

layer 2 (link)

ethernet

local network topologies use **topologies**

machines on layer 2 communicate directly with eachother

addressing

definition

MAC address: media access control address, 48 bits, hex stored permanently on the machine



types of LAN communication

unicast

definition

unicast: send packet to single recipient

destination is recipient's MAC address

everybody gets signal on shared medium, ignore packet if not intended for you

broadcast

definition

broadcast: send packet to everyone

destination is FF:FF:FF:FF:FF:FF (broadcast address)

ensure everyone knows its intended for them

multicast

definition

multicast: send packet to everyone in group

special case of **broadcast**, everyone is in a group, each with a group address

(individual computer has first bit as 0, group address has first bit as 1)

packet structure

definition

frame: data packets (at layer 2)

additional fields to separate packets on the wire (Preamble, IPG)



multiple access protocols

multiplexing

allocate fixed slice to resources to each node, (can be wasteful, if node doesn't need to transmit, frequency/time unused)

definition

frequency-based: divide medium into frequency channels

time-based: give each node fixed time slots

taking turns

definition

token passing: pass virtual token around, only node with token can speak (for as long as it needs, then gives next node)

polling protocols: coordinator decides when each node can speak

→ no more time wasted on idling, but complex to implement

random access

definition

random access: deal with whenever they need to say something, then deal with collisions

ALOHA

definition

ALOHA: if you have packet, send packet, recipient replied with ACK

If two nodes send simultaneously, collision corrupts → no ACK, so wait **random** time then resend

ALOHAnet used two frequencies, hub transmits on own frequencies → no collisions, remote sites transmit on the other frequency → can collide, use ALOHA

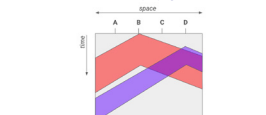
CSMA

definition

carrier sense multiple access: polite version of **ALOHA**

first listen to see if anyone is sending, and only send if quiet

doesn't avoid collisions due to **propagation delay**



CSMA/CD

definition

carrier sense multiple access with collision detection: listen before sending **and** listen while sending

stop sending if hear someone sending (during)

binary exponential backoff: after every collision, wait up to **twice** as long before resending

resends fast when possible, slowing down when necessary

topologies



definition

mesh: link between every pair of machines (complete graph)

bus: single wire for all machines, shared media

shared media: many machines using same wire signals will interfere/collide when multiple machines transmit at same time

mesh topology

benefits: good if we need dedicated high bandwidth links between every pair of machines

downsides: doesn't scale well, as you have to connect every pair to each other (complete graph)

single-link topology

benefits: scales much better than full-mesh topology

downsides: much less bandwidth available, everyone must share the single link

link layer

can use more than one wire in local network

switch

switches to forward packets towards destination

if switch receives broadcast packet, send to every neighbour

multicast more complicated, need to know who is in the group

routing protocols: from layer 3 can be used in layer 2, but MAC addresses cannot be aggregated (= layer 2 cannot scale to Internet)

layer 2 routing

flooding

send received packet to all ports, except incoming port

wastes bandwidth, and can create **broadcast storms** if there are loops, when everyone floods the packet and don't know where receiver is, so packet gets flooded forever

→ use **STP** to remove

learning switches

when receiving a data packet, has clue about where the sender is

creating the table

add table entry for sender, and as more packets sent, more table entries added



using the table

destination-based forwarding:

1. if destination exists in table, forward to next-hop
2. if doesn't exit, flood out to all ports (except incoming)

expiring routes

if TTL reaches 0, entry deleted → invalid routes eventually expire

STP

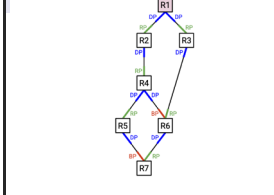
uses STP to create spanning tree

definition

spanning tree protocol: remove loops

method

1. elect one router as the root switch (pick lowest ID → use priority manually assigned, then lowest MAC address to tiebreak)
2. each router labels each port as one of three types
 - 1. **root port:** distance < x from root, along least-cost path (best path, tiebreak with this port)
 - 2. **blocked port:** distance < x from root but not on least-cost path (do not send or receive from this port)
 - 3. **designated port:** distance > x from root (leave enabled for further-away routers)
3. disable links → disable **blocked ports**



implementation

implement with **BPDU** exchanges, exchange control plane messages

BDPU

definition

BDPU: bridge protocol data units, similar to **distance:** vector protocol

each router keep tracks of (1, best root) and (2, distance of this root), and include when sending BPDU to neighbours

updating state

when receiving BPDU, accept it and update state, if (1, advertised root has lower ID) or (2, advertised root is same, but cost better)

initially every router only knows about itself, believes it is its own root, then because of criteria 1 from above makes **lowest ID** the root

each node sends advertisements to neighbours about (1, its best root) and (2, cost to reach that root)

convergence

after convergence, use periodic advertisements to learn neighbour's cost-to-root values

translation to layer 3

ARP

definition

ARP: address resolution protocol, layer 2 protocol translates layer 3 (**IP**) addresses to layer 2 MAC addresses (connects these two layers)

A sends messages to B on local network, must know recipient's MAC address from their IP address

ARP converts IP to MAC

if the **next hop** (in the **routing table**) is **direct** (in same local network) then use ARP to send to correct MAC address

ARP steps

1. Alice checks cache to see if there already is
2. if not, Alice **broadcasts** to everyone on the LAN, asking for the MAC address corresponding to Bob's IP address
3. Bob responds by sending a message to Alice (**unicast**), everyone else ignores the request
4. Alice caches Bob's MAC address

if Bob is outside the LAN, the router responds with its MAC address

→ Alice sends packet to router, router forwards packet to other LANs to reach Bob

the router will then perform ARP to find recipient, gets ARP response, and sends to destination

- source IP will be the original source IP (A's IP)
- source MAC will be router's MAC

all ARP replies are cached (even if no request was sent)

neighbour discovery

definition

neighbour discovery: layer 2 protocol translates layer 3 **IP** addresses to layer 2 MAC

uses **multicast** request to the group that destination

layer 7 (application)-merged

DHCP

when host connects to a network: user needs an **IP** address (other users can contact it), the **IP** address of the DNS server, and the **IP** address of the router

definition

DHCP: dynamic host configuration protocol, layer 7 application that acts on layer 3 information (**IP** addresses)

gives the user a configuration when they first join a network

runs on top of UDP/IP

DHCP server must know the subnet mask, gateway router **IP** address, DNS resolver **IP** address, pool of usable **IP** addresses

IP addresses are **leased** to hosts, hosts can only used for a limited time, and must return to keep using the address server will not offer the same address to other clients if currently being leased

DHCP handshake steps

1. **client discover:** client broadcasts request for configuration
2. **DHCP offer:** DHCP responds with configuration offer
 - normally only one DHCP server responds
 - offer includes IP address for client, DNS server's IP address (the recursive resolver), and router's IP address
 - offer has expiration time
3. **client request:** client broadcasts which configuration it chooses
 - if multiple DHCP offers made, the other ones are discarded
4. **DHCP acknowledgement:** chosen server confirms its configuration has been given to the client

DHCP packet



IPv6

extend **neighbour discovery** to learn other information (router address, DNS resolver address, subnet)

uses **SLAAC** to give yourself unique address

additional mechanisms in place for duplicate address detection if needed



domains

DNS

definition

DNS: domain name server, layer 4 protocol for **translating domain names to IP address**

DNS name server: server responsible for answering DNS requests

hierarchy

definition

zone: each node in the tree (area it's responsible for) → corresponds to an administrative authority response for some part of the hierarchy

- authoritative for how the names within that hierarchy are controlled
- and can delegate the task downward in the tree to a child name server

authoritative name server: each zone has an authoritative name server, which knows the domains in that zone



names are hierarchical: **domains** form a hierarchy

name servers are also hierarchical

authority is hierarchical, each organisation manages a zone and can delegate parts of the zone to other organisations

infrastructure is hierarchical, each name server only knows about a subset of domains

authorities

1. root authority = ICANN
2. top level domains (TLDs) directly below the root
3. then second level domains, third level domains, etc

each TLD can decide its own structure

users can lease specific domains from **domain registries**

organisations can operate their own zones (eg: Google, Amazon, ...) with their own name servers → tell parent about its name servers so the parent can resolve requests to the name server

zone availability

each zone must have several name servers for **redundancy**

must have 2 authoritative name servers to ensure availability

1. primary name server manages and updates actual records
2. secondary name server is a read-only copy of the primary server (mirror)
 - periodically transfers records to secondary servers
 - uses TCP instead of UDP (as transfers can be large)

definition

anycast: use same IP address for many servers on the Internet

very bad if root servers go down, so root servers use anycast to ensure high availability

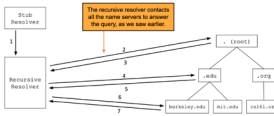
so thousands of servers can be a root server with the same address, may hear advertisements from many different servers, and just pick shortest one

resolvers

definition

stub resolver: resolver on computer, contacts recursive resolver and receives answer

recursive resolver: resolver that makes the DNS queries



getting a resolver

can use well-known resolvers (eg: 1.1.1.1 or 8.8.8.8, run by ISPs or application providers)

or can be given a resolver address when joining the network

benefits of recursive resolvers

1. can build a larger cache from multiple users' requests
2. reduces load on name servers

DNS implementation

protocol

DNS uses **UDP** as designed to be lightweight and fast

- no 3-way handshake needed
- no need for servers to maintain state
- most queries/responses fit within a single UDP packet (no bytestream abstraction needed)

DNS packet

UDP packet

- source port: chosen by client
- destination port: 53

DNS header (part of UDP payload):

- **ID number:** associates queries with responses, client picks this
 - name server uses some ID number in response
- **flags:** QR (query/response), RD (if need recursive resolver)
- **counts:** question, answer, authority, additional count: number of records of each type in the DNS payload

DNS payload (part of UDP payload):

- **resource records (RRs):** name-value pairs (question, answer, authority, additional section), the **DNS records** being sent

DNS record

definition

DNS record: name-value pair

type:

- **A:** answer, maps domain name to **IPv4** address
- **AAAA:** maps domain name to **IPv6** address
- **NS:** name server, maps zone to domain, designates another DNS server to handle
- **CNAME:** alias for another domain (eg: repo.eecs.berkeley.edu → repo2.eecs.berkeley.edu, basically a redirect)

metadata:

- time-to-live (TTL): how long record can be cached for, how long the answer supplied is **valid** for

sections:

- **question section:** what's being asked
- **answer section:** direct response to question
- **authority section:** delegation of authority (usually NS type record with zone and domain of child name server)
- **additional section:** glue records, additional information, eg: A type record with domain and **IP address** of child name server

DNS record caching

performance, resolves cache records → reduces load of name servers

DNS lookup

client sends **DNS query** to DNS name server when performing DNS lookup

name server sends **DNS response** back with answer (corresponding IP address)

use **dig** in unix to see the DNS lookup queries

authority section will contain the names of the (next) authoritative nameservers (NS records), and the additional section will contain the IP address of the nameservers to be delegated to (A records), so it doesn't need to make another query

dig +norecursive repo.eecs.berkeley.edu AAAA

0128.32.136.3

the +norecursive flag means for us to recurse manually, the AAAA flag requests for an IPv6 response

other uses

1. **email:** uses MX type records to map domain to mail server
 - contains priority in value, use lowest number first
2. **load-balancing:** return multiple servers in random order, automatically load-balances
 - or give an IP that is a geographically different server depending on where user is

domains

```
#run/csa-161 #run/csa-161/web
#run/csa-168 #run/csa-168/applications #run/csa-168/applications/dns
```

domain hierarchy



.edu is top-level domain (TLD), as directly below root

berkeley.edu is a **subdomain** of .edu

DNS resolves domains

HTTP

CDN

definition

content-delivery networks: deployments of servers that can serve content

placed close to end users (geographically and number of hops)

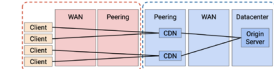
primarily caches static data (since doesn't change much)

benefits

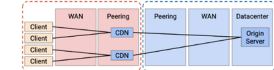
1. gives higher performance for users (with low-latency and high throughput)
2. reductions in bandwidth needed in network
3. reduces scaling needed for server infrastructure
4. provides better redundancy

deploying CDNs

- application could deploy CDNs in its own network (at the edge)



– or they could be **'sleeper'** in the network and deployed in the ISP's network reducing their backbone network cost (less peering infrastructure needed)



cannot push too deep: if we push into the WAN, higher cost and not justified

commercial model

mutually beneficial as gives content provider better performance and reduces cost for ISP (lower bandwidth costs)

content provider provides servers for free → ISPs host servers for free

HTTP optimisation

pipelining

multiple HTTP requests would need a separate TCP connection in naive approach

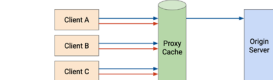
instead of this, allow multiple requests to be **pipelined** over the same TCP connection (server maintains open connections)

caching

cache data to avoid duplicate requests for same content

types of caches

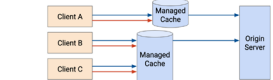
1. **private caches:** tied to specific end client, when user requests the same content for the second time
 - first client goes to origin server
 - subsequent requests served by proxy cache
2. **proxy caches:** in the network (not end host)
 - first client goes to origin server
 - subsequent requests served by proxy cache



which is operated by a **third party**

- therefore the data could be outdated/incorrect, and server has no control over it

3. **managed caches:** in the network, but operated by the server
 - gives server more control, and server can redirect client to cache (as server knows about them)



and there could be multiple proxy/managed caches in the network

HTTP implementation

resources can be **static** or **dynamic**

definition

static: stays the same for a long time (eg: a logo)

dynamic: generated on-demand for every request (eg: search results)

Expires: implements this, when the content can be cached until (HTTP/1.0)

servers cannot enforce clients or caches obey the header, just a request

Cache-Control: server gives more details on how to cache (HTTP/1.1)

- eg: Cache-Control: private, max-age=86400 → use private caches for 24 hours
- eg: Cache-Control: no-store → content cannot be cached

can have both headers for compatibility between HTTP/1.0 and HTTP/1.1

benefits of caching

1. client loads pages faster (proxy/managed caches → client gets data from closer source = lower RTT = higher TCP throughput)
2. less network bandwidth needed
3. reduces load on origin server

most larger objects are often static and so this nearly always provides benefit

CDNs

infrastructure for large scale caching

directing clients to caches

1. **anycast:** advertise some IP prefix from multiple locations, then least-cost routing automatically chooses best server to use
 - problem: routing can change in middle of a long-lived connection → the new receiver is confused as there was no TCP connection open with it
2. **DNS-based load balancing:** map some domain to different IP addresses, depending on where query came from (geographical caching)
 - solves anycast problem, since different servers have different addresses
 - problem: granularity → many users query through same resolver, all get same address
3. **application-level mapping:** application directs user to cache
 - different server for each user (eg: sf.google.com vs nz.google.com)
 - application knows end users address
 - allows for per-content item granularity
 - problems: still need to figure out best cache, and decide right strategy for failures

HTTP

definition

HTTP: a web protocol (**protocol**), used to request and retrieve data from a web server, Hypertext Transfer Protocol

HTTPS: secure version of HTTP uses **cryptography** to secure data with **TLS** protocol

- client-server protocol and runs over **TCP**
 - client and server runs a **TCP** handshake and sends data over the **bytestream**
 - server listens for HTTP on **port 80**

HTTP is a **request-response** model:

- web browser sends (one) **request** to web server
- then web server processes request and sends (one) **response** back to web browser

HTTP request

request syntax is human-readable plaintext

- **version**: version of HTTP
- **URL**: the resource we want to retrieve/interact with
- **method**:
 - GET / POST / HEAD / PUT, etc specifies what request is being made
 - GET: modifies server-state, and using it instead of POST can have security issues
- **data**: the data being sent, if applicable (only for POST and other methods that send data)
- **metadata**: eg: what browser, protocol, version

HTTP response

response also in plaintext

- **version**: version of HTTP / protocol
- **status code**: number indicating what happened → tell client what happened so it can respond correctly
 - 100s: informational responses
 - 200s: successful responses
 - 200: OK
 - 201: created / request
 - 300s: redirection messages
 - 301: moved permanently → always go to new location
 - 302: found, moved temporarily → check again in future
 - 400s: client error responses
 - 401: unauthorised
 - 403: forbidden
 - 404: file not found
 - 500s: server error responses
 - 500: internal server error
 - 503: service unavailable
- **data**: response data, eg: webpage, image, audio, PDF, etc
- **metadata**: **HTTP headers**, metadata about response, eg: date and time, length of content, etc

HTTP headers

aren't required, but can contain optional information / additional metadata

request headers

information about client to the server

- **User-Agent**: from what browser/program the client is using → can result in different processing of the request
- **Accept**: file type client is expecting in response (eg: text/html, image/*)
- **Host**: identifies which website the client is aiming to access (if server hosting multiple)
- **Referer**: how client triggered this request (where from) (used in **CSP**)

response headers

not directly related to the content but useful information

- **Date**: when server generated response
- **Location**: where content moved to (for 300 responses)
- **Server**: software server used to generate response

representation headers

used in both requests and responses, describes how content is represented

- **Content-Type**: File type of response
- **Content-Encoding**: how response is encoded into bits

underlay continues

scales by aggregating physical addresses (ignore virtual addresses)

overlay

definition

overlay network (VMs): virtual addresses based on organisational hierarchy (applications/companies)

scales as any-to-any routing not needed

encapsulation/decapsulation

definition

encapsulation: adding extra header
decapsulation: removing extra header, exposing original header underneath



when sending to a virtual address, encapsulate physical address on top of it
then receiving, decapsulate to get original header underneath

datacenters-merged

datacenter addressing

#uni/cs-168 #uni/cs-168/datacenter

still **hierarchical addressing**, but hierarchy **based on physical organisation** in the datacenter
→ address gives information about where in the building the host is, given that topology is regular (racks organised in rows)



can **aggregate** addresses by server / topology

datacenter delay & congestion control

#uni/cs-168 #uni/cs-168/datacenter

delay

generates internet delay has high transmission and propagation delay and small queuing delay
datacenters have very small transmission and propagation delay and high queuing delay

- transmission is low as high-capacity links
- propagation delay is low as everything in same building
- queuing is dominant source of delay

congestion control

TCP deliberately tries to fill up queues, much worse issue in datacenters as limited types of flows
most flows are **mouse** short and latency-sensitive (eg: queries)

some flows are **elephant**, very large, throughput-sensitive (eg: backups)
elephant flows fill up queues, mice get stuck in queues

algorithms

definition

BBR: reacts to delay instead of loss
measures RTT_s when RTT_s increase → slow down (detects congestion before queues fill up)

definition

DTCP: use **ECN** bit
since all routers managed by single datacenter, can update all hosts/routers to use ECN → efficiently makes hosts slow down before fully full

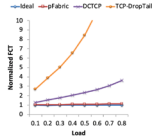
measure **FCT** (flow completion time): time from flow starting to last byte being received at destination

$$\text{normalised FCT} = \frac{\text{FCT}}{\text{Ideal FCT}}$$

definition

pFabric: gives mice a way to skip to front of queue
priority = remaining flow size
lower number → higher priority → mice have higher priority

senders transmit at full rate, only drop transmission rate under extreme loss (timeouts)
switches send highest priority packet, drop lowest priority packet



datacenter routing

#uni/cs-168 #uni/cs-168/datacenter

want routing protocol to find multiple paths between two hosts

ECMP load balancing

definition

ECMP: equal cost multi-path, finds **all** shortest paths with equal cost
then load-balance packets across those paths (uses **per-flow load balancing**)

load balancing

definition

load balancing: a function that takes as input a packet and outputs a link to send on

$$f: \text{packet} \rightarrow \text{link}$$

problem: TCP packet reordering

equal-cost paths might have different latency, recipient has to buffer packets → poor performance

destination based / source based

use destination IP/source IP to choose link
problem: if lots of sources sending to that destination/from that source, overloaded

per-flow load balancing

all 5 values, the unique 5-tuple (source/destination IP, source/destination port, protocol)
all packets in same flow use same link, no reordering problems

multi-path protocols

multi-path distance-vector

adjust **distance-vector protocol**
forwarding table stores multiple next-hops per destination, and if **same cost** → accept
use **load balancing** to choose next-hop for each packet

multi-path link-state

each router uses full network graph to calculate shortest path → extend protocol to calculate all shortest paths to each destination

datacenter topologies

#uni/cs-168 #uni/cs-168/datacenter

definition

radix: number of physical ports on a switch

big switch

connected to all TOR switches

- 1. high switching speed needed (N servers × B bandwidth)
- 2. large radix needed (1 port per rack)

tree

solved problems

- 1. reduced bandwidth
- 2. reduced radix

issues

- 1. low bisection bandwidth (top links oversubscribed)

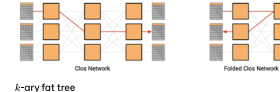
fat tree

increase link bandwidth at top layer
solves problem of low bisection bandwidth



clos networks

use **commodity switches** to build network, every switch identical (same number of ports, same link speeds, cheap switches)
high performance comes from **many paths** → high capacity through this
folded clos network: traffic goes in then back out, bidirectional links



- 1. k pods, each pod has k switches (half in each layer)
 - k/2 switches in upper aggregation layer
 - k/2 switches in lower edge layer
- 2. each switch has k links, half up and half down
 - (k/2) servers per pod

datacenter

#uni/cs-168 #uni/cs-168/datacenter

definition
datacenter:



east-west traffic: internal, between machines

north-south traffic: external, from datacenter to external location

datacenters

hosts servers and application infrastructure
peering locations host network interconnection infrastructure (ISPs and large companies (who host datacenters) install routers here)



owned by the datacenter (eg: Facebook)

properties

- 1. exists in single physical location
- 2. centralised control
- 3. homogeneous
- 4. high performance requirements (high bandwidth, reliability)
 - important as datacenter has high amounts of **internal traffic**
 - east-west traffic orders of magnitude larger than north-south traffic
- 5. organised in physical racks (each fits ~40 units)
- 6. each rack has a **top-of-rack switch**, wired connections to every switch (uplink, access link, ~100Gbps)

network connectivity

bisection bandwidth → measures network connectivity

definition

bisection bandwidth: delete minimum links to split network into two halves (bisection), sum of bandwidths on deleted links is the bisection bandwidth

bisection bandwidth (alternative): partition network into two halves, minimum bandwidth that nodes can collectively send at (worst-case partition/bottleneck bandwidth)

high bisection bandwidths ↔ more connected network

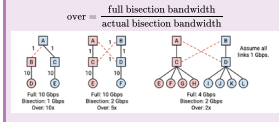
definition

full bisection bandwidth: all hosts in one partition can send at full rate
if N nodes each send at rate R, then

$$\text{full bisection bandwidth} = \frac{N}{2} \cdot R$$

definition

oversubscription: measure of how overloaded the network is



physical datacenter limitations

- 1. physical datacenters are rigid & structured, but applications constantly changing (hosts rapidly added/removed/changed)
 - changing physical infrastructure is hard
 - application grows → requires more servers to be installed
 - link goes down → need to fix/move server
- 2. scaling is difficult
 - lightweight application may not need all resources, but want separate server (security) → inefficient use of resources

- 3. routing is difficult
 - application may want their own IP addresses, may not be consistent with datacenter addresses

solution is virtualisation

host networking

#uni/cs-168 #uni/cs-168/datacenter

definition

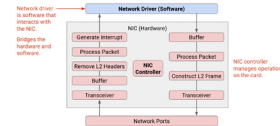
host networking: optimisations at end hosts

shared memory in user space

move networking stack (Layer 3/4) out of kernel space into user space
Layers 3-7 can access shared address space (instead of user ↔ kernel)

offloading to NIC

offload network stack out of CPU, into hardware



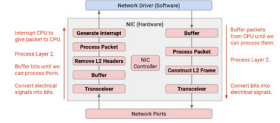
definition

network driver: software in OS that programs/manages the NIC

driver provides API for higher-level programs to interact with the NIC (hardware-software bridge: **SOFTWARE**)
network driver: software in OS that programs/manages the NIC

driver provides API for higher-level programs to interact with the NIC (hardware-software bridge)

before offloading



stateless operations

offload stateless operations, independent per packet (eg: checksum operations, validating, computing, segmentation)
segmentation → OS sends large packet, NIC segments multi-queue support, load balancing on the NIC, packet scheduler

- hashes packet to decide which CPU handles the packet (similar to ECMP)

stateful operations

firewalls, bandwidth management, checked by hardware instead
match-action pair table (similar to **OpenFlow API**)
software can program different policies onto hardware

protocols

offload TCP/protocols onto NIC

RDMA

definition

RDMA: remote direct memory access
server A can directly access memory in server B

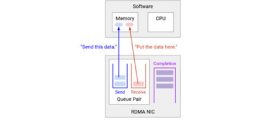
implemented directly in hardware, replace TCP/IP networking stack software (responsible for reliability, congestion control, etc)



queue pair abstraction

- send work queue has pending jobs to be transferred out
- receive work queue has pending jobs to be received

single NIC has multiple queue pairs, asynchronous each element in queue is a WQE (work queue element), eg: "Take 100 MB starting from address 0xHFF234 on the remote server, and write them to address 0xHFF7890 in my local memory"



when job finished, WQE removed from queue, NIC creates CQE (completion queue element), all jobs (success/failure) that happened

software defined networking

#uni/cs-168 #uni/cs-168/datacenter

definition

SDN: software defined networking
having **centralised control centre** routes, distribute to individual routers

vertical integration

routers are vertically integrated, standardised

- data plane standardised by IEEE
- control plane, management plane standardised by IETF

issues with standardisation

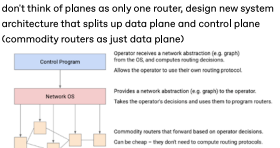
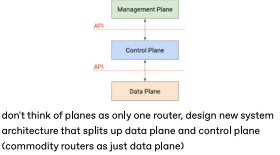
- 1. less flexible to custom solutions
- 2. unnecessarily complex for purposes of specific network
- 3. makes experimentation and research difficult

issues with vertical integration

- 4. router has all three **planes** in one chip, cannot just swap out one

disaggregating routers

separate router planes, use API between each layer of abstraction



OpenFlow API

API for interacting between control plane (operator code) and data plane (commodity routers)
defines **flow table** abstraction to describe routers, forwarding rules

- each entry consists of key-value pairs
- key = what to match the packet against (eg: 5-tuples, IP destinations, exact matches, etc)
- value = action when a packet matches

- output format is a sequence of 1/more numbered flow tables

example

Match	Action
Destination IP is in 192.168.0.0/16	Forward to next hop R2
5-tuple is exactly (1.1.1.1, 2.2.2.2, TCP 255, 80)	Add encapsulation and forward to R3
Source IP is exactly 10.2.15.3	Decapsulate and forward to R3

gives operator total freedom at control plane, not limited to **distance-vector protocol** or **link-state protocol**, can implement custom routing protocols

advantages

- 1. can simplify, remove unnecessary features, more maintainable
- 2. enables centralised computation at control program (not distributed)
- 3. more intelligent routing, better performance, easiest convergence

traffic engineering

route traffic in more intelligent way (eg: force longer path for better bandwidth utilisation)

definition

cSPF: constrained shortest path first algorithm
shortest path, subject to constraints

split paths can be used, implemented through encapsulation (label to route path along)

centralised control

centralisation ↔ globally optimal decisions, best for whole network

SDN in datacenter overlay

each tenant operates own controller
when new VM created, SDN learns about virtual and physical addresses → updates forwarding tables in other virtual switches, adding rules

example

"If you receive a packet with destination 192.0.2.1, add a header with Coke's virtual network ID of 42. Also, add a header with the corresponding physical address 2.2.2.2"

advantages

- 1. clearly split overlay and underlay networks into two scalable layers
- 2. implement control plane at end hosts, without complicated routing protocols
- 3. SDN only needs to know about VMs of a specific tenant

SDN in datacenter underlay

efficiently route packets through datacenter (eg: mice flows / elephant flow optimisation)

SDN in WANs

traffic engineer for more efficient bandwidth utilisation across cables

drawbacks

disadvantages

- 1. reliability, if central controller fails, everything fails
- 2. scalability problems, controller makes decisions for many routers → expensive
- 3. different kinds of complexity, infrastructure

SDN in management and data plane

data modeling: YANG
OpenConf: standardised APIs to interact with management plane

virtualisation

#uni/cs-168 #uni/cs-168/datacenter

definition

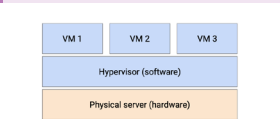
virtualisation: run 1/more virtual machines on a single physical machine

each VM provides the illusion of a dedicated physical machine

definition

hypervisor: physical machine runs the hypervisor (software) that gives each VM its own operating system

- manages resources between VMs
- enforces separation of servers



advantages

- 1. can rapidly add, remove, move VMs without touching physical servers
- 2. multiple applications can run on a single server, more efficient use of resources

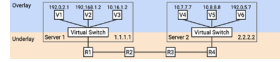
virtual switch

physical server has a single NIC, single IP address
give VM illusion of its own dedicated NIC and IP address → physical NIC receives packets meant for several VMs
solution: run **virtual switch** in software, on the server, which does **same tasks** as real switch
each VM connected to virtual switch, and uses physical NIC to send/receive packets

overlay/underlay

2 different addressing systems (sub-layers within layer 3) think about each layer separately (only thinks about its about addressing system)

virtual switch is bridge between the two layers
IP layer split into two IP headers (over, under)



underlay

definition

underlay network: physical addresses
physical servers/routers in datacenter, based on datacenter topology

congestion control-merged

congestion control

```
#uni/cs-168 #uni/cs-168/transport #uni/cs-168/transport/congestion-control
```

fundamentally a resource allocation problem
bandwidth is a limited resource → we need to allocate it to each connection well
a good congestion control algorithm should maximise link utilisation, minimise packet delay & loss, and connections should share the available bandwidth

congestion control protocols

1. reservations
2. pricing
3. dynamic adjustment → best fit for existing Internet, doesn't require knowing bandwidth ahead of time, doesn't require payment model

dynamic adjustment

1. **discovery**: discover links and a rate you can send data at
2. **detection & adjustment**: as traffic happens, network changes, adjust rate to send traffic at

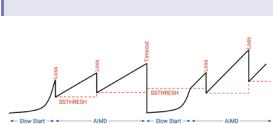
two types: **host-based congestion control** (used by TCP) and **router-assisted congestion control**

```
host-based congestion control
#uni/cs-168 #uni/cs-168/transport #uni/cs-168/transport/congestion-control
```

implemented at **and hosts**, no support from routers
adjusts rate based on implicit feedback from routers
TCP uses host-based congestion control
every host runs algorithm independently

algorithm

1. pick initial rate R (slow start)
2. try sending rate R for some period of time (event-driven updates)
 - check if congestion was experienced in this time period (detecting congestion)
 - if no, increase R (additive increase)
 - if yes, reduce R (multiplicative decrease)
 - repeat step 2



initial rate

discover initial rate by starting slow, then ramp up quickly for efficiency

algorithm

conceptual slow start:

1. start with small rate, could be much less than actual bandwidth
2. increase exponentially (double $cwnd$) each time, each RTT until loss detected
 - safe rate is most recent rate before loss detected ($0.5 \times \text{last loss}$)

algorithm

event-driven slow start: increase $cwnd$ by 1 after each ACK

intuitively, each ACK received, we can send one more packet (sliding window, unlocks next packet to send) can also increase $cwnd$ by 1 to send another packet (grows)

SSTRESH is variable in TCP of last safe rate

event-driven updates

hard to determine the period of time to send for → instead, $cwnd$ updates are **event-driven**
sender adjusts $cwnd$ each time something happens in TCP:

- 1. ACK for new data received
 - no congestion
 - increase $cwnd$
- 2. get 3 duplicate ACKs, declare packet lost (small issue)
 - probably isolated loss, still getting subsequent ACKs
 - decrease $cwnd$
- 3. timer expires, declare packet lost (bad issue)
 - probably lost several packets, don't get duplicate ACKs
 - abandon current rate, start over from slow-start

detecting congestion

use either **loss based** or **delay based** clues to detect congestion

loss based

(what **TCP** uses)

- if a packet is lost, declare congestion
- duplicate ACK
 - timer went off, didn't get ACK

benefits:

1. unambiguous signal, every packet is either lost or not lost
2. TCP already detects loss for reliability → code already exists to detect loss

cons:

1. sometimes loss not caused by congestion, but other errors/corruption
2. delayed packet could be mistakenly declared lost
3. when loss detected, network already congested

delay based

if packet is delayed, declare congestion
expect a certain return time, and compare
cons:

- 1. hard to measure packet delays, varies with queue size/other traffic (not yes/no answer)

reacting to congestion

determine how bandwidth is shared, how effectively bandwidth consumed, how quickly a host adapts to changes in available bandwidth
want efficiency (high utilisation of link bandwidth) and fairness (each flow gets equal share of bandwidth)
fast changes: multiplicative changes (doubling/halving)

$cwnd \leftarrow cwnd + \frac{1}{cwnd}$

slow changes: additive changes (adding/subtracting)

$cwnd \leftarrow \min\left(1, \frac{cwnd}{2}\right)$

$SSTRESH \leftarrow \frac{cwnd}{2}$

combine in four ways: AIMD, AIMD, MIAD, MIMD

AIMD

AIMD is best, sending too much worse than sending too little
→ gentle increase when uncongested, then rapidly decrease when congestion detected
only one that ensures fairness, converges to most fair point

derivation

two flows using single link of capacity C , flows independently running identical dynamic adjustment algorithms
flows sending at rates X and Y
if $X + Y > C$, network congested
if $X + Y < C$, network underloaded

goals are full utilisation $X + Y = C$ and fair sharing $X = Y$



additive adjustments itself always moves on line of slope 1 → on the fairness line, never approaches the fairness line
multiplicative adjustments itself

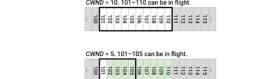
AIMD will never converge as **difference** in allocations always remains the same (increase/decrease by same additive amount)

MIMD will also not converge as the **ratio** in allocations remains the same (maintains some unfairness across iterations)
MIAD is maximally unfair, gap increases until one person has all the bandwidth ($0 - m = 0$), can never increase since multiplicative increase)
AIMD converges the best point, to fairness



fast recovery

after $cwnd$ decreases, the window becomes too small to send packets after the last packet (110) and the sender must wait (sending nothing) until the resent first dropped packet (101) is ACKed

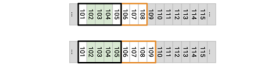


then when the first dropped packet is ACKed, the window jumps forward and the next packets (111 - 115) are sent all at once (burst after idle)

solution

the sender can deduce from the duplicate ACKs that fewer packets are in flight

→ grant the sender "temporary credit" for each duplicate ACK (because we know those packets got through)
when the duplicate ACK arrives, we know one fewer packet is in flight, so **artificially extend** the window to let the sender send one more packet



sender doesn't know which packets were ACKed, but knows how many packets were ACKed

new $cwnd$ = size of extended window - number of duplicate ACKs

new implementation

receiving duplicate ACKs

$cwnd \leftarrow \frac{cwnd}{2} + \text{number of duplicate ACKs}$

in **fast recovery mode**, receiving one duplicate ACK, artificially extend

$cwnd = cwnd + 1$

exit fast recovery mode, when new non-duplicate ACK received

$cwnd = SSTRESH$

router-assisted congestion control

```
#uni/cs-168 #uni/cs-168/transport #uni/cs-168/transport/congestion-control
```

routers signal congestion back to end hosts → hosts pick rate from explicit feedback from routers

routers can **enforce fairness**

multiple additive changes (doubling/halving)

$cwnd \leftarrow cwnd + \frac{1}{cwnd}$

$cwnd \leftarrow \min\left(1, \frac{cwnd}{2}\right)$

$SSTRESH \leftarrow \frac{cwnd}{2}$

combine in four ways: AIMD, AIMD, MIAD, MIMD

round-robin

assume packets are same size

definition

round-robin scheduling: take turns sending packets from each queue

max-min fairness

definition

max-min fairness: if one didn't get full demand, no one gets more than you

input: C , total available bandwidth and r_i , bandwidth demand for each flow i

output: fair allocation a_i to each flow i

max-min allocations are this, such that f is the "fair share" if you don't get full demand

$a_i = \min(f, r_i)$ s.t. $\sum a_i = C$

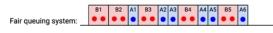
unequal packet size

bit-by-bit round-robin (fluid system), approximate this as cannot actually do it

fair queueing

most fair system is too complex to implement at high speeds, use approximations → DRR

(fluid + packet divided)
Fluid system:



Fair queueing system: **still needs dynamic adjustment**
gives **per-flow** fairness, ensures isolation → no one person hogs all bandwidth

fair queueing vs FIFO queues

advantages

1. isolation: cheating flows don't benefit (routers allocate) and flows can pick any rate adjustment scheme
2. bandwidth share doesn't depend on RTT

disadvantages

1. more complex than FIFO, must separate queues per flow, additional info needed per packet
2. can only be approximated in practice

rate adaptation

routers tell end hosts what rate to use

ECN

definition

explicit congestion notification: single bit in **E** packet header to indicate congestion

routers send this bit → when received, both packet and ACK has ECN on

sender should treat ECN bit as packet drop

advantages

1. doesn't confuse corruption and congestion
2. allows routers to warn about congestion earlier, reduces delays
3. lightweight to implement

cons

→ complicates evolution → if switch to selective ACKs, must change congestion control algorithm → failure of modularity, not layering

TCP

```
#uni/cs-161 #uni/cs-161/networking
#uni/cs-168 #uni/cs-168/transport
```

definition

TCP: transmission control protocol, layer 4 protocol for transmitting variable-length data

properties

- **bytestream abstraction**: automatically breaks input stream into segments, sent as layer 3 packets
- **provides ordering**: segments contain sequence numbers, allows destination to re-assemble
- **provides reliability**: destination sends **acknowledgements**, ensures packets are delivered
- **provides ports**: multiple services can share same IP address
- **full-duplex connections**: both hosts are "senders" and "receivers" and can send simultaneously in same connection

TCP implemented with **bytes** as the primary unit of data (bytestream)

- each byte has a number, packets are defined by number of the first byte inside
- ACKs reference byte numbers
- window size expressed in terms of number of bytes

segments

data may be streaming in from another application, when the segment is full (has **maximum segment size (MSS)**), it creates a TCP segment and adds a TCP header
the TCP payload is **one segment**

TCP header

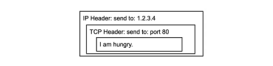
Source Port (16)	Destination Port (16)
Sequence Number (32)	Acknowledgment Number (32)
Header Len (4)	Flags (8)
Checksum (16)	Advertised Window (16)
Checksum (16)	Urgent Pointer (8)
Options (variable-length)	
Payload	

TCP functionality

1. **demultiplexing** (ports)
 - Source Port and Destination Port
2. **reliability** (checksum, sequence, ACK numbers)
 - Sequence Number, Acknowledgement Number, Checksum
3. **connection setup and teardown** (flags)
 - ACK: acknowledgement
 - SYN: synchronisation flag, indicates beginning of connection (synchronise ISN)
 - FIN: finish, **one way** and to connection, requires acknowledgement
 - no longer sends packets, but will receive (half-closed)
 - if **both sides** send a FIN, then connection is fully closed
 - RST: reset, **one way** and to connection, doesn't require acknowledgement
 - something wrong happened / abort connection, no longer send or receive packets
4. **flow control** (advertised window)
 - Advertised window: **advertised window**

TCP-over-IP packets

TCP (layer 4) is built on-top of IP (layer 3) → IP header (IP address) is still present



- E** packet with TCP header and payload
- IP packet limited by MTU
 - TCP segment limited by MSS (maximum segment size)
 - MSS = MTU - size of IP header - size of TCP header

TCP state

to have **reliability**, must maintain state at the **and hosts** (not the network)

sender must remember what packets have been sent and notacked, and length of timers
receiver has to buffer out-of-order packets

TCP sliding window

the **window** but W defines the maximum number of contiguous bytes in-flight, starting at the first unacked byte
the window slides right if and only if all leftmost bytes are acked

setting window size

- sender maintains window of W packets in flight
1. take advantage of network capacity
 2. don't overload recipient
 - receiver window $RWND$ reports how much space is left
 3. don't overload links
 - congestion window $cwnd$: sender runs congestion control algorithm

$W = \min(RWND, cwnd)$

rate and window size

$\text{rate} = \frac{cwnd}{RTT}$

larger window = higher rate, directly correlated

detecting loss

always send first unacked packet (leftmost part of the current window)

1. ack-based: if 3 duplicate acks are received, resend
2. timer-based: keep a single timer, if timer expires, resend (no longer one timer per packet)

retransmission

- if packet dropped, or ACK dropped, sender will resend (since acknowledgement was not received)
- when packets dropped a lot, TCP assumes congestion → sends data at slower rate

security

- TCP provides no **confidentiality** or **integrity** → higher layers (TLS) responsible for this
- defend against off-path attackers with random ISNs

TCP handshake, sending data, setup & teardown

```
#uni/cs-168 #uni/cs-168/transport
```

TCP 3-way handshake

SYN, SYN-ACK, ACK

synchronising ISN



1. client chooses **ISN** x , and sends SYN packet to server (synchronise)
2. server chooses **ISN** y , responds with SYN-ACK (acknowledgement) and is expecting $x + 1$ next
3. client sends ACK (acknowledges server's ISN)
4. **connection established** → server and client has synchronised sequence numbers

note
for **HTTP** requests, the ACK and HTTP request can be sent **together** in one packet (piggybacking)

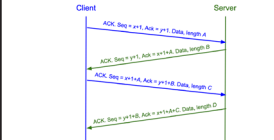
sending data

sequence numbers

initial sequence numbers: numbering starts at a **randomly-generated** (for security) initial sequence number (ISN)

first byte is ISN + 1, then ISN + 2, ...

segment sequence number is the number of the **first byte** in the segment



if the sender sends packet with sequence number j containing B bytes, then the bytes will be the numbers

$[j, j + B - 1]$

the cumulative ACK will be the **number of the next unacknowledged** (highest byte received + 1), and so the ack number is

adding on the length acknowledges the amount of data received

TCP setup/teardown transition diagram



reliability

```
#uni/cs-168 #uni/cs-168/transport
```

types of reliability:

1. best-effort delivery
2. at-least-once delivery: every packet arrives, but may have duplicates
3. exactly-once delivery: every packet arrives **exactly** once

use at-least-once delivery, then get rid of duplicates
TCP is allowed to announce that it didn't deliver → can never falsely claim it delivered a packet

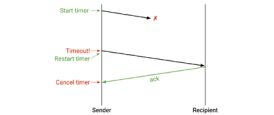
acknowledgement

definition

acknowledgement: reply from recipient to tell sender information
positive acknowledgement (ack): that packet was received successfully
negative acknowledgement (nack): that packet was **not** received successfully

individual packet ACKs

use a timer, start when packet is sent, cancel when ack received
if timer expires, resend the packet (could happen if ack dropped as well)



timer length cannot be too long (delays delivery) or too short (unnecessary retransmission)
→ should be **proportional to the round-trip time** (but difficult to measure)

full information ACKs

send an ack listing **every packet received** → but the list gets very long (eg: [1, 4, 7, 9])

cumulative ACKs

send an ack with the **highest sequence number** of which all previous packets have been received
TCP uses cumulative acks

loss

detecting loss early
individual or full information acks: can declare a packet lost if k subsequent packets are acked

example
received acks for packets 1, 2, 3, 4, ..., 6, 7, 8
means we probably lost packet 5

cumulative acks: declare loss if k duplicate acks are received (since highest sequence number cannot increase)
however, doesn't tell sender which packets were received, so leads to duplicate acks and ambiguity

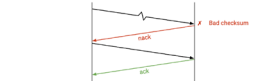
responding to loss

keep separate timer for each packet, resend corresponding packet if timer expires
fast retransmit: subsequent acks, resend the un-acked packet

reliability for single packet issues

[1/4] corrupted packets

detect corruption with checksums, if corrupted, send negative acknowledgement (nack)



or alternatively, ignore and will time out
TCP does not use nacks

[2/4] delayed packets

minor delay, ack will still be sent back
major delay, two acks will be sent back both are fine

duplicate packets

two acks will be sent back, which is fine

TCP with congestion control

```
#uni/cs-168 #uni/cs-168/transport #uni/cs-168/transport/congestion-control
```

values

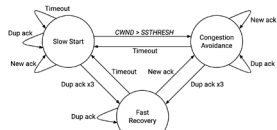
- sender maintains 5 values
- 1. dupACKcount: for detecting loss (initialised to 0)
- 2. timer for detecting loss
- 3. `rttwnd` for flow control
- 4. `CWND` for congestion control (initialised to 1)
- 5. `SSHTRESH` to remember last safe rate (initialised to ∞)

recipient maintains buffer of out-of-order packets

recipient replies with ACK and `rttwnd` value when responding to a packet

events

- sender responds to 3 events:
- 1. ACK for new data (not previously ACKed)
- 2. duplicate ACK
- 3. timeout



new ACK

- for new data
- if in slow-start mode, double per RTT
- $CWND \leftarrow CWND + 1$
- increase per ACK, so doubles over a RTT (same as $\times 2$ per RTT)
- if in fast-recovery mode, lost segment was received, leave fast recovery, enter congestion avoidance
- $CWND \leftarrow SSHTRESH$
- if in congestion avoidance mode, increase by 1 per RTT
- $CWND \leftarrow CWND + \frac{1}{CWND}$

then reset timer, reset duplicate ACK count, send new data if window allows

duplicate ACK

- when duplicate ACK received
- increment duplicate ACK count
- if `dupACKcount`: (third duplicate = 4th of the same) hits the threshold (3), do fast retransmit, then resend leftmost packet in window and enter fast recovery state
- $SSHTRESH \leftarrow CWND/2$
- $CWND \leftarrow CWND/2 + 3$
- $CWND \leftarrow CWND + 1$

timeout

when timer expires, reset to 1 packet

$SSHTRESH \leftarrow CWND/2$

$CWND \leftarrow 1$

then switch to slow-start mode, resend leftmost packet in window

congestion control variants

1. TCP Tahoe: `CWND = 1` after triple duplicate ACKs
2. TCP Reno: `CWND = 1` only on timeout, `CWND = CWND/2` after triple duplicate ACKs
3. TCP New Reno (using this one in class): TCP Reno + improved fast recovery
4. TCP-SACK: adds selective acknowledgements, ACKs describe byte ranges received

congestion control implemented at the sender, sender can run whatever code they want

works fine if using different variants – packet format is the same, just changes the rate

only issue is using SACK, must expect selective ACKs

UDP

```
#uni/cs-161 #uni/cs-161/networking #uni/cs-168 #uni/cs-168/transport
```

definition

UDP: user datagram protocol, layer 4 protocol

properties

- provides datagram (same as packet) abstraction
- messages sent as layer 3 packets
- applications break data into datagrams, then sent and received
- different to TCP in that the application breaks it up – no reliability or ordering, best-effort delivery – can be dropped / arrive in wrong order
- but much faster than TCP, since no 3-way handshake needed

UDP does not use sequence numbers, is connectionless

TCP throughput

```
#uni/cs-168 #uni/cs-168/transport
```

- using properties of
- 1. RTT (fixed)
- 2. μ (packet loss rate)

assumptions:

1. non-changing path through network
2. single TCP connection
3. ignore slow-start phase

throughput as packets

formula

$$RTT \cdot \text{bandwidth} = W_{\max}$$

if $W_{\max}$ is exceeded, one packet lost (since window size exceeded)

definition

average window size (packets): given $W_{\max}$ and $W_{\min}$

$$\frac{W_{\max} + W_{\min}}{2}$$

since it increases linearly from $W_{\min}$ to $W_{\max}$, drops back down and repeats

throughput as window size and loss

each packet is MSS bytes, so a

definition

average window size (bytes):

$$\frac{W_{\max} + W_{\min}}{2} \cdot \text{MSS}$$

definition

throughput (in terms of μ):

$$\text{throughput} = \sqrt{\frac{3}{2}} \cdot \frac{\text{MSS}}{\text{RTT} \cdot \sqrt{\mu}}$$

throughput is inversely proportional to RTT and inversely proportional to the square root of loss rate

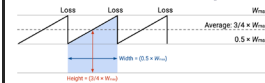
therefore flow with shorter RTT gets higher throughput, feature of TCP

derivation

window size is how much data can be sent in each RTT, so rate/bandwidth is

$$\text{bandwidth} = \frac{W_{\max}}{\text{RTT}} = \frac{2}{4} W_{\max} \cdot \frac{\text{MSS}}{\text{RTT}}$$

incorporating in loss, one packet lost every $\frac{1}{2} W_{\max}$ RTTs



each RTT, $\frac{1}{2} W_{\max} \cdot \frac{1}{2} W_{\max} = \frac{1}{4} W_{\max}^2$ packets are sent

loss rate is

$$p = \frac{8}{3W_{\max}}$$
$$\implies W_{\max} = \frac{2\sqrt{2}}{\sqrt{p}}$$
$$\implies \text{throughput} = \frac{3}{4} \cdot \frac{2\sqrt{2}}{\sqrt{p}} \cdot \frac{\text{MSS}}{\text{RTT}}$$

TCP throughput is choppy, rate oscillates between $\frac{1}{2} W$ and W

solution would be equation-based congestion control

issues

1. congestion detected even if just corrupted
 - loss could occur due to corruption, TCP confuses corruption with congestion
 - higher loss = lower throughput (but not because of congestion)
2. most connections are short
 - very few packets are sent, always stuck in slow-start
 - and not enough packets to trigger duplicate ACKs (can't get 3) - isolated loss leads to timeouts
 - partial fix, start with higher initial `CWND`
3. router queues get filled up
 - TCP overshoots capacity until packets get dropped
 - loss occurs when queue already full, so delays large
 - if routers keep long queues -> bufferbloat (routers have excessive memory, maintain long queues)
 - possible solution: BBR algorithm, detects congestion using delay instead of loss
4. cheating
 - each client runs separate algorithm, malicious users could change their algorithm
 - eg: increase window faster (rate increases faster), or start with large initial `CWND`
 - eg: open many connections, TCP shares bandwidth between connections
 - very difficult to cheat though
5. congestion control $\longleftrightarrow$ reliability
 - detects loss/congestion through duplicate ACKs, as TCP uses cumulative ACKs

TLS

```
#uni/cs-161 #uni/cs-161/networking #uni/cs-168 #uni/cs-168/routing
```

definition

TLS: transport layer security, layer 4/7 protocol secure communication on top of TCP

(relies on byte stream abstraction and provides byte stream abstract, appears the same to the client, but provides confidentiality and integrity & authenticity)

provides end-to-end security, and helps defend against lower-level network attacks (eg: to TLS, UDP) and prevents relay attacks

applications

used in HTTPS to secure communication over HTTP

TLS handshake

1. exchange random numbers R_B, R_S ("hellos"), which help prevent replay attacks (random for every handshake)
 - R_B, R_S both 256-bit randoms
2. **certificate step**: server sends its certificate, client validates certificate and therefore obtains server's public key
3. **premaster secret**: ensures client is talking to legitimate server (server must prove it has the corresponding private key)
 - **RSA approach**: client generates premaster secret (PS), encrypts with server's public key and sends to server -> server decrypts and now they share PS
 - **DHE approach**: server generates secret a , computes $g^a \text{ mod } p$, and signs with private key -> client verifies signature (proves server owns private key) and generates secret b and $g^b \text{ mod } p$, sending it to server -> they share $g^{ab} \text{ mod } p$
4. **derive symmetric keys**: each derive 4 symmetric keys (C_S, C_B, I_S, I_B) for encrypting client-to-server, server-to-client messages and MACing client-to-server, server-to-client
 - derived from R_B and R_S , with premaster secret PS
5. **exchange MACs**: exchange MACs on all messages of handshake in steps 1-4, detect tampering
6. **secure communication channel established**: encrypted and MAC'd messages can now be sent (uses MAC: then-encrypt, even though worse than encrypt-then-MAC)

replay attacks

- safe against replay attacks, as every connection uses different R_B and R_S
- add record numbers in encrypted TLS message -> since every message has unique record number, replayed messages will be detected

forward secrecy

- not guaranteed with RSA encryption approach
- guaranteed with Diffie-Hellman key exchange approach

TLS doesn't provide anonymity & availability

- does not provide anonymity, as client/server information are not encrypted in the TLS message
 - attacker can see IP addresses and ports in IP/TCP headers
- Information here is not confidential
- Information here is confidential
- Encrypted TLS message, MAC TLS tag
- does not provide availability, as attacker can drop encrypted TLS packets, or inject RST to abort TCP connection

issues

revocation

- attack can steal private key
- **defence**: certification revocation lists
- **defence**: online certificate status protocol (OCSP), browser queries CA on validity of certificate, CA can respond with 'revoked'

trust anchors

- if you trust too many certificate authorities -> CA might issue malicious certificate (if hacked or bribed, etc.)
- **defence**: certificate pinning, browser restricts which CAs are able to issue certificate for each website
- **defence**: certificate transparency, public logs provided by CAs

transport layer

```
#uni/cs-161 #uni/cs-161/networking #uni/cs-168 #uni/cs-168/transport
```

features

1. provides **demultiplexing** with ports (both TCP and UDP)
2. provides abstraction to applications (bytestream or datagram abstraction)
3. only TCP offers reliability

both TCP and UDP implemented at and hosts, not at intermediate routers

de-multiplexing gives packet to the appropriate application (on the end host)

and end-to-end principle, only end hosts need to use it

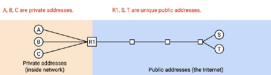
NAT

```
#uni/cs-168 #uni/cs-168/routing
```

definition

NAT: network address translation, uses a single public IP address to represent many hosts in the local network

middlebox (more functionality than just forwarding) so breaks end-to-end principle



outgoing packets: router changes private addresses to public addresses

incoming packets: router changes public address back to private addresses

PAT

definition

PAT: port address translation

most common, widely-used mode of NAT

steps

1. when A sends outgoing packet, $R1$ rewrites header to change to public IP
2. $R1$ keeps a table to remember where to send replies
 - keeps track of port numbers (inside and outside)
 - eg: if A: 50000 -> S: 80 then $R1$ will route the packet from S: 80 -> A: 50000 internally
3. $R1$ rewrites to change back to the private IP of A

NAT table entries

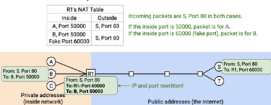
each entry represents a connection

a layer 4 connection is uniquely identified by a 5-tuple: (inside IP, inside port, protocol, outside IP, outside port)

re-writing ports

if A and B use the same inside ports then conflicts

the router will rewrite the inside port, makes up a fake port number so it is unique



implementation

1. home router sends DHCP request, asks ISP for an IP address (between router and another router)
2. ISP sends DHCP response, allocates IP address to home router
3. host sends DHCP request to home router, asks for an IP address
4. home router sends DHCP response, allocates private IP address to the host

home router converts private IP to public IP

rewrites header, read/rewrite layer 4 header, maintains connection state table

other info

- NAT used as close to edge of network as possible
- ISPs use CGNAT (carrier grade NAT) as ISPs ran out of IP addresses (even more complex)
- NAT is not used for IPv6: addresses as there are enough addresses

inbound connections

if connection not initiated by client, but by outside connection

NAT doesn't support inbound connections, breaks end-to-end principle

router needs port mapping table to support dynamic protocols exist for auto-configuring open ports (UPnP, NAT-PMP)

security implications

NAT disallows inbound connections by default, security feature similar to firewalls (side effect)

helps preserve client privacy as server sees router IP address only and not the host's address

links & delay-merged

delay

definition

propagation delay: time it takes a bit to travel along the link

$$\frac{\text{length of link}}{\text{link speed}}$$

definition

transmission delay: time to put the bits on the wire, (dependent on the bandwidth)

$$\frac{\text{packet size}}{\text{bandwidth}}$$

definition

packet delay: time for entire packet to be sent

$$\text{packet delay} = \text{transmission delay} + \text{propagation delay}$$

packet delay (with queueing delay at router):

packet delay = transmission delay + propagation delay + queueing delay

can be visually represented with timing and pipe diagrams

link tradeoff

depends on how large the packets sent are

example

- link 1: 10Mbps bandwidth, 10ms propagation delay
- link 2: 1Mbps bandwidth, 1ms propagation delay
- 10-byte packet: link 2 better (10ms with link 1, 1ms with link 2)
- 1000-byte packet: link 1 better (18ms with link 1, 81ms with link 2)

links

```
#uni/cs-168
```

definition

link: connects two devices

bandwidth: number of bits sent/received per unit time

- "width" of the link
- measured in bps (bits/s)

bandwidth-delay product

- "capacity" of the link
- bandwidth $\times$ delay

overloaded links

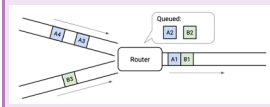
routers take many incoming streams of packets, processes them and forwards them to their destinations

the packet will either be forwarded, queued then forwarded, or dropped

definition

transient overload: overloads when two packets arrive simultaneously, as router cannot process both at same time, so must queue one for later

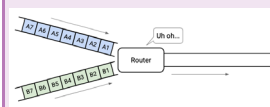
- when no incoming packets, router can drain the queue



definition

persistent overload: not enough capacity to handle incoming packets, queueing won't help

- solve by detecting overload and upgrading the link
- or router can tell senders to slow down
- can drop packets if persistent overloaded



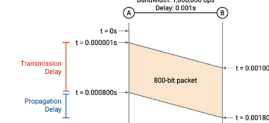
timing and pipe diagrams

```
#uni/cs-168
```

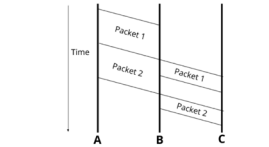
timing diagrams

Packet Delay = Transmission Delay + Propagation Delay

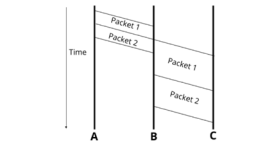
Packet Delay = (Packet Size / Bandwidth) + Propagation Delay



without queueing delay



with queueing delay

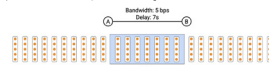


queueing delay happens when transmission time of packet 2 along the A - B link is less than the transmission time of packet 1 along the B - C link

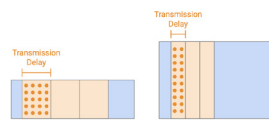
pipe diagrams

shows the bits on the link at a given moment in time

height = bandwidth, width = propagation delay, area = product (how many bits fit at any instant)



width of the packet in the pipe represents transmission delay (how long it takes to put all bits in pipe)



Send Sequence Space: TXControlBlock

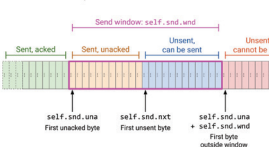
The instance variable `self.tx_data` is a byte array representing what you're sending. The user will append bytes to the back of this array, and you will remove bytes from the front of this array as you send them out.

The instance variable `self.send` is a `TXControlBlock` object, which has information about the outgoing byte stream that you are sending.

The `TXControlBlock` object has the following relevant instance variables:

- `self.send.iss`: Your initial sequence number.
- `self.send.una`: The oldest unacknowledged sequence number that you sent.
- `self.send.next`: The next sequence number you should send.
- `self.send.wnd`: The current size of your send window, determined by how much buffer space the other side (recipient) has left.

This diagram shows the byte stream that you are sending and what the instance variables represent:



Receive Sequence Space: RXControlBlock

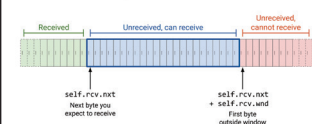
The instance variable `self.rx_data` is a byte array representing what you're receiving. As you receive packets, you should append their payloads (in the correct order) to this byte array so that the user can read them.

The instance variable `self.rcv` is a `RXControlBlock` object, which has information about the incoming byte stream that you are receiving.

The `RXControlBlock` object has the following relevant instance variables:

- `self.rcv.next`: The next sequence number you expect to receive.
- `self.rcv.wnd`: The current size of your receive window, determined by how much buffer space you have left.

This diagram shows the byte stream that you are sending and what the instance variables represent:



Receiving Incoming Segments

The packets you send and receive are TCP/IP packets.

If you receive an IP packet `p`, you can extract the TCP header and payload like this:

```
seg = p.tcp
```

If you receive a TCP segment `seg`, you can extract these fields from its header:

- `seg.seq`: The sequence number.
- `seg.ack`: The ack number.
- `seg.wnd`: The advertised window (how much buffer space left on the other side).
- `seg.ack`: Whether this segment has the ACK flag set (Boolean value).
- `seg.syn`: Whether this segment has the SYN flag set (Boolean value).
- `seg.fin`: Whether this segment has the FIN flag set (Boolean value).

If you receive a TCP segment `seg`, you can extract the payload like this:

```
seg.payload # The payload,
len(seg.payload) # Length of the payload.
```

Sending Outgoing Segments

To create a new packet, you can use the function `self.new_packet(ack=True, data=None, syn=False)`, which takes in the following arguments:

- `ack`: Whether to set the ACK flag. True by default.
- `data`: The TCP payload to send. None by default.
- `syn`: Whether to set the SYN flag. False by default.

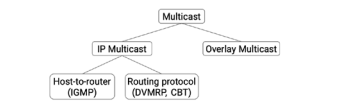
24 Multicast

#cs168 #beyond-client-server

Go beyond the unicast, client-server model. Instead, implement group communication via **multicast model**.

Multicast

- hosts can choose to join/leave groups at any time
- you can send a packet to a group, even if you are not a member of that group



- **IP multicast**: implemented in routers on L3
 - host-to-router (IGMP)
 - routing protocol (DVMRP, CBT)
- **Overlay multicast**: implemented in hosts on L7

Note

Packet delivery models

Unicast

Send the packet to exactly one destination.

Anycast

Send the packet to one of a set of possible destinations.

Broadcast

Send the packet to all destinations.

Multicast

Send the packet to all members of a group.

IP multicast

Each multicast group has an IP address. Reserved for multicast groups: 224.0.0.0-239.255.255.255.

Hosts can send 3 types of packets: **Join, Leave, Data** (send a data packet to a multicast group).

- Routers must know how to process Join, Leave, and Data packets from hosts.
- routers must **store states** e.g. to know members of group
 - routers must **run protocols** to learn how to forward packets
 - forward packet to all group members
 - packet is only forwarded once on each link

Routers implement two protocols:

- **Host-to-Router (IGMP)**: learns the group(s) that your directly-connected hosts belong to
 - static routing in unicast model
- **Routing (DVMRP, CBT)**: routers talk to each other to build forwarding tables
 - distance-vector/link state in unicast model
 - CBT gives better scaling than DVMRP at the cost of worse efficiency

Note

Pro: CBT scales better than DVMRP

- DVMRP needs one tree per sender, per destination group.
- DVMRP periodically clears prune state, causing packets to be broadcast to everyone (including non-members).
- DVMRP gives us the shortest paths from the sender to all group members.
- DVMRP doesn't require a specific choice of core.

Con: CBT is less efficient than DVMRP

- CBT needs one tree per destination group.
- CBT never broadcasts packets to everyone. The tree is constantly updated to only touch current group members.
- There is no guarantee that packets are taking the shortest path from the sender.
- CBT requires choosing a good core.
- A bad choice of core can lead to suboptimal paths.

Other comparisons between the two protocols:

- DVMRP con: The unicast and multicast protocols are tightly-coupled.
 - DVMRP is an extension to distance-vector. If we changed the unicast protocol (e.g. to link-state), we'd have to change the multicast protocol too.
 - CBT works with any unicast protocol.
- CBT creates a single point of failure at the core.
 - Can be solved by using multiple cores, but introduces more complexity. Worth discussing further.
- DVMRP is better if you have one source sending to a large group.
 - You get the shortest paths to all group members.
 - The group is large, so you get lots of bandwidth savings.
 - If the group is large, the occasional broadcasting (after pruning is cleared) isn't a big problem.
- CBT is better if you're sending to a small group.
 - CBT avoids broadcasting to non-members.

Both are used today. PIM (Protocol-Independent Multicast) has two variants:

- PIM-DM (Dense Mode): uses DVMRP for large groups
- PIM-SM (Sparse Mode): uses CBT for small groups

IGMP: directly-connected hosts

Exchange messages:

- **query**: router periodically asks hosts which groups they belong to
- **reports**: hosts periodically tell the router
- if the router doesn't receive a report for a long time, the membership has expired

DVMRP

Summary

DVMRP

One tree per sender, per destination group.

Flood. Issues:

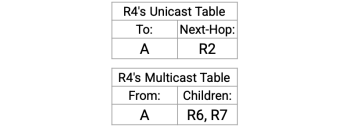
1. packet is sent along multiple paths to the same router.
 - fix with **RPB**: send packets along a spanning tree rooted at the sender
2. packet is sent to non-members
 - fix with **RPM**: prune branches of the tree with no members

Note

Reverse path broadcasting

- build a tree rooted at the sender
- if you receive a packet from your parent, send it to all your children
 - in multicast, children report themselves to their parents (extra tables)
- otherwise, drop the packet

In unicast, packet flow **up** toward the root. In multicast, packet flow **down**, from the root to everyone.



One tree per sender.
(In distance vector, there is one tree per destination.)

Core-based trees

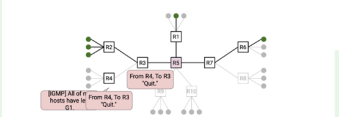
Summary

CBT

One tree per destination group, rooted at a specific **core group**. Touches every member of that destination group. (Initially, the core is the only node on the tree.)

If a router needs to join the tree: unicast a **Join** message to the core. Routers along the path can see the message. They join the tree by enabling the incoming link and the outgoing link.

If you are a leaf with no directly-connected members: send a **Quit** message to your parent (next-hop node). Parent receives Quit message and disables their link to that child. If you have only 1 link left, you are a leaf.



Summary

Sending a packet

1. From a group member
 - broadcast message along the tree to reach every group member
2. From not a group member
 - encapsulate the packet, and *unicast it to the core*
 - then, the core broadcasts the packet to everybody on the tree

IP multicast challenges

1. DVMRP, CBT do not scale to inter-domain routing.

Note

Today, IP multicast is mostly used within a single network. Applications can write custom solutions based on the unicast model.

Today, IP multicast is mostly used within a single network. Applications can write custom solutions based on the unicast model.

- e.g. use a central relay server, sender unicasts packet to server, and server unicasts packet to all group members

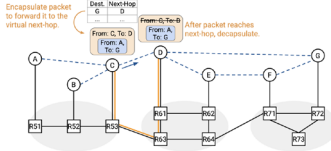
Problem: no inter-domain support for multicast.

Overlay-based multicast

Key idea: build a **virtual network topology** to connect the hosts together.

Hosts use **virtual forwarding tables** to route packets through the overlay network.

Virtual routers use **encapsulation** to send packets to the next-hop virtual routers.



Note

Implementation

Implemented in hosts, not routers:

- routers just forward unicast packets, as usual
- **virtual routers** in the hosts run routing protocols and populate multicastrules tables
- hosts are now forwarding packets between each other!

Summary

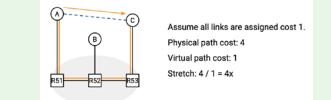
Pro: overlay multicast is easy to deploy and accommodates diversity in goals

Con: overlay multicast introduces additional overhead

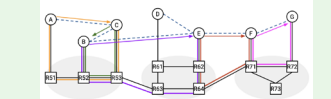
- packets require extra processing, e.g. en/decapsulating
- no guarantee that packets take least-cost paths to destinations
- app developers must implement multicastrules themselves

Note

Stretch: ratio of physical path cost to the corresponding virtual path cost. Lower is better, 1 is perfect.



Overlay multicast performance depends on how well the virtual topology matches the physical topology.



Currently, solved by manual configuration by operator. Common in dedicated networks like datacenters, where the operator controls the topology.

Note

Today, IP multicast is mostly used within a single network. Applications can write custom solutions based on the unicast model.

25 Collectives

#cs168 #beyond-client-server

Defining collectives

Summary

Collective communication: a group of nodes that exchange data in a coordinated manner as part of a group computation

- highly structured communication
- dedicated network infrastructure
- data is transformed as it's exchanged

Modern implementations: **NCCL** (Nvidia), **MSCL** (Microsoft), etc.

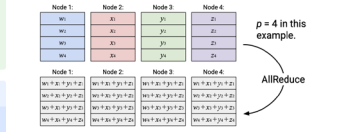
Summary

Collective operations: a set of basic communication patterns. "API" for distributed communication

- **orchestrated**: a centralized controller plans out the job ahead of time
- **synchronized**: every node starts the operation at the same time
- **blocking**: must wait for all nodes to finish before proceeding

Roughly three categories of collective operations: **redistribution** and **consolidation**.

Consider this example. There are $p = 4$ nodes. Each node specifies a p -element vector of input data, and receives a p -element vector of output data:



- Before the operation, a centralized controller does some setup:
- tell each node its number ("you are Node 1")
 - tell each node the value of p ("there are 4 nodes in total")
 - specify the memory addresses of the read buffer and write buffer

Note

Redistribution operations

Broadcast: the target sends its vector to everybody

Scatter: the target node sends its i th element to i th node

Gather: node i sends its i th element to the target node

AllGather: node i sends its i th element to everybody

Note

Consolidation operations

Reduce: sum up the vectors, then send the resulting sum vector to a target node

AllReduce: sum up the vectors, then send the resulting sum vector to everybody

Note

Naive ring is very busy:

Ring-Based AllReduce (Naive)

1. Send your vector to your left. When you receive a vector, add it to yours.

Note

Ring-Based AllReduce (Optimized)

1. Send your vector to your left, one element at a time (i.e. delaying later elements).
2. Send the overall sum to the left.

Note

Ring-Based AllReduce (Optimized)

1. Send your vector to your left, one element at a time (i.e. delaying later elements).
2. Send the overall sum to the left.

Note

Ring-Based AllReduce (Optimized)

1. Send your vector to your left, one element at a time (i.e. delaying later elements).
2. Send the overall sum to the left.

Note

Ring-Based AllReduce (Optimized)

1. Send your vector to your left, one element at a time (i.e. delaying later elements).
2. Send the overall sum to the left.

Note

Ring-Based AllReduce (Optimized)

1. Send your vector to your left, one element at a time (i.e. delaying later elements).
2. Send the overall sum to the left.

Note

Ring-Based AllReduce (Optimized)

1. Send your vector to your left, one element at a time (i.e. delaying later elements).
2. Send the overall sum to the left.

Note

Ring-Based AllReduce (Optimized)

1. Send your vector to your left, one element at a time (i.e. delaying later elements).
2. Send the overall sum to the left.

Note

Ring-Based AllReduce (Optimized)

1. Send your vector to your left, one element at a time (i.e. delaying later elements).
2. Send the overall sum to the left.

Note

Ring-Based AllReduce (Optimized)

1. Send your vector to your left, one element at a time (i.e. delaying later elements).
2. Send the overall sum to the left.

Note

Ring-Based AllReduce (Optimized)

1. Send your vector to your left, one element at a time (i.e. delaying later elements).
2. Send the overall sum to the left.

Summary

Overlay/underlay topologies

Overlay and Underlay Topologies

Suppose Hosts A, B, C, D want to run an AllReduce operation. A, B, C, D are not in a ring topology. But, we can add virtual links to create a ring!

- overlay ring topology
- average stretch = $(2 + 3 + 2 + 3)/4 = 2.5$

How you draw the overlay network (e.g. the order you connect the hosts) has a big impact on performance.

Note

Layers of abstraction for collectives

1. Definitions: users just need to know the input/outputs to use the operations
2. Overlay: the data exchanged over virtual links
3. Underlay: how the virtual links correspond to actual physical links

Note

Layers of abstraction for collectives

1. Definitions: users just need to know the input/outputs to use the operations
2. Overlay: the data exchanged over virtual links
3. Underlay: how the virtual links correspond to actual physical links

Note

Layers of abstraction for collectives

1. Definitions: users just need to know the input/outputs to use the operations
2. Overlay: the data exchanged over virtual links
3. Underlay: how the virtual links correspond to actual physical links

Note

Layers of abstraction for collectives

1. Definitions: users just need to know the input/outputs to use the operations
2. Overlay: the data exchanged over virtual links
3. Underlay: how the virtual links correspond to actual physical links

Note

Layers of abstraction for collectives

1. Definitions: users just need to know the input/outputs to use the operations
2. Overlay: the data exchanged over virtual links
3. Underlay: how the virtual links correspond to actual physical links

IP multicast is widely deployed across different domains for global Internet communication.

False: IP multicast is primarily used within a single domain, such as enterprise networks or ISPs, and is rarely deployed across multiple autonomous systems due to challenges like scalability, billing disputes, and inter-domain coordination required for protocols like PIM and MSDP.

In the DVMRP protocol, the multicast forwarding table requires one entry per source, per multicast group.

True: DVMRP builds a source-specific shortest-path tree for each (source, multicast group) pair, requiring multicast forwarding table entries that specify the Reverse Path Forwarding (RPF) interface and outgoing interfaces (children) for each pair, updated via flood-and-prune mechanisms.

Core-Based Trees (CBT) guarantee that packets are forwarded along the least-cost paths to all group members.

False: CBT uses a single shared tree per multicast group rooted at a core router, prioritizing scalability over optimality. Paths depend on the core's location and may be suboptimal compared to shortest unicast paths, unlike source-specific trees in protocols like DVMRP.

Overlay multicast requires routers to implement specialized multicast protocols, unlike IP multicast.

False: Overlay multicast operates at the application layer, with end hosts or proxy servers managing multicast routing using unicast packets. Routers only need standard unicast protocols, unlike IP multicast, which requires specialized protocols like PIM or DVMRP. Examples include P2P streaming or ALM protocols like Narada.

The AllReduce collective operation can be implemented efficiently using a ring topology, where each node sends and receives data to/from its neighbors.

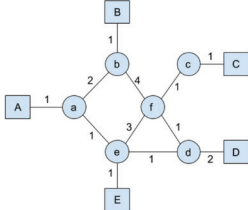
True: Ring-based AllReduce involves nodes exchanging partial sums in a logical ring, completing in $2(n-1)$ steps (reduce and broadcast phases) for n nodes. This reduces bandwidth compared to full-mesh approaches, though performance depends on the logical ring's mapping to the physical network. It is widely used in MPI and AI training frameworks like Horovod.

The stretch factor in overlay multicast measures the ratio of the overlay path cost to the underlay path cost, with lower stretch indicating better performance.

True: The stretch factor is the ratio of the overlay path cost (latency or hops in the overlay tree) to the underlay path cost (optimal unicast path). A lower stretch, closer to 1, indicates that the overlay topology closely matches the underlay, reducing latency and improving packet forwarding performance.

Collective operations in AI training, such as Broadcast and Reduce, are unrelated and cannot be combined to form other operations like AllReduce.

False: Broadcast and Reduce are complementary operations in collective communication. AllReduce can be implemented as a Reduce phase (aggregating data, e.g., summing gradients) followed by a Broadcast phase (distributing the result), commonly using tree-based algorithms in frameworks like MPI or Horovod.



DVMRP

All parts of this section are cumulative. At the start, A and E are members of group 1.

- (a) A sends a multicast packet to Group 1. After all Non-Membership Reports (NMRs) have propagated, which links remain active (i.e., used for forwarding packets) in the source-specific multicast tree for (Source A, Group 1)?

A-a, a-e, e-E

- (b) Suppose C adds itself to group 1. Immediately after this, who knows C joined group 1?

c. Only the first hop router is notified at first.

- (c) E sends a message to group 1. Which links remain active for (Source E, Group 1)?

A-a, a-e, e-E, d-e, d-f, c-f, c-C

- (d) B sends a message to group 1, is this possible?

Yes, non-members can send messages to a group.

- (e) What is the route that the packet takes from B to E?

B-b-a-e-E

- (f) How long does it take for B's message to reach all members of group 1?

7 seconds, the path to C (B-b-f-c-C) is the longest.

- (g) Suppose that the pruning information expires. A, B, and D form a new group 2. A then sends messages to groups 1 and 2. B sends a message to group 2, and D sends a message to group 1. What state does router f have? (Give a list of (Source X, group Y)).

(A, 1), (D, 1)

- (h) What message(s) took the longest to send?

B → D took 7 seconds.

- (i) What message(s) took the least time to send?

A → E took only 3 seconds.

CBT

All parts of this section are cumulative, but independent from the previous section. a is the core for group 1.

- (a) C decides to join group 1. What routers does its request go through?

c, f, d, e, a

- (b) D decides to join group 1. What routers does its request go through?

None

- (c) B sends a message to group 1. How long does it take to reach all members?

8 seconds. 3 seconds to reach a from B, and then another 5 to reach C from a (reaching D only took 4 seconds).

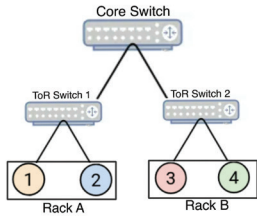
- (d) C sends a message to group 1. How long does it take to reach all members?

5 seconds. C → c → f → d → D.

- (e) The link between A and a goes down, but no other changes in control plane state occur. Who can still send/receive messages to/from group 1, if any?

B, C, D, E can send messages to Group 1. C, D can receive messages from Group 1. A cannot send or receive.

You are designing a distributed AI training system in a datacenter with 4 GPUs (nodes 1, 2, 3, 4) performing collective operations. Each node holds a 4-element vector of data (total size D bytes). The datacenter uses a Clos topology, with nodes 1 and 2 in Rack A, and nodes 3 and 4 in Rack B, connected via a core switch. Intra-rack links (e.g., 1 ↔ 2) have 1 hop (stretch = 1); inter-rack links (e.g., 1 ↔ 3) have 3 hops (stretch = 3). Assume intra-rack links have unlimited bandwidth, inter-rack links have a bandwidth capacity of B bytes per time step, and nodes can send/receive on all links simultaneously unless otherwise specified.



The Clos topology has nodes 1 and 2 in Rack A, nodes 3 and 4 in Rack B, connected via a core switch. Intra-rack links have 1 hop (stretch = 1); inter-rack links have 3 hops (stretch = 3).

Part A: Full-Mesh AllReduce Analysis

Consider an AllReduce operation where each node computes the element-wise sum of all vectors and receives the sum vector.

- (a) Explain why a full-mesh topology is suitable for AllReduce in AI training, and discuss its scalability limitations for large p .

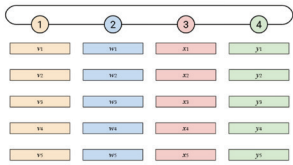
A full-mesh topology enables each node to send its vector directly to all other nodes simultaneously, minimizing latency to one time step, which is ideal for AI training's predictable, high-bandwidth needs. However, it requires $O(p^2)$ links, making it impractical for large p due to high wiring complexity and resource demands.

- (b) Calculate the total bandwidth and number of steps for a full-mesh AllReduce with $p = 4$. If inter-rack link bandwidth is limited to $B = D$ bytes per time step and intra-rack links have unlimited bandwidth, how many time steps are required?

Each node sends its vector (D bytes) to $p - 1 = 3$ other nodes, so each node sends $D \cdot 3$ bytes. With $p = 4$, total bandwidth is $4 \cdot D \cdot 3 = 12D$ bytes, completed in 1 step without bandwidth limits. With inter-rack link bandwidth limited to $B = D$ bytes per time step and intra-rack links unlimited, each node sends to 1 intra-rack node (D bytes, 1 step) and 2 inter-rack nodes (D bytes each, 1 step each). Since nodes can send on all links simultaneously, the operation completes in 1 step.

Part B: Naive Ring AllReduce

Assume a naive ring-based AllReduce (e.g., Node 1 → Node 2 → Node 3 → Node 4 → Node 1).



The ring topology connects nodes 1 → 2 → 3 → 4 → 1 in a closed loop.

- (a) How many steps are required for a naive ring AllReduce with $p = 4$? Describe the data exchange process.

The naive ring AllReduce requires $2(p - 1) = 2(4 - 1) = 6$ steps. The process consists of two phases, each taking $p - 1 = 3$ steps:

- Reduction Phase (Steps 1-3):** In each step, each node sends its current vector (D bytes) to its neighbor (e.g., Node 1 sends to Node 2, Node 2 to Node 3, etc.) and receives a vector from its other neighbor (e.g., Node 1 receives from Node 4). The received vector is added element-wise to the node's current vector. After $p - 1 = 3$ steps, each node has computed the sum of all vectors (e.g., Node 1 has $v_1 + w_1 + x_1 + y_1$).

- Distribution Phase (Steps 4-6):** Each node now sends the computed sum to its neighbor in the same manner. After another $p - 1 = 3$ steps, all nodes have received the same final sum, completing the AllReduce operation.

- (b) Calculate the total bandwidth and bandwidth per node per step. If inter-rack links are limited to $B = \frac{D}{2}$ bytes per step, how does this affect the number of steps?

- (b) Calculate the total bandwidth and bandwidth per node per step. If inter-rack links are limited to $B = \frac{D}{2}$ bytes per step, how does this affect the number of steps?

Each node sends one vector (D bytes) per step. The full AllReduce takes $2(p - 1) = 6$ steps with $p = 4$ nodes, so the total bandwidth (bytes sent) is $4 \cdot D \cdot 6 = 24D$ bytes (4 nodes send D bytes each step for 6 steps). The bandwidth per node per step is D bytes (sent only).

With inter-rack links limited to $B = \frac{D}{2}$ bytes per step, sending D bytes over an inter-rack link (e.g., 2 → 3) takes $\frac{D}{B} = 2$ steps. The ring has 2 intra-rack links (1 → 2, 3 → 4, 1 step each) and 2 inter-rack links (2 → 3, 4 → 1, 2 steps each). For one vector to traverse the ring once (one phase), the steps are: 1 (1 → 2) + 2 (2 → 3) + 1 (3 → 4) + 2 (4 → 1) = 6 steps. Since the AllReduce has two phases (reduction and distribution), the total steps increase to $6 \times 2 = 12$ steps.

Part C: Optimized Ring vs. Tree-Based AllReduce

Compare naive ring, optimized ring, and tree-based AllReduce implementations for $p = 16$.

- (a) For an optimized ring AllReduce, describe how it differs from the naive ring and calculate its total bandwidth and steps for $p = 16$.

The optimized ring performs Reduce-Scatter (each node gets one summed element) followed by AllGather (each node gets all summed elements), using $p - 1 = 15$ steps per phase, totaling $2(p - 1) = 30$ steps. Each node sends/receives $\frac{D}{p} \cdot 2$ bytes per step, unlike the naive ring, which sends the entire vector (D bytes) per step over two phases (reduction and distribution, $2(p - 1) = 30$ steps). For $p = 16$, each node sends $(p - 1) \cdot \frac{D}{p} \cdot 2 = 15 \cdot \frac{D}{16} \cdot 2 = 15 \frac{D}{8}$ bytes over both phases. Total bandwidth is $p \cdot (15 \frac{D}{8}) = 16 \cdot 15 \frac{D}{8} = 30D$ bytes, and steps are $2(p - 1) = 30$ (15 for Reduce-Scatter, 15 for AllGather).

- (b) For a tree-based AllReduce, calculate the total bandwidth and steps for $p = 16$, assuming a balanced binary tree.

In a tree-based AllReduce, leaf nodes send vectors up to the root, which computes the sum, then sends it down. For $p = 16$, a balanced binary tree has height $\log_2(16) = 4$. Each edge carries D bytes up and down, and there are $p - 1 = 15$ edges, so total bandwidth is $2 \cdot (p - 1) \cdot D = 2 \cdot 15 \cdot D = 30D$ bytes. Steps are $2 \cdot \log_2(p) = 8$ (4 up, 4 down).

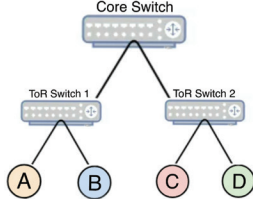
- (c) Compare the three implementations (naive ring, optimized ring, tree-based) for $p = 16$, discussing trade-offs for AI training.

- Naive Ring:** Bandwidth = $p \cdot D \cdot 2 \cdot (p - 1) = 16 \cdot 2 \cdot D \cdot 30 = 960D$ bytes, steps = $2(p - 1) = 30$. Simple but slow due to high bandwidth per step ($2D$) and sequential exchanges.
- Optimized Ring:** Bandwidth = $30D$ bytes, steps = 30. More steps but lower bandwidth per step ($\frac{D}{8}$), better for bandwidth-constrained links and scalability.
- Tree-Based:** Bandwidth = $30D$ bytes, steps = 8. Fastest due to fewer steps, but the root can be a bottleneck and tree construction may be complex.

For AI training, tree-based is preferred for low latency (fewer steps), but optimized ring is better for scalability and fault tolerance in large systems due to lower bandwidth per step and no single point of failure.

Part D: Overlay Topology Optimization

Optimize an overlay ring for AllReduce over the Clos network, with nodes A, B in Rack A, and C, D in Rack B.



Nodes A and B are in Rack A, nodes C and D are in Rack B, with intra-rack links (stretch = 1) and inter-rack links (stretch = 3).

- (a) Propose an overlay ring topology by numbering nodes (A, B, C, D) as Nodes 1, 2, 3, 4 to minimize the average stretch factor. Calculate the stretch for each link and the average.

Number nodes as: Node 1 = A, Node 2 = B, Node 3 = C, Node 4 = D, forming A → B → C → D → A. Stretches: A → B (1, intra-rack), B → C (3, inter-rack), C → D (1, intra-rack), D → A (3, inter-rack). Average stretch = $\frac{1+3+1+3}{4} = 2$.

- (b) Consider two alternative topologies: (i) A → C → D → B → A, and (ii) A → C → B → D → A. Calculate their average stretches and compare to your proposed topology. If inter-rack links have bandwidth $B = \frac{D}{4}$ bytes per step, which topology minimizes total latency for an optimized ring AllReduce, assuming each hop adds 1 time unit of latency?

- Alternative (i)** A → C → D → B → A: Stretches: A → C (3, inter-rack), C → D (1, intra-rack), D → B (3, inter-rack), B → A (1, intra-rack). Average = $\frac{3+1+3+1}{4} = 2$.
- Alternative (ii)** A → C → B → D → A: Stretches: A → C (3, inter-rack), C → B (3, inter-rack), B → D (3, inter-rack), D → A (3, inter-rack). Average = $\frac{3+3+3+3}{4} = 3$.
- Comparison:** The proposed topology (A → B → C → D → A) and alternative (i) both have stretch = 2, alternating intra- and inter-rack links, while alternative (ii) has higher stretch = 3, using only inter-rack links, making it less efficient.
- Latency Analysis:** For an optimized ring AllReduce ($p = 4$), total steps are $2 \cdot (p - 1) = 6$ (3 for Reduce-Scatter, 3 for AllGather). Each step sends $\frac{D}{4}$ bytes. With inter-rack bandwidth $B = \frac{D}{4}$, intra-rack links (unlimited) send $\frac{D}{4}$ in 1 step (1 hop = 1 time unit), and inter-rack links send $\frac{D}{4}$ in $\frac{D}{B} = 1$ step (3 hops = 3 time units). For A → B → C → D → A (links: intra, inter, intra, inter), steps are: 1 (1 hop), 3 (3 hops), 1 (1 hop), 3 (3 hops), total latency = $1 + 3 + 1 + 3 + 1 = 9$ time units. Alternative (i) has the same sequence (intra, inter, intra, inter), yielding 12 time units. Alternative (ii) (all inter-rack) has steps: 3, 3, 3, 3, total = 18 time units. The proposed topology and alternative (i) minimize latency equally; the proposed topology is chosen for its sequential ordering (A, B in Rack A; C, D in Rack B), simplifying configuration.

Part E: ReduceScatter Implementation

Consider a ReduceScatter operation, where each node receives one summed element of the AllReduce output.

- (a) Explain how ReduceScatter differs from AllReduce and how it can be implemented using a naive ring for $p = 4$. Calculate the total bandwidth and steps.

ReduceScatter sums each element across nodes, with node i receiving the i th summed element, unlike AllReduce, where all nodes receive the full summed vector. For a naive ring ($p = 4$), each node sends its entire vector (D bytes) to its neighbor in each of $p - 1 = 3$ steps, accumulating the i th element's sum at node i . Total bandwidth = $D \cdot (p - 1) \cdot p = 3 \cdot 4 \cdot D = 12D$ bytes (each of 4 nodes sends D bytes for 3 steps). Steps = 3.

- (b) If you implement ReduceScatter using the optimized ring topology from Part D.1 (A → B → C → D → A) with inter-rack bandwidth $B = \frac{D}{2}$, calculate the total latency (in steps) and compare to the naive ring.

For an optimized ring ReduceScatter ($p = 4$), only the ReduceScatter phase is needed ($p - 1 = 3$ steps), sending $\frac{D}{p} = \frac{D}{4}$ bytes per step. With inter-rack bandwidth $B = \frac{D}{2}$, intra-rack links (unlimited) send $\frac{D}{4}$ in 1 step, and inter-rack links (e.g., B → C, D → A) send $\frac{D}{4}$ in $\frac{D}{B} = 1$ step. The ring (A → B → C → D → A) has 2 intra-rack (1 → 2, 3 → 4) and 1 inter-rack (2 → 3) links in 3 steps, yielding 3 steps total. The naive ring also takes 3 steps but sends D bytes per step, increasing congestion. The optimized ring is more efficient due to lower bandwidth per step ($\frac{D}{4}$ vs. D), reducing congestion in bandwidth-constrained settings.

Summary of DVMRP Rules

Routing Rules:

For each source's spanning tree, you need to learn your parents and your children.

- Learning your parents: No action needed. Your unicast forwarding table already identifies your parent.
- Learning your children: Everyone sends an advertisement to their parent. When you receive these advertisements, you learn who your children are.

Forwarding Rules:

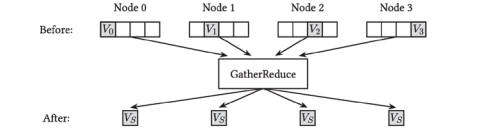
- When you receive a packet, use the unicast forwarding table for the given source to check if the packet is from your parent.
- If the packet is from your parent, use the new multicast forwarding table to forward it to your children. Only forward to the non-pruned children for the given destination.
- Otherwise, if the packet is not from your parent, then just drop the packet.

Pruning Rules:

For each (destination group, source) pair:

- If you receive a pruning message from a child, remove that child from your multicast forwarding table entry for this destination group.
- If none of your descendants (directly-connected hosts or children) belong to this group, send a pruning message to your parent.
- Periodically clear all pruning information (revert to forwarding to all children).

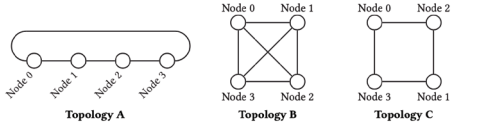
In the next two subparts, consider a new collective operation called GatherReduce. Using the representation from lecture, the GatherReduce operation is depicted as follows:



GatherReduce should compute $v_0 + v_1 + v_2 + v_3$ and send this sum to each node.

Note: Each individual square, i.e. v_0, v_1, v_2, v_3 , represents a vector of size D bytes.

We have three topologies to choose from:



We want to implement this collective, while satisfying the following efficiency constraint: For each link $X \rightarrow Y$, at most D bytes are sent from X to Y on this link, and at most D bytes are sent from Y to X on this link.

(2 points) Which topologies would allow us to implement the GatherReduce collective, while satisfying the efficiency constraint above? Select all that apply.

■ Topology A ■ Topology B ■ Topology C □ None
Out of the topologies you chose in the previous subpart (i.e. the topologies satisfying the constraint), which topology allows GatherReduce to complete in the fastest time?

Assume all links are identical (i.e. same bandwidth, same propagation delay).

○ Topology A ● Topology B ○ Topology C

links & delay-merged

delay

#uni/cs-168

definition

propagation delay: time it takes a bit to travel along the link

length of link
link speed

definition

transmission delay: time to put the bits on the wire, (dependent on the bandwidth)

packet size
bandwidth

definition

packet delay: time for entire packet to be sent

packet delay = transmission delay + propagation delay

packet delay (with queueing delay at router):

packet delay = transmission delay + propagation delay + q

can be visually represented with **timing and pipe diagrams**

link tradeoff

depends on how large the packets sent are

→ e.g:

link 1: 10Mbps bandwidth, 10ms propagation delay
link 2: 1Mbps bandwidth, 1ms propagation delay
10-byte packet: link 2 better (10ms with link 1, 1ms with link 2)
10,000-byte packet: link 1 better (18ms with link 1, 81ms with link 2)

links

#uni/cs-168

definition

link: connects two devices

bandwidth: number of bits sent/received per unit time
– "width" of the link
– measured in bps (bits/s)

bandwidth-delay product
– "capacity" of the link

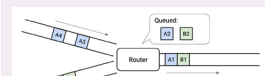
bandwidth × delay

overloaded links

routers take many incoming streams of packets, processes them and forwards them to their destinations
the packet will either be forwarded, queued then forwarded, or dropped

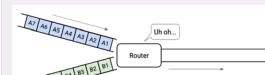
definition

transient overload: overloads when two packets arrive simultaneously, as router cannot process both at same time, so must queue one for later
– when no incoming packets, router can drain the queue



definition

persistent overload: not enough capacity to handle incoming packets, queuing won't help
– solve by detecting overload and upgrading the link
– or router can tell senders to slow down
– can drop packets if persistent overload

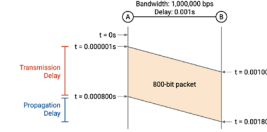


timing and pipe diagrams

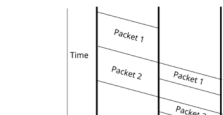
#uni/cs-168

timing diagrams

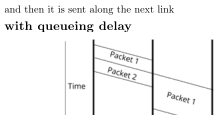
Packet Delay = Transmission Delay + Propagation Delay
Packet Delay = (Packet Size / Bandwidth) + Propagation Delay



without queueing delay



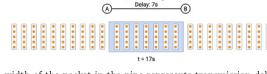
with queueing delay



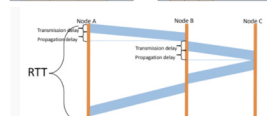
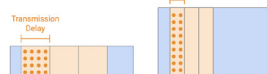
queueing delay happens when transmission time of packet 2 along the A – B link is less than the transmission time of packet 1 along the B – C link

pipe diagrams

shows the bits on the link at a frozen moment in time
height = bandwidth, width = propagation delay, area = product (how many bits fit at any instant)



width of the packet in the pipe represents transmission delay (how long it takes to put all bits in pipe)

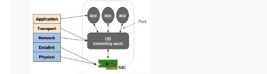


burly traffic: when packet arrivals are not evenly spaced in time

Summary

socket: an OS mechanism to connect a process to the networking stack via TCP/UDP protocols. **sockets are identified by unique IP/port pairs**

– **port (logical):** a number that the OS associates with a socket when it is created.
– **open a socket** between (source IP address source port) and (destination IP address destination port)
– allow L7 applications to operate on data streams (awareness of layers below packet handling, routing, congestion control, etc.)
– connect() : establishes connection to remote server
– listen() : waits for incoming connections (server side)
– accept() : accepts incoming connections and creates a new socket for a client session
– send() / receive() : sends/receives data through the connection



Note: NetworkInterface Card (NIC) is a hardware responsible for communication between a computer and a network medium (e.g., Ethernet cable, Wi-Fi) or L2.

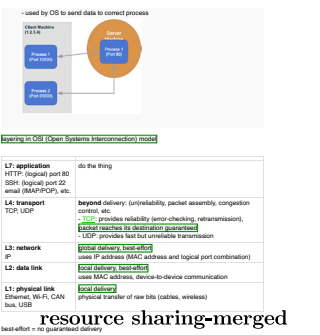
– each NIC is assigned a unique 48-bit **MAC address** (Media Access Control)

socket is OS abstraction for connections

connection pipe between two processes (or different hosts) data flows both ways



data is sent out as a stream of bits
– receiver knows IP and port of endpoints
– the Internet knows nothing about what IP forwarding (unless you implement security)



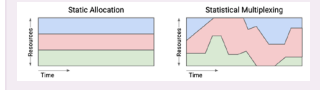
resource sharing-merged

resource sharing

#uni/cs-168

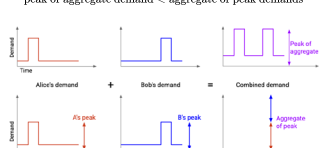
definition

statistical multiplexing: dynamically allocate resources to users based on their demand



statistical multiplexing is more efficient → as we need to satisfy everyone's **peak demand**

peak of aggregate demand < aggregate of peak demands



both circuit and packet switching are statistical multiplexing methods

circuit switching

definition

circuit switching: (reservations), at start of connection, end-hosts explicitly request and reserve resources

during connection, use reserved resources to send packets

at end of connection, release resources

resources shared between all active flows, reserves peak demand for the flow
not allocating for all flows that might ever exist, just currently established one
benefits:
– more convenient, guarantee of reserved resources
– more predictable and understandable behaviour

drawbacks:

– harder to implement as needs to establish dedicated links
– bursty traffic will lead to inefficient resource allocation, because needs to cater to peak demand

packet switching

definition

packet switching: (best-effort delivery), just use resources and send packets and hope for best (switching through whichever router is best)

resources shared between packets, given out on a packet-by-packet basis
no fixed allocation of resources

benefits:
– more efficient, as circuit switching has to use time to setup/teardown
– also no wasted resources, as the resources are only used when packets are sent
– automatically handles failure, end hosts don't need to do anything extra (where as circuit switching needs to detect) packet switching satisfies all demands

burstiness

data applications are bursty → such as web browsing, you download the entire webpage, then browse, so there is an initial large amount of traffic
smooth applications are like phone networks, which are pretty much constant, so phone network is circuit switching based → reserves the connection

demultiplexing

layer 3

add new header field that tells us what the next/higher layer protocol is → allows the IP code to pass the rest of the packet to the appropriate layer 4 code

→ e.g:

next layer is TCP / next layer is UDP, so the next layer knows which path to route through

layer 4

each running application (layer 7) is associated with a **port number**
when layer 4 receives the packet and forwards onto layer 7, it needs to pass the packet to the correct application

→ e.g:

to port 40000, or to port 60000, so it is passed to the correct application

design principles

#uni/cs-168 | #uni/cs-168/design-principles

- decentralised control:** each network device runs on its own
- best-effort service model:** routers only offer best-effort delivery
- route around trouble:** must be resilient to failures, if router link goes down, must try to find different path through network
- dumb infrastructure, with smart endpoints:** routers don't care about what's inside, just forwards packets
- end-to-end principle:** implement features at end hosts, not the routers
- layering** (layers of internet)
- ederation via narrow-waist interface** (narrow waist principle): federation works because all operators speak the same Layer 3 protocol

end-to-end principle

#uni/cs-168 | #uni/cs-168/design-principles

only have end hosts check (instead of routers)
strictly better than having every router check, as end hosts must check either way, so ends hosts checking alone is sufficient
routers checking = extra complexity

definition

end-to-end principle: certain application features must be implemented at the end host for correctness

definition

fate-sharing: either nothing fails, or everything fails

breaking for performance

could implement more checks to performance optimisation, done in addition to end-to-end checks (for correctness)
need for this must be evaluated on case-by-case basis (eg: if links are very lossy, sending duplicate packets can improve performance for end hosts)

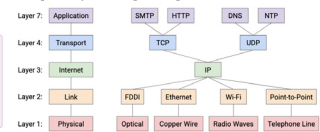
narrow waist principle

#uni/cs-168 | #uni/cs-168/design-principles

narrow waist: IP is the only protocol at Layer 3

definition

all hosts and routers understand IP ⇒ unifies the internet and enables federation
layers 1, 2, 4, 7 can have different protocols, but will all converge at Layer 3 and go through IP



ports

#uni/cs-168 | #uni/cs-168/design-principles

logical port

definition

logical port: number identifying an application, software port

logical ports identify the attachment point of the application and the OS

ports at layer 4

both ends hosts in a connection have a port number
private client can use randomly-generated port number
public server must use fixed, well-known port number (to be accessed)

both ends hosts in a connection have a port number
private client can use randomly-generated port number
public server must use fixed, well-known port number (to be accessed)

physical port

definition

physical port: hole you plug a cable into, hardware port

layers of internet

#uni/cs-161 | #uni/cs-161/networking

#uni/cs-168

layers

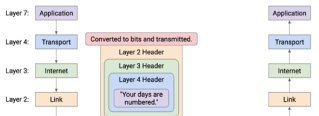
layers of abstraction, modularity in design
each layer only needs to think about itself (modular)
each layer has its own protocol, relies on service from layer below, and provides services for layer above

layers of abstraction

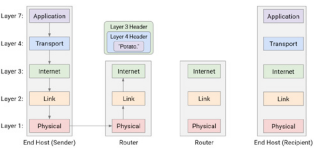
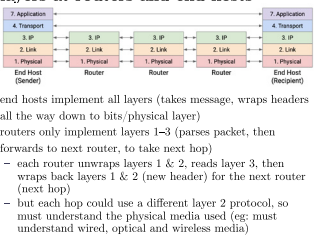
importantly, change in layer 1 implementation (wireless vs wired) doesn't affect other layers
change in layer 2 protocols doesn't affect the other layers, etc

headers

when moving to lower layers, **wrap additional headers**
when moving to higher layers, **peel off headers**
at each layer, only the headers for that layer are used, the rest are ignored
different layers use different **addressing schemes**
sending: go from highest to lowest layer
receiving: go from lowest to highest layer



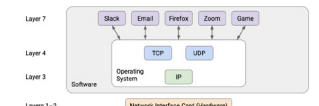
layers at routers and end hosts



then it is re-wrapped to be sent to the next router
when it arrives at the end host, all the layers are unwrapped to recover the original message sent

implementation of layers in end host

- layers 1 and 2 → implemented in **hardware**, on the NIC (network interface card)
- layers 3 and 4 → implemented in **software**, in the OS
- layer 7 is the applications running in software



OSI model

definition

OSI model: open systems interconnection model, layered model of internet communication (originally 7 layers, but 5 and 6 aren't used)



layer 1 (physical)

physical technology to move data across space
– eg: voltages an electrical wire, light signals on optical fibre, wireless radio waves, etc

provides: sending bits from one device to another

– encodes bits to send them over physical link (eg: patterns of voltage levels, RF modulation, etc)
protocols: Wi-Fi radios (IEEE 802.11), ethernet voltages (IEEE 802.3)

layer 2 (link)

local networks, connects devices locally, creating **links** between machines (LAN)

provides: sending frames directly from one device to another
– encodes messages into groups of bits (**frames**), defining the format of the data to be sent
relies upon: layer 1, physical technology to send bits
protocols: ethernet frames (IEEE 802.3)

frames

– must have **source**, **destination**, and **data**

MAC address

definition

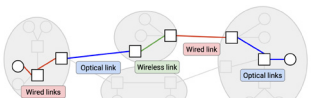
MAC address: media access control address
– 6-byte address, uniquely identifies network equipment (first 3 = manufacturer ID, last 3 = device ID)

layer 3 (network)

connecting local networks together, builds internet (network of networks) (WAN)

each operator runs its own local network, local networks connect together to form the internet

global delivery: packet takes multiple hops to reach destination, routers forwards packet closer to its destination
each local network could use a different layer 2 protocol (since it is abstracted away)



best effort service model: packets limited in size, could get lost/reordered/corrupted
network will try its best to deliver, but no guarantees (and cannot tell you it failed)

provides: sending packets from any device to another device
– encodes messages into groups of bits (**packets**)
– bridges multiple LANs for global addressing
relies upon: layer 2, frames to be sent correctly
protocols: Internet Protocol (IP)

routing

– allows packets to be routed across different devices to reach destination (not all devices are directly connected to one another, would not scale)

packets

– like frames, must have **source**, **destination**, and **data**
– may be fragmented into smaller packets (different links support different packet sizes)
– each router forwards packet to the next **hop**
– packets aren't guaranteed to take a specific route

reliability

– ensures packets are received correctly
– implemented with **checksum**
– IP is **unreliable**
– packets can be dropped/corrupted/delivered out of order

layer 4 (transport)

provides more mechanisms for reliable packet delivery
splits up and reassembles packets to send large data
provides: transportation of variable-length data from any point to any other point
relies upon: layer 3, packets sent correctly from device to another device
protocols: TCP (provides reliability and ports), UDP

layer 7 (application)

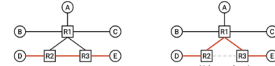
– provides: applications/services to users
– **relies upon:** layer 4, transportation of variable-length data
– **applications:** any online application! web browsing, messaging services, video calling

links, delays, pipe diagrams, switching, layers of internet, demultiplexing, ports

routing

#uni/cs-168

need for routing → seeing both mesh and single-link topologies, we can use fewer links than full-mesh and have more capacity/bandwidth than single link
more benefits, if a link goes down, traffic can just take another path (not reliant on specific links)



definition

intra-domain (routing protocols): compute routes within a single network
– called **IGPs** (interior gateway protocols)
– each network chooses their own intra-domain protocol, based on their needs

definition

inter-domain (routing protocols): computes routes between networks
– called **EGPs** (exterior gateway protocols)
– everybody has to agree on same protocol
– **internet** uses the **BGP** protocol

forwarding vs routing

forwarding → look up packet's destination in table, send packet to neighbour (**local** procedure), occurs every time a packet arrives

routing → communicates with other routers to determine how to populate tables (**global** procedure), occurs every time the network changes

destination based forwarding

each router keep table, mapping destinations to next hops (mapping of physical ports)
decision only depends on destination field of the packet

routing state validity

definition

valid routing state: if packets can actually reach their destinations
local routing state is a table in a single router, but can't be evaluated (as it's a global property)

theorem

global routing state is valid if and only if there are no dead ends and no loops

definition

dead end: packet arrives at router, but no next hop to forward it to
(but packet arriving at destination is not a dead end)
loop: packet cycles around the same set of routers

verifying state validity

check each destination separately, for each destination, use tables to see how router forward packets
next-hop becomes outgoing arrow, builds **directed delivery tree** for that destination

- properties:
- at each router, 1 next-hop per destination (=1 outgoing arrow per node)
 - once paths meet, they next split (same destination = same next hop)
 - all paths end in the destination



theorem

a **directed delivery tree** is an **oriented spanning tree** (rooted at the destination)

least-cost routing

assign costs to every edge (we can choose how to define "good", could be any metric)
costs are local to a router, router knows costs of its links
good paths (least-cost) must also be **valid paths**

properties of costs

1. always positive integers (can't traverse an edge and make cost cheaper)
2. costs are always symmetrical ($c(A,B) = c(B,A)$)

static routing

some table entries can be manually hard-coded
connected/direct routes let us forward things we're connected to directly, no routing protocol needed

routing

#uni/cs-168

#uni/cs-168/routing

	intra-domain (IGP)	inter-domain (EGP)
distance-vector protocols	RIP	
link-state protocols	IS-IS, OSPF	
path-vector		BGP

communication between routers

eBGP: routers in different ASes exchanging routes
IGP: routers distributing externally-learned routes to interior routes (for external routes)
→ cares only about AS and reachability
IGP: intra-domain routing protocols for internal reachability (for internal routes)
→ cares about cost to internal nodes

distance-vector vs link-state

link-state holds global data, where each node knows the full network graph → gives routers more control over the path they choose
distance-vector holds local data, and so must trust what their neighbour says (don't know what path they're using)
link-state may converge better, as information is flooded before recomputing, where as distance-vector needs to wait for neighbour to recompute and re-advertise path

distance-vector protocols

#uni/cs-168

#uni/cs-168/routing

#uni/cs-168/routing/intra-domain

local data: each node only knows about part of the network (its part)

global (distributed) computation: each node computes part of the solution
general idea, if you hear about a path to the destination, tell all your neighbours



in this image, R2 and R3 hear that R1 is able to reach A, so tells all of its neighbours (R4, R5, R6, R7) about how they know how to reach A, and propagates onwards
routing announcements propagates outward (away from A, destination)

forwarding packets travel inward (towards A)

if multiple paths are advertised, **accept the shorter path** but, a router may not hear about multiple paths simultaneously, so record the best-known cost to a destination if the table doesn't have a path to destination, accept any path you hear about
then update the table if you hear a better path later, same relaxation operation

cost to neighbour + cost from neighbour to destination < best-known distance-vector algorithms are a **distributed, asynchronous** version of **Bellman Ford**, as each router relaxes its own links and can happen at any time

base algorithm

```
1: procedure BellmanFord(dat, routers, links)
2:   for each  $r$  in routers do
3:     distance[ $r$ ] ← ∞
4:     nexthop[ $r$ ] ← None
5:   end for
6:   distance[dat] ← 0
7:   while distance[dat] < ∞
8:     for each router do
9:       for  $r_1, r_2, c$  in links do //relaxation operation
10:        if distance[ $r_1$ ] + c < distance[ $r_2$ ] then
11:          distance[ $r_2$ ] = distance[ $r_1$ ] + c
12:          nexthop[ $r_2$ ] =  $r_1$ 
13:        end if
14:      end for
15:    end for
16:  end while
17: end procedure
```

except it is decentralised, the algorithm does not have access to routers, links so it will just be done locally

advertising a route, updates from next-hop

if you hear an advertisement, update the table if
– the destination isn't in the table
– or the cost is better

advertised cost + link cost to neighbour < best-known cost

then advertise to all your neighbours (if something changes)

changing topology

if our current next hop (best next route) sends is an announcement, accept it, even if the path is worse → allows next-hop to notify us that the topology has changed

convergence

after running this protocol for some time, it will **converge**,

if our current next hop (best next route) sends is an announcement, accept it, even if the path is worse → allows next-hop to notify us that the topology has changed

convergence

after running this protocol for some time, it will **converge**, everyone's forwarding table has the least-cost next hop if a change happens, new announcements are sent and forwarding tables are updated, and will eventually converge again to new routing state

definition

steady-state: occurs when network has converged, everything stays same until next topology change

reliability

packets can be dropped and never reaches destination
solution is to resend advertisements every X seconds (advertisement interval)

expiry

links and routers can fail, so each route has a **finite time to live** (TTL)

solution is to use periodic advertisements ⇒ confirms that a route still exists
→ when we get advertisement, reset the TTL

if a link goes down, updates will stop, so TTL expires → if TTL expires, delete entry from table

poison

when links go down, routers still need to wait for TTL to be over until it's refreshed/updated ⇒ much faster than waiting for timeouts

solution is to **poison** → explicitly advertise that link went down

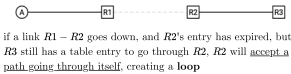
path with cost ∞ represents a broken path, path will propagate just like any other path
– routers always accepts poisoned path (∞) to invalidate the route
– then update the table, reset the TTL
– advertise the poison to all neighbours, so they can know of the broken route

sources of poison: one of your routes times out (since we know all connected links) → poison the entry, set cost to ∞ and reset TTL = advertising poison to neighbours

timers

- therefore we have two timers
1. to resend advertisements
 2. to hold the TTL

split horizon & poison reverse

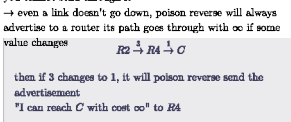


if a link $R1 - R2$ goes down, and $R2$'s entry has expired, but $R3$ still has a table entry to go through $R2$, $R2$ will accept a path going through itself, creating a loop

split horizon solution is to never advertise a path back to the source
→ in this case, $R3$ should never advertise the path back to $R2$

poison reverse solution is to explicitly advertise poison for all entries
→ in this case, $R2$ will poison reverse $R3$, and advertise that you cannot send through it
→ even a link doesn't go down, poison reverse will always advertise to a router its path goes through with ∞ if some value changes $R2 \rightarrow R4 \rightarrow C$

then if 3 changes to 1, it will poison reverse send the advertisement
"I can reach C with cost ∞" to $R4$



split horizon is the passive solution, where it waits for expiry (loop stays until routes expire), but poison reverse is the active solution, and explicitly terminates the loop as it poisons the entry (later)

count to infinity

a very special case of 3 routers, where a link goes down, BUT the routing packet gets dropped



as the **poison packet** is dropped, $R1$ is not notified of this link failure and announces that it can still reach A
if using **split horizon**, $R1$ will not announce to $R3$ BUT will announce to $R2$ and remove the poison
likewise, $R2$ will not announce to $R1$ BUT will announce to $R3$ and remove its poison

now all routers have positive value entries, and the costs will keep increasing (infinite loop)

solution is to enforce a max cost, and generally consider anything $\geq 16 = \infty$
this will make all entries eventually become ∞ and expire all

routing

#uni/cs-168

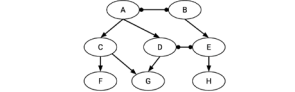
#uni/cs-168/routing

1. **customer-provider** relationship
– customer pays the provider
– provider offers to forward traffic to/from the customer
2. **peer-to-peer** relationship
– peers don't pay each other, exchange roughly equal traffic

Provider → Customer

Peer → Peer

The arrows do not represent direction of packets (e.g. F can send packets to C).



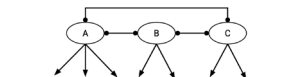
packets can go in either direction of an arrow, just represents the business relationship

properties

- **autonomous systems** > **stub** AS at the bottom of the graph, as they're not providing service to anybody
- **autonomous systems** > **transit** AS in all other nodes (and flow links) as service is being provided
- **asyclic** (since cannot pay yourself)
- **service** a hierarchy (all arrows point down)
- money flows up

hierarchy

tier 1 AS at the top of hierarchy, no providers, peer-to-peer relationship with every other AS (so graph is connected)



non-tier 1 AS has at least one provider, walking up from any AS will lead to a tier 1 AS eventually

goals

1. **scalability**: routing must scale to entire internet
– use hierarchical IP addressing
2. **privacy**: AS don't want to explicitly announce sensitive information
3. **autonomy**: AS have freedom to choose their own policies (based on business goals)
– good paths ⇒ paths that respect AS's policies (instead of least-cost for intra-domain)

autonomous systems

#uni/cs-168

#uni/cs-168/routing

#uni/cs-168/routing/inter-domain

definition

autonomous system (AS): one or more local networks, under a single administrative control
each AS is assigned a unique AS number (ASN)

inter-domain topology (AS graph): graph of ASes, each node is an AS, abstracts away individual routers/hosts

types

stub AS

only sends/receives packets on behalf of hosts inside the AS similar to end hosts in intra-domain model, doesn't forward packets between other ASes
most ASes are stubs

transit AS

forwards packets on behalf of other ASes similar to routers in intra-domain model, can still send/receive packets for hosts inside the AS

BGP-merged

BGP export policy

#uni/cs-168

#uni/cs-168/routing

#uni/cs-168/routing/inter-domain

choosing routes

therefore ASes should choose the most profitable path
– best: path where next hop is a customer = they pay
– neutral/less good: path where next hop is a peer = no money made
– worst: path where next hop is a provider = you pay

when offered multiple paths, pick the most profitable path

rule

only participate in a route if you get paid ⇒ don't do work for free
→ check if you get paid by looking at neighbours along the path, you get paid if and only if one of the neighbours is a customer

properties

- based on this, routes are **valley-free**, you cannot travel downhill then uphill again
- routes are **single-peaked**, climb strictly up, traverse peer-to-peer links, then climb strictly down
- **reachability**: any two ASes in graph can communicate

routing

#uni/cs-168

#uni/cs-168/routing

- properties
- based on this, routes are **valley-free**, you cannot travel downhill then uphill again
 - routes are **single-peaked**, climb strictly up, traverse peer-to-peer links, then climb strictly down
 - **reachability**: any two ASes in graph can communicate
 - **convergence**: all ASes agree on paths

caveats

if ASes pick arbitrary policies, then cannot guarantee these properties (only for Gao-Rexford rules)

export policy

based on **Gao-Rexford** rules, we only export a learned route if we make a profit

route advertised by

customer

peer

provider

export route to

everyone

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only

customers only