Lecture 14: The transport layer

Different types of abstractions, reliability techniques and tradeoffs

Lecturer: Venkat Arun

We roughly follow chapter 5.2, but there are differences

Recap

How sequence numbers are send and ACKed

Sequence numbers of the bytes that the sender sends	**Sequence numbers the receiver acknowledges (it does not send any data bytes)**
0-1000	0-1000
1000-2000	0-2000
2000-3000	Lost. Did not receive
3000-4000	0-2000, 3000-4000
4000-5000	0-2000, 3000-5000
⋮ after several more packets, sender receives the ACKs and realizes bytes 2000-3000 were lost. This takes time because it takes one RTT (Round Trip Time) for the ACKs to reach the sender. Plus, it must wait for either dupACKs or a timeout to declare loss. Say it has sent 15 more packets by then, we will continue as shown below	
20000-21000	0-2000, 3000-21000
2000-3000	0-21000 We're back to normal now 🎉

Objective for today

There are lots of engineering tradeoffs that are possible. The design presented yesterday is not the only possible design or even the best one.

Today, we will look at lots of alternate designs, all of which have been/are used:

- Other reliability mechanisms
- Alternative transport abstractions
- Reconsider the internet’s end-to-end and best effort principles

Cumulative ACKs

- The method described in the previous lecture is *not* how TCP works, but it is quite similar to how a newer protocol, QUIC, works.
- TCP originally did not give receivers the ability to acknowledge ranges of bytes. Instead, it could only send a *cumulative* acknowledgment
- A cumulative ACK is a single sequence number that acknowledges all bytes sent from the beginning to that sequence number

Cumulative ACK example

How the same circumstance would be handled by cumulative ACKs

Sequence numbers of the bytes that the sender sends	Sequence number the receiver acknowledges (it does not send any data bytes)
0-1000	1000
1000-2000	2000
2000-3000	Lost. Did not receive
3000-4000	2000
4000-5000	2000
⋮ after several more packets, sender receives the ACKs and realizes bytes after 2000 were lost. However, it does not know how many were lost. One approach is to be optimistic and assume only one packet was lost and retransmit it	
20000-21000	2000
2000-3000	21000 We're back to normal now 🎉
21000-22000	22000 The sender does not yet know that things are back to normal, but it chooses to be optimistic

TCP Fast Retransmit

When only cumulative ACKs are available, TCP uses two retransmission strategies

- If it obtains 3 duplicate ACKs (dupACKs), it retransmits the segment immediately following the ACKed sequence number and continues with fresh transmissions afterward
  - Note: if the sender is getting dupACKs, it still knows that packets are still getting through. Thus, the network has not completely died and there is reason for optimism
- If it times out, things are much worse since it means no packets are getting through (why?). It assumes all subsequent packets are lost and takes drastic steps, which we will discuss when we discuss congestion control
  - Why does a timeout mean that no packets are getting through? If they were getting through, loss would have been detected with dupACKs first
- There are lots of corner cases here. In Linux, the logic is implemented in net/ipv4/tcp.c. It is ~5000 lines of complicated spaghetti code

Where cumulative ACKs go wrong

How the same circumstance would be handled by cumulative ACKs

Sequence numbers of the bytes that the sender sends	Sequence number the receiver acknowledges (it does not send any data bytes)
0-1000	1000
1000-2000	2000
2000-3000	Lost. Did not receive
3000-4000	Lost. Did not receive
4000-5000	2000
⋮ Here, the sender's view is exactly the same as before. However, 2 packets got lost. Thus, here being optimistic is unwarranted	
20000-21000	2000
2000-3000	3000 We have not made much progress and are headed toward a long recovery period and probably a timeout
21000-22000	3000 Alas.

TCP was later modified

- Cumulative ACKs are clearly suboptimal. Thus, TCP Selective ACKnowledgments (SACKs) were standardized
- However, the header format was already, so people adopted the "TCP Options" fields. These allowed for an extra 40 bytes of fields, with each field being 32 bits.
- SACKs were standardized in IETF RFC2018 in 1996
- This allowed for up to 3 ranges
- This is smaller from the arbitrarily many ranges allowed in QUIC, but is nevertheless almost as good
- However, all TCP implementations must handle SACK-free packets because:
  - The other end need not implement SACK
  - Routers are allowed to drop options fields
- **Note:** You do not need to remember all this detail. A vague recollection is probably enough for your career. You can always look things up later. Even I had to look up some of it. I have spent 9 years (and counting) of my life thinking about TCP

An even simpler design: Go-back-n

- Suppose the receiver gets bytes 0-9000 and 10000-20000
- Next, it receives 9000-20000. It can now reconstruct the entire stream
- However, maintaining the state required to do this can sometimes be expensive. An alternate simpler design for the receiver is the following:
  - If the received packet is the one immediately following the bytes already delivered to the application, deliver this new data to the application
  - Otherwise, drop the packet
- This is called the go-back-n protocol and enables extremely receivers. However, now the sender must retransmit all packets after every loss event.
- Where might you use this?
- Reference: https://www.geeksforgeeks.org/sliding-window-protocol-set-2-receiver-side/

An even simpler design: Go-back-n

- Go-back-n is used in some link layer protocols (I believe)
- It was used recently in Microsoft datacenters. Why?
- Datacenter servers need to handle extremely high data rates. Using a software-based TCP stack can be inefficient. Thus, they tried implementing the transport stack in hardware
- However, hardware has a hard time being dynamic. Thus, they used the go-back-n protocol
- This was ok since they put in a lot of effort to make the rest of the network highly reliable making packet losses unlikely
- I believe that they have now gone back to a more “normal” protocol, but am not sure

Takeaway

There is no “best” design. Different circumstances call for different choices.

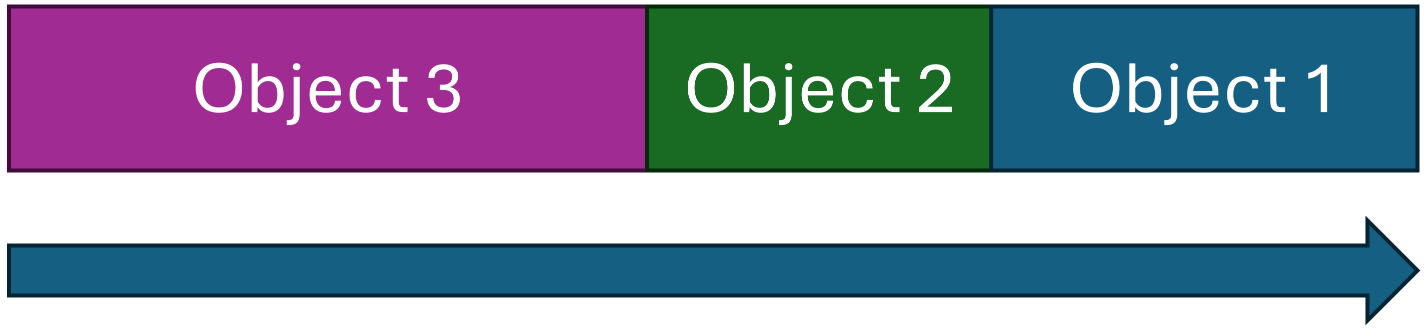
The end-to-end principle of the internet and the layered architecture makes it (slightly) simpler to modify protocols independently of each other. However, the sheer scale of the internet still makes it one of the hardest places in which to change fundamental protocols

Alternate transport abstractions

Three transport ←→ application interfaces

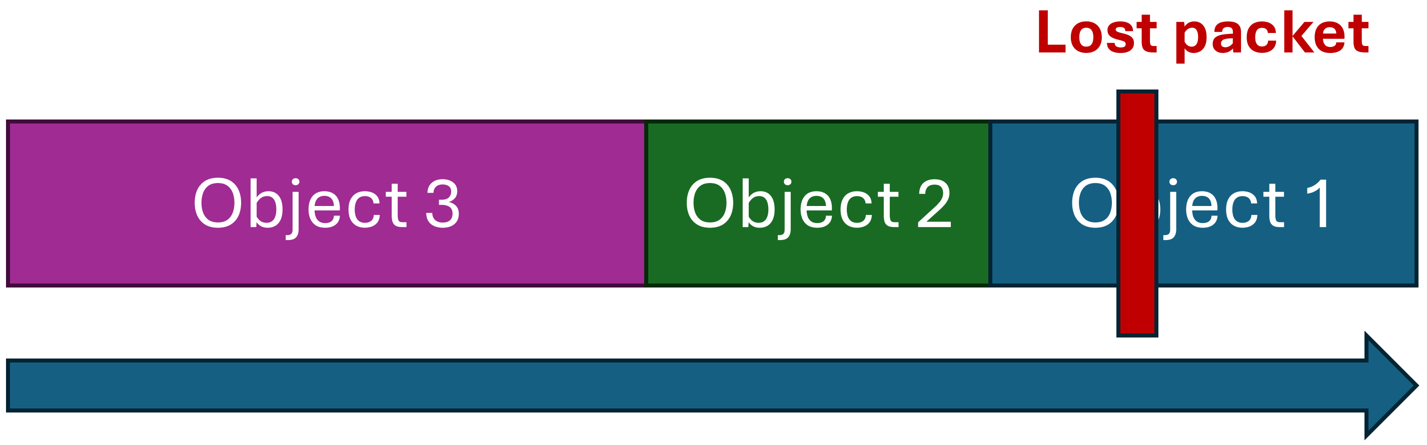
- UDP: best effort, packet oriented
- TCP: reliable stream oriented
- QUIC: Message oriented (chapter 5.2.10)
- Unreliable, message-oriented protocols are useful for video streaming

Head of line blocking



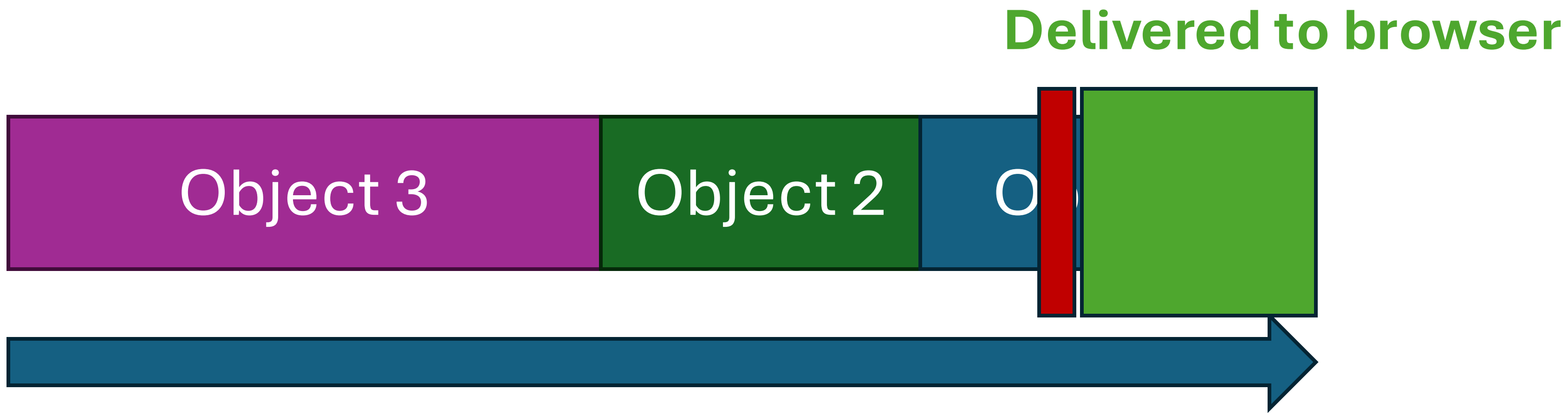
Let the above denote bytes received by a TCP receiver. It contains three objects. For example, web pages constitute a lot of different objects (code, images etc), so loading a single page often involves downloading 100s-1000s of distinct objects. Many objects are downloaded over a single TCP connection

Head of line blocking



Suppose one packet of object 1 is lost

Head-of-line blocking



Suppose one packet of object 1 is lost

TCP will deliver only the first part of object 1, even though objects 2 and 3 have already been received. The page would load faster if the browser could start processing them while we wait for the retransmitted packet

QUIC – a brief history

- This was enough of a problem that Google built QUIC, a new transport protocol
- Normally, developing new protocols for the internet is hard
- But Google is a large company. They control both the browser and their own servers, so they can do it
- However, while they control the operating systems (OS) of their own servers, they do not control the users’ OS
- TCP and other transport protocols are usually implemented in the OS and user-space applications like web browsers do not have permission to modify them. Another blocker was that they would have had to standardize a new protocol number in IP (although this is an easier problem to solve)
- Thus, they built QUIC on top of UDP. TCP encapsulates inside IP packets. QUIC encapsulates inside UDP packets which in turn is inside IP. This works just fine. OSes allow applications to send receive UDP packets.

QUIC – a brief history contd…

- Now QUIC is also standardized by IETF and everyone has their own implementations
- In addition to offering message orientation, they added a few more features:
  - Unlimited SACK ranges as we've already discussed
  - Different type of cryptography that is sometimes faster
  - Support for FEC (Forward Error Correction). FEC is done via Error Correcting Codes (ECC) and support for unreliable data delivery
  - A different congestion control algorithm, BBR (although this does not require a new protocol)

A case for unreliability

- In live video streaming (e.g. zoom) if a packet for a frame is lost, it is better to just drop the whole frame and work toward displaying the next frame than to incur the overhead, and latency, of retransmission

- FEC is useful here to minimize the effects of loss. FEC allows for low latency delivery since we need not wait for retransmissions

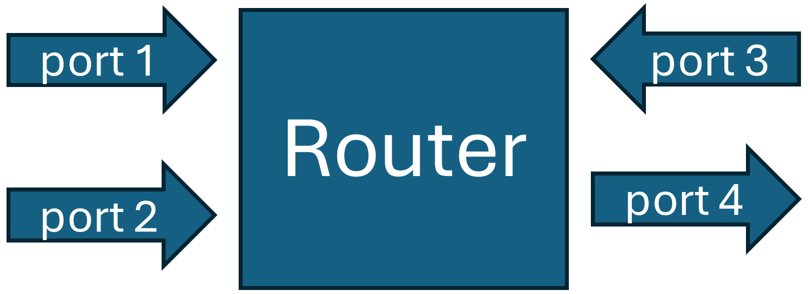
Note

The following content is only for fun and will not appear in exams.

The intention is to ensure that you are not left with the impression that there is only one “right” way to do things. Even the most sacred principles of internet design—the end-to-end principle and best effort delivery—can be violated when necessary

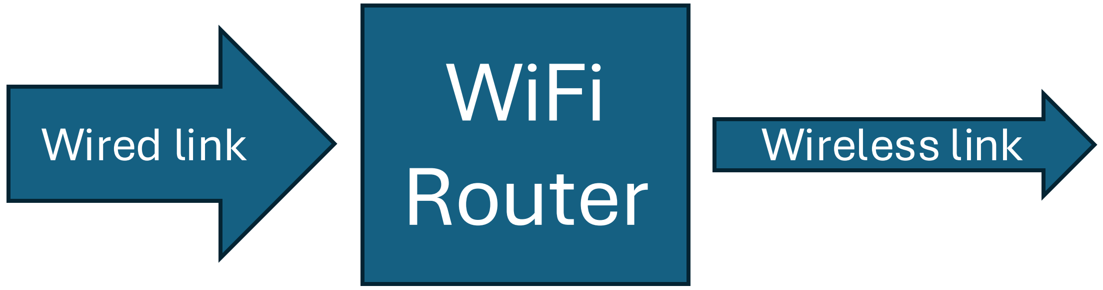
Like all engineering, network design is about balancing tradeoffs

Why packets get lost



- Physical noise
- Routers are getting packets faster than they can send them out. Eventually, they run out of memory space for their packet buffer, and are forced to drop the packet
  - In the picture above, the router is receiving traffic from all ports with a destination to port 4

Why packets get lost



Sometimes, the one port is a lot slower than the other. In this case, even a 2-port “router” can drop packets

An example is WiFi where the wireless link is much slower than the wired one

A second look at flow control

Flow control in TCP ensures that the receiver always has enough memory allocated to receive all packets sent by the sender

If the application does not consume packets at the same rate, it signals the sender to stop

Wait a minute…

A completely reliable network

Each link in the network implements flow control to ensure its neighbors never send more than what it can handle

(backpressure and deadlocks discussed on board)

This violates the end-to-end principle and forces all nodes to implement this complex protocol. Thus, it is not used on the internet

However, it is used by some datacenters because they own all the nodes. It is also used in networks inside chips (e.g. CPUs) since the hardware design is simpler if they assume packets never get lost

This is why Microsoft dared to use the go-back-N reliability protocol, since they built an (almost) loss free network