Communication Channels

February 22, 2013

Roadmap

Message-based communication

Properties of a Communication Channel

Internet Protocols
- UDP
- TCP

Roadmap

Message-based communication

Properties of a Communication Channel

Internet Protocols
- UDP
- TCP

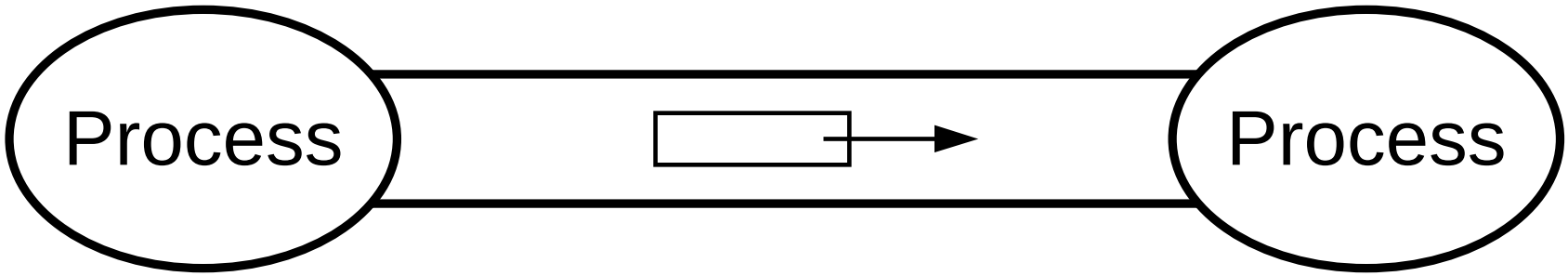
Distributed Application

What is?

Is an application with 2 or more processes that execute on different processors that do not share memory

A corollary of this definition is:

In a distributed application, processes communicate with each other by exchanging messages

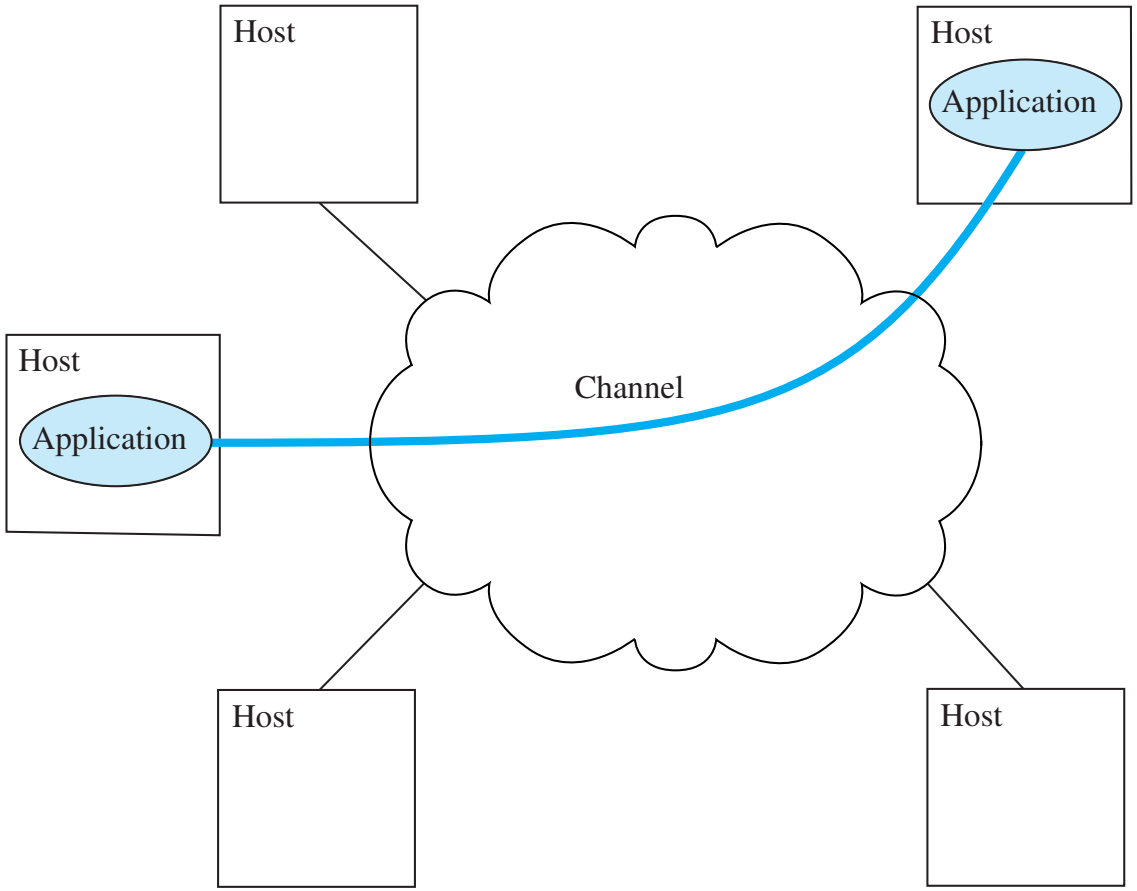


What is a message? is an **atomic** bit string

- Its format and its meaning are specified by a communications protocol

Message-based Communication and Networking

- The transport of a message from its source to its destination is performed by a computer network.



- The network can be abstracted as a communication channel
  - What are the properties of such a channel?

Distributed Computing Fallacies (Sun People)

1. The network is reliable.
2. Latency is zero.
3. Bandwidth is infinite.
4. The network is secure.
5. The topology does not change.
6. There is one administrator.
7. Transport cost is zero.
8. The network is homogeneous.

Some assumptions proved wrong.

Roadmap

Message-based communication

Properties of a Communication Channel

Internet Protocols

- UDP
- TCP

(Some) Properties of a Communication Channel

- Connection-based vs. connectionless
- Reliable vs. unreliable (may loose messages)
- Ensures order (or not)
- Message-based vs. stream-based
- With or without flow control
- Number of ends of the channel

Connection-based *vs.* Connectionless

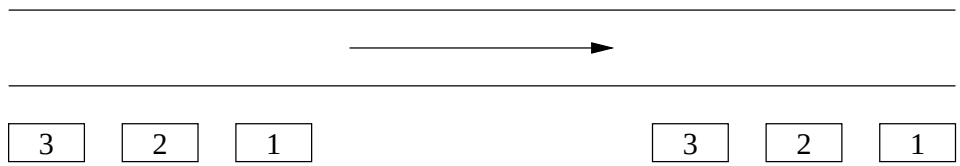
Connection-based: the processes must setup the channel before exchanging data – analogous to the telephone network;

Connectionless: the processes need not set up the channel, can exchange data immediately – analogous to mail

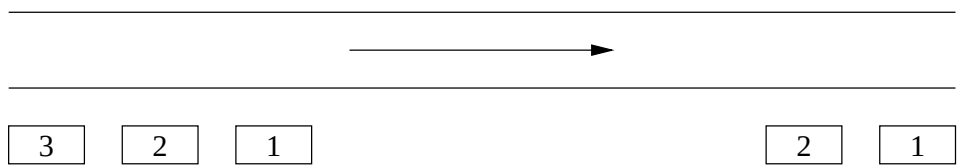
Reliability (loss of packets)

Reliable: ensure that the data sent is delivered to the respective destination

- under some assumptions;
- if not, the communicating processes are notified

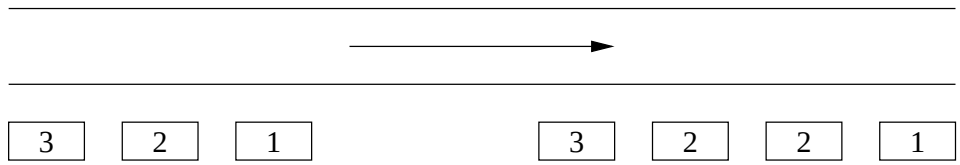


Unreliable: it is up to the communicating processes to detect the loss of messages and proceed as required by the application

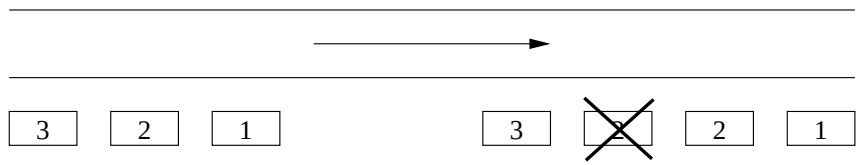


Reliability (duplication)

"Generates" duplicates: the channel may deliver duplicated messages to the destination – it is up to the recipients to detect the duplicates:

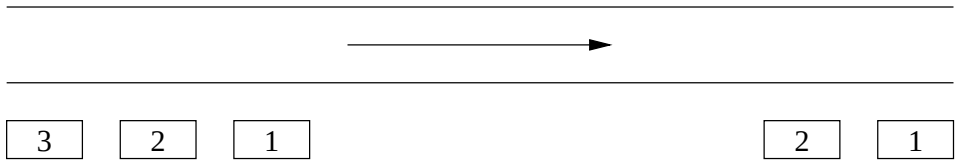


No duplicates: the channel ensures that it delivers each message to its recipients at most once:

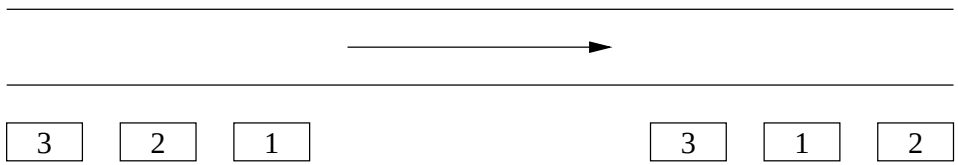


Order

Ordered: ensures that the data is delivered to its recipients in the order in which it was sent:



Unordered:



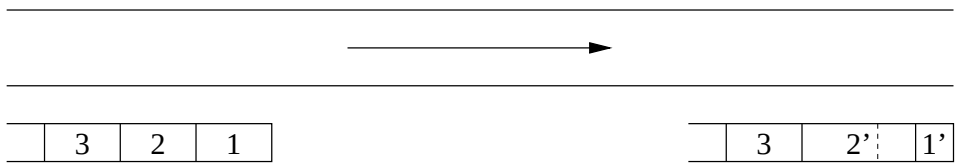
If it is important to preserve the order, it is up to the application to detect that the data is out of order and if necessary to reorder it

IMP order and reliability are orthogonal

Comunication Abstraction

Message (datagram): the channel supports the transport of messages – sequences of bits processed atomically – anlogous to mail

Stream the channel does not support messages. Essentially, it works as a pipe for a sequence of bytes – analogous to Unix pipes



Other Properties of a Communication Channel

Flow control: prevents "fast" senders from overflowing with data "slow" recipients

- it does not necessarily mean that the sender has more computing power than the receiver

Number of ends of the communication channel

unicast, or point-to-point only two ends;
broadcast all nodes in the "network";
multicast subset of nodes in the network

Identification of the communicating processes

- "Name" of the process itself
- "Name" of the channel endpoint, e.g. phone number

(Process Identification on the Internet (IPv4))

Question How do identify a process on the Internet?

Answer By a pair of identifiers:

IP Address :

- Identifies the computer on the Internet;
- It is a 32-bit, usually represented in *dotted decimal*, e.g.: `193.136.28.31`.

Port Number :

- Identifies the endpoint of a communication channel inside a computer (transport layer identifier)
- It is a 16-bit integer (between 0 and 65535);
- By convention, some ports are reserved for some services.

Clarification The (*IPAddress*, *PortNumber*) pair actually identifies the endpoint of the communication, the process may change

This is similar to a residence phone number: the person that answers a call may change.

Roadmap

Message-based communication

Properties of a Communication Channel

Internet Protocols
- UDP
- TCP

Internet Protocols

Aplication	Specific communication services
Transport	Communication between 2 (or more) processes.
Network	Communication between 2 computers not directly connected with each other.
Interface	Communication between 2 computers directly connected.

- On the Internet, the properties of the communication channel provided to an application depend on the transport protocol used (UDP or TCP):
  - The design of a distributed application depends on the properties provided by the chosen transport protocol

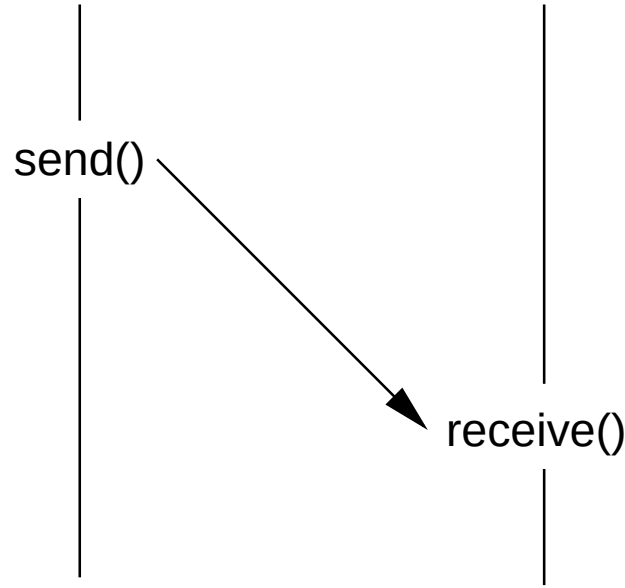
Summary of the Properties of the Internet Transport Protocols

Property	UDP	TCP
Abstraction	Message	Stream
Connection-based	N	Y
Reliability (loss & duplication)	N	Y
Order	N	Y
Flow control	N	Y
Number of recipients	n	1

- The abstraction provided by TCP stems from the API, or is it intrinsic to the protocol?

UDP: Observation (1/3)

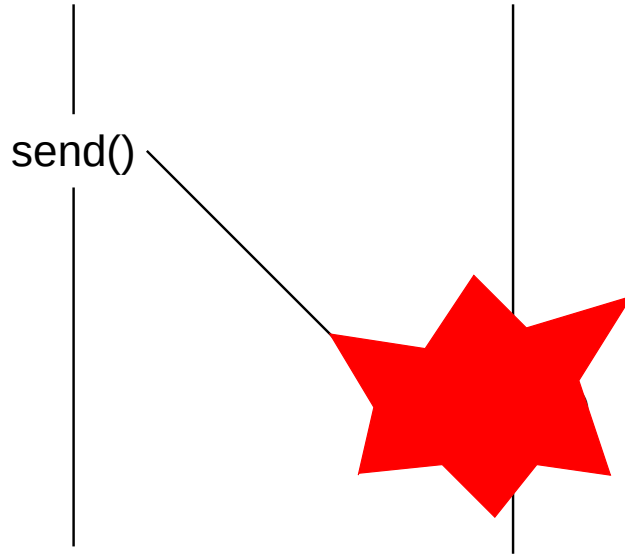
- UDP channels transport messages – UDP datagram:
  - Basically, its API supports two operations: *send()* e *receive()*;
  - Each message is transmitted by invoking `send()` once;
  - If delivered, the message will be delievered atomically, in a single invocation of `receive()`



- *Datagrams* have a maximum size of 65535 bytes:
  - Applications may have to **split** the data to send in datagrams before transmitting, and **reassemble** the data from the fragments after receiving

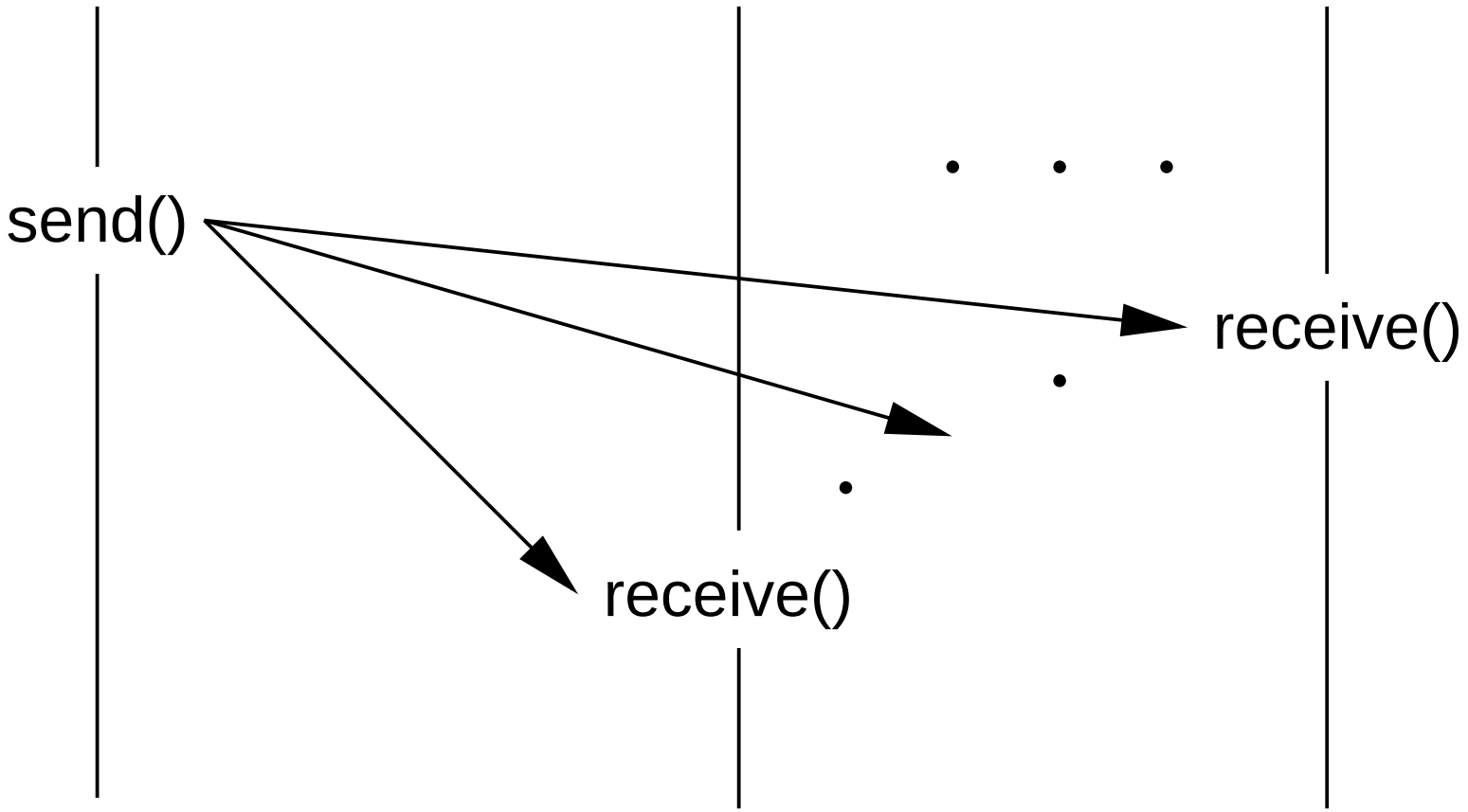
UDP: Observations(2/3)

- UDP being connectionless:
  - \+ allows a process to start transmitting data immediately;
  - − requires the specification of the the other channel endpoint on every invocation of *send()*.
- UDP provides no reliabity guarantee
  - UDP datagrams may be lost or even duplicated;
  - If the application cannot tolerate the loss, or the duplication, of datagrams, it will have to detect and recover from such an event



UDP: Observations(3/3)

- UDP has no flow control
  - A receiver may be flooded with requests and run out of resources (e.g. buffers) to receive other messages.
- UDP supports *multicast*, by invoking `send()` once, it is possible to send a copy of a given message to several processes.



TCP: Observations (1/3)

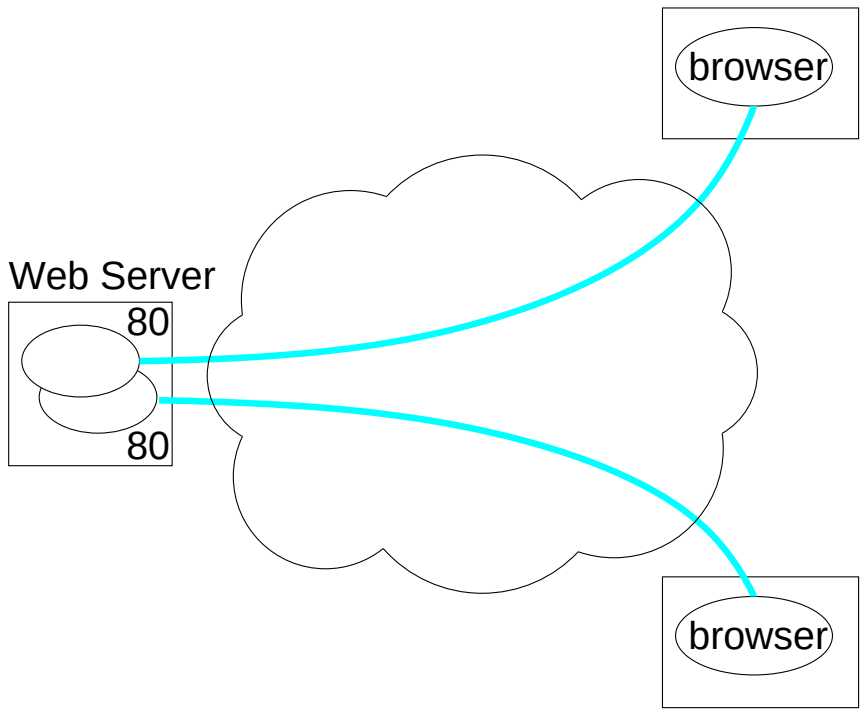
- TCP channels are **stream** channels, i.e.:
  - They are similar to Unix *pipes* on Unix, except:
    - They can be used for communication between processes in different computers;
    - They are bidirectional channels – i.e. it is possible to send data in both directions simultaneously
- Although we can also use `send()` and `receive()` to exchange data:
  - TCP does not ensure the "separation" between *bytes* sent by invoking two `send()` calls;
  - `write()` and `read()` are more intuitive.
- If it is essential to preserve message boundaries, it is up to the application to implement it. How?

TCP: Observations(2/3)

- TCP is connection-oriented. Communication with TCP has 3 phases:
  1. Connection set up
  2. Data exchange
  3. Connection tear down
- TCP ensures reliability (both loss and duplication of data):
  - In the event of communication problems, the connection may be aborted and the processes on its ends notified
- TCP ensures flow control
  - Prevents data loss because of insufficient resources
  - Nevertheless, TCP may be vultnerable to *denial-of-service* attacks, i.e. *SYN attacks*.

TCP: Observations(3/3)

- TCP channels have only two endpoints, supporting the communication between only two processes
- Unlike what happens with UDP, TCP channels on the same computer may have the same port number:
  - A TCP channel is identified by the pairs (IP Address, TCP Port) of its two endpoints;
  - This allows the concurrent service of several clients in client-server applications, for example on the Web:



TCP vs. UDP

Why not always use TCP?

- It provides "more" than UDP

Can you pay the cost?

- Connection must be set up before data exchange
- Recovery of a lost segment affects those that follow it
  - TCP ensures in-order delivery

Some applications cannot E.g. Internet telephony

- It is very sensitive to delays
- But can tolerate some loss

TCP provides a service that it does not need (recovery of lost data), at a cost that may be too high

Distributed Computing Fallacies (Sun People)

1. The network is reliable.
2. **Latency is zero.**
3. **Bandwidth is infinite.**
4. The network is secure.
5. The topology does not change.
6. There is one administrator.
7. Transport cost is zero.
8. The network is homogeneous.

Multimedia Applications

Classes

Streaming Stored Audio and Video e.g. YouTube

- Streaming means that the contents is played before completing the reception of the entire contents

Streaming Live Audio and Video Internet radio/television

- Contents is generated as it is being sent

Real-time Interactive Audio and Video e.g. Skype

- Two-way communication

Requirements

Bandwith do not tolerate large variations

Packet delay and also its jitter (i.e. its variation)

Packet loss not that stringent

Internet Protocols and Multimedia Applications

- But the Internet is designed on the **best-effort** principle
  - It does not provide any guarantees, especially regarding packet-loss rate, bandwidth, packet delay and its jitter
- "Tricks people play"

  Bandwidth Use compression

  Delay
    - For non-RT applications we can use buffering
    - For RT applications use **also** forward-error-correction (FEC)
      - Compression standards (e.g. MPEG) often support FEC

  Delay jitter Engineer a delay (may be fixed or adaptive)

  Packet loss
    - Streaming apps can use plain TCP, as long as the buffers are large enough
    - Interactive RT apps:
      - Use forward-error-correction
      - Rely on the end-to-end argument (this is explicitly referred in the article)

End-to-End Argument (around 1980)

- This is a design principle for layered systems, and states:

  > If you have to implemente a function **end-to-end**
  > don't implement it on the lower layers
  > unless there is a compelling performance enhancement

  Saltzer, Reed and Clark, "End-to-End Arguments in System Design"
- The main examples in the paper are drawn from data communication systems, but the authors also give other examples

  > "Therefore, providing that questioned function as a feature of the communication system itself is not possible. (Sometimes an incomplete version of the function provided by the communication system may be useful as a performance enhancement.)"
- Why is this relevant for distributed applications?
  - Distributed applications are often layered
  - On the Internet, you have to choose between TCP and UDP
- This is a design principle, not a physics law

Dave Andersen's (?) Algorithm

Do you need everything TCP provides?

- If yes, you are done.

If not: can you pay the cost?

- If yes, you are done

If not Use UDP

- Implement what you need on top of UDP

What about other protocols? There are some

- **But** they are not standard
  - If you are targeting the consumer market and you are not Microsoft or Apple (why?) you have no chance ... may be Google