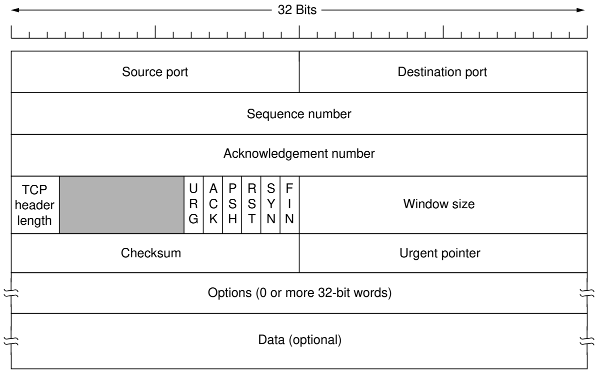
Transmission Control Protocol (TCP)

- Connection-oriented service that supports **byte streams** (not message streams). A sender may send eight 512-byte packets that are received as two chunks of 1024 bytes, and one of 2048 bytes.
- Transport address consists of a 16-bit **port number**, which augments the underlying IP address.
- TCP ensures reliable, point–to–point connections. No support for multicasting or broadcasting.
- A TCP TPDU is called a **segment**, consisting of (minimal) 20-byte header, and maximum total length of 65,535 bytes. A segment is fragmented by the network layer when it is larger than the network's **maximum transfer unit** (MTU).

TCP Header



- Acknowledgments are piggybacked when ACK = 1.
- SYN is for connection setup (ACK = 0: request; ACK = 1: accepted).
- FIN is for connection release. Data sent before the release is not lost.
- URG indicates immediate processing and transmission: the receiver is signalled.

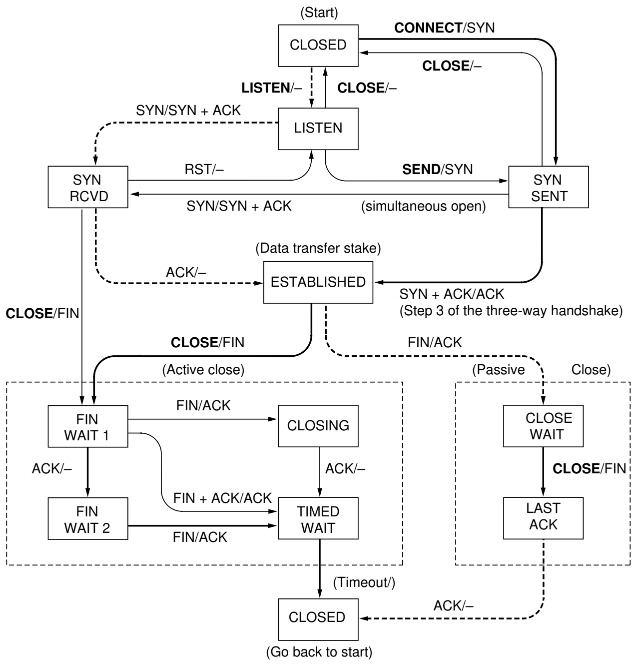
TCP Connection Management

Connection establishment: Uses three-way handshake protocol.

Connection release: To be thought of as independent releases of two simplex connections:

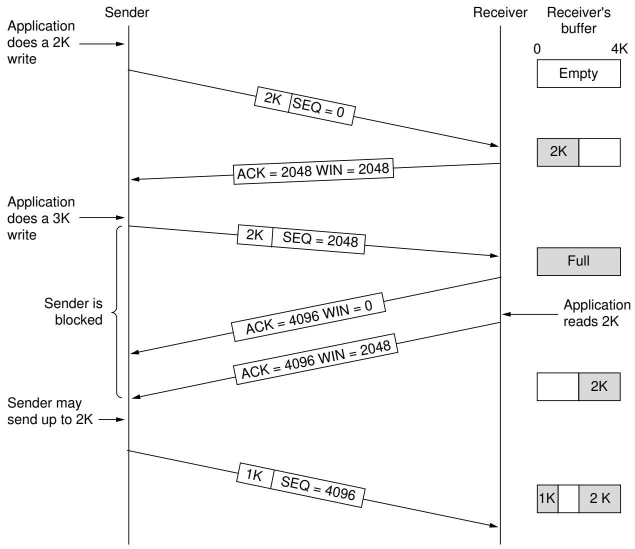
State	Description
CLOSED	No connection active or pending
LISTEN	Server waiting for conn. request
SYN RCVD	Conn. request has arrived; wait for ACK
SYN SENT	Conn. request just sent; wait for SYN+ACK
ESTABLISHED	Data can be sent and received
FIN WAIT 1	Client just sent conn. release
FIN WAIT 2	Server just agreed to release connection
TIMED WAIT	Wait for all packets to die
CLOSING	Client & server both tried to close
CLOSE WAIT	Other side initiated release
LAST ACK	Wait for all packets to die

TCP Connection Management – Release



TCP Window Management

Basic idea: The receiver sends an acknowledgment for the next byte that can be sent in the current stream, and the maximum number of bytes that may be sent.



TCP Window Management II

Important: The TCP entity is not obliged to immediately transmit data that the application hands over: it can do as much buffering as it likes. Same goes for acknowledgments.

Example: Interactive, character-oriented applications. Rather than sending one byte at a time, buffer as much characters as possible until the previous batch is acknowledged (Nagle's algorithm). **Note:** we're always stuck to at least 40 bytes of overhead per TPDU.

Example: Avoid the **silly window syndrome** where the server is reading one byte at a time (and acknowledges one at a time). Instead, the receiver should wait until it can receive a reasonable amount of bytes in a row.

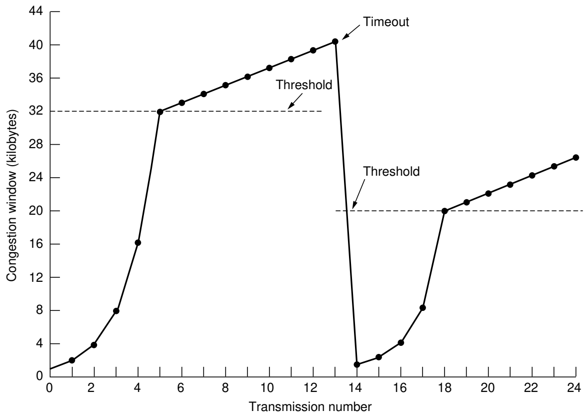
TCP Congestion Control

Problem: As before, the transport layer has to take into account that the underlying network can be the bottleneck. Question is how to detect and react to congestion.

Solution: use a **congestion window** next to the window granted by the receiver. The actual window size is the minimum of the two.

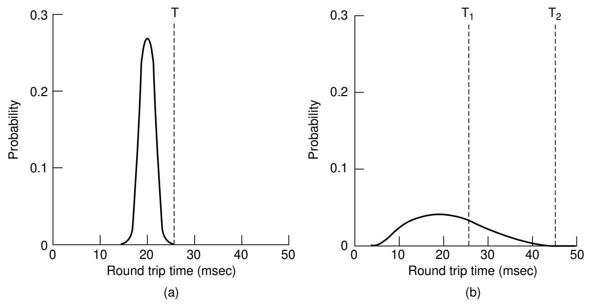
- Initialize congestion window to maximum segment size to be used in the connection. Send it off. If it gets acknowledged, double the size. Repeat until failure. Leads to initial congestion window size (**slow start**).
- In addition, use a **threshold**. On a timeout, lower the threshold to 50 % of the congestion window size, do a slow start (exponential) until new threshold, and *add* maximum segment size to congestion window size after that (linear growth).

TCP Congestion Control II



TCP Timer Management

Main issue: How do we determine the best timeout value for retransmitting segments in the face of a large standard deviation of round-trip delays:



RTT	best current estimate of round-trip delay
D	estimate of deviation of round-trip delays
M	measured round-trip delay

$$\begin{aligned} RTT &= \alpha RTT + (1-\alpha)M \\ D &= \alpha D + (1-\alpha)|RTT - M| \\ timeout &= RTT + 4 \cdot D \end{aligned}$$

ATM Transport Layer: AAL

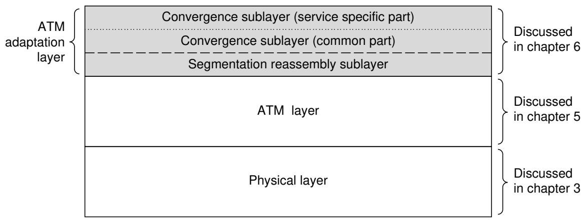
Basic idea: ATM provides just cell-based virtual circuits, which need enhancements to be really supportive for applications. There is no error control and no flow control. So what we need is:

	A		B	C				D
Timing	Real time	None	Real time	None	Real time	None	Real time	None
Bit rate	Constant		Variable		Constant		Variable	
Mode	Connection orientated				Connectionless			

Note: The initial four classes (A,B,C,D) have been abandoned ☹

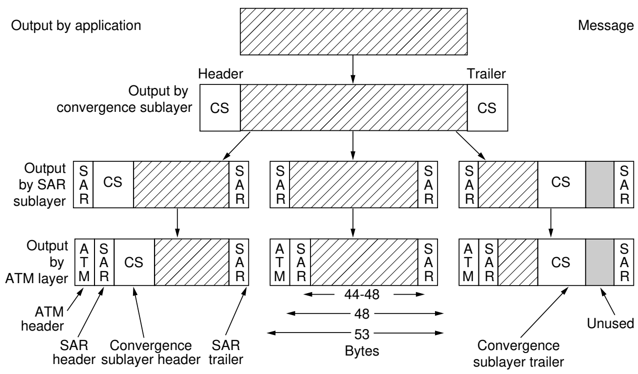
CBR	Constant Bit Rate
RT-VBR	Real Time Variable Bit Rate
NRT-VBR	Non-Real Time Variable Bit Rate
ABR	Available Bit Rate
UBR	Unspecified Bit Rate

ATM Adaptation Layers



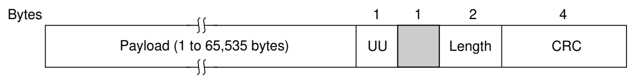
- **Convergence sublayer:** Make the message that is passed by the application suitable for transmission across the ATM network: (1) offer an application interface, and (2) process the message into something that can be fragmented into cells.
- **Segmentation And Reassembly sublayer:** Do the actual segmentation and reassembly, and construct suitable ATM cells.

ATM Adaptation Layers II



- Each class (A,B,C,D) got its own adaptation protocol. AAL 1 is probably not really necessary. AAL 2 has been left unfinished, and had a lot of problems. AAL 3 and AAL 4 have been merged (because they were so similar). Let's just call it a mess!
- AAL 5 has been invented by the computer people. It'll probably work just fine for data communication. It's also really simple.

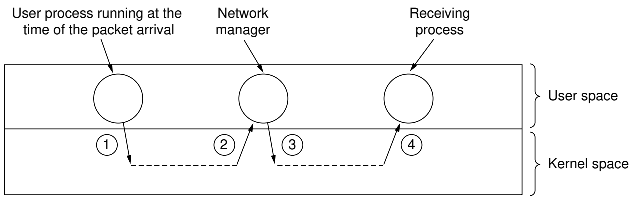
AAL 5



- Offers reliable unicast service (guaranteed delivery), unreliable service (unicast and multicast). Supports messages and streams.
- The trailer is added by the convergence layer; the result is passed to the SAR sublayer, which just breaks it into 48-byte pieces, no headers/trailers added. It sets a bit in the header of the last cell.

Making Computer Networks Fast

- *CPUs form the bottleneck.* In nearly all networks, the operating system and protocol overhead dominate performance figures.
- *Number of packets should be kept low.* The point to note is that there is a fixed overhead per byte, as well as per TPDU. Incoming packets also always generate an interrupt, which costs a lot.
- *Minimize context switches.* This is really bad with microkernels where a lot of the network software is in user space. Imagine what might happen when a packet arrives:

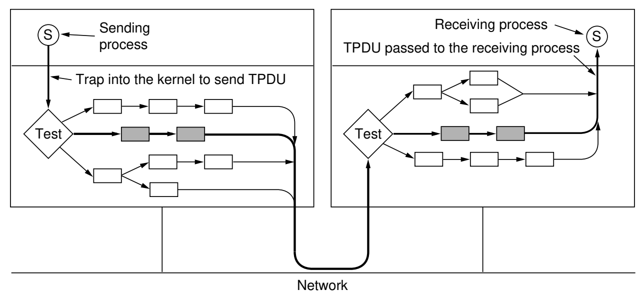


Making Computer Networks Fast II

- *Minimize copying.* This is perhaps the worst one: all this layering stuff means that (if we don't do it right) we'll be copying packets between layers. Copying can make it hard to get at 50 Mbps.
- *Concentrate on lowering delays.* Increasing bandwidth is never a problem: you just buy the hardware. Lowering latencies is a protocol and implementation issue. It's very hard.
- *Avoid congestion.* As always: repairing things may be more costly than avoiding it. Especially if repairs are often needed.
- *Avoid timeouts.* Because timeouts generally lead to repeating some action, the consumed bandwidth and processing time of the repeated action are to be considered wasted.

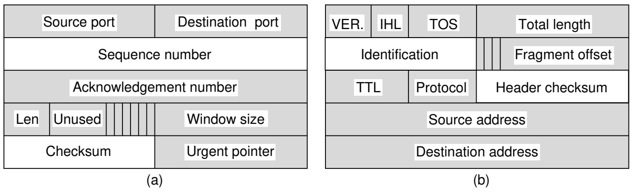
Fast TPDU Processing

Basic idea: Separate the normal case from the special ones, and guide the normal case along a **fast path** from sender to receiver. Separation generally requires a very simple test.



Fast TCP Processing

Example: Successive segments in a TCP connection generally have the same header. So, store a **prototype header** once in the output buffer, and just fill in the changed fields:



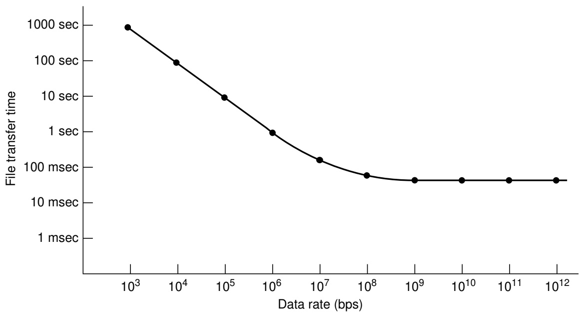
Note: we're trying to get performance by minimizing copying.

Gigabit Networks: The Problems

- **Sequence numbers.** The 32-bit numbers won't do: they'll wrap around far too soon. For example, at a data rate of 1 Gbps it takes 34 seconds to send 2^{32} bytes. TCP, with byte sequence numbering, is going to have a problem.
- **Processing power.** Packets of 4 KB may now come in at a speed of 30,000 per second, so each packet has to be processed in 33 µsec (assuming that the CPU has nothing else to do). For a 100-MIPS computer, this leaves us 3300 RISC instructions. Protocols will have to be made simpler.
- **Inappropriate protocols.** A lot of protocols use go-back-N error control. On a 4000-km line with a round-trip delay of 40 msec, this means that 5 Mbyte of data may be on its way when an error is reported at the sender.

Gigabit Networks: The Problems II

- **Delay versus bandwidth.** Transfer across gigabit lines is limited by delay, not bandwidth (the length of the line can hold the entire chunk of data):



Consequently, stop–and–wait protocols (like RPC) have an upperbound on their performance: we can't let them perform any better.

- **New applications.** As it turns out, not only speed, but the variance is equally important. Real-time stuff like multimedia requires stable, continuous transmission.

Gigabit Networks – Some Solutions

- **Avoid feedback.** We can't afford probing the network to see how well it is doing, nor can we inform the sender to slow down. Instead, sender and receiver have to make reservations beforehand. Connection-oriented communication will help here.
- **Simple headers.** To avoid excessive processing times, packets should have simple headers. Checksum the header separate from the payload to avoid unnecessary copying of the latter.
- **Maximize payload.** It should be possible to send very large chunks of data in a single packet to avoid the processing overhead, and to use bandwidth efficiently.
- **Focus on success.** Simply assume that, normally, everything goes as planned. Current protocols do their best to detect and correct errors. This costs too much for the general case when things go as they should.