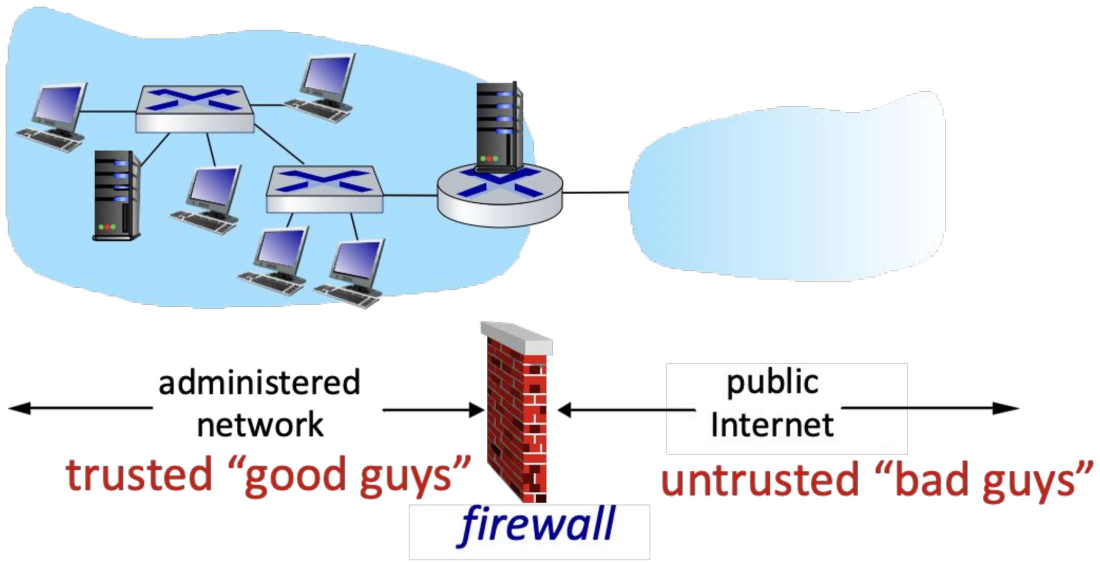
Operational Network Security

Firewalls

- isolates organization's internal network from larger Internet, allowing some packets to pass, blocking others

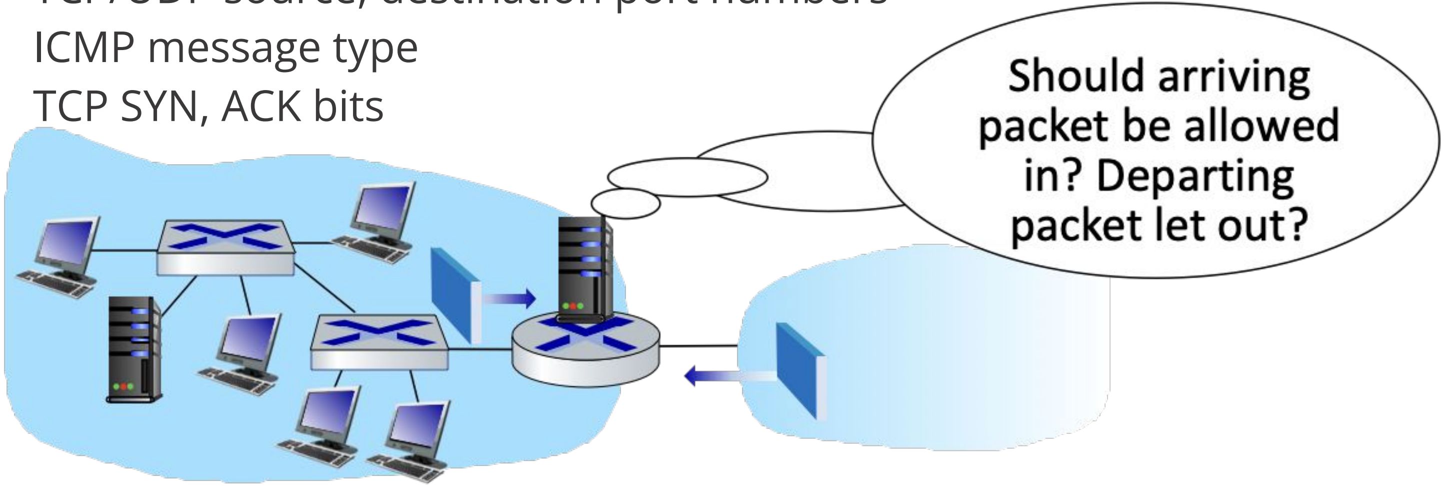


Firewalls

- prevent denial of service attacks:
  - SYN flooding: attacker establishes many bogus TCP connections, no resources left for “real” connections
- prevent illegal modification/access of internal data
  - e.g., attacker replaces CIA’s homepage with something else
- allow only authorized access to inside network
  - set of authenticated users/hosts
- three types of firewalls:
  - stateless packet filters
  - stateful packet filters
  - application gateways

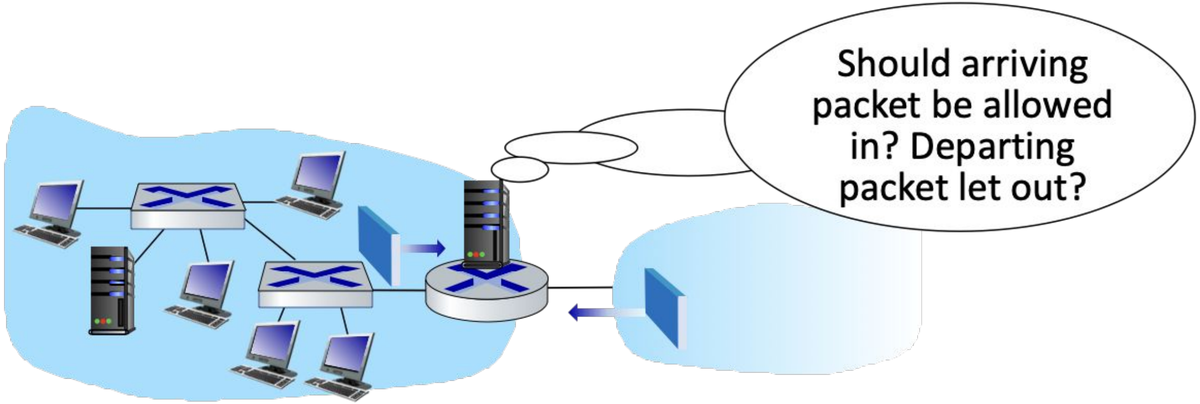
Stateless Packet Filtering

- internal network connected to Internet via router firewall
- filters packet-by-packet, decision to forward/drop packet based on:
  - source IP address, destination IP address
  - TCP/UDP source, destination port numbers
  - ICMP message type
  - TCP SYN, ACK bits



Stateless Packet Filtering

- example 1: block incoming and outgoing datagrams with IP protocol field = 17 and with either source or dest port = 23
  - result: all incoming, outgoing UDP flows and telnet connections are blocked
- example 2: block inbound TCP segments with ACK=0
  - result: prevents external clients from making TCP connections with internal clients, but allows internal clients to connect to outside



Stateful Packet Filtering

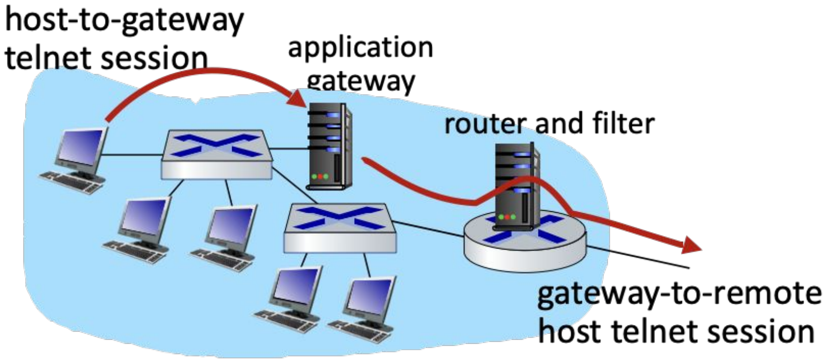
- stateless packet filter: heavy handed tool
    - admits packets that “make no sense,” e.g., dest port = 80, ACK bit set, even though no TCP connection established
- stateful packet filter: track status of every TCP connection
    - track connection setup (SYN), teardown (FIN): determine whether incoming, outgoing packets “makes sense”
    - timeout inactive connections at firewall: no longer admit packets

Application gateways

filter packets on application data as well as on IP/TCP/UDP fields.

example: allow select internal users to telnet outside

1. require all telnet users to telnet through gateway.
2. for authorized users, gateway sets up telnet connection to dest host
   a. gateway relays data between 2 connections
3. router filter blocks all telnet connections not originating from gateway



Limitations of firewalls, gateways

- IP spoofing: router can't know if data "really" comes from claimed source
- if multiple apps need special treatment, each has own app. gateway
- client software must know how to contact gateway
  - e.g., must set IP address of proxy in Web browser
- filters often use all or nothing policy for UDP
- tradeoff: degree of communication with outside world, level of security
- many highly protected sites still suffer from attacks

Intrusion detection systems

- packet filtering:
  - operates on TCP/IP headers only
  - no correlation check among sessions
- IDS: intrusion detection system
  - deep packet inspection: look at packet contents (e.g., check character strings in packet against database of known virus, attack strings)
  - examine correlation among multiple packets
    - port scanning
    - network mapping
    - DoS attack

Transport Layer

Transport

Network layer: communication between hosts

Transport layer: communication between processes

Transport

Network layer: communication between hosts

Transport layer: communication between processes

Muxing / demuxing across many processes

Unit of data: segment

Transport

- Two principal transports: TCP and UDP
- 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

routing

reliable delivery

How do you ensure reliable transport on top of best-effort delivery?

Transport

In the Internet, reliability is ensured by the end hosts, not by the network

Reliability is left to L4, the Transport Layer

Why?

Reliability is left to L4, the Transport Layer

goals	**Keep the network simple, dumb**
	make it relatively "easy" to build and operate a network
	Keep applications as network "unaware" as possible
	a developer should focus on its app, not on the network
design	**Implement reliability in-between, in the networking stack**
	relieve the burden from both the app and the network

Network stack - reliability in L4

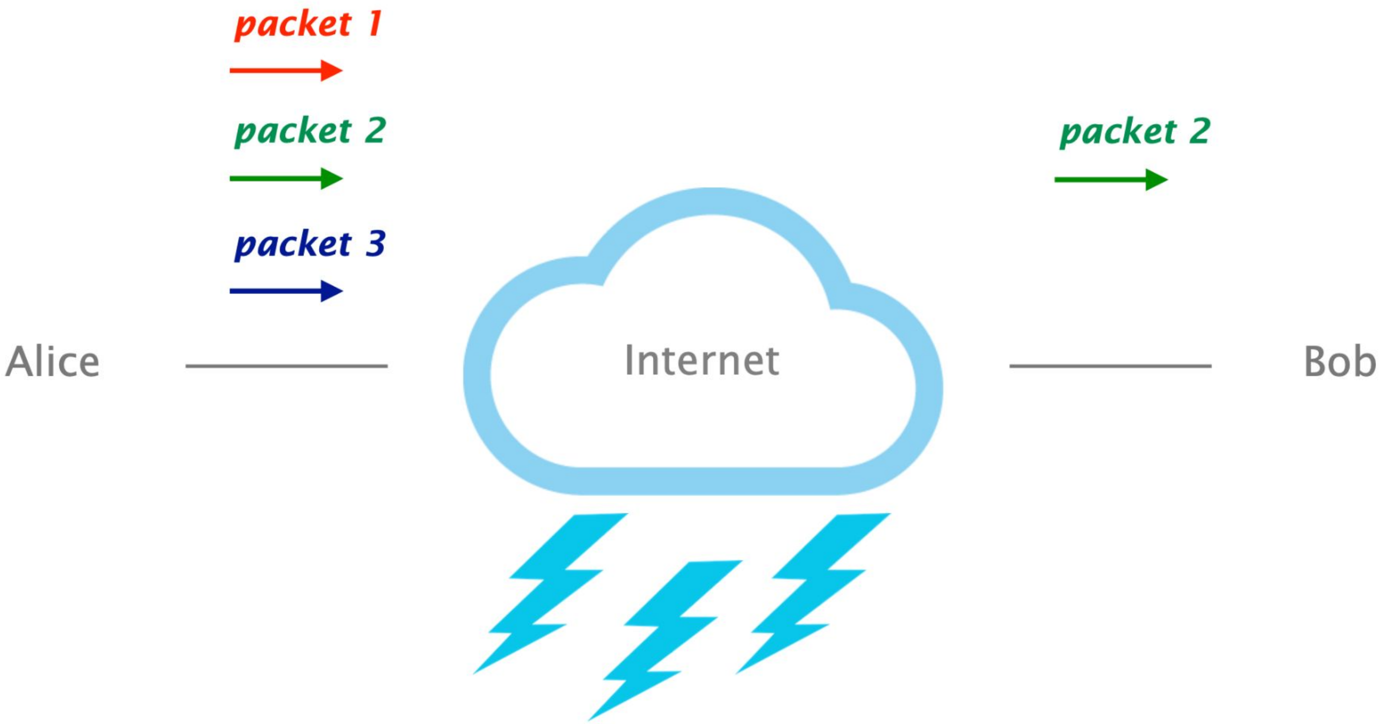
	layer	
	Application	
L4	Transport	reliable end-to-end delivery
L3	Network	global best-effort delivery
	Link	
	Physical	

Network stack - reliability in L4

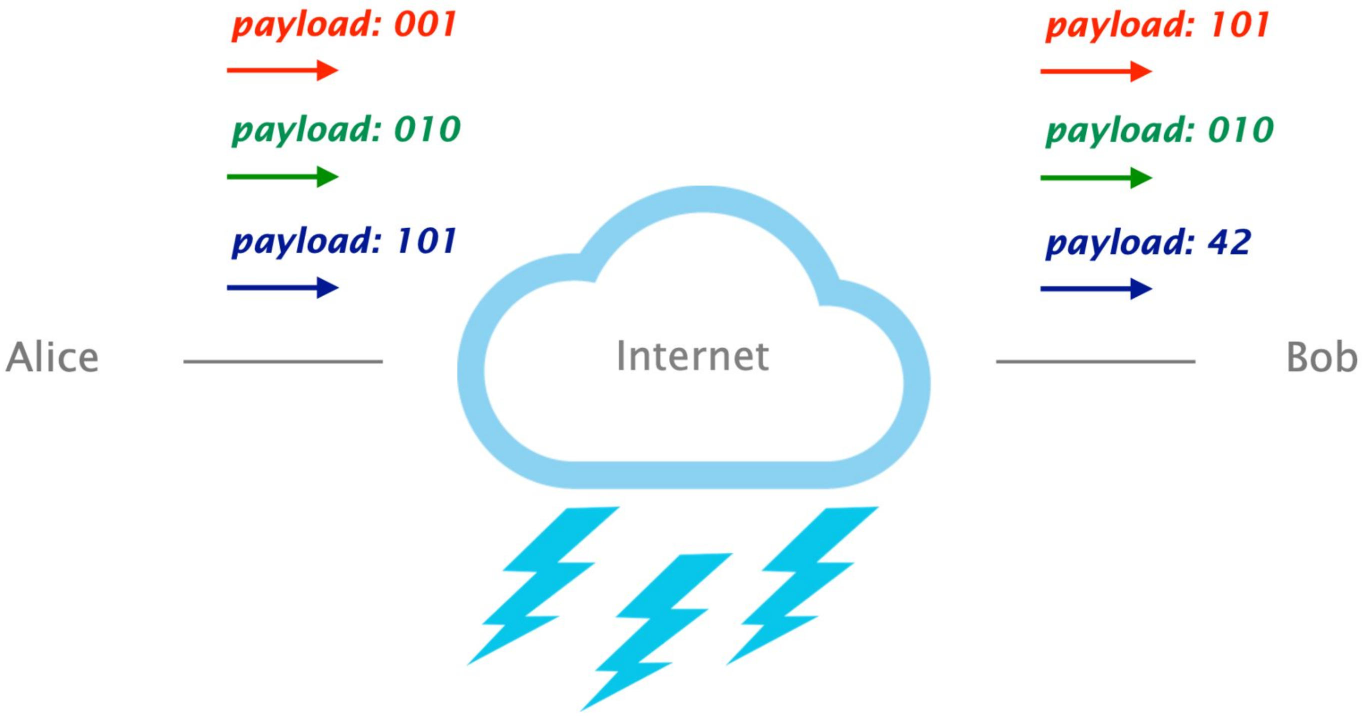
	layer		
	Application		
L4	Transport	reliable end-to-end delivery	
L3	Network	global best-effort delivery	IP is “best-effort”
	Link		
	Physical		

Example: Alice and Bob

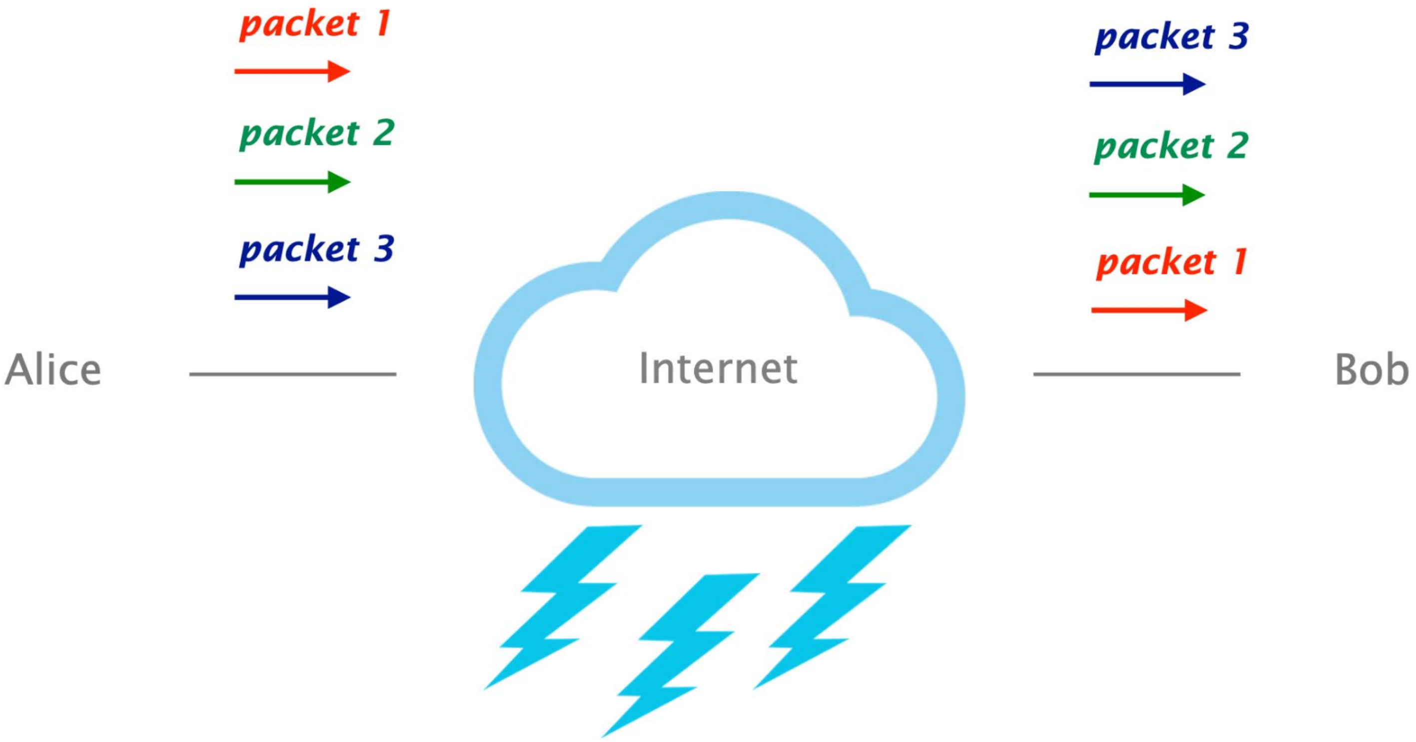
IP packets can be lost or delayed



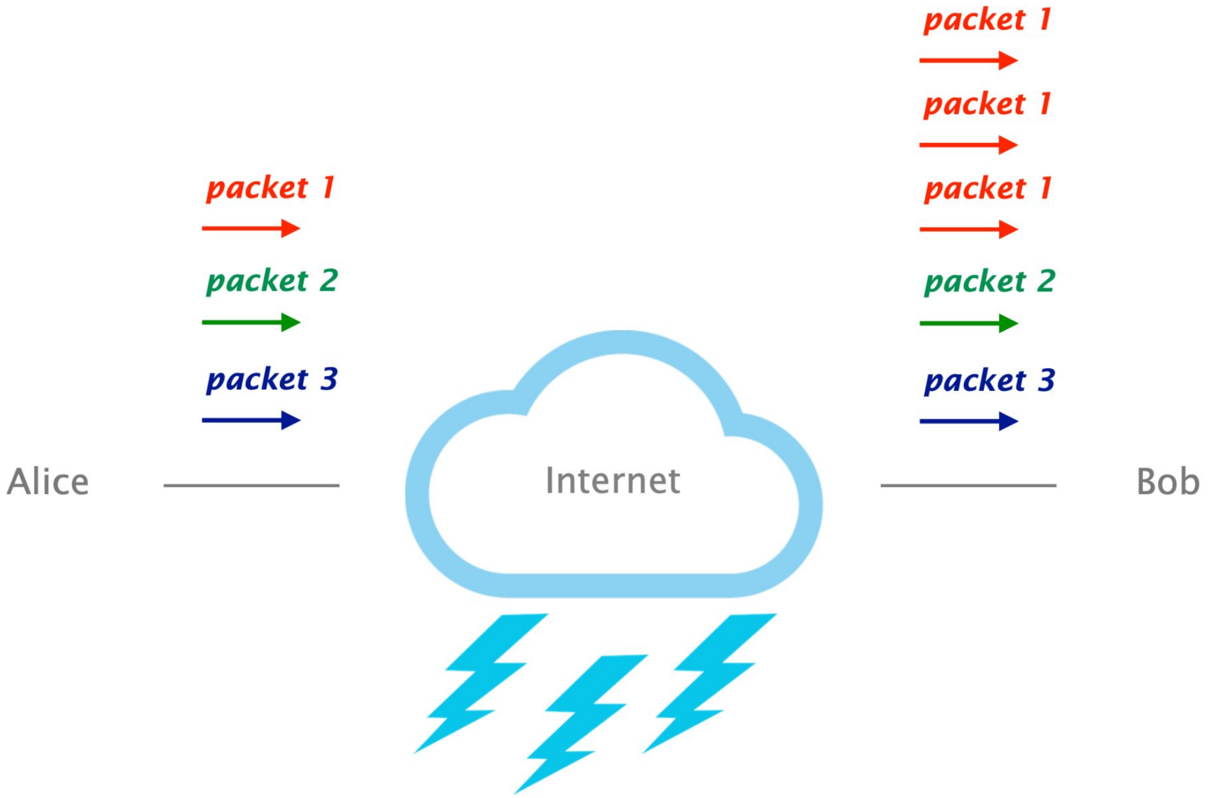
IP packets can get corrupted



IP packets can get reordered



IP packets can be duplicated



The four goals of reliable transport

goals	
correctness	ensure data is delivered, in order, and untouched
timeliness	minimize time until data is transferred
efficiency	optimal use of bandwidth
fairness	play well with concurrent communications

The four goals of reliable transport

goals	
correctness	ensure data is delivered, in order, and untouched
timeliness	minimize time until data is transferred
efficiency	optimal use of bandwidth
fairness	play well with concurrent communications

Correctness is clean / easy with routing

sufficient and necessary condition

Theorem

a global forwarding state is valid **if and only if**

- there are no dead ends

  no outgoing port defined in the table

- there are no loops

  packets going around the same set of nodes

Correctness is clean / easy with routing

sufficient and necessary condition

Theorem

How can we come up with a similarly clean definition for transport?

- there are no loops

  packets going around the same set of nodes

Correctness in reliable transport

A reliable transport design is correct if…

attempt #1 — packets are delivered to the receiver

Wrong — Consider that the network is partitioned

We cannot say a transport design is *incorrect*
if it doesn't work in a partitioned network…

Correctness in reliable transport

A reliable transport design is correct if…

attempt #2

packets are delivered to receiver if and only if it was possible to deliver them

Wrong

If the network is only available one instant in time, only an oracle would know when to send

We cannot say a transport design is *incorrect* if it doesn't know the unknowable

Correctness in reliable transport

A reliable transport design is correct if…

attempt #3

It resends a packet if and only if the previous packet was lost or corrupted

Wrong

Consider two cases

- packet made it to the receiver and all packets from receiver were dropped
- packet is dropped on the way and all packets from receiver were dropped

Correctness in reliable transport

A reliable transport design is correct if…

attempt #3 — It resends a packet if and only if the previous packet was lost or corrupted

In both cases the sender has no feedback. How can it know to re-send???

Wrong — Consider two cas[…]

- packet made it to the receiver and all packets from receiver were dropped
- packet is dropped on the way and all packets from receiver were dropped

Correctness in reliable transport

A reliable transport design is correct if…

attempt #4

A packet is always resent if
the previous packet was lost or corrupted

A packet may be resent at other times

Correct!

A transport mechanism is only correct if and only if it resends all dropped or corrupted packets

Sufficient “if”	algorithm will always keep trying to deliver undelivered packets
Necessary “only if”	if it ever let a packet go undelivered without resending it, it isn’t reliable
Note	it is ok to give up after a while but must announce it to the application