Networking part 3: the transport layer

Juliusz Chroboczek
Université de Paris-Diderot (Paris 7)

September 2011

Summary of the previous episodes

Episode 1: switching, packet switching and the Internet.

Episode 2: the network layer: routing.

Episode 3: the transport layer: end-to-end communication.

Episode 1(1): circuit switching

Circuit switching:

Switching is what makes networking possible.

Episode 1(2): message switching

Message switching: telegraph.

- data is in the form of discrete messages;
- messages are forwarded over multiple hops;
- each message is routed independently.

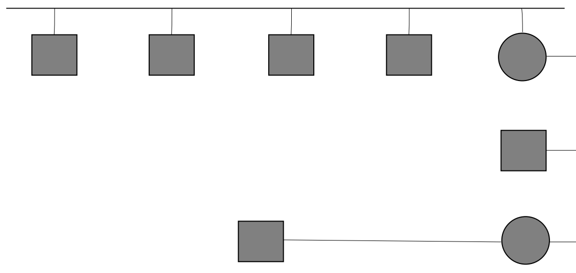
The different segments are never connected to make a physical circuit: virtual circuit.

Episode 1(3): packet switching

Packet switching: internet.

- data is segmented into bounded size packets;
- packets are forwarded over multiple hops;
- each message is routed independently.

Packet switching is what makes it possible to interconnect networks: an internet.



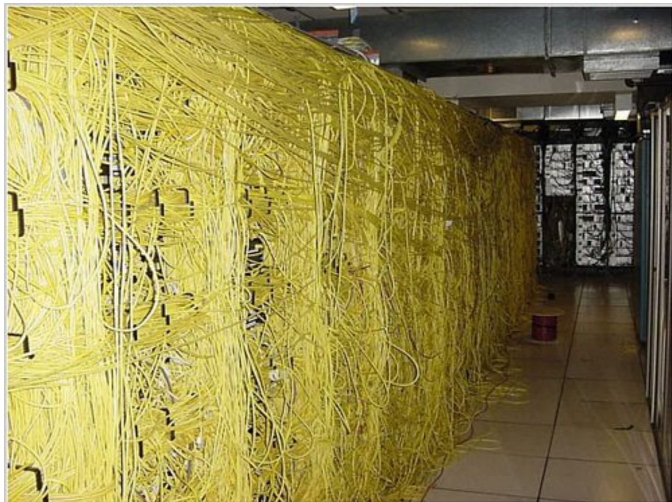
The largest internet is called The (Global) Internet.

Episode 2: routing

Routing is the process of deciding where packets go.

In the Internet, routing is hop-to-hop: every router makes an autonomous decision.

Episode 2: routing (2)



We really want routing to be automated.

Episode 2: routing (3)

We really want automated routing.

This is the role of a routing protocol.

An example was described in detail in Tsvaneti; we can now assume that we know how to route.

Layering

In Episode 2, we assumed that we know how to communicate on a single link.

In Episode 3, we assume that we know how to communicate across the Internet.

This is analogous to how:

- mathematicians assume that a lemma is correct;
- computer programmers assume that a library works.

In networking, this kind of modularity is called layering.

Layering (2)

Layering follows a strict structure: the simplified OSI model:

Application	(7)
Transport	(4)
Network	(3)
Link	(2)
Physical	(1)

Layer 2 is responsible for sending a packet over a single link.
Layer 3 is responsible for sending a packet over the Internet.
Layer 4 is responsible for internal multiplexing, sequencing (if desired), reliability (if desired) etc.
(Layers 5 and 6 don't exist any more.)

Layering (3)

Individual protocols fit in the OSI model:

NTP, DNS, FTP, SMTP, HTTP, ed2k, Bittorrent etc.	(7)
UDP, TCP	(4)
IP	(3)
SLIP, PPP, Ethernet, 802.11 (WiFi) etc.	(2)

- every protocol uses the service provided by a lower layer (only);
- the model has the structure of an hourglass;
- there is a convergence layer: there is only one protocol at layer 3.

The network layer

Service provided by the network layer:

- communication across the Internet;
- communication endpoints are hosts (interfaces);
- communication is packet-based;
- communication is unreliable;
- communication is unordered;
- communication is uncontrolled.

The network layer (2)

Service provided by the network layer:

- communication across the Internet
  routing is transparent to the higher layers;
- communication endpoints are hosts (interfaces)
  there is no finer structure;
- communication is packet-based
  the network retains packet boundaries;
- communication is unreliable
  the network is allowed to drop packets;
- communication is unordered
  the network is allowed to reorder packets;
- communication is uncontrolled.

This is not a useful service for the application layer.

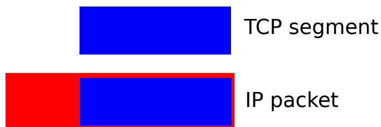
The transport layer: TCP

Service provided by the **TCP protocol**:

- communication across the Internet;
- communication endpoints are ports;
- communication is stream-based;
- communication is reliable;
- communication is ordered;
- communication is flow-controlled and congestion-controlled.

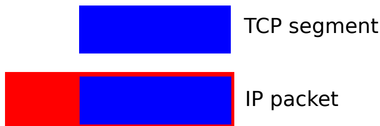
Encapsulation

A TCP segment is encapsulated in the IP packet:

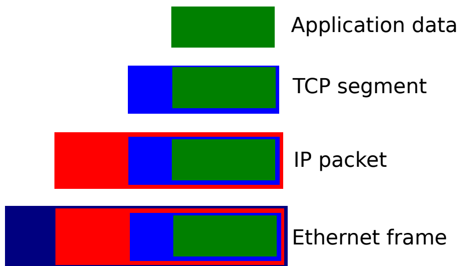


Encapsulation

A TCP segment is encapsulated in the IP packet:



Since the IP packet is itself encapsulated in an Ethernet frame, we have recursive encapsulation — one level per layer:



Ordering

The network can reorder the packets:

- because of the implementation of buffering;
- because of routing instabilities.

Ordering

The network can reorder the packets:

- because of the implementation of buffering;
- because of routing instabilities.

Solution: number the segments.
The receiver reorders back the received packets.

Ordering is performed by the endpoints, not the routers.

Digression: state

Computer programs maintain state. State causes bugs:

- state needs to be maintained;
- state needs to be preserved.

Programming guideline: minimize the amount of state.

Two kinds of state:

- hard state needs to be preserved;
- soft state can be recovered if it is lost.

Soft state is not as evil as hard state. (Not really state?)

The end-to-end principle

The end-to-end principle states that all (hard) state should be at the communication endpoints.

Equivalently, no (hard) state in routers.

In the OSI model, routers are pure Layer 3 devices (in principle).

This implies that most intelligence is at the endpoints. Consequences:

- new applications are easy to deploy;
- the network survives a router crash (fate sharing);
- routers are fast, cheap and reliable (pick two).

This is an important architectural principle of the Internet. This is the opposite of the telephone network.

Reliability

The network can drop packets:

- because of link-layer issues (radio links);
- because of buffers overflowing.

The network is unreliable.

Reliability

The network can drop packets:

- because of link-layer issues (radio links);
- because of buffers overflowing.

The network is unreliable.

What does it mean to have reliable communication?

Reliability (2)

Definition (wrong): communication is reliable when all sent data arrives to the destination.

This is obviously impossible to achieve when the lower layers are unreliable (unplugged network jack).

Reliability (2)

Definition (wrong): communication is reliable when all sent data arrives to the destination.

This is obviously impossible to achieve when the lower layers are unreliable (unplugged network jack).

Definition: communication is reliable when

- sent data arrives to the destination; or
- the network returns an error indication.

(Note that this implies that always returning an error indication provides reliable commmunication.)

Reliability (3)

Is it possible to require a stronger condition?

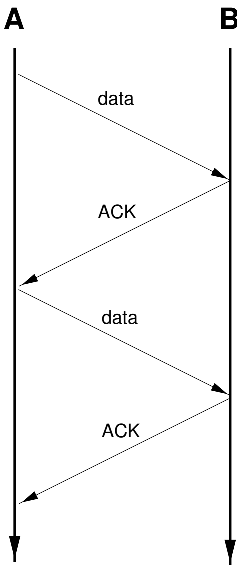
Condition: the network only returns an error indication when the sent data didn’t arrive.

Equivalently, sent data arrives or the network returns an error indication, but not both.

This condition is impossible to achieve.

Reliability (4)

Reliability is achieved by the receiver sending end-to-end acknowledgments to the sender.



Reliability (5)

Hop-to-hop acknowledgments don't work: what if a router crashes after sending an acknowledgment?

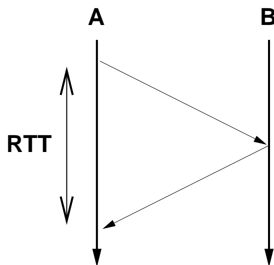
(Remember the end-to-end principle?)

Digression: throughput and latency

There are two measures of the “speed” of a network link: throughput and latency.

Throughput measures how much data you can push into the network. It is measured in bits per second (bit/s) or bytes per second (B/s).
Example: 1.5 Mbit/s.

Latency measures how long it takes for data to arrive to the other end. It is usually expressed as the Round-Trip Time (RTT, or ping time):



Pipelining

The “synchronous” protocol described above is extremely inefficient.

Suppose a Round Trip Time (RTT) of 30 ms and a Maxium Segment Size (MSS or MTU) of 1500 bytes.

Then this protocol’s maximum throughput is

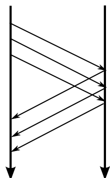
$$\frac{1500}{0.03} = 50\,\text{kB/s}$$

no matter how large the throughput of the link.

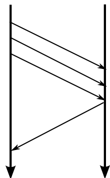
Solution: pipeline multiple packets before receiving the first acknowledgment.

Pipelining (2)

A pipelined protocol sends multiple pieces of data before receiving a single reply:



With pipelining, it is possible to have cumulative acknowledgments:



Unreliable communication: UDP

Reliable, ordered communication implies that packets are sent later:

- lost packets are resent later;
- lost packets delay subsequent ones.

This is not suitable for real-time communication:

- time distribution;
- real-time Internet games;
- voice over IP.

Unreliable communication: UDP (2)

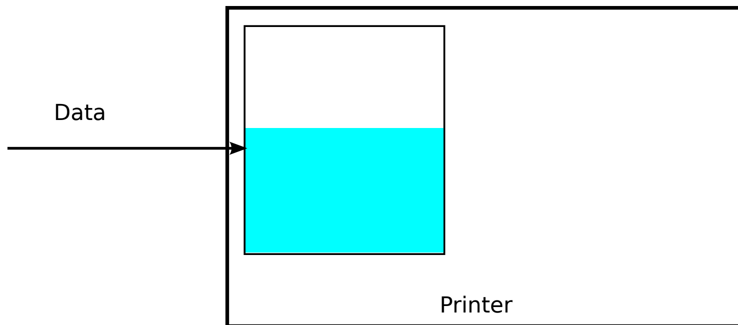
For real-time applications, we use UDP:

- communication across the Internet;
- communication endpoints are ports;
- communication is packet-based;
- communication is unreliable;
- communication is unordered;
- communication is uncontrolled.

Unlike TCP, UDP is a thin layer over IP.

Buffering

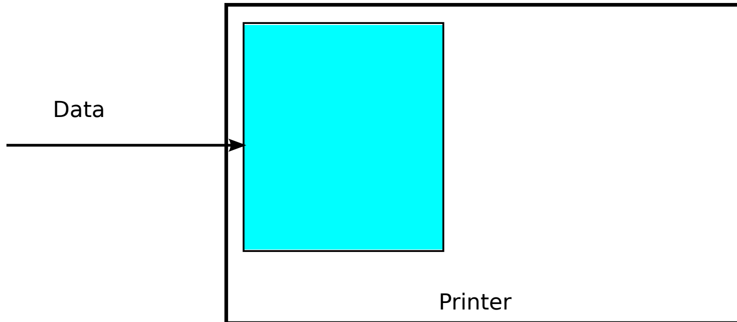
A buffer is an area of data that is used for holding data undergoing input/output.



Buffers make it possible for the sender to send data faster than the receiver can consume it: bursty traffic.

Buffer overflow

When the sender sends data too fast for the receiver, buffers overflow.



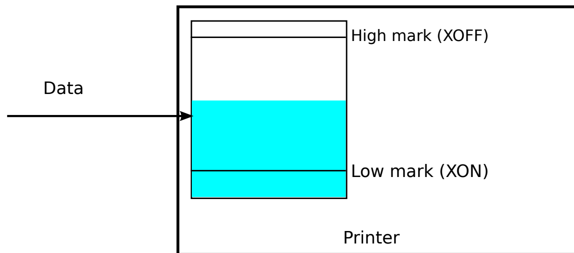
Avoiding buffer overflow in the receiver requires moderating the sending rate (slowing down): this is flow control.

Flow control: XON-XOFF

The simplest flow control technique is XON-XOFF flow control. (Not used in networking.)

In XON-XOFF flow control, the receiver sends two messages to the sender:

- XOFF: “my buffer is almost full, please stop sending data”;
- XON: “my buffer is almost empty, please send data again”.

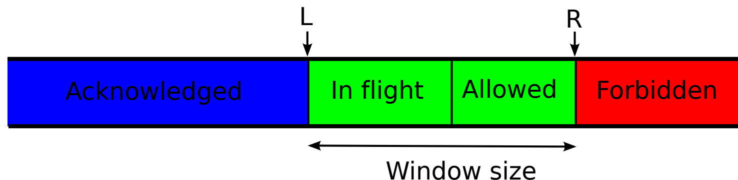


What if XOFF/XON is lost? Not suitable for networks.

Flow control: windowing

In windowing flow control, the sender maintains a window of sequence numbers that it is allowed to send.

- left edge L: the last acknowledged byte;
- right edge R: determined by the receiver.



The window size is $Rwin = R - L$.

Every ACK packet carries a window update that specifies the new value of the right edge.

What if a window update gets lost? It still works out.

Aside: a few values

Time:

- $1\,\mathrm{ns} = 10^{-9}\,\mathrm{s}$; $1\,\mathrm{ns} \cdot c \simeq 30\,\mathrm{cm}$;
- $1\,\mu\mathrm{s} = 10^{-6}\,\mathrm{s}$; $1\,\mu\mathrm{s} \cdot c \simeq 300\,\mathrm{m}$;
- $1\,\mathrm{ms} = 10^{-3}\,\mathrm{s}$; $1\,\mathrm{ms} \cdot c \simeq 300\,\mathrm{km}$;
- $100\,\mathrm{ms} = 0.1\,\mathrm{s}$: noticeable by humans.

Throughput:

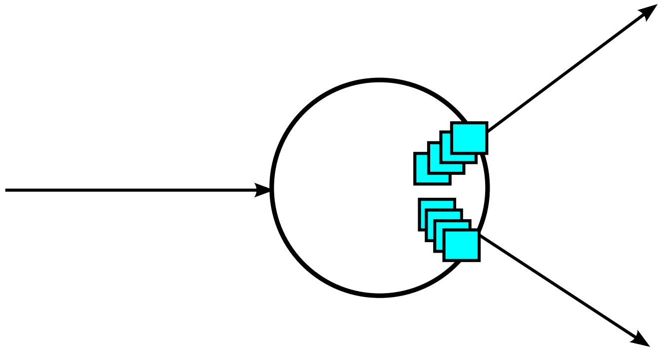
- $10\,\mathrm{kbit/s}$: a slow telephone modem;
- $1\,\mathrm{Mbit/s}$: a slow ADSL line;
- $1\,\mathrm{Gbit/s}$: a fast Ethernet;
- $1\,\mathrm{Tbit/s}$: the fastest networks in the world.

$$\frac{1\,\mathrm{Tbit/s}}{10\,\mathrm{kbit/s}} = 10^8.$$

Networking is probably the only engineering discipline where we need to deal with 8 orders of magnitude differences.

Buffering in routers

A router maintains a buffer of outgoing packets with each interface.

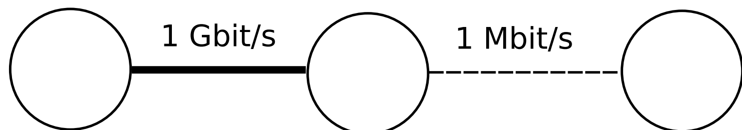


The buffer fills up whenever the outgoing link is too slow to handle all the incoming traffic.

Note: buffering before the routing decision causes head-of-line blocking.

Congestion

When a router's buffers fill up, we say that the outgoing link is congested.



In the presence of congestion, the router's buffers fill up and the router starts dropping packets.

Congestion control is about avoiding congestion inside the network. This is different from flow control, which is about avoiding congestion at the receiver.

Congestion collapse

Congestion causes dropped packets; dropped packets are resent, which in turn causes further congestion.

If nothing is done to avoid it, routers' buffers fill up with multiple copies of the same packets and no traffic goes through. This condition is called congestion collapse, and is stable.

Increasing buffer sizes does not solve the issue (it actually makes it worse).

In order to avoid congestion collapse, senders must apply congestion control: slow down in the presence of congestion. This requires:

1. detecting congestion;
2. reacting to congestion.

Signalling congestion: source quench

Idea: the router sends a "source quench" packet to request that the sender should slow down.
Problems:

- if source quench is lost, congestion will still occur;
- the more congested the network, the more likely packet loss becomes. Source Quench only works when it is not useful.

Source Quench is not used any more.

Congestion control: loss signalling

Idea: use packet loss as an indicator of congestion.

Sender slows down whenever it detects that a packet has been lost.

Advantage: a loss event cannot be lost.

Disadvantages:

- congestion is detected late, after a packet has been lost;
- lost packets must be resent, which increases latency and jitter (latency variation);
- non-congestion-related packet loss causes spurious reductions in throughput.

Congestion control: congestion window

We want to reduce the sending rate whenever we detect a loss event. This is done using the congestion window $Cwin$, maintained by the sender.
The window effectively used is

$$Ewin = \min(Rwin, Cwin).$$

The congestion window obeys Additive Increase Multiplicative Decrease (AIMD):

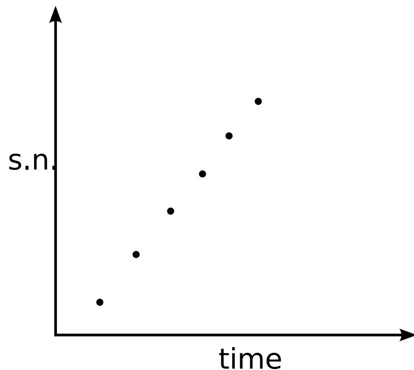
- on every acknowledgment, $Cwin := Cwin + \mathrm{MSS}$;
- on every loss event, $Cwin := Cwin/2$.

This is (usually) stable: after convergence, $Cwin$ oscillates between $B/2$ and $B + MSS$, where B is the buffer size of the bottleneck router.

Time-sequence graph

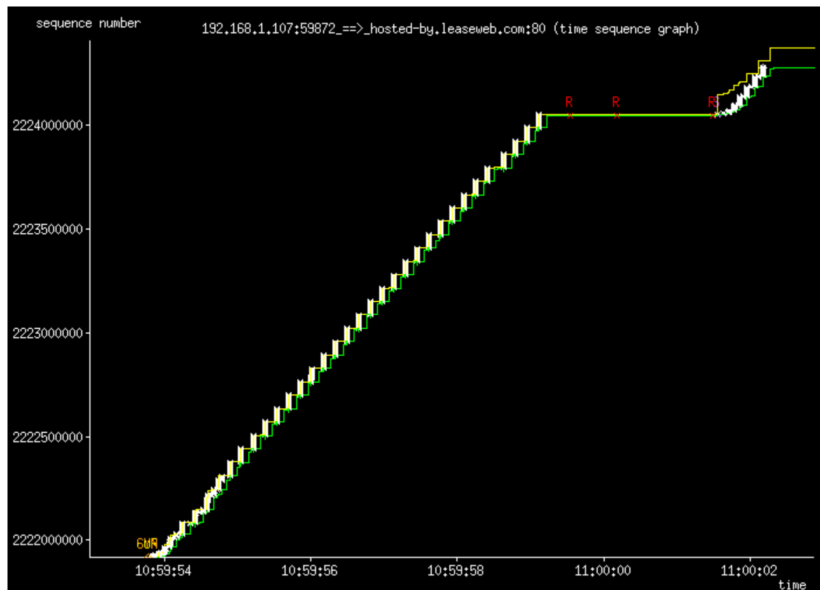
A time-sequence graph is a graph on which:

- every dot represents a sent packet;
- the x coordinate represents time;
- the y coordinate represents sequence numbers.



The slope of the resulting curve is the throughput.

Time-sequence graph (2)



Current work: lossless congestion control

Current congestion control relies on packet loss; this makes packet loss a common occurrence in normal operation.

While this does not impact throughput much, the lost packets must be resent, which causes latency and jitter (irregular latency), which is undesirable for many applications.

Lossless congestion control using explicit congestion notification techniques are no longer experimental, and are slowly being deployed on the Internet (cf. ECN).

Further research: fighting buffer bloat

Router buffers are a necessary evil: they absorb bursts of traffic, but while doing so they increase latency.

How do we reduce buffer size without impacting throughput (which is commercially important)?

Buffer bloat is an area of (currently fashionable) active research.

Conclusions

Congestion control is difficult, and there is a lot of unanswered questions:

- how do we distinguish congestion-related and unrelated packet loss?
- how useful is explicit congestion notification?
- what about delay-based congestion control?
- is AIMD the best we can do?
- what queueing strategies are best for routers?
- how do we fight buffer bloat?