, determines segment header fields values, creates segment, passes segment to IP, transport, T, h, T, h, app. msg, Transport Layer: 3-, 7.
Key point: a segment is application data plus a transport header; an IP datagram carries that segment at the network layer.
Close the slide by connecting it to the next one: this mechanism is one part of the

control information the transport layer adds so applications can communicate in a more organized and manageable way.

Transport Layer Actions

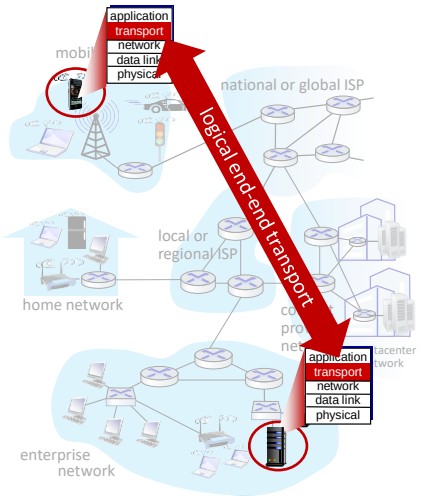


Slide 8 - Transport Layer Actions
Book reference (Kurose & Ross, Chapter 3, p. 182): short quote: "transport-layer packets, known as transport-layer segments".
Walk through the sender and receiver actions step by step. At the sender, the application message is accepted, a transport header is added, and the resulting segment is passed to IP. At the receiver, the segment arrives from IP, header values are checked, the payload is extracted, and the message is delivered through the correct socket.
Explain why header fields exist: carrying data is not enough. Source and destination ports, checksums, sequence numbers, and acknowledgments are control information that define protocol behavior. This slide prepares students for the UDP and TCP segment formats.
Visible slide keywords: physical, link, network (IP), application, physical, link, network (IP), application, transport, Transport Layer Actions, transport, Receiver:, app. msg, extracts application-layer message, checks header values, receives segment from IP, T, h, app. msg, demultiplexes message up to application via socket, Transport Layer: 3-, 8.
Key point: a segment is application data plus a transport header; an IP datagram carries that segment at the network layer.

Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

Two principal Internet transport protocols

- **TCP:** Transmission Control Protocol
  - reliable, in-order delivery
  - congestion control
  - flow control
  - connection setup
- **UDP:** User Datagram Protocol
  - unreliable, unordered delivery
  - no-frills extension of “best-effort” IP
- services not available:
  - delay guarantees
  - bandwidth guarantees



Transport Layer: 3-9

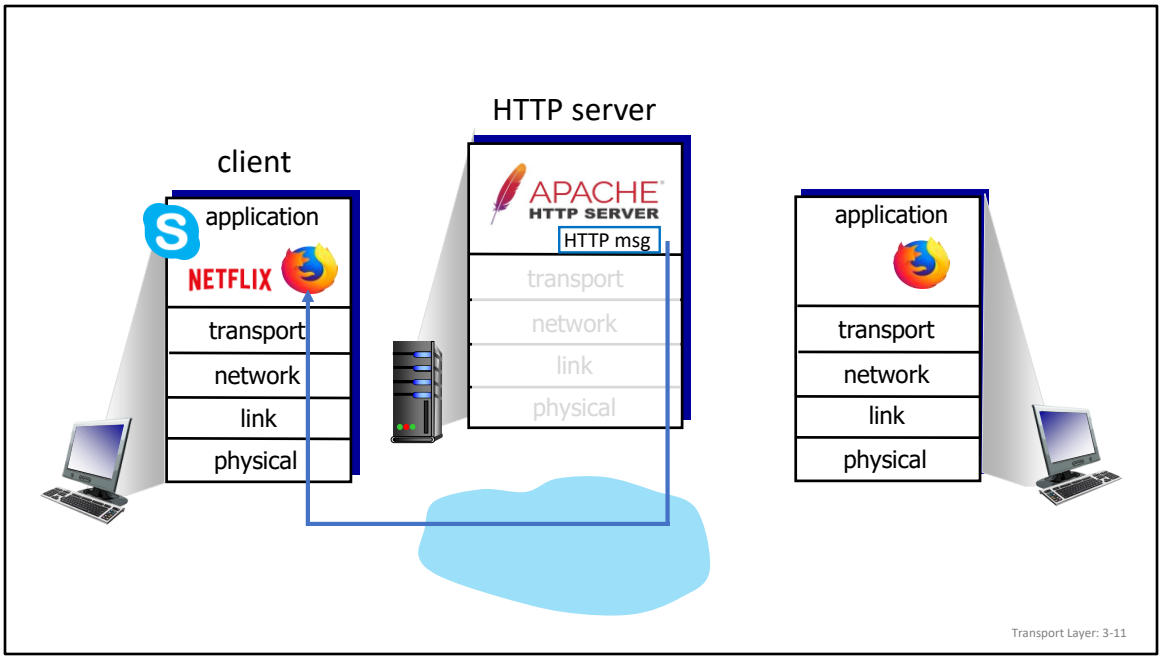
Slide 9 - Two principal Internet transport protocols
Book reference (Kurose & Ross, Chapter 3, p. 194): short quote: "no-frills, bare-bones transport protocol".
Compare TCP and UDP as two service models, not as simply good versus bad. TCP provides reliable, in-order delivery, flow and congestion control, and connection setup. UDP is leaner: it does not establish a connection and adds only a small amount of functionality on top of IP best effort.
Stress a key idea from the book: the services TCP and UDP can provide are limited by what IP provides. For example, because IP does not guarantee delay, TCP cannot give an absolute delay guarantee. TCP mainly adds reliability and control mechanisms.
Visible slide keywords: Two principal Internet transport protocols, mobile network, home network, enterprise, network, national or global ISP, local or regional ISP, datacenter, network, content, provider, network, application, transport, network, data link, physical, application, transport, network, data link, physical, logical end-end transport, TCP:, Transmission Control Protocol, reliable, in-order delivery, congestion control, flow control, connection setup, UDP:, User Datagram Protocol, unreliable, unordered delivery, no-frills extension of “, best-effort” IP, services not available:, delay guarantees, bandwidth guarantees, Transport Layer: 3-, 9.
Use examples: DNS fits UDP well because it is often a short query-response exchange;

a web page download usually needs TCP because the byte stream must arrive completely and in order.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

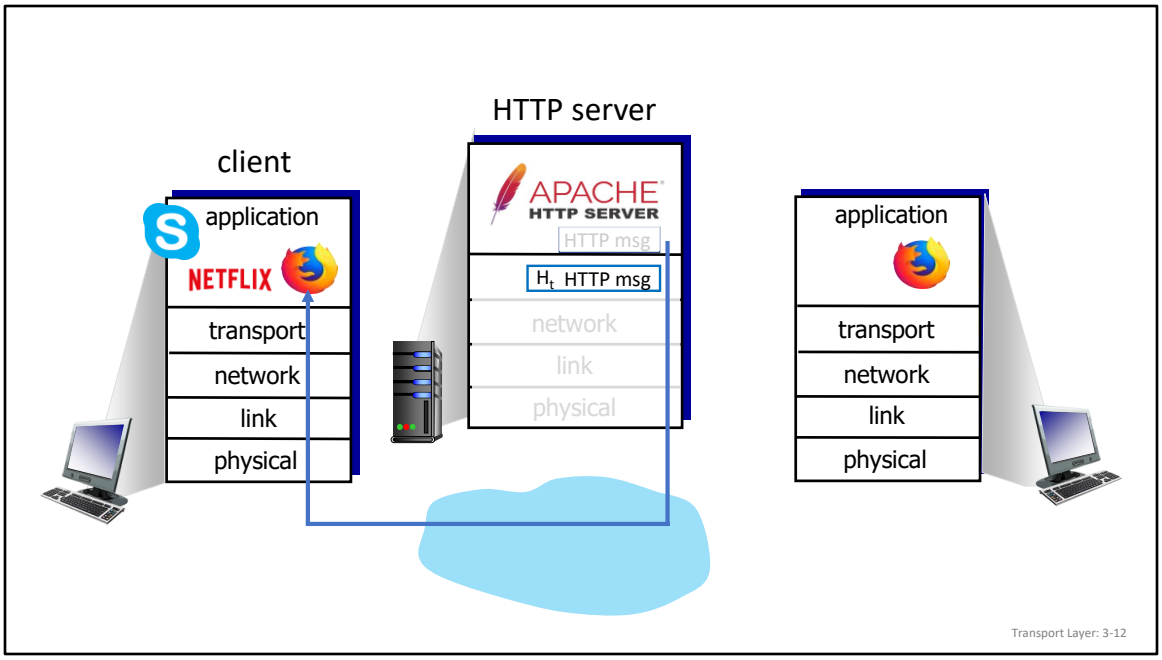
Chapter 3: roadmap

- Transport-layer services
- Multiplexing and demultiplexing
- Connectionless transport: UDP
- Principles of reliable data transfer
- Connection-oriented transport: TCP
- Principles of congestion control
- TCP congestion control
- Evolution of transport-layer functionality

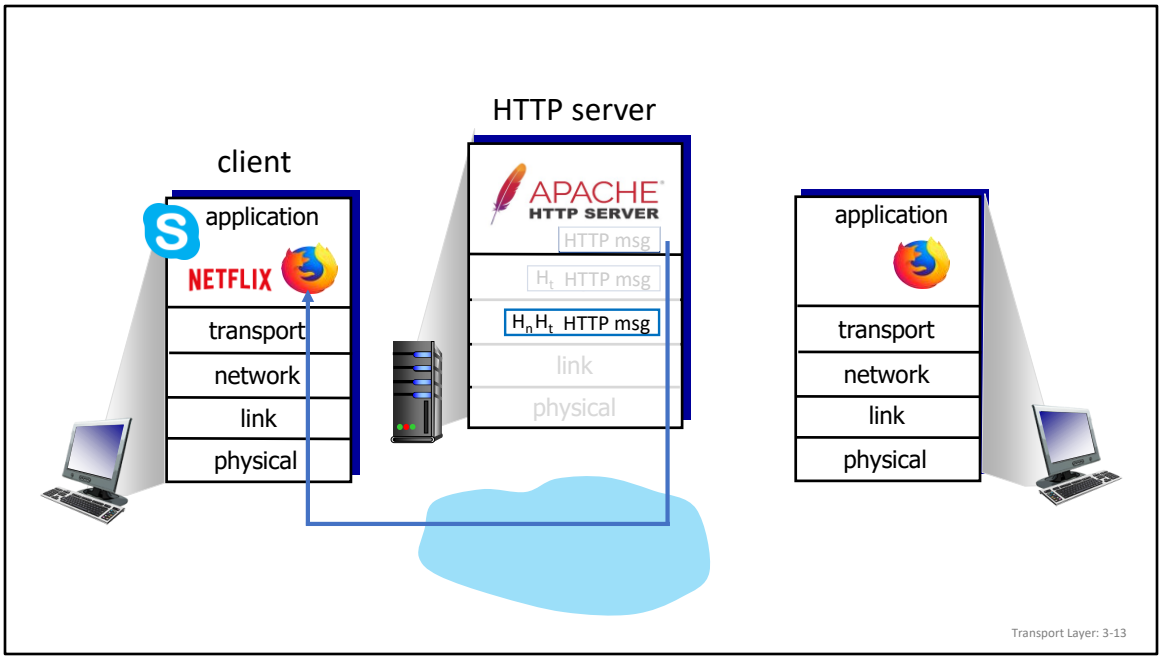
Slide 10 - Principles of reliable data transfer
Book reference (Kurose & Ross, Chapter 3, p. 186): short quote: "host-to-host delivery to process-to-process delivery".
Use this summary slide to consolidate ports and sockets. UDP uses a simpler delivery rule based on the destination port. TCP separates connections using a 4-tuple. That difference prepares the class for TCP connection management later.
Students should memorize not only the field names but also the reason for the difference: UDP carries individual datagrams; TCP carries an ongoing byte-stream connection.
Visible slide keywords: Chapter 3: roadmap, Transport-layer services, Multiplexing and demultiplexing, Connectionless transport: UDP, Principles of reliable data transfer, Connection-oriented transport: TCP, Principles of congestion control, TCP congestion control, Evolution of transport-layer functionality, Transport Layer: 3-, 10.
Closing question: which two source-side values distinguish two TCP connections arriving at the same server port? Source IP and source port.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.



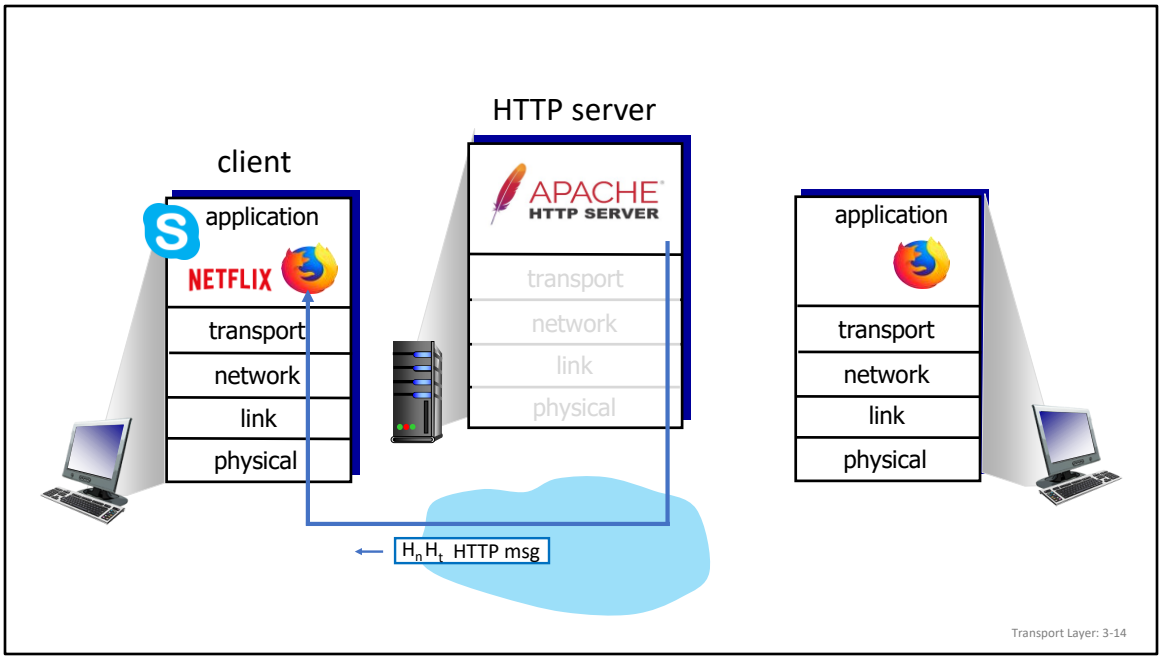
Slide 11 - transport
Book reference (Kurose & Ross, Chapter 3, p. 189): short quote: "Each socket in the host could be assigned a port number".
Use this sequence to show encapsulation and demultiplexing. The HTTP message becomes part of a TCP or UDP segment at the transport layer, is carried inside an IP datagram at the network layer, and is unpacked in reverse order at the destination.
Make sure students do not treat symbols such as Ht as decorative details. Each header helps answer the receiver-side question, "Which socket should this data go to?" That is why demultiplexing is a core transport-layer responsibility.
Visible slide keywords: transport, physical, link, network, transport, application, physical, link, network, transport, application, physical, link, network, HTTP server, client, HTTP msg, Transport Layer: 3-, 11.
Key point: multiplexing gathers data from many applications at the sender; demultiplexing distributes incoming segments to the correct sockets at the receiver.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.



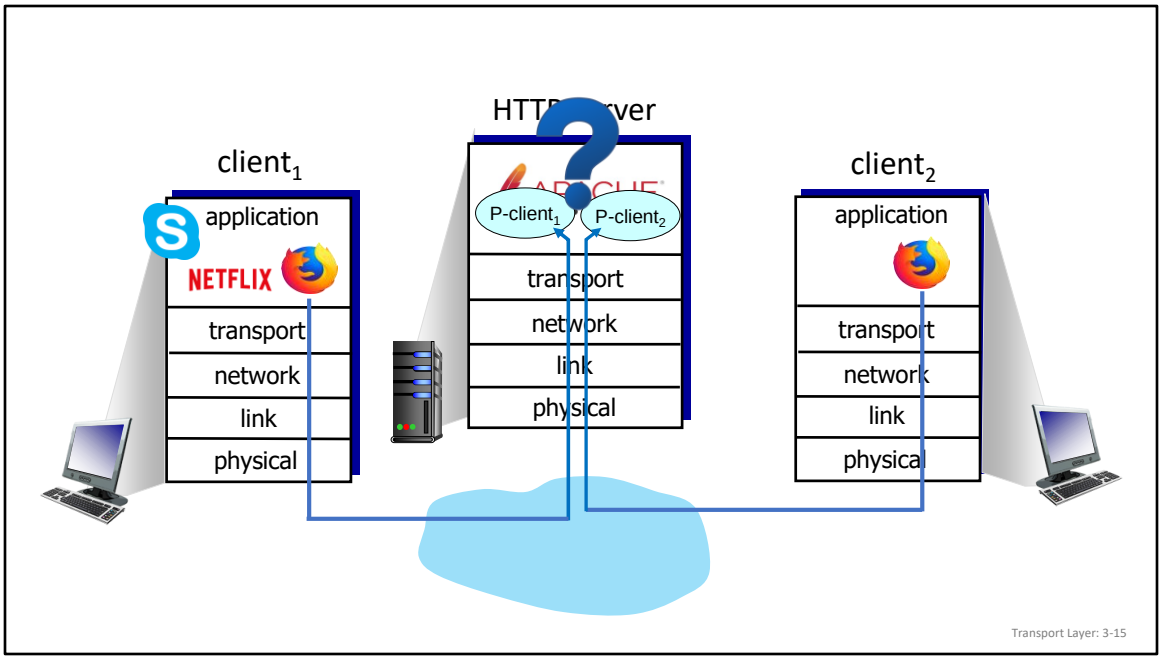
Slide 12 - transport
Book reference (Kurose & Ross, Chapter 3, p. 189): short quote: "Each socket in the host could be assigned a port number".
Use this sequence to show encapsulation and demultiplexing. The HTTP message becomes part of a TCP or UDP segment at the transport layer, is carried inside an IP datagram at the network layer, and is unpacked in reverse order at the destination.
Make sure students do not treat symbols such as Ht as decorative details. Each header helps answer the receiver-side question, "Which socket should this data go to?" That is why demultiplexing is a core transport-layer responsibility.
Visible slide keywords: transport, physical, link, network, transport, application, physical, link, network, transport, application, physical, link, network, HTTP server, client, HTTP msg, H, t, HTTP msg, Transport Layer: 3-, 12.
Key point: multiplexing gathers data from many applications at the sender; demultiplexing distributes incoming segments to the correct sockets at the receiver.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.



Slide 13 - transport
Book reference (Kurose & Ross, Chapter 3, p. 189): short quote: "Each socket in the host could be assigned a port number".
Use this sequence to show encapsulation and demultiplexing. The HTTP message becomes part of a TCP or UDP segment at the transport layer, is carried inside an IP datagram at the network layer, and is unpacked in reverse order at the destination.
Make sure students do not treat symbols such as Ht as decorative details. Each header helps answer the receiver-side question, "Which socket should this data go to?" That is why demultiplexing is a core transport-layer responsibility.
Visible slide keywords: transport, physical, link, network, transport, application, physical, link, network, transport, application, physical, link, network, HTTP server, client, HTTP msg, H, t, HTTP msg, H, t, H, n, HTTP msg, Transport Layer: 3-, 13.
Key point: multiplexing gathers data from many applications at the sender; demultiplexing distributes incoming segments to the correct sockets at the receiver.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.



Slide 14 - transport
Book reference (Kurose & Ross, Chapter 3, p. 189): short quote: "Each socket in the host could be assigned a port number".
Use this sequence to show encapsulation and demultiplexing. The HTTP message becomes part of a TCP or UDP segment at the transport layer, is carried inside an IP datagram at the network layer, and is unpacked in reverse order at the destination.
Make sure students do not treat symbols such as Ht as decorative details. Each header helps answer the receiver-side question, "Which socket should this data go to?" That is why demultiplexing is a core transport-layer responsibility.
Visible slide keywords: transport, physical, link, network, transport, application, physical, link, network, transport, application, physical, link, network, HTTP server, client, HTTP msg, H, t, H, n, Transport Layer: 3-, 14.
Key point: multiplexing gathers data from many applications at the sender; demultiplexing distributes incoming segments to the correct sockets at the receiver.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.



Slide 15 - transport
Book reference (Kurose & Ross, Chapter 3, p. 189): short quote: "Each socket in the host could be assigned a port number".
Use this sequence to show encapsulation and demultiplexing. The HTTP message becomes part of a TCP or UDP segment at the transport layer, is carried inside an IP datagram at the network layer, and is unpacked in reverse order at the destination. Make sure students do not treat symbols such as Ht as decorative details. Each header helps answer the receiver-side question, "Which socket should this data go to?" That is why demultiplexing is a core transport-layer responsibility.
Visible slide keywords: transport, physical, link, network, transport, application, physical, link, network, transport, application, physical, link, network, HTTP server, client, 1, client, 2, P-client, 1, P-client, 2, Transport Layer: 3-, 15.
Key point: multiplexing gathers data from many applications at the sender; demultiplexing distributes incoming segments to the correct sockets at the receiver.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

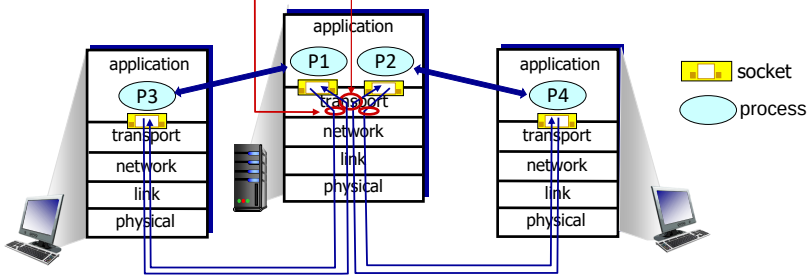
Multiplexing/demultiplexing

multiplexing at sender:

handle data from multiple sockets, add transport header (later used for demultiplexing)

demultiplexing at receiver:

use header info to deliver received segments to correct socket

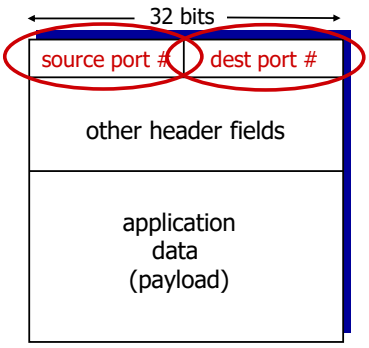


Slide 16 - Multiplexing/demultiplexing
Book reference (Kurose & Ross, Chapter 3, p. 186): short quote: "host-to-host delivery to process-to-process delivery".
Explain multiplexing and demultiplexing in both directions. At the sender, data from many sockets is turned into segments with header information. At the receiver, that header information is inspected so the segment can be delivered to the correct socket.
IP datagrams contain source and destination IP addresses; transport segments contain source and destination port numbers. Together, these identify the remote host and the application process within the host.
Visible slide keywords: Multiplexing/demultiplexing, process, socket, use header info to deliver, received segments to correct, socket, demultiplexing at receiver:, transport, application, physical, link, network, P2, P1, transport, application, physical, link, network, P4, transport, application, physical, link, network, P3, handle data from multiple, sockets, add transport header (later used for demultiplexing), multiplexing at sender:, Transport Layer: 3-, 16.
Mini example: if two browser tabs connect to the same web server, how does the operating system keep the connections separate?
Close the slide by connecting it to the next one: this mechanism is one part of the

control information the transport layer adds so applications can communicate in a more organized and manageable way.

How demultiplexing works

- host receives IP datagrams
  - each datagram has source IP address, destination IP address
  - each datagram carries one transport-layer segment
  - each segment has source, destination port number
- host uses *IP addresses & port numbers* to direct segment to appropriate socket



TCP/UDP segment format

Slide 17 - How demultiplexing
Book reference (Kurose & Ross, Chapter 3, p. 189): short quote: "Each socket in the host could be assigned a port number".
Explain multiplexing and demultiplexing in both directions. At the sender, data from many sockets is turned into segments with header information. At the receiver, that header information is inspected so the segment can be delivered to the correct socket.
IP datagrams contain source and destination IP addresses; transport segments contain source and destination port numbers. Together, these identify the remote host and the application process within the host.
Visible slide keywords: How demultiplexing, w, orks, host receives IP datagrams, each datagram has source IP address, destination IP address, each datagram carries one transport-layer segment, each segment has source, destination port number, host uses, IP addresses & port numbers, to direct segment to appropriate socket, source port #, dest port #, 32 bits, application, data, (payload), other header fields, TCP/UDP segment format, Transport Layer: 3-, 17.
Mini example: if two browser tabs connect to the same web server, how does the operating system keep the connections separate?
Close the slide by connecting it to the next one: this mechanism is one part of the

control information the transport layer adds so applications can communicate in a more organized and manageable way.

Connectionless demultiplexing

Recall:

- when creating socket, must specify *host-local* port #:

```
DatagramSocket mySocket1
 = new DatagramSocket(12534);
```

- when creating datagram to send into UDP socket, must specify
  - destination IP address
  - destination port #

when receiving host receives *UDP* segment:

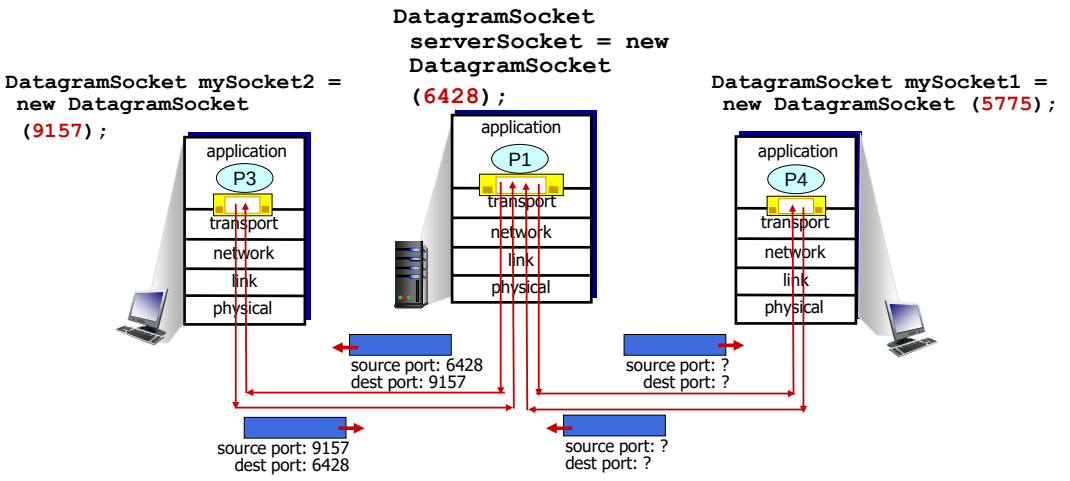
- checks destination port # in segment
- directs UDP segment to socket with that port #

IP/UDP datagrams with *same dest. port #,* but different source IP addresses and/or source port numbers will be directed to *same socket* at receiving host

Slide 18 - Connectionless demultiplexing
Book reference (Kurose & Ross, Chapter 3, p. 189): short quote: "Each socket in the host could be assigned a port number".
For UDP demultiplexing, state the basic rule: at the receiving host, the destination port number determines the UDP socket that receives the data. The source IP and source port are important for replies, but the main matching key for UDP socket delivery is the destination port.
In the example, serverSocket is bound to port 6428. UDP datagrams from different clients all reach the same server socket if their destination port is 6428. This matches UDP's connectionless nature: each datagram is handled independently.
Visible slide keywords: Connectionless demultiplexing, Recall:, when creating socket, must specify, host-local, port #:, DatagramSocket, mySocket1 = new, DatagramSocket, (, 12534,);, when receiving host receives, UDP, segment:, checks destination port # in segment, directs UDP segment to socket with that port #, when creating datagram to send into UDP socket, must specify, destination IP address, destination port #, IP/UDP datagrams with, same, dest, . port #,, but different source IP addresses and/or source port numbers will be directed to, same socket, at receiving host, Transport Layer: 3-, 18.
Ask: if a UDP server receives requests from many clients on the same port, which

fields help it send replies? Source IP and source port.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

Connectionless demultiplexing: an example



Slide 19 - Connectionless demultiplexing
Book reference (Kurose & Ross, Chapter 3, p. 189): short quote: "Each socket in the host could be assigned a port number".
For UDP demultiplexing, state the basic rule: at the receiving host, the destination port number determines the UDP socket that receives the data. The source IP and source port are important for replies, but the main matching key for UDP socket delivery is the destination port.
In the example, serverSocket is bound to port 6428. UDP datagrams from different clients all reach the same server socket if their destination port is 6428. This matches UDP's connectionless nature: each datagram is handled independently.
Visible slide keywords: Connectionless demultiplexing: an example, DatagramSocket, serverSocket, = new, DatagramSocket, (, 6428,);, transport, application, physical, link, network, P3, transport, application, physical, link, network, P1, transport, application, physical, link, network, P4, DatagramSocket, mySocket1 = new, DatagramSocket, (, 5775,);, DatagramSocket, mySocket2 = new, DatagramSocket, (, 9157,);, source port: 9157, dest port: 6428, source port: 6428, dest port: 9157, source port: ?, dest port: ?, source port: ?, dest port: ?, Transport Layer: 3-, 19.
Ask: if a UDP server receives requests from many clients on the same port, which fields help it send replies? Source IP and source port.

Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

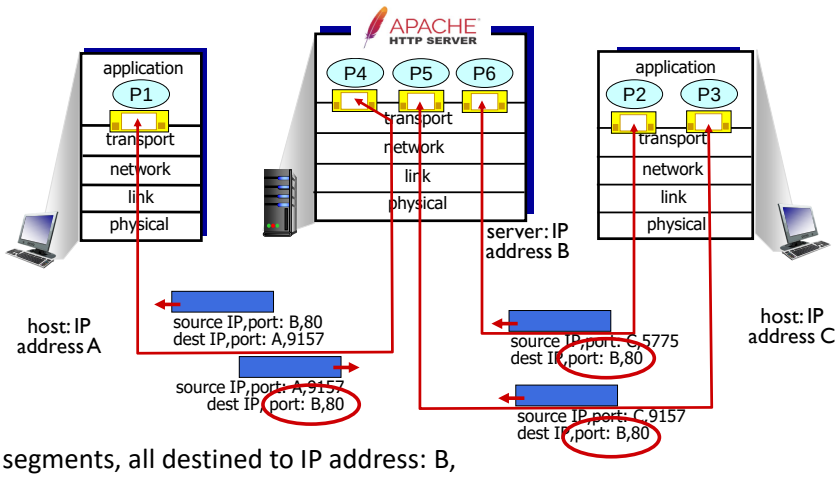
Connection-oriented demultiplexing

- TCP socket identified by 4-tuple:
  - source IP address
  - source port number
  - dest IP address
  - dest port number
- demux: receiver uses *all four values (4-tuple)* to direct segment to appropriate socket
- server may support many simultaneous TCP sockets:
  - each socket identified by its own 4-tuple
  - each socket associated with a different connecting client

Slide 20 - Connection-oriented demultiplexing
Book reference (Kurose & Ross, Chapter 3, p. 189): short quote: "Each socket in the host could be assigned a port number".
Highlight what makes TCP demultiplexing different from UDP. A TCP socket is identified by a 4-tuple: source IP address, source port number, destination IP address, and destination port number. This allows one web server to distinguish thousands of simultaneous client connections.
In the server example, each client connection has a separate connection socket. The destination port may remain the same, such as 80 or 443, but the source IP and source port differ, so each connection socket is distinct.
Visible slide keywords: Connection-oriented demultiplexing, TCP socket identified by, 4-, tuple:, source IP address, source port number, dest, IP address, dest, port number, server may support many simultaneous TCP sockets:, each socket identified by its own 4-tuple, each socket associated with a different connecting client, demux: receiver uses, all four values, (4-tuple), to direct segment to appropriate socket, Transport Layer: 3-, 20.
Key point: because TCP is connection-oriented, it carries richer information about who is talking to whom within which connection.
Close the slide by connecting it to the next one: this mechanism is one part of the

control information the transport layer adds so applications can communicate in a more organized and manageable way.

Connection-oriented demultiplexing: example



Three segments, all destined to IP address: B, dest port: 80 are demultiplexed to *different* sockets

Transport Layer: 3-21

Slide 21 - Connection-oriented demultiplexing
Book reference (Kurose & Ross, Chapter 3, p. 189): short quote: "Each socket in the host could be assigned a port number".
Highlight what makes TCP demultiplexing different from UDP. A TCP socket is identified by a 4-tuple: source IP address, source port number, destination IP address, and destination port number. This allows one web server to distinguish thousands of simultaneous client connections.
In the server example, each client connection has a separate connection socket. The destination port may remain the same, such as 80 or 443, but the source IP and source port differ, so each connection socket is distinct.
Visible slide keywords: Connection-oriented demultiplexing: example, transport, application, physical, link, network, P1, transport, application, physical, link, P4, transport, application, physical, link, network, P2, host: IP address A, host: IP address C, network, P6, P5, P3, source, IP,port, : A,9157, dest, IP, port: B,80, source IP,port: B,80, dest IP,port: A,9157, source IP,port: C,5775, dest IP,port: B,80, source, IP,port, : C,9157, dest, IP,port, : B,80, server: IP address B, Three segments, all destined to IP address: B,, dest, port: 80 are demultiplexed to, different, sockets, Transport Layer: 3-, 21.
Key point: because TCP is connection-oriented, it carries richer information about

who is talking to whom within which connection.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

Summary

- Multiplexing, demultiplexing: based on segment, datagram header field values
- **UDP:** demultiplexing using destination port number (only)
- **TCP:** demultiplexing using 4-tuple: source and destination IP addresses, and port numbers
- Multiplexing/demultiplexing happen at *all* layers

Slide 22 - Multiplexing/demultiplexing
Book reference (Kurose & Ross, Chapter 3, p. 186): short quote: "host-to-host delivery to process-to-process delivery".
Use this summary slide to consolidate ports and sockets. UDP uses a simpler delivery rule based on the destination port. TCP separates connections using a 4-tuple. That difference prepares the class for TCP connection management later.
Students should memorize not only the field names but also the reason for the difference: UDP carries individual datagrams; TCP carries an ongoing byte-stream connection.
Visible slide keywords: Summary, Multiplexing, demultiplexing: based on segment, datagram header field values, UDP:, demultiplexing using destination port number (only), TCP:, demultiplexing using 4-tuple: source and destination IP addresses, and port numbers, Multiplexing/demultiplexing happen at, all, layers, Transport Layer: 3-, 22.
Closing question: which two source-side values distinguish two TCP connections arriving at the same server port? Source IP and source port.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

Chapter 3: roadmap

- Transport-layer services
- Multiplexing and demultiplexing
- Connectionless transport: UDP
- Principles of reliable data transfer
- Connection-oriented transport: TCP
- Principles of congestion control
- TCP congestion control
- Evolution of transport-layer functionality

Slide 23 - Principles of reliable data transfer
Book reference (Kurose & Ross, Chapter 3, p. 194): short quote: "no-frills, bare-bones transport protocol".
Present UDP as intentionally simple, not as a useless protocol. It has no connection setup delay, a small header, no built-in congestion or flow control, and it gives applications speed and control.
Connect each use case to the type of problem: DNS often uses short query-response exchanges; streaming multimedia can tolerate some loss but is sensitive to delay; SNMP benefits from simplicity; HTTP/3 uses QUIC to build richer transport behavior above UDP.
Visible slide keywords: Chapter 3: roadmap, Transport-layer services, Multiplexing and demultiplexing, Connectionless transport: UDP, Principles of reliable data transfer, Connection-oriented transport: TCP, Principles of congestion control, TCP congestion control, Evolution of transport-layer functionality, Transport Layer: 3-, 23.
Key point: UDP itself does not provide reliability. If reliability is needed, the application or a protocol above UDP must add it.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

UDP: User Datagram Protocol

- "no frills," "bare bones" Internet transport protocol
- "best effort" service, UDP segments may be:
  - lost
  - delivered out-of-order to app
- *connectionless:*
  - no handshaking between UDP sender, receiver
  - each UDP segment handled independently of others

Why is there a UDP?

- no connection establishment (which can add RTT delay)
- simple: no connection state at sender, receiver
- small header size
- no congestion control
  - UDP can blast away as fast as desired!
  - can function in the face of congestion

Slide 24 - UDP: User Datagram Protocol
Book reference (Kurose & Ross, Chapter 3, p. 194): short quote: "no-frills, bare-bones transport protocol".
Present UDP as intentionally simple, not as a useless protocol. It has no connection setup delay, a small header, no built-in congestion or flow control, and it gives applications speed and control.
Connect each use case to the type of problem: DNS often uses short query-response exchanges; streaming multimedia can tolerate some loss but is sensitive to delay; SNMP benefits from simplicity; HTTP/3 uses QUIC to build richer transport behavior above UDP.
Visible slide keywords: UDP: User Datagram Protocol, "no frills," "bare bones" Internet transport protocol, "best effort" service, UDP segments may be:, lost, delivered out-of-order to app, no connection establishment (which can add RTT delay), simple: no connection state at sender, receiver, small header size, no congestion control, UDP can blast away as fast as desired!, can function in the face of congestion, W, hy, is there a UDP?, connectionless:, no handshaking between UDP sender, receiver, each UDP segment handled independently of others, Transport Layer: 3-, 24.
Key point: UDP itself does not provide reliability. If reliability is needed, the

application or a protocol above UDP must add it.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

UDP: User Datagram Protocol

- UDP use:
  - streaming multimedia apps (loss tolerant, rate sensitive)
  - DNS
  - SNMP
  - HTTP/3
- if reliable transfer needed over UDP (e.g., HTTP/3):
  - add needed reliability at application layer
  - add congestion control at application layer

Slide 25 - UDP: User Datagram Protocol
Book reference (Kurose & Ross, Chapter 3, p. 194): short quote: "no-frills, bare-bones transport protocol".
Present UDP as intentionally simple, not as a useless protocol. It has no connection setup delay, a small header, no built-in congestion or flow control, and it gives applications speed and control.
Connect each use case to the type of problem: DNS often uses short query-response exchanges; streaming multimedia can tolerate some loss but is sensitive to delay; SNMP benefits from simplicity; HTTP/3 uses QUIC to build richer transport behavior above UDP.
Visible slide keywords: UDP: User Datagram Protocol, UDP use:, streaming multimedia apps (loss tolerant, rate sensitive), DNS, SNMP, HTTP/3, if reliable transfer needed over UDP (e.g., HTTP/3):, add needed reliability at application layer, add congestion control at application layer, Transport Layer: 3-, 25.
Key point: UDP itself does not provide reliability. If reliability is needed, the application or a protocol above UDP must add it.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

UDP: User Datagram Protocol [RFC 768]

INTERNET STANDARD

RFC 768

J. Postel
ISI
28 August 1980

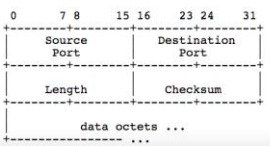
User Datagram Protocol

Introduction

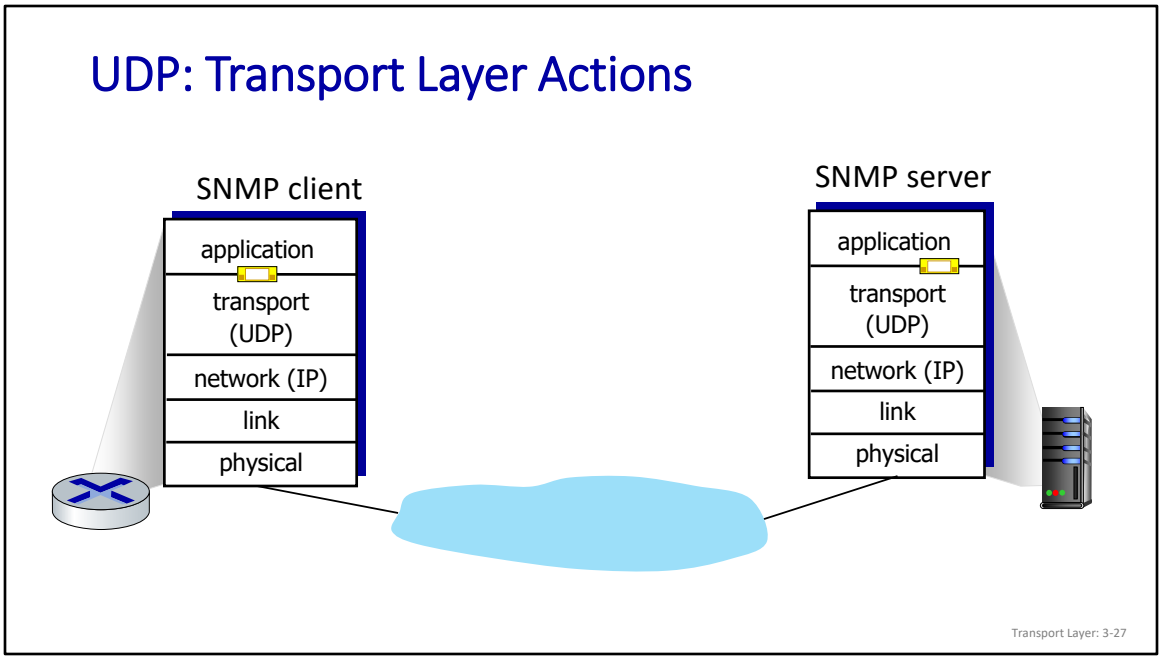
This User Datagram Protocol (UDP) is defined to make available a datagram mode of packet-switched computer communication in the environment of an interconnected set of computer networks. This protocol assumes that the Internet Protocol (IP) [1] is used as the underlying protocol.

This protocol provides a procedure for application programs to send messages to other programs with a minimum of protocol mechanism. The protocol is transaction oriented, and delivery and duplicate protection are not guaranteed. Applications requiring ordered reliable delivery of streams of data should use the Transmission Control Protocol (TCP) [2].

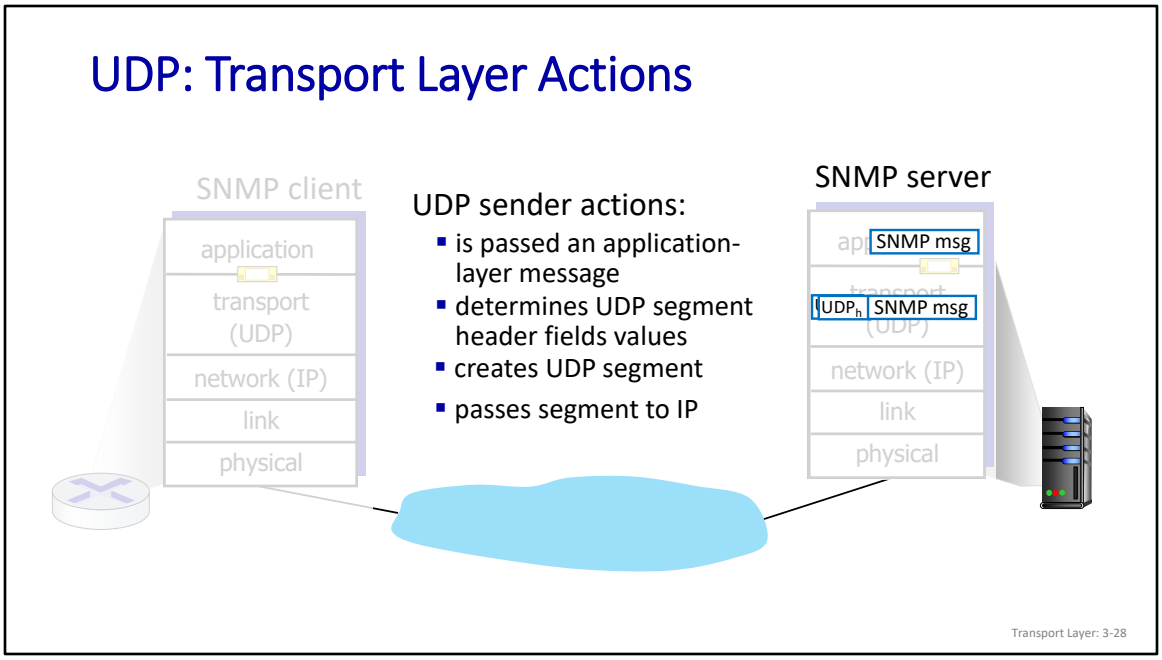
Format



Slide 26 - UDP: User Datagram Protocol
Book reference (Kurose & Ross, Chapter 3, p. 194): short quote: "no-frills, bare-bones transport protocol".
Present UDP as intentionally simple, not as a useless protocol. It has no connection setup delay, a small header, no built-in congestion or flow control, and it gives applications speed and control.
Connect each use case to the type of problem: DNS often uses short query-response exchanges; streaming multimedia can tolerate some loss but is sensitive to delay; SNMP benefits from simplicity; HTTP/3 uses QUIC to build richer transport behavior above UDP.
Visible slide keywords: UDP: User Datagram Protocol, [RFC 768], Transport Layer: 3-, 26.
Key point: UDP itself does not provide reliability. If reliability is needed, the application or a protocol above UDP must add it.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

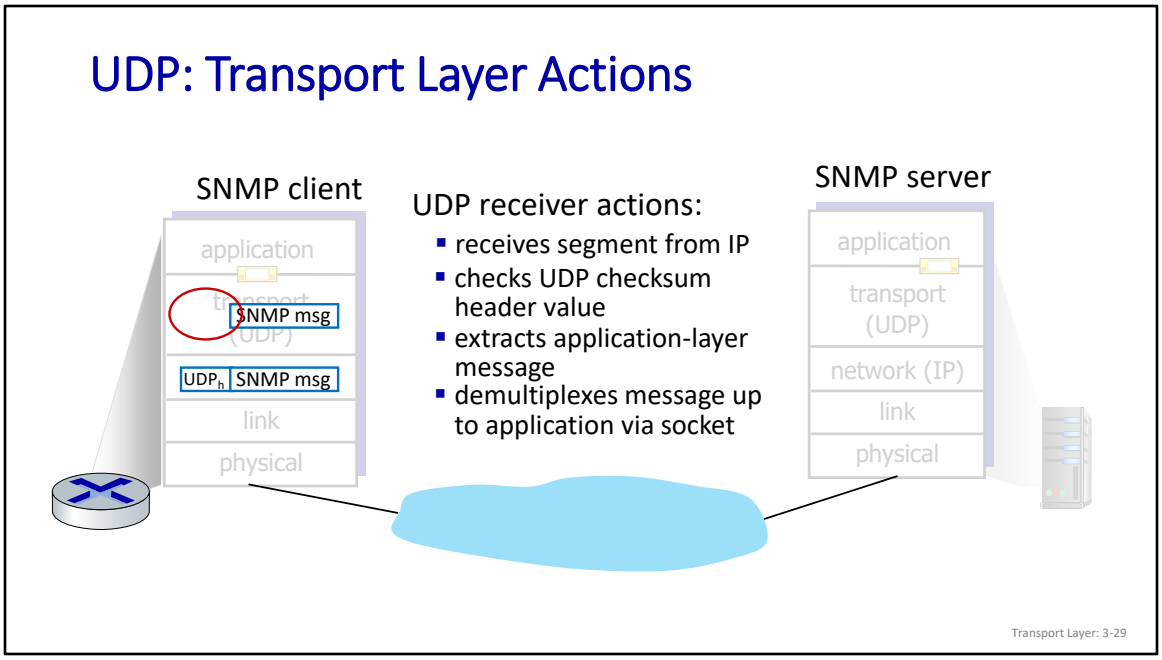


Slide 27 - Transport Layer Actions
Book reference (Kurose & Ross, Chapter 3, p. 182): short quote: "transport-layer packets, known as transport-layer segments".
Follow the SNMP example as a transport-layer action sequence. The SNMP application gives a message to UDP, UDP adds a small header and hands it to IP, and the receiver's UDP layer extracts the payload and delivers it to the SNMP application.
Remind students that UDP does not perform connection setup. Each message is treated like an independent datagram; there is no SYN/ACK preparation. This makes UDP fast, but it gives no loss or ordering guarantee.
Visible slide keywords: SNMP server, SNMP client, transport, (UDP), physical, link, network (IP), application, UDP: Transport Layer Actions, transport, (UDP), physical, link, network (IP), application, Transport Layer: 3-, 27.
Quick question: can two SNMP messages sent over UDP arrive out of order? Yes, UDP does not guarantee ordering.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.



Slide 28 - Transport Layer Actions
Book reference (Kurose & Ross, Chapter 3, p. 182): short quote: "transport-layer packets, known as transport-layer segments".
Follow the SNMP example as a transport-layer action sequence. The SNMP application gives a message to UDP, UDP adds a small header and hands it to IP, and the receiver's UDP layer extracts the payload and delivers it to the SNMP application.
Remind students that UDP does not perform connection setup. Each message is treated like an independent datagram; there is no SYN/ACK preparation. This makes UDP fast, but it gives no loss or ordering guarantee.
Visible slide keywords: SNMP server, SNMP client, transport, (UDP), physical, link, network (IP), application, transport, (UDP), physical, link, network (IP), application, UDP: Transport Layer Actions, UDP sender actions:, SNMP msg, is passed an application-layer message, determines UDP segment header fields values, creates UDP segment, passes segment to IP, UDP, h, UDP, h, SNMP msg, Transport Layer: 3-, 28.
Quick question: can two SNMP messages sent over UDP arrive out of order? Yes, UDP does not guarantee ordering.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a

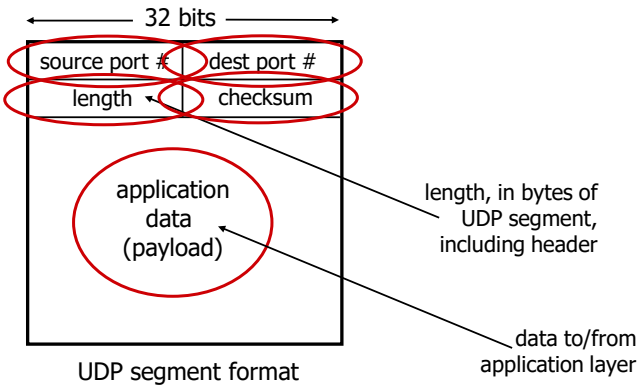
more organized and manageable way.



Slide 29 - Transport Layer Actions
Book reference (Kurose & Ross, Chapter 3, p. 182): short quote: "transport-layer packets, known as transport-layer segments".
Follow the SNMP example as a transport-layer action sequence. The SNMP application gives a message to UDP, UDP adds a small header and hands it to IP, and the receiver's UDP layer extracts the payload and delivers it to the SNMP application.
Remind students that UDP does not perform connection setup. Each message is treated like an independent datagram; there is no SYN/ACK preparation. This makes UDP fast, but it gives no loss or ordering guarantee.
Visible slide keywords: SNMP server, SNMP client, transport, (UDP), physical, link, network (IP), application, transport, (UDP), physical, link, network (IP), application, UDP: Transport Layer Actions, UDP receiver actions:, SNMP msg, extracts application-layer message, checks UDP checksum header value, receives segment from IP, UDP, h, SNMP msg, demultiplexes message up to application via socket, Transport Layer: 3-, 29.
Quick question: can two SNMP messages sent over UDP arrive out of order? Yes, UDP does not guarantee ordering.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a

more organized and manageable way.

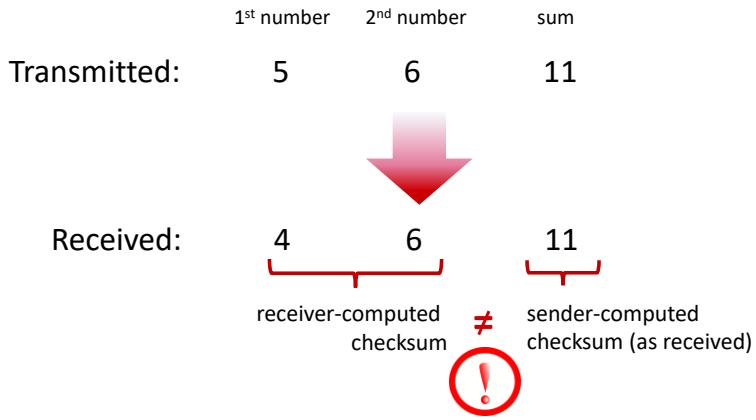
UDP segment header



Slide 30 - UDP segment
Book reference (Kurose & Ross, Chapter 3, p. 198): short quote: "UDP checksum provides for error detection".
Explain the UDP header field by field: source port is useful for replies, destination port supports demultiplexing, length gives segment size, and checksum provides error detection. The payload is the application data.
The 32-bit row layout helps students see how small the header is. A UDP header is 8 bytes, which is one reason it is leaner than TCP. The checksum also shows that UDP is not completely empty; it still provides basic error detection.
Visible slide keywords: UDP segment, h, eader, source port #, dest port #, 32 bits, application, data, (payload), UDP segment format, length, checksum, length, in bytes of UDP segment, including header, data to/from application layer, Transport Layer: 3-, 30.
Key point: the checksum can detect errors, but it does not correct them. UDP may discard the damaged segment or pass it upward with a warning.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

UDP checksum

Goal: detect errors (*i.e.,* flipped bits) in transmitted segment



Slide 31 - UDP checksum
Book reference (Kurose & Ross, Chapter 3, p. 198): short quote: "UDP checksum provides for error detection".
In the checksum slides, the goal is for students to understand bit-level error detection. The sender treats the UDP segment as a sequence of 16-bit words, computes the one's-complement sum, and stores the checksum. The receiver repeats the logic to check the segment.
Point out that the Internet checksum is weak protection. Some error patterns can leave the sum unchanged, so this is not cryptographic integrity. It is still useful because it is simple and fast.
Visible slide keywords: UDP checksum, Transmitted: 5 6 11, Goal:, detect, errors (, i.e.,, flipped bits) in transmitted segment, Received: 4 6 11, 1, st, number, 2, nd, number, sum, receiver-computed, checksum, sender-computed, checksum (as received), =, Transport Layer: 3-, 31.
Mini exercise: if two bits flip, will the checksum always detect the problem? No; slide 34 illustrates this weakness.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

UDP checksum

Goal: detect errors (*i.e.,* flipped bits) in transmitted segment

sender:

- treat contents of UDP segment (including UDP header fields and IP addresses) as sequence of 16-bit integers
- checksum: addition (one's complement sum) of segment content
- checksum value put into UDP checksum field

receiver:

- compute checksum of received segment
- check if computed checksum equals checksum field value:
  - Not equal - error detected
  - Equal - no error detected. *But maybe errors nonetheless?* More later ….

Slide 32 - UDP checksum
Book reference (Kurose & Ross, Chapter 3, p. 198): short quote: "UDP checksum provides for error detection".
In the checksum slides, the goal is for students to understand bit-level error detection. The sender treats the UDP segment as a sequence of 16-bit words, computes the one's-complement sum, and stores the checksum. The receiver repeats the logic to check the segment.
Point out that the Internet checksum is weak protection. Some error patterns can leave the sum unchanged, so this is not cryptographic integrity. It is still useful because it is simple and fast.
Visible slide keywords: UDP checksum, sender:, treat contents of UDP segment, (including UDP header fields and IP addresses), as sequence of 16-bit integers, checksum:, addition, (one', s complement sum), of segment content, checksum value put into UDP checksum field, receiver, :, compute checksum of received segment, check if computed checksum equals checksum field value:, Not equal - error detected, Equal - no error detected., But maybe errors nonetheless?, More later …., Goal:, detect, errors (, i.e.,, flipped bits) in transmitted segment, Transport Layer: 3-, 32.
Mini exercise: if two bits flip, will the checksum always detect the problem? No; slide

34 illustrates this weakness.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

Internet checksum: an example

example: add two 16-bit integers

	1	1	1	0	0	1	1	0	0	1	1	0	0	1	1	0
	1	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
wraparound	(1)	1	0	1	1	1	0	1	1	1	0	1	1	1	0	1	1
sum		1	0	1	1	1	0	1	1	1	0	1	1	1	1	0	0
checksum		0	1	0	0	0	1	0	0	0	1	0	0	0	0	1	1

Note: when adding numbers, a carryout from the most significant bit needs to be added to the result

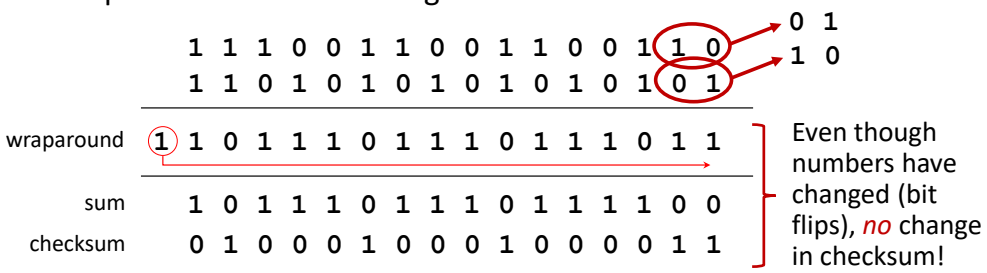
* Check out the online interactive exercises for more examples: http://gaia.cs.umass.edu/kurose_ross/interactive/

Slide 33 - Internet checksum: an example
Book reference (Kurose & Ross, Chapter 3, p. 198): short quote: "UDP checksum provides for error detection".
In the checksum slides, the goal is for students to understand bit-level error detection. The sender treats the UDP segment as a sequence of 16-bit words, computes the one's-complement sum, and stores the checksum. The receiver repeats the logic to check the segment.
Point out that the Internet checksum is weak protection. Some error patterns can leave the sum unchanged, so this is not cryptographic integrity. It is still useful because it is simple and fast.
Visible slide keywords: Internet checksum: an example, example: add two 16-bit integers, sum, checksum, Note:, when adding numbers, a carryout from the most significant bit needs to be added to the result, * Check out the online interactive exercises for more examples: h, ttp://, gaia.cs.umass.edu, /, kurose_ross, /interactive/, 1 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0, 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1, 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1, wraparound, 1 0 1 1 1 0 1 1 1 0 1 1 1 1 0 0, 0 1 0 0 0 1 0 0 0 1 0 0 0 1 1, Transport Layer: 3-, 33.
Mini exercise: if two bits flip, will the checksum always detect the problem? No; slide 34 illustrates this weakness.

Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

Internet checksum: weak protection!

example: add two 16-bit integers



Slide 34 - Internet checksum: weak protection!
Book reference (Kurose & Ross, Chapter 3, p. 198): short quote: "UDP checksum provides for error detection".
In the checksum slides, the goal is for students to understand bit-level error detection. The sender treats the UDP segment as a sequence of 16-bit words, computes the one's-complement sum, and stores the checksum. The receiver repeats the logic to check the segment.
Point out that the Internet checksum is weak protection. Some error patterns can leave the sum unchanged, so this is not cryptographic integrity. It is still useful because it is simple and fast.
Visible slide keywords: Internet checksum: weak protection!, example: add two 16-bit integers, sum, checksum, 1 1 1 0 0 1 1 0 0 1 1 0 0 1 1 0, 1 1 0 1 0 1 0 1 0 1 0 1 0 1 0 1, 1 1 0 1 1 1 0 1 1 1 0 1 1 1 0 1 1, wraparound, 1 0 1 1 1 0 1 1 1 0 1 1 1 1 0 0, 0 1 0 0 0 1 0 0 0 1 0 0 0 0 1 1, 0 1, 1 0, Even though numbers have changed (bit flips),, no, change in checksum!, Transport Layer: 3-, 34.
Mini exercise: if two bits flip, will the checksum always detect the problem? No; slide 34 illustrates this weakness.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a

more organized and manageable way.

Summary: UDP

- "no frills" protocol:
  - segments may be lost, delivered out of order
  - best effort service: "send and hope for the best"
- UDP has its plusses:
  - no setup/handshaking needed (no RTT incurred)
  - can function when network service is compromised
  - helps with reliability (checksum)
- build additional functionality on top of UDP in application layer (e.g., HTTP/3)

Slide 35 - Summary
Book reference (Kurose & Ross, Chapter 3, p. 200): short quote: "UDP provides error checking, it does not do anything to recover".
Present UDP as intentionally simple, not as a useless protocol. It has no connection setup delay, a small header, no built-in congestion or flow control, and it gives applications speed and control.
Connect each use case to the type of problem: DNS often uses short query-response exchanges; streaming multimedia can tolerate some loss but is sensitive to delay; SNMP benefits from simplicity; HTTP/3 uses QUIC to build richer transport behavior above UDP.
Visible slide keywords: Summary, : UDP, "no frills" protocol:, segments may be lost, delivered out of order, best effort service: "send and hope for the best", UDP has its plusses:, no setup/handshaking needed (no RTT incurred), can function when network service is compromised, helps with reliability (checksum), build additional functionality on top of UDP in application layer (e.g., HTTP/3).
Key point: UDP itself does not provide reliability. If reliability is needed, the application or a protocol above UDP must add it.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a

more organized and manageable way.

Chapter 3: roadmap

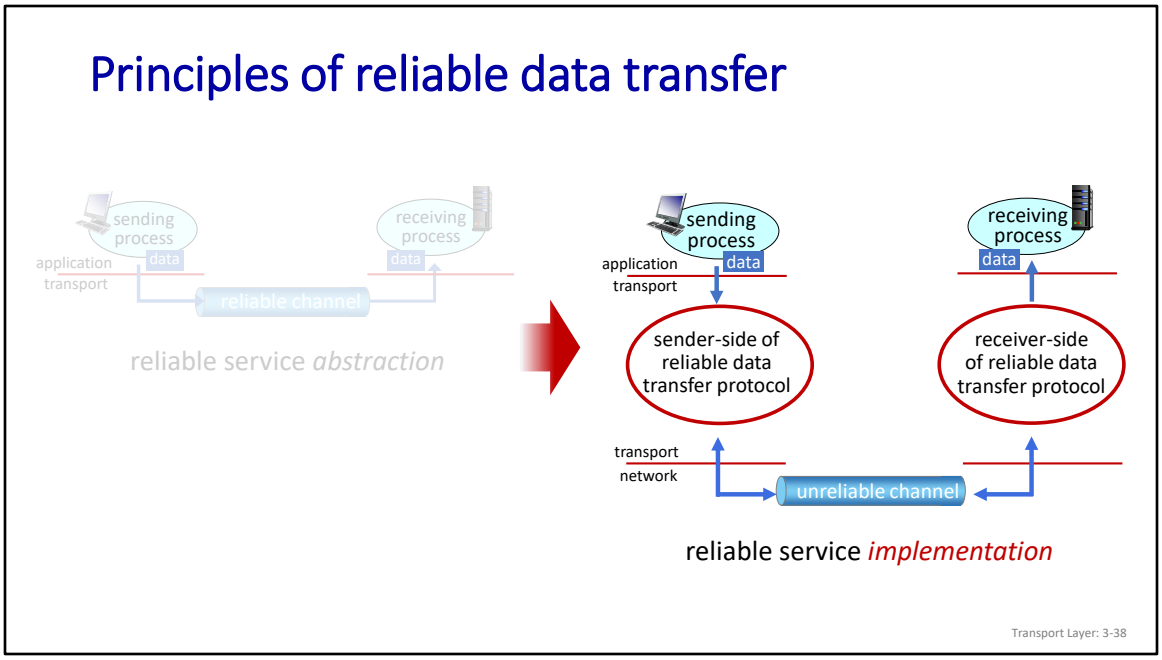
- Transport-layer services
- Multiplexing and demultiplexing
- Connectionless transport: UDP
- Principles of reliable data transfer
- Connection-oriented transport: TCP
- Principles of congestion control
- TCP congestion control
- Evolution of transport-layer functionality

Slide 36 - Principles of reliable data transfer
Book reference (Kurose & Ross, Chapter 3, p. 200): short quote: "no transferred data bits are corrupted or lost".
Introduce reliable data transfer as preparation for TCP. The goal is to give the upper layer the abstraction of a reliable channel even when the lower channel can corrupt, lose, or delay packets.
Make the distinction between reliable service and unreliable channel clear. The application wants reliable delivery, but the real network may provide only best effort. The RDT protocol fills that gap with error detection, acknowledgments, sequence numbers, timers, and retransmissions.
Visible slide keywords: Chapter 3: roadmap, Transport-layer services, Multiplexing and demultiplexing, Connectionless transport: UDP, Principles of reliable data transfer, Connection-oriented transport: TCP, Principles of congestion control, TCP congestion control, Evolution of transport-layer functionality, Transport Layer: 3-, 36.
Quick question: if the underlying channel can lose packets, what extra mechanism is needed? A timer and retransmission.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

Principles of reliable data transfer

reliable service *abstraction*

Slide 37 - Principles of reliable
Book reference (Kurose & Ross, Chapter 3, p. 200): short quote: "no transferred data bits are corrupted or lost".
Introduce reliable data transfer as preparation for TCP. The goal is to give the upper layer the abstraction of a reliable channel even when the lower channel can corrupt, lose, or delay packets.
Make the distinction between reliable service and unreliable channel clear. The application wants reliable delivery, but the real network may provide only best effort. The RDT protocol fills that gap with error detection, acknowledgments, sequence numbers, timers, and retransmissions.
Visible slide keywords: Principles of reliable, d, ata, t, ransfer, sending process, data, receiving process, data, reliable channel, application, transport, reliable service, abstraction, Transport Layer: 3-, 37.
Quick question: if the underlying channel can lose packets, what extra mechanism is needed? A timer and retransmission.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

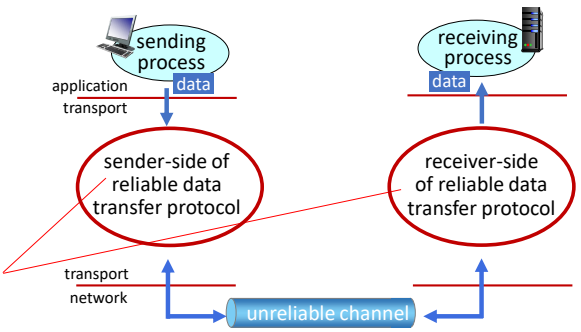


Slide 38 - Reliable data transfer protocol
Book reference (Kurose & Ross, Chapter 3, p. 200): short quote: "no transferred data bits are corrupted or lost".
Introduce reliable data transfer as preparation for TCP. The goal is to give the upper layer the abstraction of a reliable channel even when the lower channel can corrupt, lose, or delay packets.
Make the distinction between reliable service and unreliable channel clear. The application wants reliable delivery, but the real network may provide only best effort. The RDT protocol fills that gap with error detection, acknowledgments, sequence numbers, timers, and retransmissions.
Visible slide keywords: Principles of reliable, d, ata, t, ransfer, sending process, data, receiving process, data, application, transport, reliable service, implementation, unreliable channel, network, transport, sender-side of, reliable data transfer protocol, receiver-side, of reliable data transfer protocol, sending process, data, receiving process, data, reliable channel, application, transport, reliable service, abstraction, Transport Layer: 3-, 38.
Quick question: if the underlying channel can lose packets, what extra mechanism is needed? A timer and retransmission.
Close the slide by connecting it to the next one: this mechanism is one part of the

control information the transport layer adds so applications can communicate in a more organized and manageable way.

Principles of reliable data transfer

Complexity of reliable data transfer protocol will depend (strongly) on characteristics of unreliable channel (lose, corrupt, reorder data?)



reliable service *implementation*

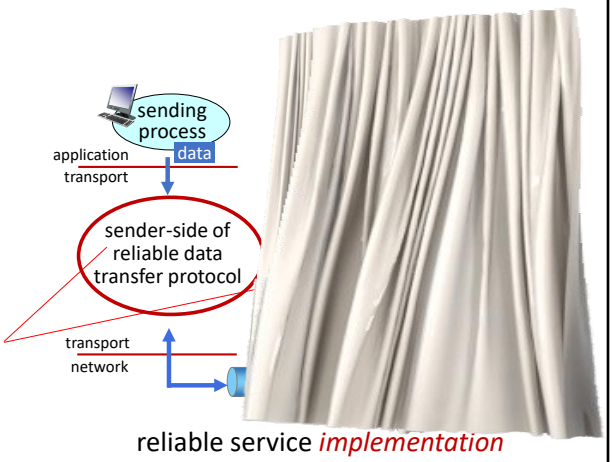
Slide 39 - Reliable data transfer protocol
Book reference (Kurose & Ross, Chapter 3, p. 200): short quote: "no transferred data bits are corrupted or lost".
Introduce reliable data transfer as preparation for TCP. The goal is to give the upper layer the abstraction of a reliable channel even when the lower channel can corrupt, lose, or delay packets.
Make the distinction between reliable service and unreliable channel clear. The application wants reliable delivery, but the real network may provide only best effort. The RDT protocol fills that gap with error detection, acknowledgments, sequence numbers, timers, and retransmissions.
Visible slide keywords: Principles of reliable, d, ata, t, ransfer, sending process, data, receiving process, data, application, transport, reliable service, implementation, unreliable channel, network, transport, sender-side of, reliable data transfer protocol, receiver-side, of reliable data transfer protocol, Complexity of reliable data transfer protocol will depend (strongly) on characteristics of unreliable channel (lose, corrupt, reorder data?), Transport Layer: 3-, 39.
Quick question: if the underlying channel can lose packets, what extra mechanism is needed? A timer and retransmission.
Close the slide by connecting it to the next one: this mechanism is one part of the

control information the transport layer adds so applications can communicate in a more organized and manageable way.

Principles of reliable data transfer

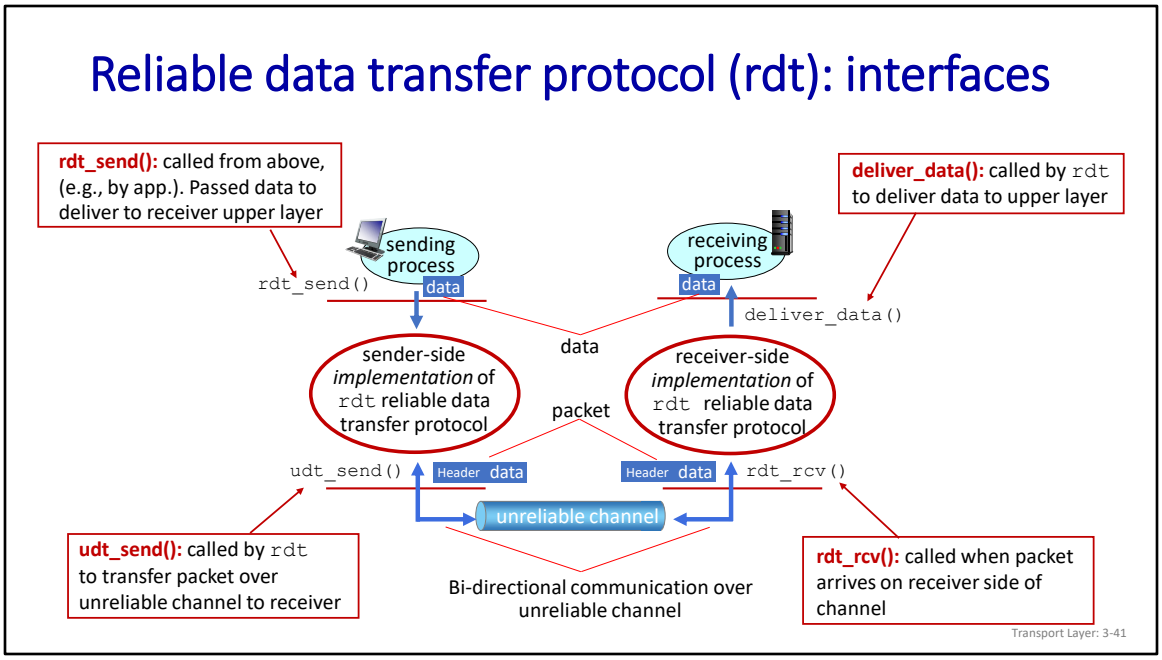
Sender, receiver do *not* know the "state" of each other, e.g., was a message received?

- unless communicated via a message



Slide 40 - Reliable data transfer protocol
Book reference (Kurose & Ross, Chapter 3, p. 200): short quote: "no transferred data bits are corrupted or lost".
Introduce reliable data transfer as preparation for TCP. The goal is to give the upper layer the abstraction of a reliable channel even when the lower channel can corrupt, lose, or delay packets.
Make the distinction between reliable service and unreliable channel clear. The application wants reliable delivery, but the real network may provide only best effort. The RDT protocol fills that gap with error detection, acknowledgments, sequence numbers, timers, and retransmissions.
Visible slide keywords: Principles of reliable, d, ata, t, ransfer, sending process, data, receiving process, data, application, transport, reliable service, implementation, unreliable channel, network, transport, sender-side of, reliable data transfer protocol, receiver-side, of reliable data transfer protocol, Sender, receiver do, not, know the "state" of each other, e.g., was a message received?, unless communicated via a message, Transport Layer: 3-, 40.
Quick question: if the underlying channel can lose packets, what extra mechanism is needed? A timer and retransmission.
Close the slide by connecting it to the next one: this mechanism is one part of the

control information the transport layer adds so applications can communicate in a more organized and manageable way.



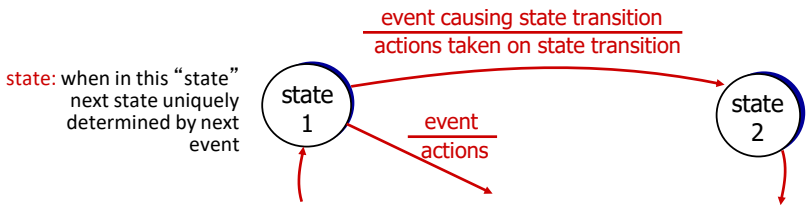
Slide 41 - Reliable data transfer protocol
Book reference (Kurose & Ross, Chapter 3, p. 201): short quote: "rdt_send()".
Introduce reliable data transfer as preparation for TCP. The goal is to give the upper layer the abstraction of a reliable channel even when the lower channel can corrupt, lose, or delay packets.
Make the distinction between reliable service and unreliable channel clear. The application wants reliable delivery, but the real network may provide only best effort. The RDT protocol fills that gap with error detection, acknowledgments, sequence numbers, timers, and retransmissions.
Visible slide keywords: Reliable data transfer protocol (, rdt,): interfaces, sending process, data, receiving process, data, unreliable channel, sender-side, implementation, of, rdt, reliable data transfer protocol, receiver-side, implementation, of, rdt, reliable data transfer protocol, rdt_send, (), udt_send, (), rdt_rcv, (), deliver_data, (), data, Header, data, Header, rdt_send, ():, called from above, (e.g., by app.). Passed data to deliver to receiver upper layer, udt_send, ():, called by, rdt, to transfer packet over, unreliable channel to receiver, rdt_rcv, ():, called when packet arrives on receiver side of channel, deliver_data, ():, called by, rdt, to deliver data to upper layer, Bi-directional communi.
Quick question: if the underlying channel can lose packets, what extra mechanism is

needed? A timer and retransmission.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

Reliable data transfer: getting started

We will:

- incrementally develop sender, receiver sides of reliable data transfer protocol (`rdt`)
- consider only unidirectional data transfer
  - but control info will flow in both directions!
- use finite state machines (FSM) to specify sender, receiver

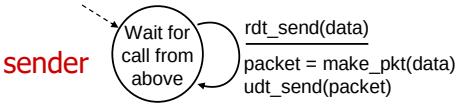


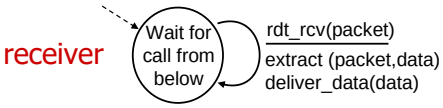
Slide 42 - Reliable data transfer: getting started
Book reference (Kurose & Ross, Chapter 3, p. 200): short quote: "no transferred data bits are corrupted or lost".
Introduce reliable data transfer as preparation for TCP. The goal is to give the upper layer the abstraction of a reliable channel even when the lower channel can corrupt, lose, or delay packets.
Make the distinction between reliable service and unreliable channel clear. The application wants reliable delivery, but the real network may provide only best effort. The RDT protocol fills that gap with error detection, acknowledgments, sequence numbers, timers, and retransmissions.
Visible slide keywords: Reliable data transfer: getting started, We will:, incrementally develop sender, receiver sides of, r, eliable, d, ata, t, ransfer protocol (, rdt,), consider only unidirectional data transfer, but control info will flow in both directions!, state, 1, state, 2, event causing state transition, actions taken on state transition, state:, when in this, “, state, ”, next state uniquely determined by next event, event, actions, use finite state machines (FSM) to specify sender, receiver, Transport Layer: 3-, 42.
Quick question: if the underlying channel can lose packets, what extra mechanism is needed? A timer and retransmission.
Close the slide by connecting it to the next one: this mechanism is one part of the

control information the transport layer adds so applications can communicate in a more organized and manageable way.

rdt1.0: reliable transfer over a reliable channel

- underlying channel perfectly reliable
  - no bit errors
  - no loss of packets
- *separate* FSMs for sender, receiver:
  - sender sends data into underlying channel
  - receiver reads data from underlying channel





Slide 43 - rdt1.0
Book reference (Kurose & Ross, Chapter 3, p. 203): short quote: "with a perfectly reliable channel there is no need".
rdt1.0 is the simplest starting point: if the channel is perfect, the sender only makes a packet and sends it, and the receiver extracts the packet and delivers the data upward. ACKs, NAKs, checksums, and sequence numbers are unnecessary under this assumption.
Keep this slide short but explain its importance: protocol design begins with assumptions. As the assumptions become more realistic, the protocol needs more states and more header information.
Visible slide keywords: rdt1.0:, reliable transfer over a reliable channel, underlying channel perfectly reliable, no bit errors, no loss of packets, packet =, make_pkt, (data), udt_send, (packet), rdt_send, (data), extract (, packet,data,), deliver_data, (data), rdt_rcv, (packet), Wait for call from below, receiver, separate, FSMs for sender, receiver:, sender sends data into underlying channel, receiver reads data from underlying channel, sender, Wait for call from above, Transport Layer: 3-, 43.
Key point: rdt1.0 is not the real Internet. Later slides add reality step by step.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a

more organized and manageable way.

rdt2.0: channel with bit errors

- underlying channel may flip bits in packet
  - checksum (e.g., Internet checksum) to detect bit errors
- *the* question: how to recover from errors?

How do humans recover from "errors" during conversation?

Slide 44 - rdt2.0
Book reference (Kurose & Ross, Chapter 3, p. 204): short quote: "positive acknowledgments and negative acknowledgments".
rdt2.0 assumes a channel that can corrupt bits but does not lose packets. Three ideas appear here: checksum for error detection, ACK/NAK for receiver feedback, and retransmission for damaged packets.
When reading the FSM, separate events from actions. If data arrives from above, the sender makes and sends a packet. If a corrupted packet arrives from below, the receiver sends NAK; if a correct packet arrives, it sends ACK. The sender waits for ACK before sending new data, so this is stop-and-wait behavior.
Visible slide keywords: rdt2.0: channel with bit errors, underlying channel may flip bits in packet, checksum (e.g., Internet checksum) to detect bit errors, the, question: how to recover from errors?, How do humans recover from ", errors", during conversation?, Transport Layer: 3-, 44.
Ask the class: if the ACK or NAK itself is corrupted, what can the sender trust? This leads directly to the fatal flaw of rdt2.0.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

rdt2.0: channel with bit errors

- underlying channel may flip bits in packet
  - checksum to detect bit errors
- *the* question: how to recover from errors?
  - *acknowledgements (ACKs):* receiver explicitly tells sender that pkt received OK
  - *negative acknowledgements (NAKs):* receiver explicitly tells sender that pkt had errors
  - sender *retransmits* pkt on receipt of NAK

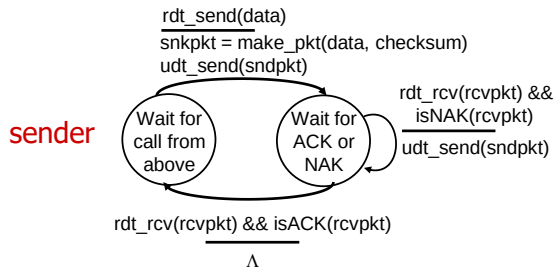
stop and wait

sender sends one packet, then waits for receiver response

Slide 45 - rdt2.0
Book reference (Kurose & Ross, Chapter 3, p. 204): short quote: "positive acknowledgments and negative acknowledgments".
rdt2.0 assumes a channel that can corrupt bits but does not lose packets. Three ideas appear here: checksum for error detection, ACK/NAK for receiver feedback, and retransmission for damaged packets.
When reading the FSM, separate events from actions. If data arrives from above, the sender makes and sends a packet. If a corrupted packet arrives from below, the receiver sends NAK; if a correct packet arrives, it sends ACK. The sender waits for ACK before sending new data, so this is stop-and-wait behavior.
Visible slide keywords: rdt2.0: channel with bit errors, underlying channel may flip bits in packet, checksum to detect bit errors, the, question: how to recover from errors?, acknowledgements (ACKs):, receiver explicitly tells sender that pkt received OK, negative acknowledgements (NAKs):, receiver explicitly tells sender that pkt had errors, sender, retransmits, pkt on receipt of NAK, stop and wait, sender sends one packet, then waits for receiver response, Transport Layer: 3-, 45.
Ask the class: if the ACK or NAK itself is corrupted, what can the sender trust? This leads directly to the fatal flaw of rdt2.0.
Close the slide by connecting it to the next one: this mechanism is one part of the

control information the transport layer adds so applications can communicate in a more organized and manageable way.

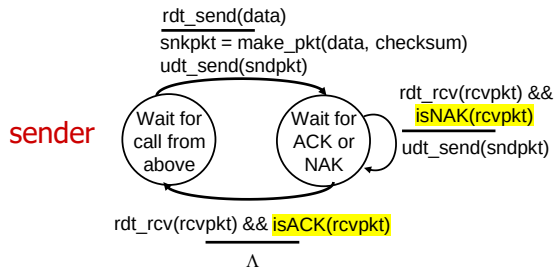
rdt2.0: FSM specifications



Slide 46 - rdt2.0
Book reference (Kurose & Ross, Chapter 3, p. 204): short quote: "positive acknowledgments and negative acknowledgments".
rdt2.0 assumes a channel that can corrupt bits but does not lose packets. Three ideas appear here: checksum for error detection, ACK/NAK for receiver feedback, and retransmission for damaged packets.
When reading the FSM, separate events from actions. If data arrives from above, the sender makes and sends a packet. If a corrupted packet arrives from below, the receiver sends NAK; if a correct packet arrives, it sends ACK. The sender waits for ACK before sending new data, so this is stop-and-wait behavior.
Visible slide keywords: rdt2.0: FSM specifications, Wait for call from above, udt_send(sndpkt), Wait for ACK or NAK, udt_send(NAK), rdt_rcv, (, rcvpkt,) && corrupt(, rcvpkt,), Wait for call from below, extract(, rcvpkt,data,), deliver_data, (data), udt_send, (ACK), rdt_rcv, (, rcvpkt,) &&, notcorrupt, (, rcvpkt,), snkpkt = make_pkt(data, checksum), udt_send(sndpkt), rdt_send(data), rdt_rcv, (, rcvpkt,) &&, isACK, (, rcvpkt,), L, sender, receiver, rdt_rcv, (, rcvpkt,) &&, isNAK, (, rcvpkt,), Transport Layer: 3-, 46.
Ask the class: if the ACK or NAK itself is corrupted, what can the sender trust? This leads directly to the fatal flaw of rdt2.0.

Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

rdt2.0: FSM specification



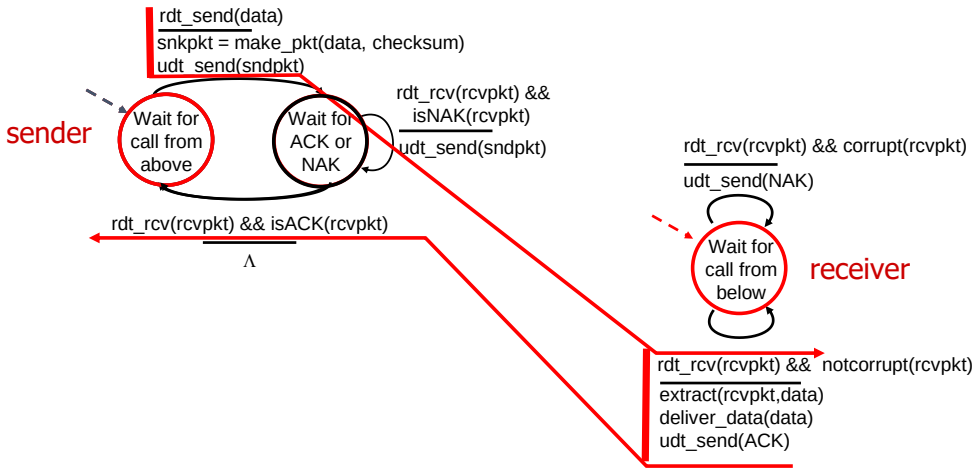
Note: "state" of receiver (did the receiver get my message correctly?) isn't known to sender unless somehow communicated from receiver to sender

- that's why we need a protocol!

Slide 47 - rdt2.0
Book reference (Kurose & Ross, Chapter 3, p. 204): short quote: "positive acknowledgments and negative acknowledgments".
rdt2.0 assumes a channel that can corrupt bits but does not lose packets. Three ideas appear here: checksum for error detection, ACK/NAK for receiver feedback, and retransmission for damaged packets.
When reading the FSM, separate events from actions. If data arrives from above, the sender makes and sends a packet. If a corrupted packet arrives from below, the receiver sends NAK; if a correct packet arrives, it sends ACK. The sender waits for ACK before sending new data, so this is stop-and-wait behavior.
Visible slide keywords: rdt2.0: FSM specification, Wait for call from above, udt_send(sndpkt), Wait for ACK or NAK, udt_send(NAK), rdt_rcv, (, rcvpkt,) && corrupt(, rcvpkt,), Wait for call from below, extract(, rcvpkt,data,), deliver_data, (data), udt_send, (ACK), rdt_rcv, (, rcvpkt,) &&, notcorrupt, (, rcvpkt,), snkpkt = make_pkt(data, checksum), udt_send(sndpkt), rdt_send(data), rdt_rcv, (, rcvpkt,) &&, isACK, (, rcvpkt,), L, sender, receiver, Note:, "state" of receiver (did the receiver get my message correctly?) isn't known to sender unless somehow communicated from receiver to sender, that's why we need a protocol!, rdt_rcv, (, rcvpkt,) &&, isNAK, (, rcvpkt,), isNAK, (, rcvpkt,), isACK, (, rcvpkt,),.

Ask the class: if the ACK or NAK itself is corrupted, what can the sender trust? This leads directly to the fatal flaw of rdt2.0.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

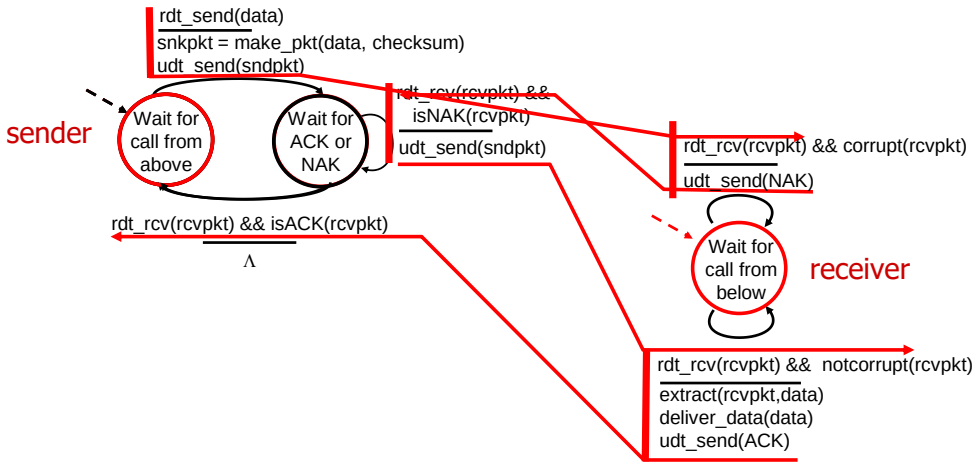
rdt2.0: operation with no errors



Slide 48 - rdt2.0
Book reference (Kurose & Ross, Chapter 3, p. 204): short quote: "positive acknowledgments and negative acknowledgments".
rdt2.0 assumes a channel that can corrupt bits but does not lose packets. Three ideas appear here: checksum for error detection, ACK/NAK for receiver feedback, and retransmission for damaged packets.
When reading the FSM, separate events from actions. If data arrives from above, the sender makes and sends a packet. If a corrupted packet arrives from below, the receiver sends NAK; if a correct packet arrives, it sends ACK. The sender waits for ACK before sending new data, so this is stop-and-wait behavior.
Visible slide keywords: rdt2.0: operation with no errors, Wait for call from above, snkpkt = make_pkt(data, checksum), udt_send(sndpkt), udt_send, (, sndpkt,), udt_send, (NAK), Wait for ACK or NAK, Wait for call from below, rdt_send(data), rdt_rcv, (, rcvpkt,) && corrupt(, rcvpkt,), rdt_rcv, (, rcvpkt,) &&, isACK, (, rcvpkt,), L, extract(, rcvpkt,data,), deliver_data, (data), udt_send, (ACK), rdt_rcv, (, rcvpkt,) &&, notcorrupt, (, rcvpkt,), sender, receiver, rdt_rcv, (, rcvpkt,) &&, isNAK, (, rcvpkt,), Transport Layer: 3-, 48.
Ask the class: if the ACK or NAK itself is corrupted, what can the sender trust? This leads directly to the fatal flaw of rdt2.0.

Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

rdt2.0: corrupted packet scenario



Slide 49 - rdt2.0
Book reference (Kurose & Ross, Chapter 3, p. 204): short quote: "positive acknowledgments and negative acknowledgments".
rdt2.0 assumes a channel that can corrupt bits but does not lose packets. Three ideas appear here: checksum for error detection, ACK/NAK for receiver feedback, and retransmission for damaged packets.
When reading the FSM, separate events from actions. If data arrives from above, the sender makes and sends a packet. If a corrupted packet arrives from below, the receiver sends NAK; if a correct packet arrives, it sends ACK. The sender waits for ACK before sending new data, so this is stop-and-wait behavior.
Visible slide keywords: rdt2.0: corrupted packet scenario, Wait for call from above, snkpkt = make_pkt(data, checksum), udt_send(sndpkt), udt_send(sndpkt), rdt_rcv, (, rcvpkt,) &&, isNAK, (, rcvpkt,), Wait for ACK or NAK, Wait for call from below, rdt_send(data), udt_send, (NAK), rdt_rcv, (, rcvpkt,) && corrupt(, rcvpkt,), extract(, rcvpkt,data,), deliver_data, (data), udt_send, (ACK), rdt_rcv, (, rcvpkt,) &&, notcorrupt, (, rcvpkt,), rdt_rcv, (, rcvpkt,) &&, isACK, (, rcvpkt,), L, sender, receiver, Transport Layer: 3-, 49.
Ask the class: if the ACK or NAK itself is corrupted, what can the sender trust? This leads directly to the fatal flaw of rdt2.0.

Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

rdt2.0 has a fatal flaw!

what happens if ACK/NAK corrupted?

- sender doesn't know what happened at receiver!
- can't just retransmit: possible duplicate

handling duplicates:

- sender retransmits current pkt if ACK/NAK corrupted
- sender adds *sequence number* to each pkt
- receiver discards (doesn't deliver up) duplicate pkt

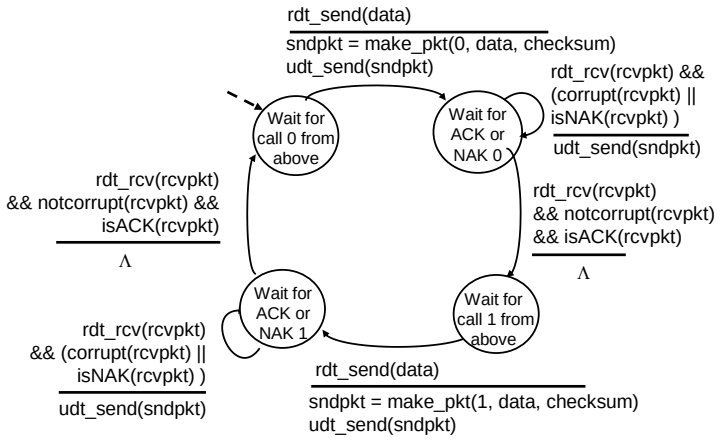
stop and wait

sender sends one packet, then waits for receiver response

Slide 50 - rdt2.0
Book reference (Kurose & Ross, Chapter 3, p. 204): short quote: "the receiver to provide explicit feedback".
The fatal flaw in rdt2.0 is corrupted ACK/NAK feedback. The sender does not know what happened at the receiver. If it blindly retransmits, the receiver may deliver the same data twice. Therefore, duplicate detection is needed.
rdt2.1 solves this by adding sequence numbers. In stop-and-wait, two sequence numbers, 0 and 1, are enough because only one packet is outstanding at a time. The receiver can tell whether an arriving packet is new or a duplicate.
Visible slide keywords: rdt2.0 has a fatal flaw!, what happens if ACK/NAK corrupted, ?, sender doesn', t know what happened at receiver!, can', t just retransmit: possible duplicate, handling duplicates, :, sender retransmits current pkt if ACK/NAK corrupted, sender adds, sequence number, to each pkt, receiver discards (doesn', t deliver up) duplicate pkt, stop and wait, sender sends one packet, then waits for receiver response, Transport Layer: 3-, 50.
Key point: sequence numbers are used not only for ordering but also for duplicate detection.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a

more organized and manageable way.

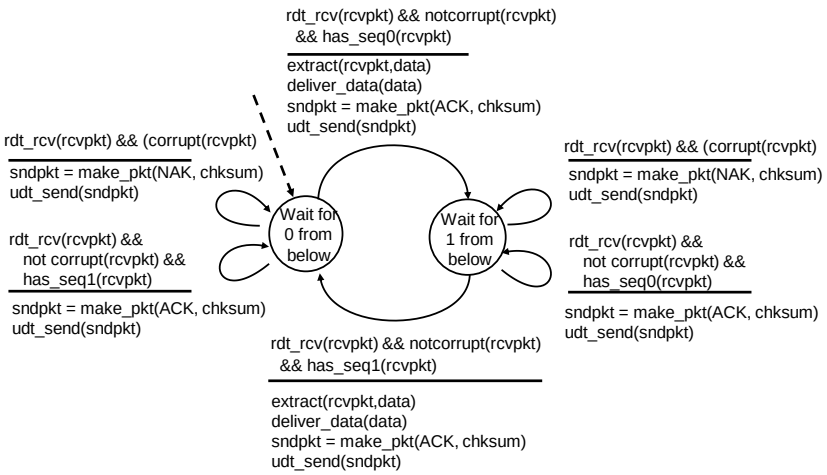
rdt2.1: sender, handling garbled ACK/NAKs



Slide 51 - rdt2.1
Book reference (Kurose & Ross, Chapter 3, p. 204): short quote: "the receiver to provide explicit feedback".
The fatal flaw in rdt2.0 is corrupted ACK/NAK feedback. The sender does not know what happened at the receiver. If it blindly retransmits, the receiver may deliver the same data twice. Therefore, duplicate detection is needed.
rdt2.1 solves this by adding sequence numbers. In stop-and-wait, two sequence numbers, 0 and 1, are enough because only one packet is outstanding at a time. The receiver can tell whether an arriving packet is new or a duplicate.
Visible slide keywords: rdt2.1: sender, handling garbled ACK/NAKs, Wait for call 0 from above, Wait for ACK or NAK 0, sndpkt, =, make_pkt, (0, data, checksum), udt_send, (, sndpkt,), rdt_send(data), udt_send, (, sndpkt,), rdt_rcv, (, rcvpkt,) && (corrupt(, rcvpkt,) ||, isNAK, (, rcvpkt,)), sndpkt = make_pkt(1, data, checksum), udt_send(sndpkt), rdt_send(data), udt_send(sndpkt), rdt_rcv, (, rcvpkt,), && (corrupt(, rcvpkt,) ||, isNAK, (, rcvpkt,)), Wait for, call 1 from above, Wait for ACK or NAK 1, rdt_rcv, (, rcvpkt,), &&, notcorrupt, (, rcvpkt,), &&, isACK, (, rcvpkt,), L, rdt_rcv, (, rcvpkt,), &&, notcorrupt, (, rcvpkt,) &&, isACK, (, rcvpkt,), L, Transport Layer: 3-, 51.
Key point: sequence numbers are used not only for ordering but also for duplicate

detection.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

rdt2.1: receiver, handling garbled ACK/NAKs



Slide 52 - rdt2.1
Book reference (Kurose & Ross, Chapter 3, p. 204): short quote: "the receiver to provide explicit feedback".
The fatal flaw in rdt2.0 is corrupted ACK/NAK feedback. The sender does not know what happened at the receiver. If it blindly retransmits, the receiver may deliver the same data twice. Therefore, duplicate detection is needed.
rdt2.1 solves this by adding sequence numbers. In stop-and-wait, two sequence numbers, 0 and 1, are enough because only one packet is outstanding at a time. The receiver can tell whether an arriving packet is new or a duplicate.
Visible slide keywords: rdt2.1: receiver, handling garbled ACK/NAKs, Wait for, 0 from below, rdt_rcv(rcvpkt) && notcorrupt(rcvpkt), && has_seq1(rcvpkt), extract(rcvpkt,data), deliver_data(data), sndpkt = make_pkt(ACK, chksum), udt_send(sndpkt), Wait for, 1 from below, rdt_rcv(rcvpkt) && notcorrupt(rcvpkt), && has_seq0(rcvpkt), extract(, rcvpkt,data,), deliver_data, (data), sndpkt, =, make_pkt, (ACK,, chksum,), udt_send, (, sndpkt,), sndpkt = make_pkt(NAK, chksum), udt_send(sndpkt), rdt_rcv(rcvpkt) && (corrupt(rcvpkt), rdt_rcv(rcvpkt) &&, not corrupt(rcvpkt) &&, has_seq0(rcvpkt), sndpkt = make_pkt(ACK, chksum), udt_send(sndpkt), rdt_rcv(rcvpkt) &&, not corrupt(rcvpkt) &&, has_seq1(rcvpkt), sndpkt = make_pkt(ACK, .

Key point: sequence numbers are used not only for ordering but also for duplicate detection.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

rdt2.1: discussion

sender:

- seq # added to pkt
- two seq. #s (0,1) will suffice. Why?
- must check if received ACK/NAK corrupted
- twice as many states
  - state must “remember” whether “expected” pkt should have seq # of 0 or 1

receiver:

- must check if received packet is duplicate
  - state indicates whether 0 or 1 is expected pkt seq #
- note: receiver can *not* know if its last ACK/NAK received OK at sender

Slide 53 - rdt2.1
Book reference (Kurose & Ross, Chapter 3, p. 204): short quote: "the receiver to provide explicit feedback".
The fatal flaw in rdt2.0 is corrupted ACK/NAK feedback. The sender does not know what happened at the receiver. If it blindly retransmits, the receiver may deliver the same data twice. Therefore, duplicate detection is needed.
rdt2.1 solves this by adding sequence numbers. In stop-and-wait, two sequence numbers, 0 and 1, are enough because only one packet is outstanding at a time. The receiver can tell whether an arriving packet is new or a duplicate.
Visible slide keywords: rdt2.1: discussion, sender:, seq # added to pkt, two seq. #, s (0,1) will suffice. Why?, must check if received ACK/NAK corrupted, twice as many states, state must “, remember” whether “expected” pkt should have seq # of 0 or 1, receiver:, must check if received packet is duplicate, state indicates whether 0 or 1 is expected pkt seq #, note: receiver can, not, know if its last ACK/NAK received OK at sender, Transport Layer: 3-, 53.
Key point: sequence numbers are used not only for ordering but also for duplicate detection.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a

more organized and manageable way.

rdt2.2: a NAK-free protocol

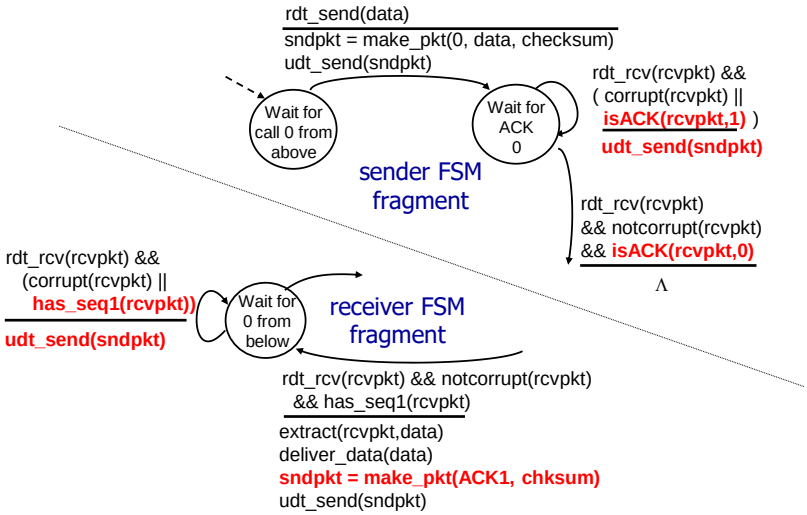
- same functionality as rdt2.1, using ACKs only
- instead of NAK, receiver sends ACK for last pkt received OK
  - receiver must *explicitly* include seq # of pkt being ACKed
- duplicate ACK at sender results in same action as NAK: *retransmit current pkt*

As we will see, TCP uses this approach to be NAK-free

Slide 54 - rdt2.1
Book reference (Kurose & Ross, Chapter 3, p. 204): short quote: "the receiver to provide explicit feedback".
rdt2.2 provides the same functionality without NAKs. When the receiver sees a corrupted or unexpected packet, it sends an ACK for the last correctly received packet. The sender interprets that duplicate ACK as a signal about what still needs attention.
This idea is close to TCP: TCP is ACK-based rather than NAK-based. Emphasize the logic of repeatedly reporting the last correct point instead of explicitly sending a negative response.
Visible slide keywords: rdt2.2: a NAK-free protocol, same functionality as rdt2.1, using ACKs only, instead of NAK, receiver sends ACK for last pkt received OK, receiver must, explicitly, include seq # of pkt being, ACKed, duplicate ACK at sender results in same action as NAK:, retransmit current pkt, As we will see, TCP uses this approach to be NAK-free, Transport Layer: 3-, 54.
Quick question: why must the receiver include a sequence number in the ACK? So the sender knows which packet has been acknowledged.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a

more organized and manageable way.

rdt2.2: sender, receiver fragments



Slide 55 - rdt2.2
Book reference (Kurose & Ross, Chapter 3, p. 204): short quote: "the receiver to provide explicit feedback".
rdt2.2 provides the same functionality without NAKs. When the receiver sees a corrupted or unexpected packet, it sends an ACK for the last correctly received packet. The sender interprets that duplicate ACK as a signal about what still needs attention.
This idea is close to TCP: TCP is ACK-based rather than NAK-based. Emphasize the logic of repeatedly reporting the last correct point instead of explicitly sending a negative response.
Visible slide keywords: rdt2.2: sender, receiver fragments, Wait for call 0 from above, sndpkt = make_pkt(0, data, checksum), udt_send(sndpkt), rdt_send(data), udt_send(sndpkt), rdt_rcv(rcvpkt) &&, (corrupt(rcvpkt) ||, isACK(rcvpkt,1),), rdt_rcv(rcvpkt), && notcorrupt(rcvpkt), &&, isACK(rcvpkt,0), Wait for ACK, 0, sender FSM, fragment, rdt_rcv(rcvpkt) && notcorrupt(rcvpkt), && has_seq1(rcvpkt), extract(rcvpkt,data), deliver_data(data), sndpkt = make_pkt(ACK1, chksum), udt_send(sndpkt), Wait for, 0 from below, rdt_rcv(rcvpkt) &&, (corrupt(rcvpkt) ||, has_seq1(rcvpkt)), udt_send(sndpkt), receiver FSM, fragment, L, Transport Layer: 3-, 55.

Quick question: why must the receiver include a sequence number in the ACK? So the sender knows which packet has been acknowledged.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

rdt3.0: channels with errors *and* loss

New channel assumption: underlying channel can also *lose* packets (data, ACKs)

- checksum, sequence #s, ACKs, retransmissions will be of help … but not quite enough

Q: How do *humans* handle lost sender-to-receiver words in conversation?

Slide 56 - rdt3.0
Book reference (Kurose & Ross, Chapter 3, p. 211): short quote: "checksums, sequence numbers, timers".
rdt3.0 adds the possibility of loss. Now not only can bits be corrupted, but a data packet or ACK can disappear entirely. The sender waits a reasonable time for an ACK; if the time expires, it retransmits the same packet.
Explain the side effect of timers: the packet may not be lost; it may only be delayed. That can create duplicates. The receiver uses sequence numbers to detect duplicates and avoid delivering the same data to the application twice.
Visible slide keywords: rdt3.0: channels with errors, and, loss, New channel assumption:, underlying channel can also, lose, packets (data, ACKs), checksum, sequence #s, ACKs, retransmissions will be of help … but not quite enough, Q:, How do, humans, handle lost sender-to-receiver words in conversation?, Transport Layer: 3-, 56.
Key point: timeout is not proof of loss; it is the sender's way of managing uncertainty.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

rdt3.0: channels with errors *and* loss

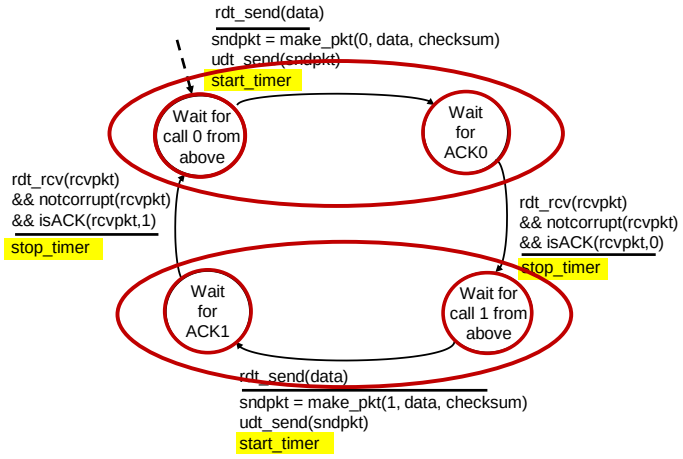
Approach: sender waits "reasonable" amount of time for ACK

- retransmits if no ACK received in this time
- if pkt (or ACK) just delayed (not lost):
  - retransmission will be duplicate, but seq #s already handles this!
  - receiver must specify seq # of packet being ACKed
- use countdown timer to interrupt after "reasonable" amount of time

timeout

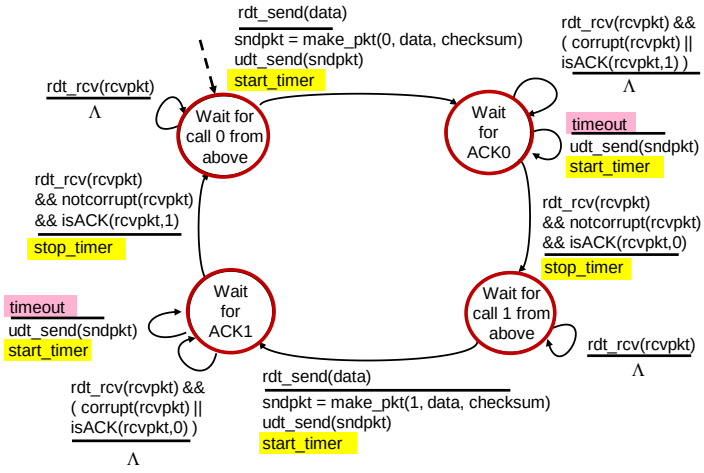
Slide 57 - rdt3.0
Book reference (Kurose & Ross, Chapter 3, p. 211): short quote: "checksums, sequence numbers, timers".
rdt3.0 adds the possibility of loss. Now not only can bits be corrupted, but a data packet or ACK can disappear entirely. The sender waits a reasonable time for an ACK; if the time expires, it retransmits the same packet.
Explain the side effect of timers: the packet may not be lost; it may only be delayed. That can create duplicates. The receiver uses sequence numbers to detect duplicates and avoid delivering the same data to the application twice.
Visible slide keywords: rdt3.0: channels with errors, and, loss, Approach:, sender waits, "reasonable" amount of time for ACK, retransmits if no ACK received in this time, if pkt (or ACK) just delayed (not lost):, retransmission will be duplicate, but seq #, s already handles this!, receiver must specify seq # of packet being, ACKed, timeout, use countdown timer to interrupt after "reasonable" amount of time, Transport Layer: 3-, 57.
Key point: timeout is not proof of loss; it is the sender's way of managing uncertainty.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

rdt3.0 sender



Slide 58 - rdt3.0
Book reference (Kurose & Ross, Chapter 3, p. 211): short quote: "checksums, sequence numbers, timers".
rdt3.0 adds the possibility of loss. Now not only can bits be corrupted, but a data packet or ACK can disappear entirely. The sender waits a reasonable time for an ACK; if the time expires, it retransmits the same packet.
Explain the side effect of timers: the packet may not be lost; it may only be delayed. That can create duplicates. The receiver uses sequence numbers to detect duplicates and avoid delivering the same data to the application twice.
Visible slide keywords: rdt3.0 sender, Wait for ACK0, sndpkt, =, make_pkt, (0, data, checksum), udt_send, (, sndpkt,), start_timer, rdt_send(data), Wait for, call 1 from above, sndpkt = make_pkt(1, data, checksum), udt_send(sndpkt), start_timer, rdt_send(data), rdt_rcv(rcvpkt), && notcorrupt(rcvpkt), && isACK(rcvpkt,0), stop_timer, rdt_rcv, (, rcvpkt,), &&, notcorrupt, (, rcvpkt,), &&, isACK, (rcvpkt,1), stop_timer, Wait for, call 0 from above, Wait for ACK1, Transport Layer: 3-, 58.
Key point: timeout is not proof of loss; it is the sender's way of managing uncertainty.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

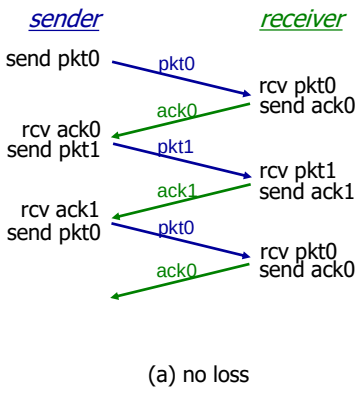
rdt3.0 sender



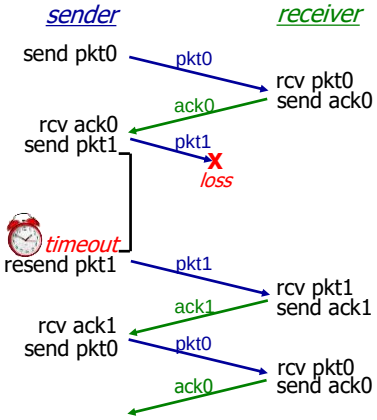
Slide 59 - rdt3.0
Book reference (Kurose & Ross, Chapter 3, p. 211): short quote: "checksums, sequence numbers, timers".
rdt3.0 adds the possibility of loss. Now not only can bits be corrupted, but a data packet or ACK can disappear entirely. The sender waits a reasonable time for an ACK; if the time expires, it retransmits the same packet.
Explain the side effect of timers: the packet may not be lost; it may only be delayed. That can create duplicates. The receiver uses sequence numbers to detect duplicates and avoid delivering the same data to the application twice.
Visible slide keywords: rdt3.0 sender, Wait for ACK0, sndpkt, =, make_pkt, (0, data, checksum), udt_send, (, sndpkt,), start_timer, rdt_send(data), Wait for, call 1 from above, sndpkt = make_pkt(1, data, checksum), udt_send(sndpkt), start_timer, rdt_send(data), rdt_rcv(rcvpkt), && notcorrupt(rcvpkt), && isACK(rcvpkt,0), stop_timer, rdt_rcv(rcvpkt), && notcorrupt(rcvpkt), && isACK(rcvpkt,1), stop_timer, udt_send, (, sndpkt,), start_timer, timeout, Wait for, call 0 from above, Wait for ACK1, L, rdt_rcv, (, rcvpkt,), rdt_rcv, (, rcvpkt,) &&, (corrupt(, rcvpkt,) ||, isACK, (rcvpkt,1)), L, rdt_rcv(rcvpkt), L, udt_send(sndpkt), start_timer, timeout, rdt_rcv, (, rcvpkt,) &&, (corrupt(, rcvpkt,) ||, isACK, (rcvpk.
Key point: timeout is not proof of loss; it is the sender's way of managing uncertainty.

Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

rdt3.0 in action

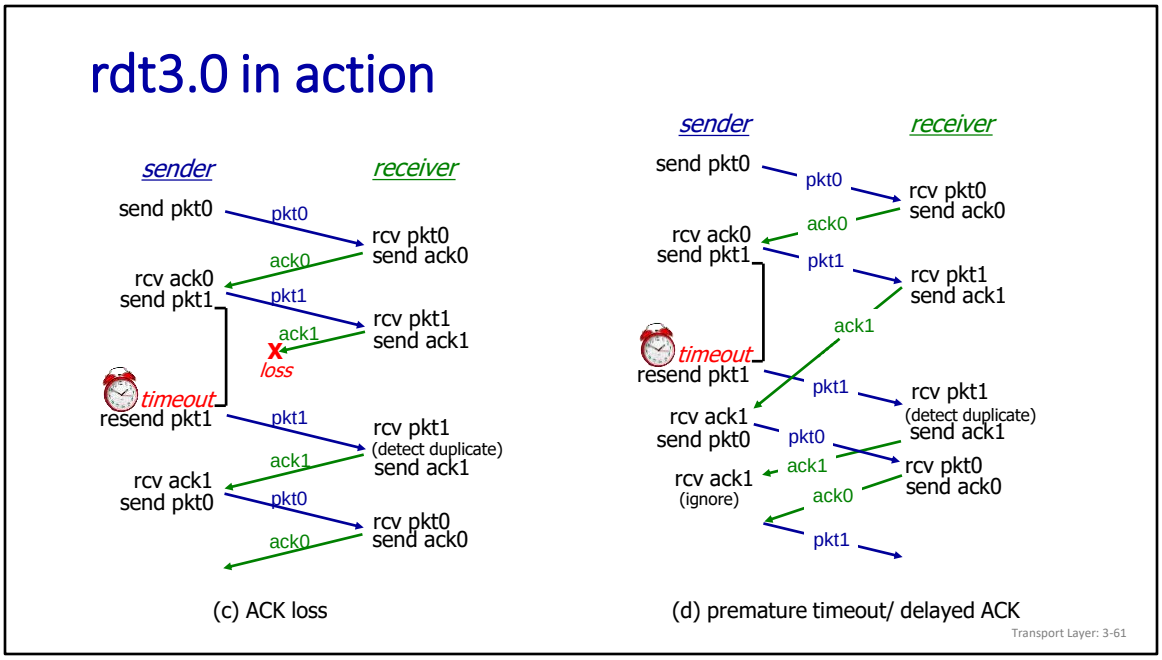


(a) no loss



(b) packet loss

Slide 60 - rdt3.0
Book reference (Kurose & Ross, Chapter 3, p. 211): short quote: "checksums, sequence numbers, timers".
rdt3.0 adds the possibility of loss. Now not only can bits be corrupted, but a data packet or ACK can disappear entirely. The sender waits a reasonable time for an ACK; if the time expires, it retransmits the same packet.
Explain the side effect of timers: the packet may not be lost; it may only be delayed. That can create duplicates. The receiver uses sequence numbers to detect duplicates and avoid delivering the same data to the application twice.
Visible slide keywords: rdt3.0 in action, sender, receiver, rcv pkt1, rcv pkt0, send ack0, send ack1, send ack0, rcv ack0, send pkt0, send pkt1, rcv ack1, send pkt0, rcv pkt0, pkt0, pkt0, pkt1, ack1, ack0, ack0, (a) no loss, sender, receiver, rcv pkt1, rcv pkt0, send ack0, send ack1, send ack0, rcv ack0, send pkt0, send pkt1, rcv ack1, send pkt0, rcv pkt0, pkt0, pkt0, ack1, ack0, ack0, (b) packet loss, pkt1, X, loss, pkt1, timeout, resend pkt1, Transport Layer: 3-, 60.
Key point: timeout is not proof of loss; it is the sender's way of managing uncertainty.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.



Slide 61 - rdt3.0
Book reference (Kurose & Ross, Chapter 3, p. 211): short quote: "checksums, sequence numbers, timers".
rdt3.0 adds the possibility of loss. Now not only can bits be corrupted, but a data packet or ACK can disappear entirely. The sender waits a reasonable time for an ACK; if the time expires, it retransmits the same packet.
Explain the side effect of timers: the packet may not be lost; it may only be delayed. That can create duplicates. The receiver uses sequence numbers to detect duplicates and avoid delivering the same data to the application twice.
Visible slide keywords: rdt3.0 in action, rcv pkt1, send ack1, (detect duplicate), pkt1, sender, receiver, rcv pkt1, rcv pkt0, send ack0, send ack1, send ack0, rcv ack0, send pkt0, send pkt1, rcv ack1, send pkt0, rcv pkt0, pkt0, pkt0, ack1, ack0, ack0, (c) ACK loss, ack1, X, loss, pkt1, timeout, resend pkt1, rcv pkt1, send ack1, (detect duplicate), pkt1, sender, receiver, rcv, pkt1, send ack0, rcv ack0, send pkt1, send pkt0, rcv pkt0, pkt0, ack0, (d) premature timeout/ delayed ACK, pkt1, timeout, resend pkt1, ack1, ack1, send ack1, send pkt0, rcv, ack1, pkt0, rcv, pkt0, send ack0, ack0, pkt1, (ignore), rcv, ack1, Transport Layer: 3-, 61.
Key point: timeout is not proof of loss; it is the sender's way of managing uncertainty. Close the slide by connecting it to the next one: this mechanism is one part of the

control information the transport layer adds so applications can communicate in a more organized and manageable way.

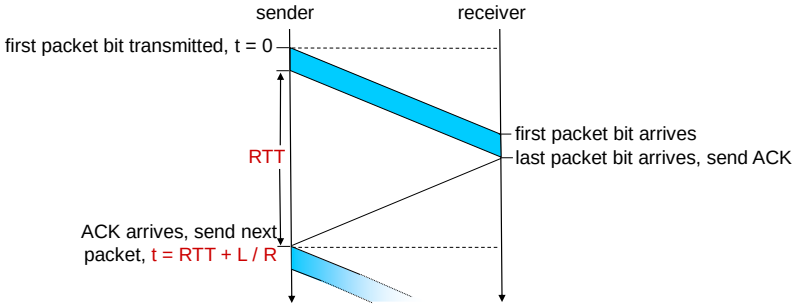
Performance of rdt3.0 (stop-and-wait)

- U_{sender}: *utilization* – fraction of time sender busy sending
- example: 1 Gbps link, 15 ms prop. delay, 8000 bit packet
  - time to transmit packet into channel:

$$D_{trans} = \frac{L}{R} = \frac{8000 \text{ bits}}{10^9 \text{ bits/sec}} = 8 \text{ microsecs}$$

Slide 62 - rdt3.0
Book reference (Kurose & Ross, Chapter 3, p. 205): short quote: "stop-and-wait protocols".
Use the performance slides to show that stop-and-wait is correct but inefficient. The sender transmits one packet and then waits through the RTT. Even with a very fast link, the sender spends most of the time idle.
In the formula, L/R is the time required to push the packet onto the link, and RTT is the round-trip waiting time. In the 1 Gbps and 15 ms propagation example, utilization is extremely low. This motivates pipelining.
Visible slide keywords: Performance of rdt3.0, (stop-and-wait), example: 1 Gbps link, 15, ms, prop. delay, 8000 bit packet, U, sender, :, utilization, – fraction of time sender busy sending, D, trans, =, L, R, 8000 bits, 10, 9, bits/sec, =, =, 8 microsecs, t, ime, to transmit packet into channel:, Transport Layer: 3-, 62.
Ask: does a fast link alone guarantee high utilization? No; protocol waiting behavior can limit performance.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

rdt3.0: stop-and-wait operation



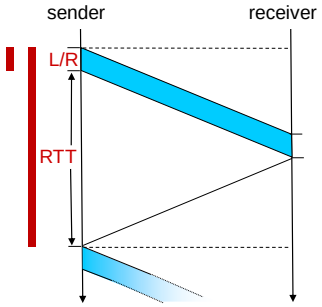
Slide 63 - rdt3.0
Book reference (Kurose & Ross, Chapter 3, p. 205): short quote: "stop-and-wait protocols".
Use the performance slides to show that stop-and-wait is correct but inefficient. The sender transmits one packet and then waits through the RTT. Even with a very fast link, the sender spends most of the time idle.
In the formula, L/R is the time required to push the packet onto the link, and RTT is the round-trip waiting time. In the 1 Gbps and 15 ms propagation example, utilization is extremely low. This motivates pipelining.
Visible slide keywords: rdt3.0: stop-and-wait operation, first packet bit transmitted, t = 0, sender, receiver, RTT, first packet bit arrives, last packet bit arrives, send ACK, ACK arrives, send next, packet,, t = RTT + L / R, Transport Layer: 3-, 63.
Ask: does a fast link alone guarantee high utilization? No; protocol waiting behavior can limit performance.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

rdt3.0: stop-and-wait operation

$$U_{sender} = \frac{L / R}{RTT + L / R}$$

$$= \frac{.008}{30.008}$$

$$= 0.00027$$



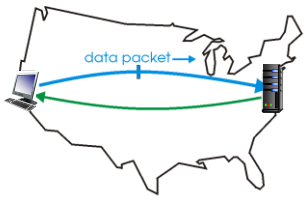
- rdt 3.0 protocol performance stinks!
- Protocol limits performance of underlying infrastructure (channel)

Slide 64 - rdt3.0
Book reference (Kurose & Ross, Chapter 3, p. 205): short quote: "stop-and-wait protocols".
Use the performance slides to show that stop-and-wait is correct but inefficient. The sender transmits one packet and then waits through the RTT. Even with a very fast link, the sender spends most of the time idle.
In the formula, L/R is the time required to push the packet onto the link, and RTT is the round-trip waiting time. In the 1 Gbps and 15 ms propagation example, utilization is extremely low. This motivates pipelining.
Visible slide keywords: rdt3.0: stop-and-wait operation, sender, receiver, U, sender, =, L / R, RTT, RTT, L/R, + L / R, =, 0.00027, =, .008, 30.008, rdt, 3.0 protocol performance stinks!, Protocol limits performance of underlying infrastructure (channel), Transport Layer: 3-, 64.
Ask: does a fast link alone guarantee high utilization? No; protocol waiting behavior can limit performance.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

rdt3.0: pipelined protocols operation

pipelining: sender allows multiple, “in-flight”, yet-to-be-acknowledged packets

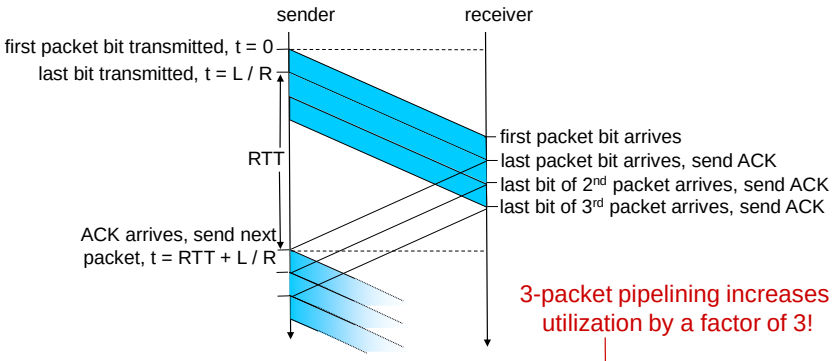
- range of sequence numbers must be increased
- buffering at sender and/or receiver



(a) a stop-and-wait protocol in operation

Slide 65 - rdt3.0
Book reference (Kurose & Ross, Chapter 3, p. 215): short quote: "this technique is known as pipelining".
Present pipelining as the answer to stop-and-wait's bottleneck. The sender transmits multiple packets before waiting for ACKs, keeping the link busier and increasing utilization.
Also explain the cost: the sequence-number range must be larger, the sender and receiver may need to buffer multiple packets, and the protocol must decide which packets to retransmit after loss.
Visible slide keywords: rdt3.0: pipelined protocols operation, pipelining:, sender allows multiple, “, in-flight”, yet-to-be-acknowledged packets, range of sequence numbers must be increased, buffering at sender and/or receiver, Transport Layer: 3-, 65.
Key point: pipelining improves performance, but it makes protocol state and error recovery more complex.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

Pipelining: increased utilization



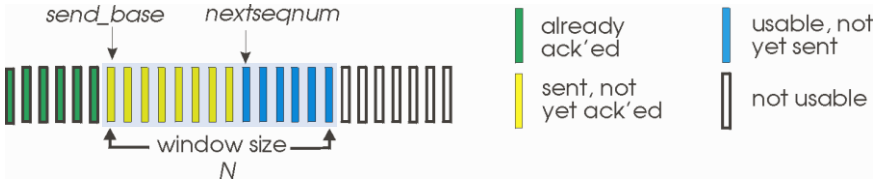
3-packet pipelining increases utilization by a factor of 3!

$$U_{sender} = \frac{3L / R}{RTT + L / R} = \frac{.0024}{30.008} = 0.00081$$

Slide 66 - Pipelining: increased utilization
Book reference (Kurose & Ross, Chapter 3, p. 215): short quote: "this technique is known as pipelining".
Present pipelining as the answer to stop-and-wait's bottleneck. The sender transmits multiple packets before waiting for ACKs, keeping the link busier and increasing utilization.
Also explain the cost: the sequence-number range must be larger, the sender and receiver may need to buffer multiple packets, and the protocol must decide which packets to retransmit after loss.
Visible slide keywords: Pipelining: increased utilization, first packet bit transmitted, t = 0, sender, receiver, RTT, last bit transmitted, t = L / R, first packet bit arrives, last packet bit arrives, send ACK, ACK arrives, send next, packet, t = RTT + L / R, last bit of 2, nd, packet arrives, send ACK, last bit of 3, rd, packet arrives, send ACK, 3-packet pipelining increases, utilization by a factor of 3!, Transport Layer: 3-, 66.
Key point: pipelining improves performance, but it makes protocol state and error recovery more complex.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

Go-Back-N: sender

- sender: “window” of up to N, consecutive transmitted but unACKed pkts
  - k-bit seq # in pkt header



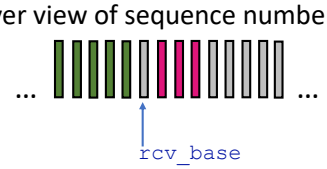
- *cumulative ACK:* ACK(*n*): ACKs all packets up to, including seq # *n*
  - on receiving ACK(*n*): move window forward to begin at *n+1*
- timer for oldest in-flight packet
- *timeout(n):* retransmit packet n and all higher seq # packets in window

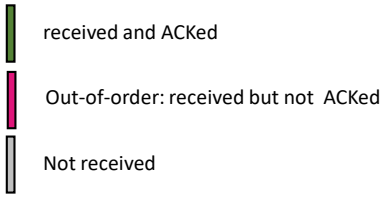
Slide 67 - Go-Back-N
Book reference (Kurose & Ross, Chapter 3, p. 216): short quote: "no more than some maximum allowable number".
Explain the Go-Back-N sender window: at most N transmitted but unacknowledged packets may be in the pipeline. ACKs are cumulative; ACK(n) implies that packets up to n have been received correctly.
The receiver is simple: it accepts the next in-order packet, discards out-of-order packets, and repeats the ACK for the last correctly received in-order packet. After a timeout, the sender may retransmit from the missing packet onward.
Visible slide keywords: Go-Back-N: sender, sender: “window” of up to N, consecutive transmitted but, unACKed, pkts, k-bit seq # in pkt header, cumulative ACK:, ACK(, n,): ACKs all packets up to, including seq #, n, on receiving ACK(, n,): m, ove, window forward to begin at, n+1, timer for oldest in-flight packet, timeout(n):, retransmit packet n and all higher seq # packets in window, Transport Layer: 3-, 67.
Quick question: why can GBN retransmit packets that already arrived correctly? Because the receiver does not buffer out-of-order packets.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

Go-Back-N: receiver

- ACK-only: always send ACK for correctly-received packet so far, with highest *in-order* seq #
  - may generate duplicate ACKs
  - need only remember `rcv_base`
- on receipt of out-of-order packet:
  - can discard (don't buffer) or buffer: an implementation decision
  - re-ACK pkt with highest in-order seq #

Receiver view of sequence number space:

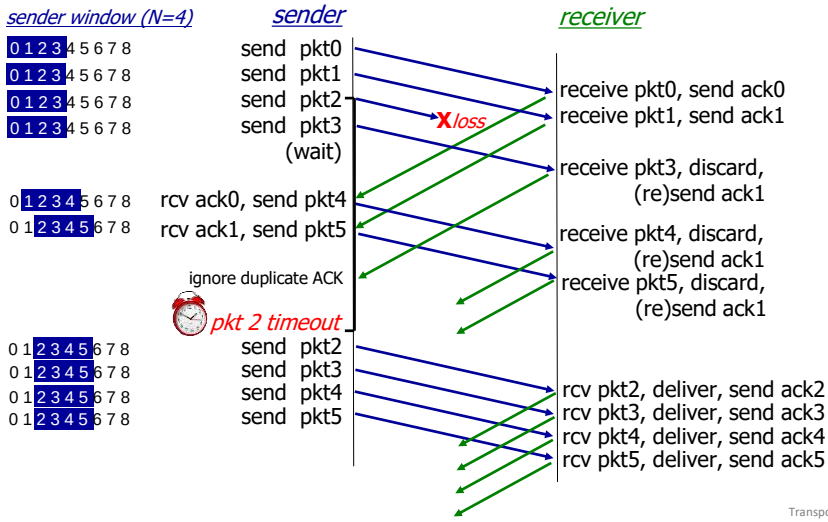




Slide 68 - Go-Back-N
Book reference (Kurose & Ross, Chapter 3, p. 216): short quote: "no more than some maximum allowable number".
Explain the Go-Back-N sender window: at most N transmitted but unacknowledged packets may be in the pipeline. ACKs are cumulative; ACK(n) implies that packets up to n have been received correctly.
The receiver is simple: it accepts the next in-order packet, discards out-of-order packets, and repeats the ACK for the last correctly received in-order packet. After a timeout, the sender may retransmit from the missing packet onward.
Visible slide keywords: Go-Back-N: receiver, ACK-only: always send ACK for correctly-received packet so far, with highest, in-order, seq #, may generate duplicate ACKs, need only remember, rcv_base, on receipt of out-of-order packet:, can discard (don', t buffer) or buffer: an implementation decision, re-ACK pkt with highest in-order seq #, rcv_base, received and, ACKed, Out-of-order: received but not, ACKed, Not received, Receiver view of sequence number space:, …, …, Transport Layer: 3-, 68.
Quick question: why can GBN retransmit packets that already arrived correctly? Because the receiver does not buffer out-of-order packets.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a

more organized and manageable way.

Go-Back-N in action



Slide 69 - Go-Back-N
Book reference (Kurose & Ross, Chapter 3, p. 216): short quote: "no more than some maximum allowable number".
Explain the Go-Back-N sender window: at most N transmitted but unacknowledged packets may be in the pipeline. ACKs are cumulative; ACK(n) implies that packets up to n have been received correctly.
The receiver is simple: it accepts the next in-order packet, discards out-of-order packets, and repeats the ACK for the last correctly received in-order packet. After a timeout, the sender may retransmit from the missing packet onward.
Visible slide keywords: Go-Back-N in action, send pkt0, send pkt1, send pkt2, send pkt3, (wait), sender, receiver, receive pkt0, send ack0, receive pkt1, send ack1, receive pkt3, discard,, (re)send ack1, send pkt2, send pkt3, send pkt4, send pkt5, X, loss, pkt 2 timeout, receive pkt4, discard,, (re)send ack1, receive pkt5, discard,, (re)send ack1, rcv, pkt2, deliver, send ack2, rcv, pkt3, deliver, send ack3, rcv, pkt4, deliver, send ack4, rcv, pkt5, deliver, send ack5, ignore duplicate ACK, sender window (N=4), 0 1 2 3, 4 5 6 7 8, 0 1 2 3, 4 5 6 7 8, 0 1 2 3, 4 5 6 7 8, 0 1 2 3, 4 5 6 7 8, rcv, ack0, send pkt4, 0, 1 2 3 4, 5 6 7 8, 0 1, 2 3 4 5, 6 7 8, 0 1, 2 3 4 5, 6 7 8, 0 1, 2 3 4 5, 6 7 8, 0 1, 2 3 4 5, 6 7 8, .
Quick question: why can GBN retransmit packets that already arrived correctly?

Because the receiver does not buffer out-of-order packets.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

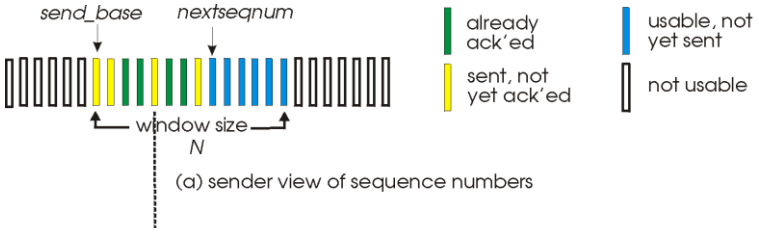
Selective repeat

- receiver *individually* acknowledges all correctly received packets
  - buffers packets, as needed, for eventual in-order delivery to upper layer
- sender times-out/retransmits individually for unACKed packets
  - sender maintains timer for each unACKed pkt
- sender window
  - *N* consecutive seq #s
  - limits seq #s of sent, unACKed packets

Slide 70 - Selective repeat
Book reference (Kurose & Ross, Chapter 3, p. 221): short quote: "selective-repeat protocols avoid unnecessary retransmissions".
Explain Selective Repeat by contrasting it with Go-Back-N. The basic idea is to avoid unnecessary retransmissions: the receiver buffers correctly received out-of-order packets and acknowledges each correct packet individually; the sender retransmits only packets suspected to be lost or corrupted.
Sender and receiver windows may move at different times. If the sequence-number space is too small, an old retransmission can be confused with a new packet inside the current window. The dilemma slides are designed to show that ambiguity.
Visible slide keywords: Selective repeat, receiver, individually, acknowledges all correctly received packets, buffers packets, as needed, for eventual in-order delivery to upper layer, sender times-out/retransmits individually for, unACKed, packets, sender maintains timer for each, unACKed, pkt, sender window, N, consecutive seq #, s, limits seq #s of sent,, unACKed, packets, Transport Layer: 3-, 70.
Key point: SR can be more efficient, but it requires receiver buffering, individual ACK/timer tracking, and a sufficiently large sequence-number space.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a

more organized and manageable way.

Selective repeat: sender, receiver windows



Slide 71 - Selective repeat
Book reference (Kurose & Ross, Chapter 3, p. 222): short quote: "The receive window is then moved forward".
Explain Selective Repeat by contrasting it with Go-Back-N. The basic idea is to avoid unnecessary retransmissions: the receiver buffers correctly received out-of-order packets and acknowledges each correct packet individually; the sender retransmits only packets suspected to be lost or corrupted.
Sender and receiver windows may move at different times. If the sequence-number space is too small, an old retransmission can be confused with a new packet inside the current window. The dilemma slides are designed to show that ambiguity.
Visible slide keywords: Selective repeat: sender, receiver windows, Transport Layer: 3-, 71.
Key point: SR can be more efficient, but it requires receiver buffering, individual ACK/timer tracking, and a sufficiently large sequence-number space.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

Selective repeat: sender and receiver

sender

data from above:

- if next available seq # in window, send packet

timeout(*n*):

- resend packet *n*, restart timer

ACK(*n*) in [sendbase,sendbase+N]:

- mark packet *n* as received
- if n smallest unACKed packet, advance window base to next unACKed seq #

receiver

packet *n* in [rcvbase, rcvbase+N-1]

- send ACK(*n*)
- out-of-order: buffer
- in-order: deliver (also deliver buffered, in-order packets), advance window to next not-yet-received packet

packet *n* in [rcvbase-N,rcvbase-1]

- ACK(*n*)

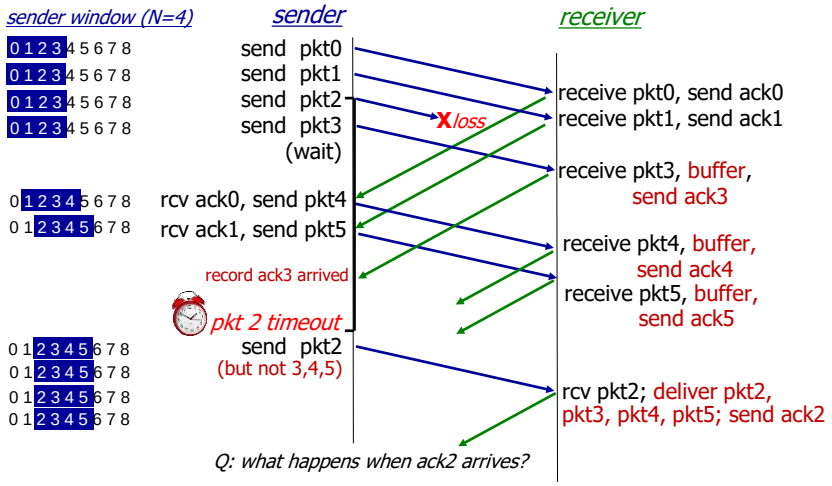
otherwise:

- ignore

Slide 72 - Selective repeat
Book reference (Kurose & Ross, Chapter 3, p. 221): short quote: "selective-repeat protocols avoid unnecessary retransmissions".
Explain Selective Repeat by contrasting it with Go-Back-N. The basic idea is to avoid unnecessary retransmissions: the receiver buffers correctly received out-of-order packets and acknowledges each correct packet individually; the sender retransmits only packets suspected to be lost or corrupted.
Sender and receiver windows may move at different times. If the sequence-number space is too small, an old retransmission can be confused with a new packet inside the current window. The dilemma slides are designed to show that ambiguity.
Visible slide keywords: Selective repeat: sender and receiver, data from above:, if next available seq # in window, send packet, timeout(, n,):, resend packet, n, , restart timer, ACK(, n,), in [, sendbase,sendbase+N,], :, mark packet, n, as received, if n smallest, unACKed, packet, advance window base to next, unACKed, seq #, sender, packet, n, in, [, rcvbase, , rcvbase+N-1], send ACK(, n,), out-of-order: buffer, in-order: deliver (also deliver buffered, in-order packets), advance window to next not-yet-received packet, packet, n, in, [rcvbase-N,rcvbase-1], ACK(, n,), otherwise:, ignore, receiver, Transport Layer: 3-, 72.
Key point: SR can be more efficient, but it requires receiver buffering, individual

ACK/timer tracking, and a sufficiently large sequence-number space.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

Selective Repeat in action



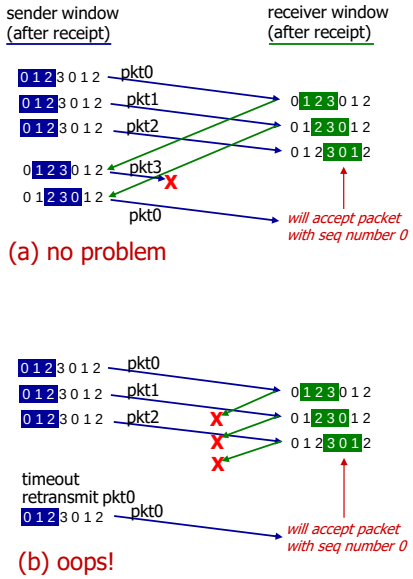
Slide 73 - Selective repeat
Book reference (Kurose & Ross, Chapter 3, p. 221): short quote: "selective-repeat protocols avoid unnecessary retransmissions".
Explain Selective Repeat by contrasting it with Go-Back-N. The basic idea is to avoid unnecessary retransmissions: the receiver buffers correctly received out-of-order packets and acknowledges each correct packet individually; the sender retransmits only packets suspected to be lost or corrupted.
Sender and receiver windows may move at different times. If the sequence-number space is too small, an old retransmission can be confused with a new packet inside the current window. The dilemma slides are designed to show that ambiguity.
Visible slide keywords: Selective Repeat in action, send pkt0, send pkt1, send pkt2, send pkt3, (wait), sender, receiver, send pkt2, (but not 3,4,5), X, loss, pkt 2 timeout, sender window (N=4), 0 1 2 3, 4 5 6 7 8, 0 1 2 3, 4 5 6 7 8, 0 1 2 3, 4 5 6 7 8, 0 1 2 3, 4 5 6 7 8, rcv, ack0, send pkt4, 0, 1 2 3 4, 5 6 7 8, 0 1, 2 3 4 5, 6 7 8, 0 1, 2 3 4 5, 6 7 8, 0 1, 2 3 4 5, 6 7 8, 0 1, 2 3 4 5, 6 7 8, 0 1, 2 3 4 5, 6 7 8, rcv, ack1, send pkt5, receive pkt0, send ack0, receive pkt1, send ack1, receive pkt3,, buffer, ,, send ack3, record ack3 arrived, receive pkt4,, buffer,, send ack4, receive pkt5,, buffer,, send ack5, rcv, pkt2;, deliver pkt2,, pkt3, pkt4, pkt5; send ack2, Q: what happens when ack2 arrives?, Transport.

Key point: SR can be more efficient, but it requires receiver buffering, individual ACK/timer tracking, and a sufficiently large sequence-number space.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

Selective repeat: a dilemma!

example:

- seq #s: 0, 1, 2, 3 (base 4 counting)
- window size=3



Slide 74 - Selective repeat
Book reference (Kurose & Ross, Chapter 3, p. 222): short quote: "The receive window is then moved forward".
Explain Selective Repeat by contrasting it with Go-Back-N. The basic idea is to avoid unnecessary retransmissions: the receiver buffers correctly received out-of-order packets and acknowledges each correct packet individually; the sender retransmits only packets suspected to be lost or corrupted.
Sender and receiver windows may move at different times. If the sequence-number space is too small, an old retransmission can be confused with a new packet inside the current window. The dilemma slides are designed to show that ambiguity.
Visible slide keywords: Selective repeat:, a dilemma!, 0 1 2, 3 0 1 2, 0 1 2, 3 0 1 2, 0 1 2, 3 0 1 2, pkt0, pkt1, pkt2, 0 1 2, 3 0 1 2, pkt0, timeout, retransmit pkt0, 0, 1 2 3, 0 1 2, 0 1, 2 3 0, 1 2, 0 1 2, 3 0 1, 2, X, X, X, will accept packet, with seq number 0, (b) oops!, receiver window, (after receipt), sender window, (after receipt), 0 1 2, 3 0 1 2, 0 1 2, 3 0 1 2, 0 1 2, 3 0 1 2, pkt0, pkt1, pkt2, 0 1, 2, 3 0, 1 2, pkt0, 0, 1 2 3, 0 1 2, 0 1, 2 3 0, 1 2, 0 1 2, 3 0 1, 2, X, will accept packet, with seq number 0, 0, 1 2, 3, 0 1 2, pkt3, (a) no problem, example:, seq #, s: 0, 1, 2, 3, (base 4 counting), window size=3, Transport Layer: 3-, 74.
Key point: SR can be more efficient, but it requires receiver buffering, individual

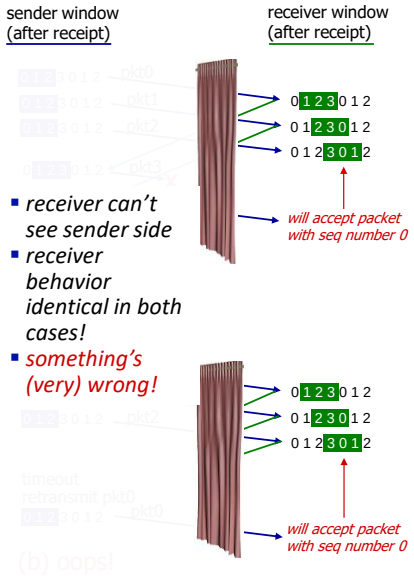
ACK/timer tracking, and a sufficiently large sequence-number space.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.

Selective repeat: a dilemma!

example:

- seq #s: 0, 1, 2, 3 (base 4 counting)
- window size=3

Q: what relationship is needed between sequence # size and window size to avoid problem in scenario (b)?



Slide 75 - Selective repeat
Book reference (Kurose & Ross, Chapter 3, p. 222): short quote: "The receive window is then moved forward".
Explain Selective Repeat by contrasting it with Go-Back-N. The basic idea is to avoid unnecessary retransmissions: the receiver buffers correctly received out-of-order packets and acknowledges each correct packet individually; the sender retransmits only packets suspected to be lost or corrupted.
Sender and receiver windows may move at different times. If the sequence-number space is too small, an old retransmission can be confused with a new packet inside the current window. The dilemma slides are designed to show that ambiguity.
Visible slide keywords: Selective repeat:, a dilemma!, Q:, what relationship is needed between sequence # size and window size to avoid problem in scenario (b)?, 0 1 2, 3 0 1 2, 0 1 2, 3 0 1 2, 0 1 2, 3 0 1 2, pkt0, pkt1, pkt2, 0 1 2, 3 0 1 2, pkt0, timeout, retransmit pkt0, 0, 1 2 3, 0 1 2, 0 1, 2 3 0, 1 2, 0 1 2, 3 0 1, 2, X, X, X, will accept packet, with seq number 0, (b) oops!, receiver window, (after receipt), sender window, (after receipt), 0 1 2, 3 0 1 2, 0 1 2, 3 0 1 2, 0 1 2, 3 0 1 2, pkt0, pkt1, pkt2, 0 1, 2, 3 0, 1 2, pkt0, 0, 1 2 3, 0 1 2, 0 1, 2 3 0, 1 2, 0 1 2, 3 0 1, 2, X, will accept packet, with seq number 0, 0, 1 2, 3, 0 1 2, pkt3, (a) no problem, example:, seq #, s: 0, 1, 2, 3, (base 4 counting).

Key point: SR can be more efficient, but it requires receiver buffering, individual ACK/timer tracking, and a sufficiently large sequence-number space.
Close the slide by connecting it to the next one: this mechanism is one part of the control information the transport layer adds so applications can communicate in a more organized and manageable way.