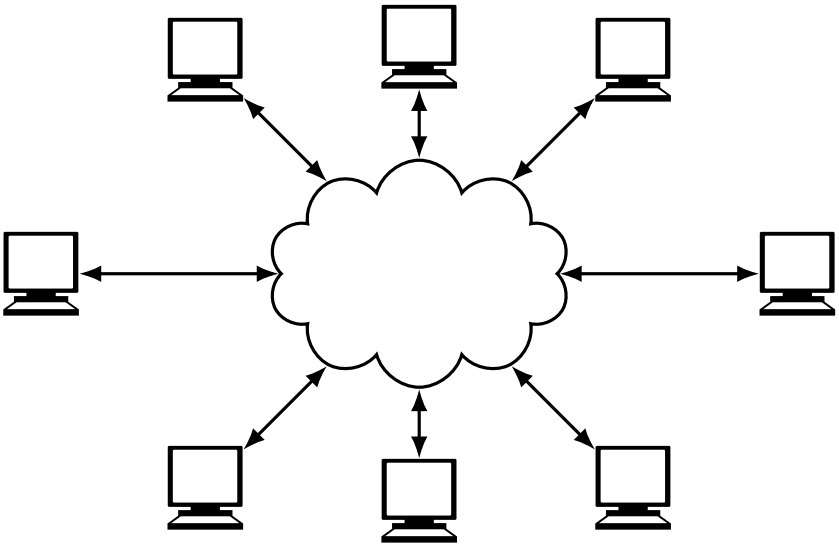
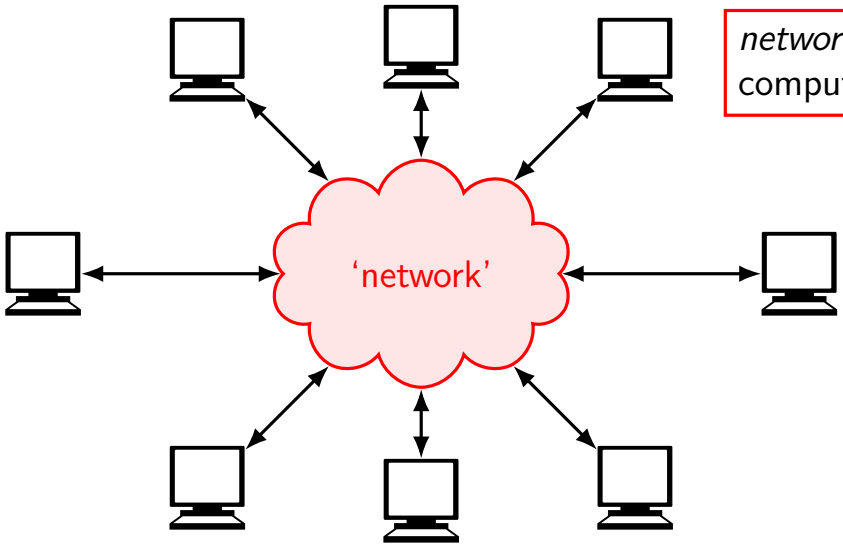
networks / hosts aka end systems

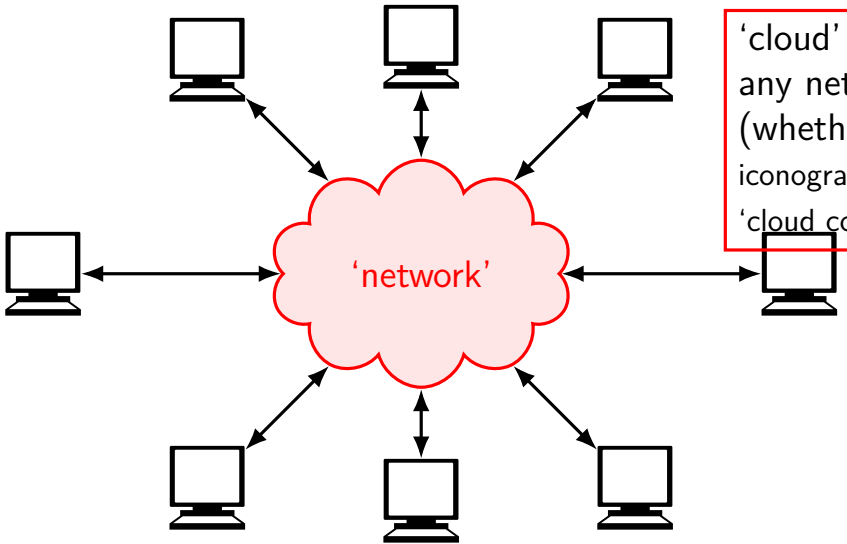


networks / hosts aka end systems



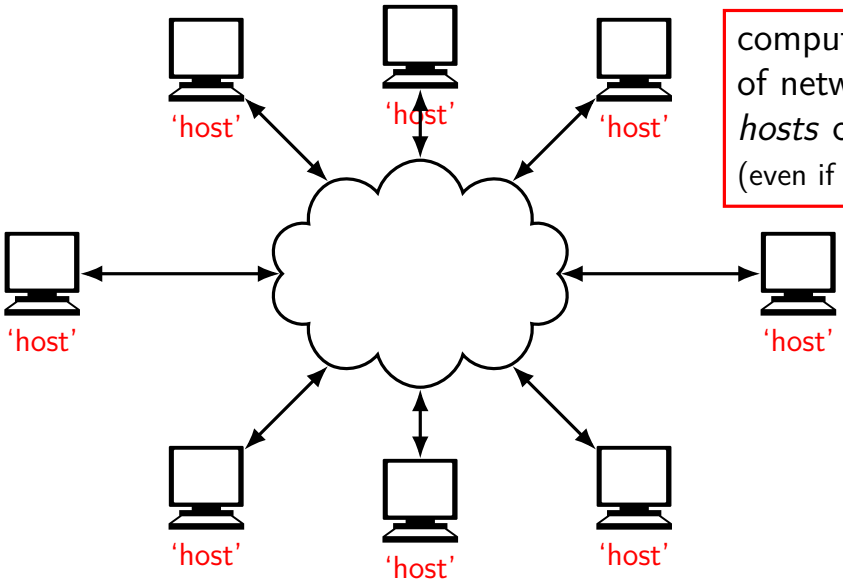
networks connect computers

networks / hosts aka end systems



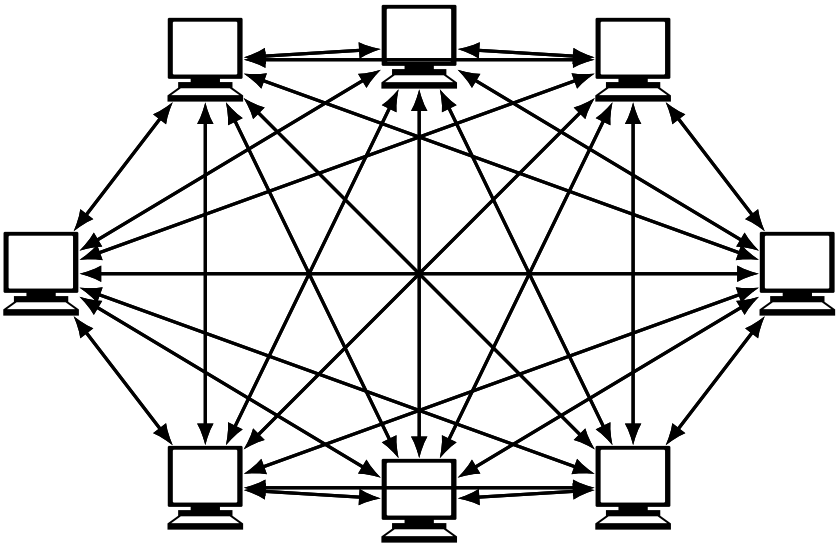
'cloud' represents
any network
(whether local or not)
iconography predates
'cloud computing'

networks / hosts aka end systems

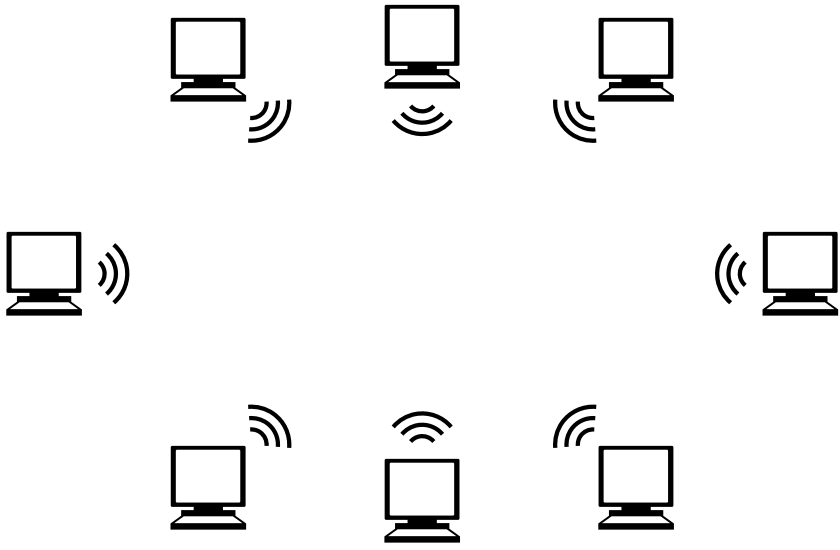


computers on edge of network called *hosts* or *end systems* (even if also 'servers')

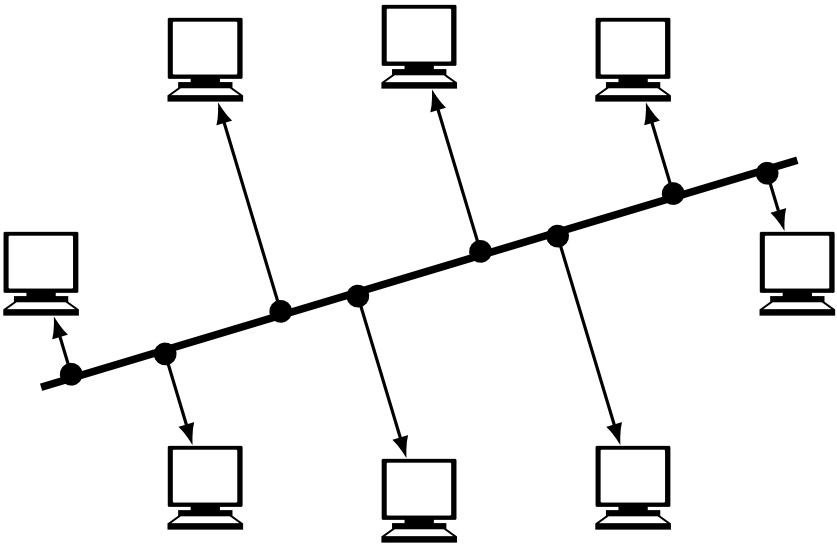
direct connections?



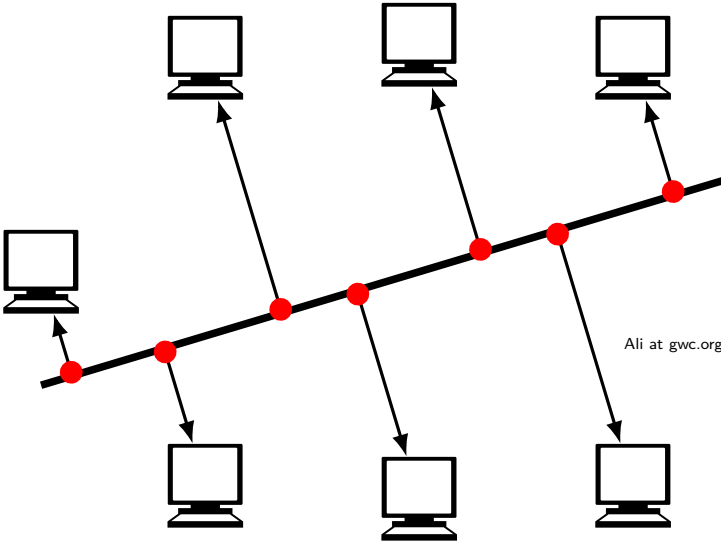
shared medium: radio?

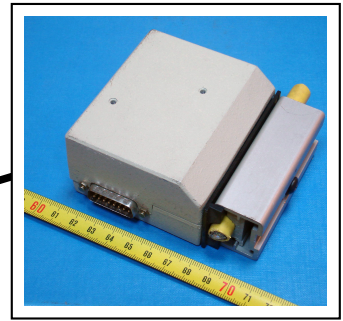


shared medium: wires



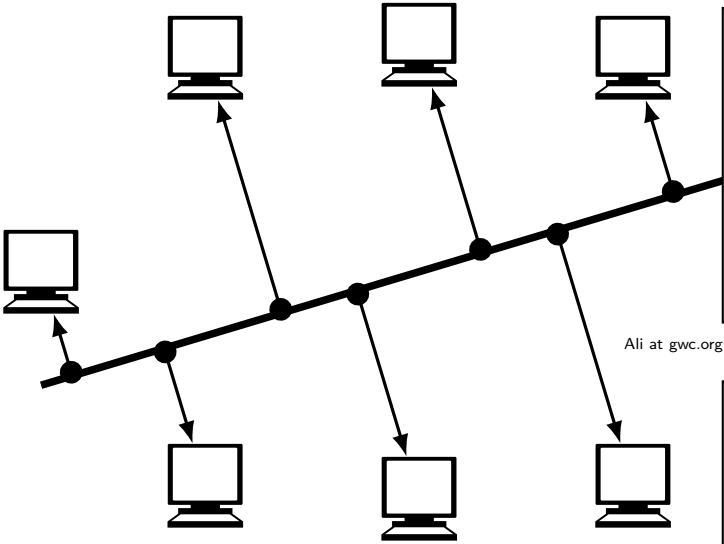
shared medium: wires

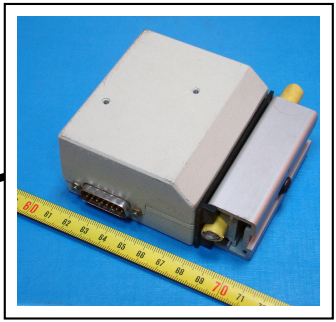




Ali at gwc.org.uk / Alistair1978 via Wikimedia commons / CC-BY-SA 2.5

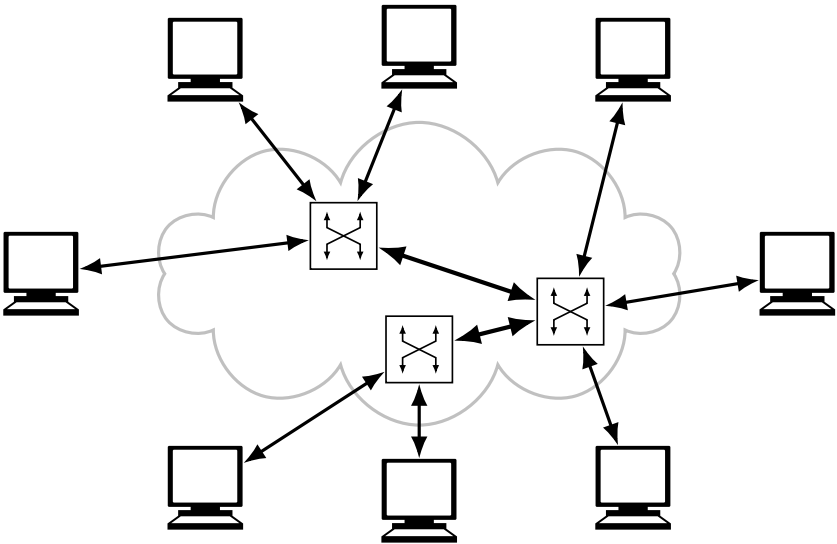
shared medium: wires



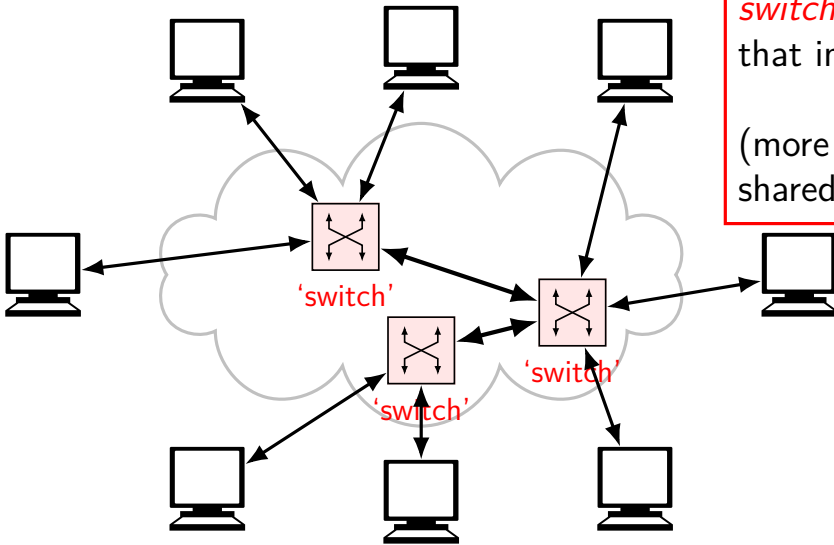


Ali at gwc.org.uk / Alistair1978 via Wikimedia commons / CC-BY-SA 2.5

switches / nodes / links



switches / nodes / links

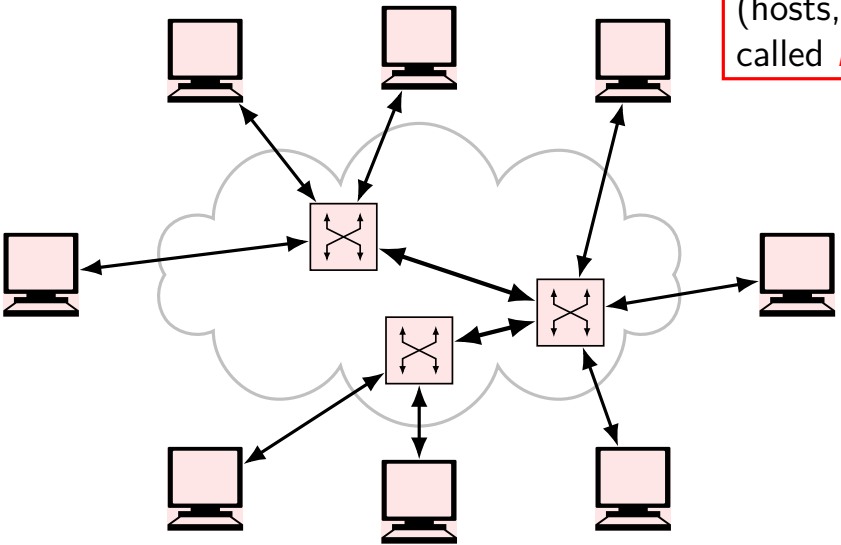


hosts directly connected to *switches*
that implement network

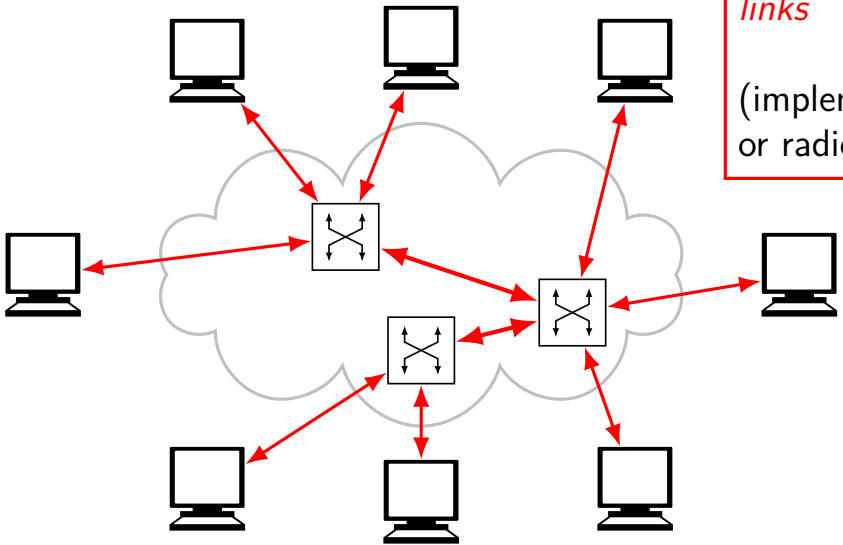
(more efficiently than shared medium)

switches / nodes / links

machines on network (hosts, switches, ...) called *nodes*



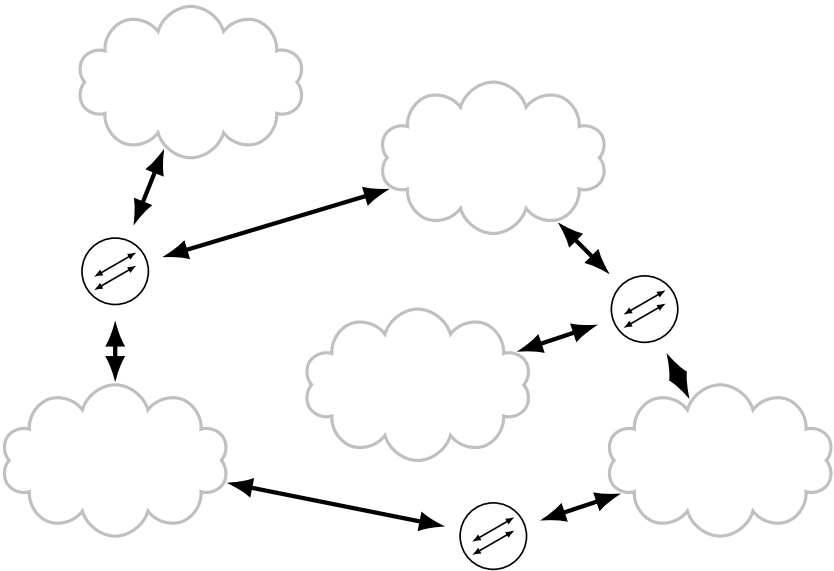
switches / nodes / links



nodes connected by *links*

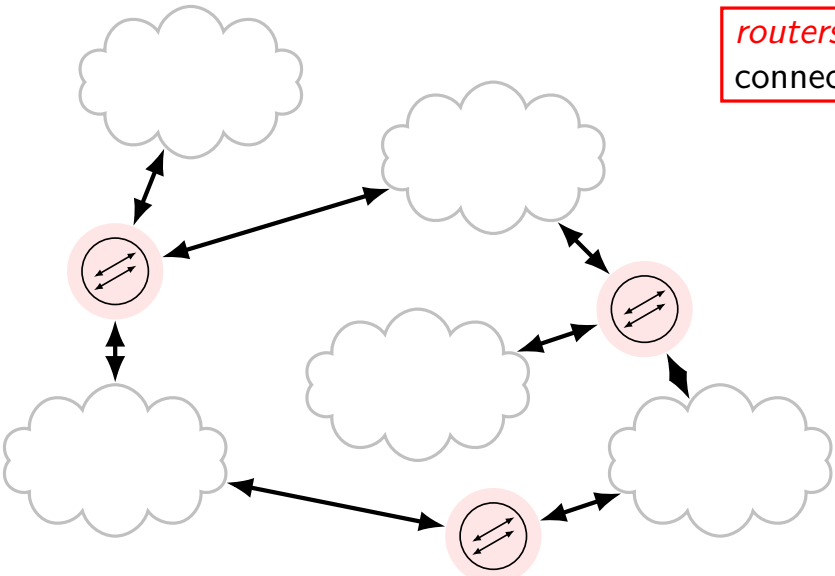
(implemented by wires or radio or ...)

routers / internetwork

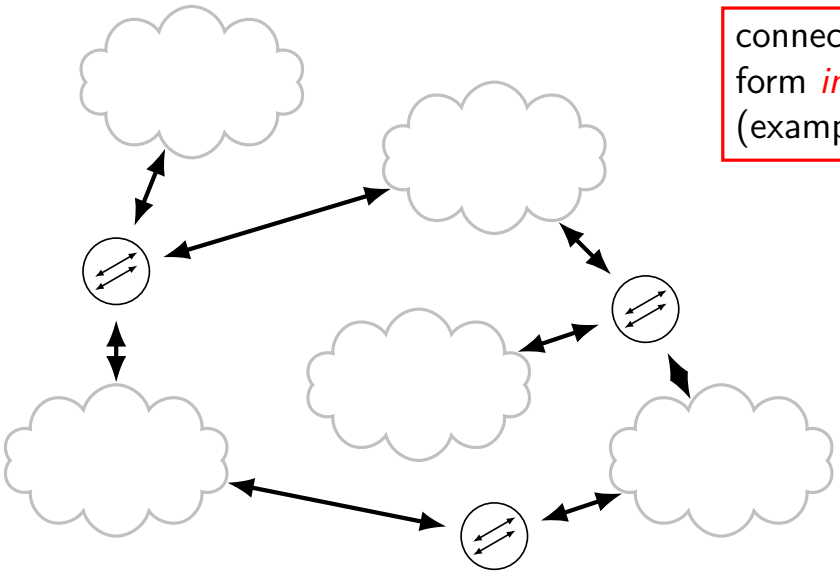


routers / internetwork

routers or *gateways* connect networks

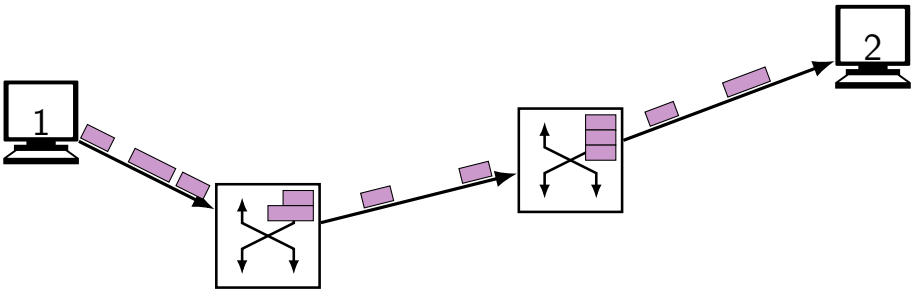


routers / internetwork



connected networks
form *internetwork*
(example: the Internet)

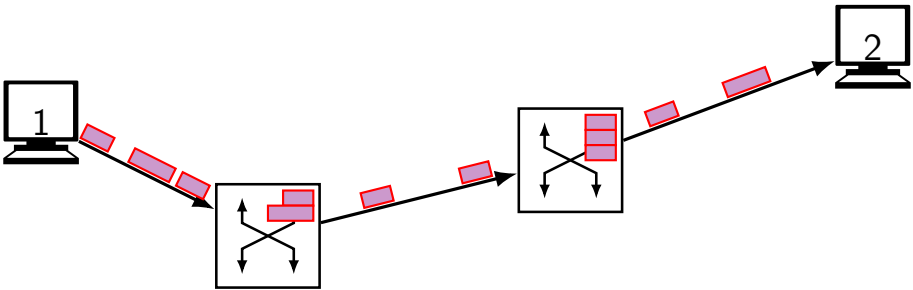
flows / packets



flow of data between two machines

flow is very general term
will depend on context how it relates to connections, sockets, etc.

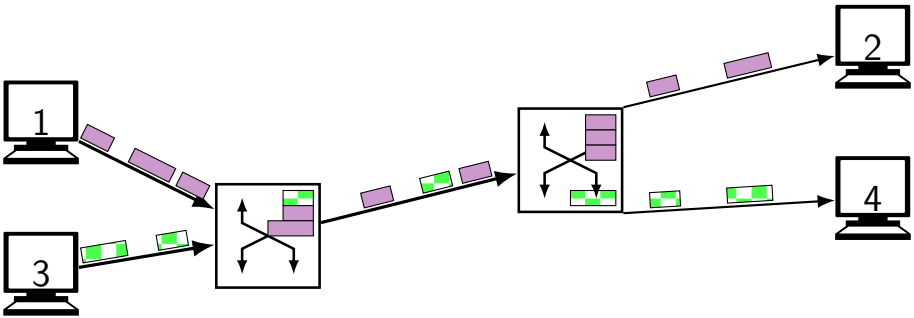
flows / packets



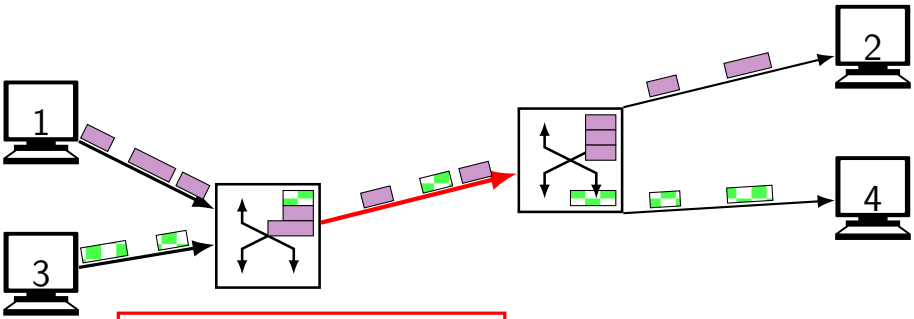
flow of data between two machines

possibly divided up into pieces,
called *packets*, *frames*, *segments*
(which name is best depends on context)

(de)multiplexing

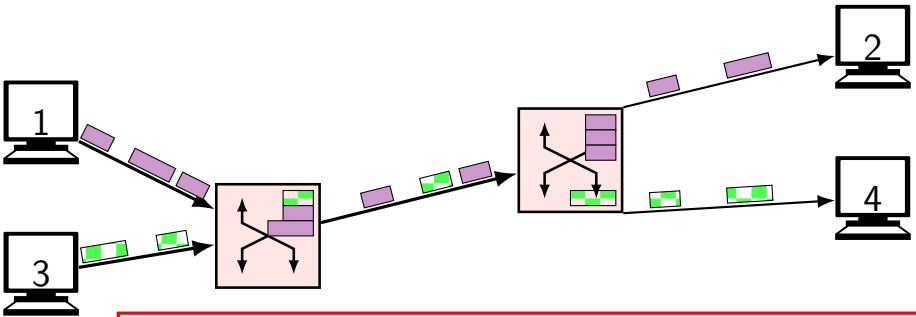


(de)multiplexing



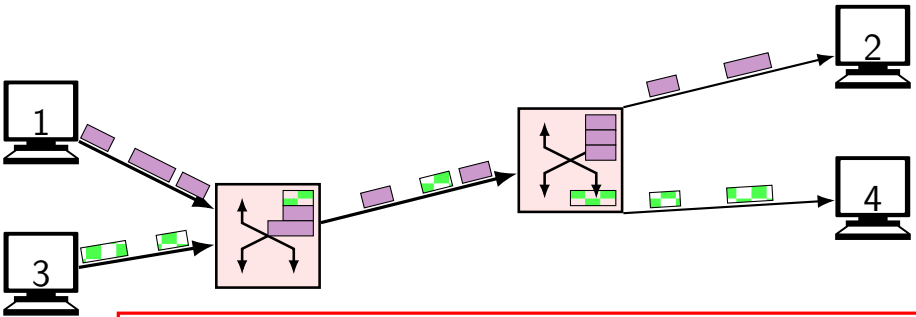
two or more flows can
share one or more links

(de)multiplexing



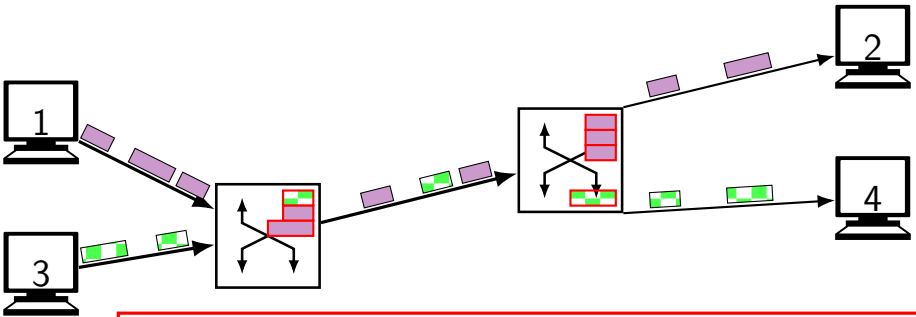
left switch *multiplexes* the two flows onto one link
right switch *demultiplexes* them to separate them

(de)multiplexing



this picture: multiplexed by dividing up *time* on link

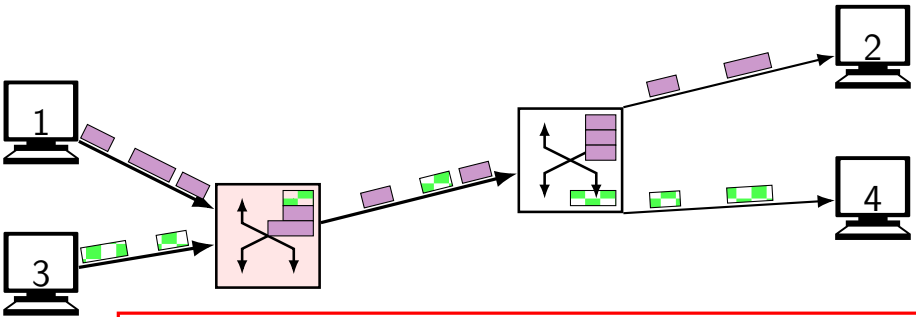
(de)multiplexing



switches usually have *buffers* (also called *queues*) hold waiting packets

absorbs temporary "bursts" where packets come faster than outgoing link can handle

(de)multiplexing

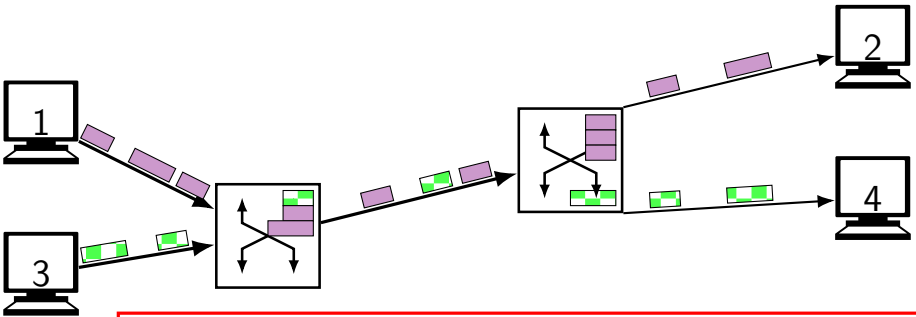


incomplete list of causes of ‘bursts’:

multiple unsynchronized flows

fast links produce packets faster for slow can send

(de)multiplexing

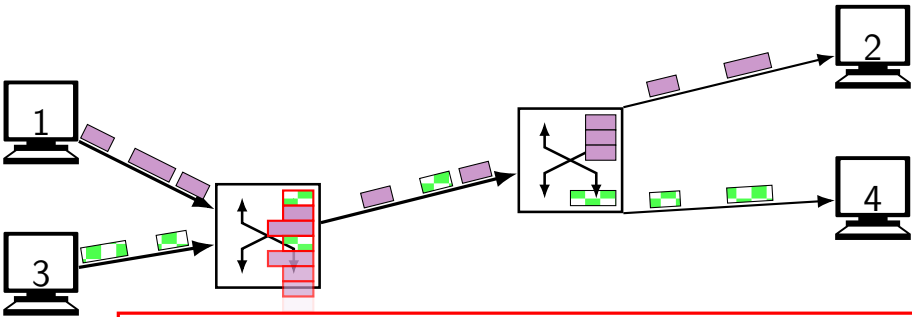


incomplete list of causes of ‘bursts’:

multiple unsynchronized flows

fast links produce packets faster for slow can send

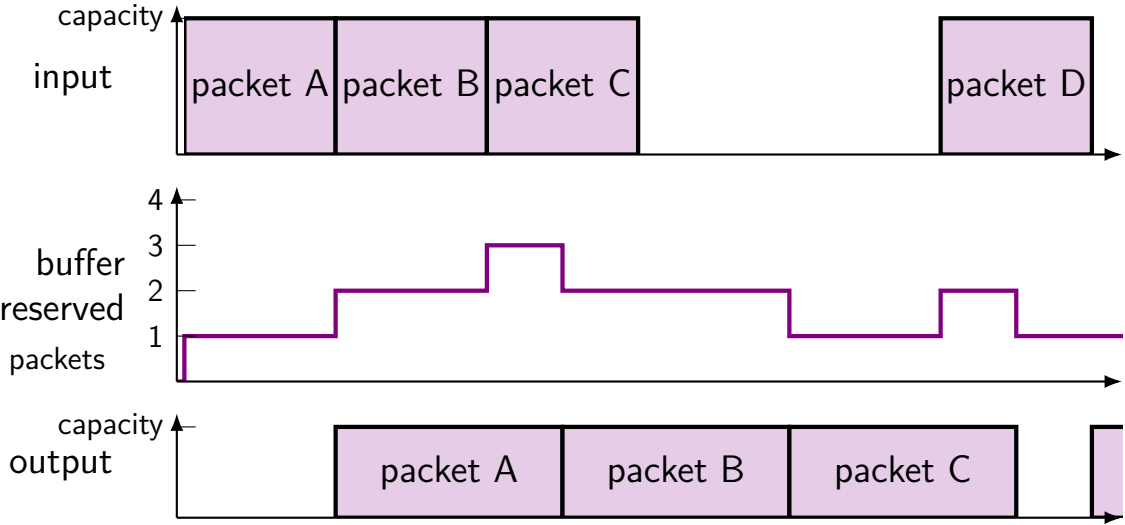
(de)multiplexing



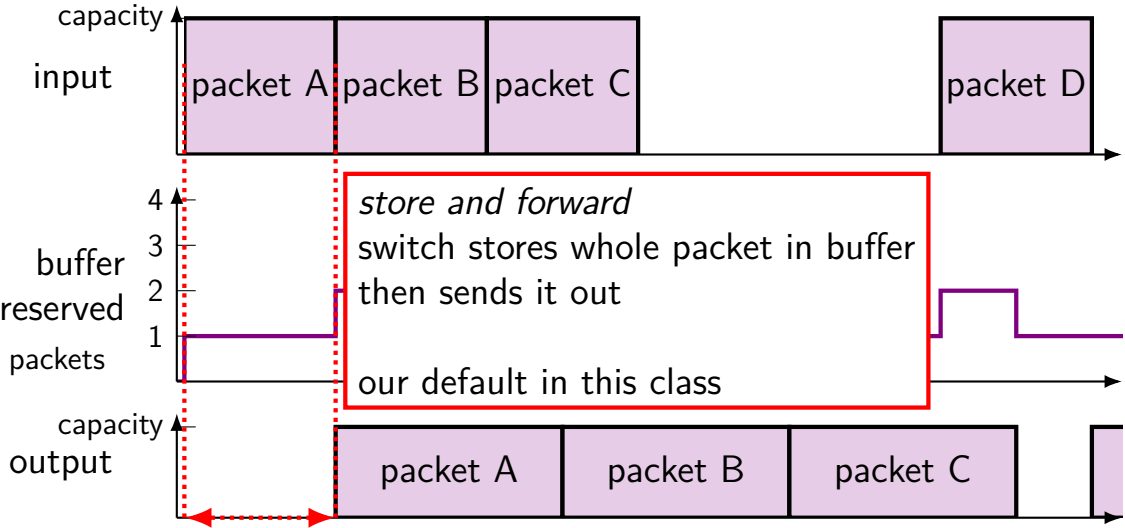
if buffer full, switch must *drop* packets
will happen eventually if overall rate faster than outgoing link

scenario is called *congestion*

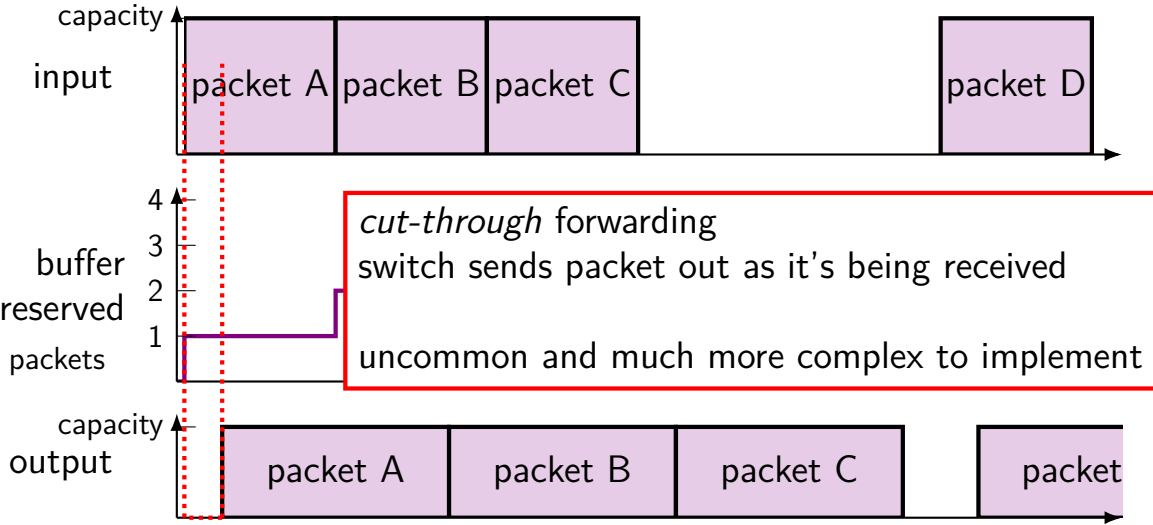
buffer usage: fast to slow, store + forward



buffer usage: fast to slow, store + forward



buffer usage: fast to slow, cut-through



channel abstractions

want to avoid custom network for each application

but applications have different needs

$\rightarrow$ multiple application interfaces to networks

common implementation of common patterns

some abstractions

stream

- continuous stream of bytes from one program to another
- 'connection' from one program to another

datagrams

- send small messages (*datagrams*)
- each datagram's destination independently set

remote procedure calls

- make function calls that run on remote machine

remote memory access

- read/write bytes of data in remote memory

...

some abstractions

stream

- continuous stream of bytes from one program to another
- 'connection' from one program to another

datagrams

- send small messages (*datagrams*)
- each datagram's destination independently set

remote procedure calls

- make function calls that run on remote machine

remote memory access

- read/write bytes of data in remote memory

...

focus on streams

this class: focus on implementing *streams of bytes*

why?

- most commonly used by applications on the Internet
- many common tasks with other abstractions

some challenges for streams

separating data into pieces network can handle

putting pieces back together

getting network to send piece to correct remote network

getting network to send piece to correct machine

getting machine to send data to correct program

getting pieces into format wires/radio/fiber/etc. can handle

handling transmission errors

some challenges for streams

separating data into pieces network can handle

putting pieces back together

getting network to send piece to correct remote network

getting ne

lots of work! don't want to implement all at once!

getting machine to send data to correct program

getting pieces into format wires/radio/fiber/etc. can handle

handling transmission errors

some challenges for streams

separating data into pieces network can handle

putting pieces in order

some parts need to be different for different local networks

getting network to send piece to correct remote network

getting network to send piece to correct machine

getting machine to send data to correct program

getting pieces into format wires/radio/fiber/etc. can handle

handling transmission errors

some challenges for streams

some parts should not concern local network implementors

separa

putting pieces back together

getting network to send piece to correct remote network

getting network to send piece to correct machine

getting machine to send data to correct program

getting pieces into format wires/radio/fiber/etc. can handle

handling transmission errors

some challenges for streams

some parts should be same for different abstraction

separating

putting pieces back together

getting network to send piece to correct remote network

getting network to send piece to correct machine

getting machine to send data to correct program

getting pieces into format wires/radio/fiber/etc. can handle

handling transmission errors

layered model

networking implemented in ‘layers’

upper layers implemented by making calls to lower layers

example: network implements ‘send data to (remote) machine’ function (“network layer”)

stream implementation calls this to implement ‘send stream to remote application’

OSI model

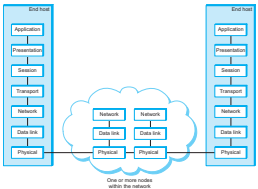


Figure 13 of Chapter 1 of Computer Networks: A Systems Approach (6th ed) (Peterson and Davie)

links types

backup slides