Internet - core networking infra that links all connected computing devices
· ties together networks
Goals: speed, cost, reliability
Purpose: allow two processes on different hosts to exchange data
Asynchronous op, limited by speed of light
★ Key to scaling systems is how to deal with failure

Network: routers/switches, links
Network Stack: networking SW on hosts
· replicates router/switch functionality
Control Plane - mechanisms used to compute routing tables
· global, must know network topology
Data Plane - using routing tables to fwd packets
· depends only on arriving packet and local routing table
Different ISPs control their own autonomous systems
Physical ports: where links connect to switches
Logical Ports - logical places where application connects to network stack
Access to network: open socket, associated w port
Network Stack:
Outgoing: chops data into payloads, puts header on packets
· ensures reliable delivery
Incoming: reconstructs packet payloads into bytestream
· delivers bytestream to appropriate socket
· ensures reliable delivery

Statistical Multiplexing:
peak of aggregate << aggregate of peaks
Sharing on nw:
Reservations: reserve bandwidth for flow
· Circuit Switching
· reserve x Mbps
· send data
· teardown
Packet Switching: each packet contains dest
· link fails, nw recomputes route
· flows do nothing

Circuit	vs Packet
· better app perf	· better efficiency
· more predictable	· simpler state in switches
+ understandable	· easier recovery from failure
	· faster startup

On-demand: higher utilization
We use packet: better for bursty loads
· use gateways to translate between networks via universal protocol

3 Design Principles
Layering: how to break system into modules
7 Application: networking support for apps
4 Transport: (reliable) end to end delivery
3 Network: Global best effort delivery
2 Data link: Local best effort delivery
1 Physical: bits on wire
Host: 1,2,3,4,7 Router: 1,2,3

End to End Principle
· what the network does and doesn't do
Reliability: · quick recovery from failure in hosts
· new failures shouldn't interfere w endpoint semantics
Only if sufficient: implement only if it can be completely done
Necessary: don't do anything hosts can do
Useful: only if it enhances performance w/out burden w/out burdening apps that don't need
E2E ignores operation, security

Fate Sharing Principle
· when storing state, locate it w entities that rely on that state
· failure can only cause loss of critical state if entity that cares about it also fails
IP is unifying protocol that connects networks
Architect for flexibility!
Edge Router - connected to end hosts
Border - connected to routers in another network
Core/backbone - connected to other internal routers
"Valid" Routing State - produces forwarding decisions that always deliver packets to their destinations. no dead ends - no loops
Dest-Based Routing -
· produces spanning tree rooted at destination
if at src, no go
∃ 1 delivery tree ∀ hosts deterministic

Internet Design Goals:
· connect existing networks
· robust in face of failure
· support multiple types of delivery
· accommodate a variety of networks
· allow distributed management
· easy host attachment
· cost effective
· allow resource accountability
Global routing state is valid iff no dead ends, no loops

Tree Topologies - LAN (1)
Algo:
1) Pick a root - smallest ID
2) Compute shortest paths to root
- only keep links on shortest path
- break ties by choosing path w lowest ID neighbor
Spanning Tree Protocol:
· Messages (Y, D, X)
- from node X
- Proposing Y as root
- advertising distance D to root
1) each switch thinks it's the root
2) switches update their view of root based on ID
3) switches compute distance from root (D+1)
4) send update
Robustness:
· root switch sends periodic messages
· detect failures thru timeout - no word from root
Weaknesses:
- requires loop free - waste bandwidth
- slow to react to failures · must recompute tree
- slow to react to host movement · entries must time out
Spanning tree proves!
routing can be done by hosts

Routing on Trees
- flooding
↳ allows every switch to see on which port all other switches are
2 Approaches:
- run routing protocol
· send flood packet to create routing state
- learn on the fly:
· nodes flood packets when routing state isn't there
Each switch can flood or forward depending on whether it has routing information
Learning Switch: when packet arrives, inspect src ID, associate w incoming port
Flooding usually needs a tree
BUT we can use duplicate suppression for non-trees (choose shortest path copy)

Global View (2) intradomain
· create global view in every router
⇒ Link state routing minimize SOME metric
· every router knows its link state
→ state of links to neighbors: up/down, cost
→ Flood link state to all other routers
→ run route computation locally, computing $\boxed{\text{least cost}}$ paths
★ ONLY routing messages are flooded in link state
· don't need tree: routers remember which updates they've seen so they don't resend
Initiate flood when: MAY HAVE DIFF GLOBAL VIEW
· topology change
- link/node up/down
· configuration change
· periodically
There can be transient loops when some routers know about failures before others
- inconsistent link state
↳ performance problems during convergence period

Distance Vector (3) intradomain
· each node announces distance to each dest
neighbor w gives me vector $d(w,v)$ for all nodes v
$d(u,v) = \min_w (c(u,w) + d(w,v))$
u tells neighbors about $d(u,v)$
DS needed:
· cost table - costs to all neighbors
· peer table - copies of each neighbor's forwarding tables
· forwarding table - routes/costs to each host
Send messages when any of your distances change
"Count to Infinity" scenario if link goes down
Poisoned reverse: if you are using next hop's path, tell them not to use your path by setting cost to ∞

Split Horizon: if you use neighbor for path to x, don't send a routing entry for x to that neighbor FULL UPDATES 168
Route Poisoning - send ∞ when you no longer have a path
only need to poison if route advertise
More Complete Version of Protocol:
· Send DV based on timer
· send updates when any value in DV changes
- only send elements that changed
Poisoning necessary w partial updates bc silence means no change

Path Vector (4) interdomain
· Send your paths when advertising to neighbor
· each router makes its own decisions about paths...
Issues: lack of connectivity, lack of convergence

Reliable Transport
Transport mechanism is reliable iff
· resends all dropped/corrupted packets
· attempts to make progress
★ cannot falsely claim to have delivered a packet
Single Packet Case:
· set timer, send packet, resend if no ACK when timer expires
Multiple Packets:
· allow W packets to be in flight at once
· optimal W: $\frac{RTT \cdot B}{\text{Packet Size}}$
"Sliding Window"

Designing Feedback: ACKs
1) ACK individual packets
+ know fate of each packet
+ impervious to reordering
+ simple window
- loss of ACK requires retransmission
2) Full Info Feedback
· Give highest cumulative ACK plus any additional packets
+ as much info as you could hope for
+ resilient form of individual ACKs
- could require sizable overhead in bad cases
3) Cumulative ACK: smaller ACK pkts
· ACK the highest sequence number for which all previous packets have been received
+ resilient to lost ACKs
- confused by reordering
- incomplete info about which packets have been received

Loss
Individual ACKs
· resend when ≥k subsequent ACKs are received
2nd/Full Response: · resend missing packet · continue window based protocol
Full Info
· resend when k subsequent ACKs are received
Cumulative ACKs
· Duplicate ACKs are sign of isolated loss
· stream of duplicate ACKs means some packets are being delivered
Hard to deal w loss

Go Back N
- sliding window
- when loss is detected, resend all W packets starting w loss
- receiver discards all out-of-order packets

Bandwidth: Bits/sec

Propagation Delay: time for one bit to move thru link

Bandwidth Delay Product: # bits in flight at any time. bandwidth · prop delay

Window Sizing Rule: total bits in flight = amount that fits into forward and reverse pipes

Reliability belongs in network stack

Routers care about L2, L3 header
NW stacks care about L2, L3, L4
- Source address is necessary for routers to respond to source with errors

IP Packet Header
- Header Length: # 32 bit words in header
  - usually 5
- Total length: # bytes in packet
  - max size is 65,535 bytes
- Protocol - how to handle packet (higher level protocol) TCP, UDP
- Potential Problems
  - Corrupted Header: checksum — recalculated at every router
  - Loop: TTL (decremented at each hop)
  - Packet too big: Fragmentation
    - reassembly done at destination

IPv4 Frag fields:
- Identifiers: which fragments belong together
- Flags:
  MF: more fragments coming
- offset:
  portion of original payload this fragment contains in 8 byte units

L2: deliver packet within a network
L3: deliver packet between networks

Addresses: used by routers to fwd packets to dest
- used by dest to know packet is for them

Want: scalable routing, efficient fwding

To scale L2:
- nonstandard learning switch routing algorithm between switches
- First hop router adds tag to dst addr, source addr

Which switch is this host at?
If you don't know, send packet to all edge switches

Address: Switch: Host

Mapping between hosts, switches

L3:
- use addr - Network: Host

2 Keys:
1) aggregation
   - single fwding entry for many hosts → ~~Blahs~~ network IP
2) DNS ⇒ IP name
   - IP is locator, contains both network + host

Extending L3:
- add layer: autonomous system
AS: Network: Host

above: ideal, but not reality

Original Internet Address:
1st eight: network
Last 24, host

Next: Classful Addressing

A:	/8	0***
B:	/16	10**
C:	/24	110*
D:	/	1110*
E:		11110

Problem: wasted address space

today: CIDR
∃ address, mask
1s = nw, 0s = host
- mask carried in routing algo
"Prefix" = nw, "suffix" = host

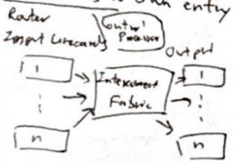
For routing, orig, classful had network addr as entries

Aggregation in CIDR is much more fluid - aggregate by prefix

- CIDR routes on prefixes: key to internet scalability

- Hierarchical address allocation helps routing scalability

- BUT "multi-homed" nw means we need to represent networks w own entry



Control Processor puts fwding table on line cards

Input linecards: receive incoming pkts
- update IP header
  - TTL
  - checksum
  - options
  - fragment

If dest IP not in Fwd task, select default route

Orig, Classful Lookup
- know network address, use to index

CIDR - packet doesn't tell you NW address

Use prefix tree for aggregates, overlaps

Longest Prefix Match - compact representation of tree

In CIDR, must walk down bit by bit
- LPM decreases size of routing task

Problems w IP addr
- multihoming not supported
  - aggregation issues
  - aggregation relies on contiguity

- forwarding hard
- no binding to identity

Solution: hierarchical addressing

MAC Addr: Numerical address associated with network adapter
- 48 bits

Hierarchical Allocation — depend on interface
- 24 bit blocks assigned to vendors
- last 24 bits assigned by vendors

Broadcast Address: FF:FF:FF:FF:FF:FF
↳ Goal is to reach everyone

NW Interfaces only pass up packets that
- match MAC of host
- match MAC of broadcast

⇒ flooded packets read by one host
⇒ broadcast packets read by all — flooded too.

MAC requires no configuration

Tell if destination host is remote:
- mask your IP w dest IP
- if different, remote
: L3 hands packet to L2 w nexthop IP addr as param
- L2 ARP resolves IP into MAC

Discover: — Scalable L2 Routing: Switch: Host (MAC)
IP addr: DHCP
B's IP: DNS — Map host → switch
B's MAC (B = local) ARP
First Hop Router IP: DHCP
First Hop Router MAC: ARP

DHCP allows host to discover
- IP
- Netmask
- IP of local DNS server
- IP of first hop router

DHCP: application layer
1) server maintains IP pool, mask, etc
2) Client broadcasts DHCP discover
   - L2 broadcast to FF:FF..
3) ≥1 server responds w offer (broadcast) IP
4) Client broadcasts DHCP request
   - specifies which offer
   - echos accepted params
5) Selected server responds w ACK: broadcast

Src IP of host during DHCP: 0.0.0.0

IP Level: IP broadcast addr is 255.255.255.255
- uses soft state
- IP has lease
- client must request refresh before lease expires
  ↳ server ACKs
- No ACK ⇒ server down
  - client leases new IP

If nw fails, lease expires

ARP
- each host has list of IP→MAC
- if IP not in table: — all hosts read, but one responds
  - broadcast who has IP?
  - receiver unicast MAC
  - result stored in list

If dest remote:
- set MAC to be MAC addr of first hop router

ARP caches populated only on:
- direct ARP response
- ARP requests for own addr

Broadcasting - scalable bc of limited size

Soft State: forget past, eventually
KEY FOR ROBUSTNESS

IETF: Internet Engineering Task Force
RFC: Request for comments

IPv6 vs v4
- don't deal w problems
  - eliminate fragmentation, checksum
- simplify handling
  - new options mechanism — NEXT HEADER
  - eliminate header length
- provide general flow label
1) expand to dest & back
2) reduce dealing w problems
3) reduce read correctly
4) similar special handling

IPv6

Version	Traffic Class	Flow Label	
Payload Length		Next Header	Hop Limit
Src Addr			
Dest Addr			

Transmission Delay - how long it takes for all bits to get on wire

$$\frac{\text{packet size (bytes)}}{\text{bandwidth}} \text{ (transmission delay)}$$

Propagation Delay - how long it takes to transmit bit from one end to other

$$\frac{\text{length of link (m)}}{\text{speed of light (m/s)}} \text{ (speed of link)}$$

Bandwidth Delay Product: ~~opt~~ window size
Bandwidth · Prop Delay

Queueing Delay: how long packet waits to get transmitted on wire

E2E Delay: sum of all nodal delays

RTT: 2 · E2E delay

Delay = Trans + Prop

Multihoming breaks aggregation

switches do not mess w packets, don't need to change MAC for it.

- L4 bridges gap between abstraction designers want and abstractions networks can support

Optimal window size = bandwidth · RTT

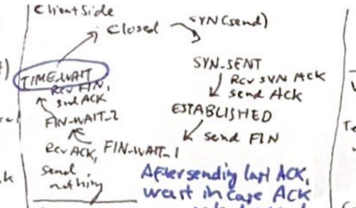
$$\text{delay} = \frac{\text{size}}{\text{throughput}} \text{ (get on wire)} + \text{propagation}$$

Control Program — (Specification) Abstract NW View
Virtualization Layer — Global NW View
Nos — OpenFlow (Fwding)
Switches

NAT - enables many hosts to share 1 addr
- uses port #s, private addr
- socket opened by PS maps to port (transport)
3 special purpose addr blocks:
255.255.255.255/32 - every host on local network: broadcast
127.0.0.0/8: loopback
169.254.0.0/16: not fwded, for single link comm only
10.0.0.0/8, 172.16.0.0/12, 192.168.0.0/16
↳ not routed in public internet
→ assign private addr to hosts behind same NAT
- multiplex using port #s
NAT Port ⇒ IP, port (original)
- violates layering (middlebox) - nw implementing L4
Network stack - shared networking code on host
· relieves burden from host + network L4
Transport Layer:
Demultiplexing: from IP addr to process (es)
Packets → bytestreams (segmentation + assembly)
Reliability
Corruption
Flow Control - rcvr
Congestion Control - network
connection identified by src IP, port, dst IP, port, protocol
UDP - multiplexing between processes
discarding corrupted packets
TCP - ^^
- retransmission of lost/corrupted pkts
- flow/congestion control
- connection set up/tear down
socket ⇔ port (es)
UDP - no connection establishment delay
no connection state (less overhead = more clients)
small packet header overhead
e.g. interactive streaming apps, DNS
TCP: stream-of-bytes service
TCP (segment) data - ≤ MAX segment size
= Max Transmission Unit - IP hdr - TCP header
Seg Num X, Pkt contains B bytes (data)
X...X+B
ACK: next expected byte - always set after SYN
Seg of next packet = last ACK field
Sliding Window Flow Control:
- advertised wnd W
- can send W bytes beyond next expected byte
- prevent overflowing rcvr buffer
No faster than $\frac{W}{RTT}$ $\frac{bytes}{sec}$
Sliding Window: data that has been sent but not ACK'd
- max data in flight
ISN: set from 32 bit clock
- prevents confusion if reusing old port
- rcvr will ignore packet if it doesn't fit in window

Client ——SYN, seq num = X——→ Server
←——SYNACK SeqNum = Y, ACK = X+1——
——ACK ACK = Y+1——→

- timer to wait on SYN
- reclick hyperlink ⇒ new socket, new SYN
Close connection - FIN ← A
- other host ACKS FIN
↳ closes A's side of connection not B's
Both together:
FIN, FIN ACK, ACK
A B A
For abrupt termination:
A sends RESET to B
B doesn't ACK
if RST lost + B sends another, A sends another RST

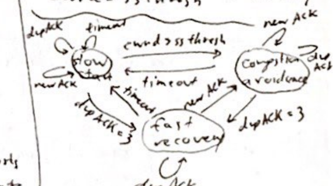


After sending last ACK, wait in case ACK gets dropped for another FIN ACK

Retransmission
- loss w cum ACKs
- dup ACKs show isolated loss
Timeout (RTO)
- retransmit on timeout
- reset when new data ACK'd
short: dup pkts
long: inefficient
· base RTO on RTT estimation
SampleRTT = AckRcvdTime - SendPktTime
EstimatedRTT = α · EstimatedRTT + (1-α) · SampleRTT
Timers: Estimated Deviation = α · Est Dev + (1-α) · Sample Dev
RTO - value you set timer to for timeouts
ETO - current estimate of appropriate "raw" timeout
Use exponential averaging to estimate:
· RTT
Sample Deviation = | Estimated RTT - SampleRTT |
ETO = EstimatedRTT + 4 · Estimated Deviation
- only use clean samples for ETO ☆
ACK includes no retransmitted segments
- when RTO expires, set RTO ← 2 · RTO
- when clean sample arrives, set RTO to ETO
- timeouts are EXPENSIVE
When ACK arrives:
- ACKing data that has already been ACKed
· nothing happens
- ACKing new data:
· RTO set to ETO
· timer reset to RTO, applied to next unACKed
- if none of new data has been retransmitted
- this is a clean sample
- ETO is recalculated, used in value of RTO
DNS
· addresses can change underneath name
· name can map to multiple IP addr
· multiple names for same addr
Goals:
- scaling
- ease of management
- availability + consistency
- fast lookups
Hierarchical namespace
- domains are subtrees
- name is leaf to root path
Zone corresponds to administrative authority responsible for contiguous portion of hierarchy
DNS: hierarchy Root TLD
Authoritative DNS server
· DNS servers are replicated
· every server knows address of root name server
· every node knows address of children
Logical Hierarchy:
- name resolution - walk up/down hierarchy
- name allocation: control over namespace partitioned
DNS records:
(name, value, type, TTL)
Type: A address
hostname ⇒ IP addr
Type: NS: nameserver
domain ⇒ name of DNS server for domain
Type: mail exchanger
domain in email ⇒ name of mail server
canonical name
alias ⇒ actual name
pointer
reversed IP ⇒ hostname
Anycast: routing finds shortest path to dst
- several locations given same addr,
nw delivers packet to closest location
→ 13 root servers, replication via any-cast
recursive vs iterative DNS query
straight shot
local DNS server initiates every request to other DNS servers

DNS cache
- DNS servers cache responses to queries
- response includes TTL
Web is successful bc self-publishing is easy, independent, free
Ted Nelson, Xanadu - proposed network that would allow everything to be connected.
· owners of docs would be automatically paid
Content Display - HTML
Content Reference - URLs
Download Management - HTTP
· turn URLs into TCP flows
URL - naming content
protocol://hostname[:port]/dir path/resrc
HTTP: TCP surrounding HTTP
- client:server architecture
- synchronous request/reply protocol
↳ same HTTP session used
- stateless 1st HTTP request sent w ACK
↳ each request/response treated independently
+ improves scalability
- some apps need state
Cookies
- client side state maintenance
- Sends state in future requests
Use caching + replication for fast downloads, high availability, and to avoid network overload
Naive:
- retrieve each object on page individually
Can do concurrent requests in parallel
- multiple connections
Persistent Connections
- maintain TCP connection across multiple requests
→ avoid overhead
→ more accurate RTT estimate
→ allow TCP congestion wnd to increase
Pipelined Requests/Responses:
- batch requests/responses to reduce num packets
Caching:
- if-modified-since
Send data request at same time as ACK of handshake
Request...
· within TTL, local client cache responds
· else, send if-modified-since to server
Reverse Proxies - done by content provider
- cache documents close to server to decrease server load
Fwd Proxies - cache close to client to reduce nw traffic, latency | ISPs, enterprises
Replication - spread loads across servers
· direct client to particular replica by DNS
Content Distribution Network
- caching/replication aaS
Pull caching - direct result of client requests
Push replication - expectation of high access rate
CDN creates new domains
- client content provider (CNN) modifies content to point to new domains
☆ multiple sites hosted on same infra
Hit rate of cache grows logarithmically w size
CDN: initial request goes to CNN
- embedded links go to CDN
ICMP - Internet control message protocol
→ tell host about nw problems; diagnosis
→ can exploit to elicit nw info; discovery
types: need fragmentation
TTL expired
unreachable
source quench - slow down!
redirect
Path MTU: min e2e MTU uses NF
- guess + check
- can see which routers are how far using time exceeded
Quality of Service - priority scheduling
Congestion Control:
- end hosts adjust sending rate
- based on implicit feedback from network

Detect Congestion Thru Packet Loss 168
· fail-safe that TCP has to detect
· non-congestive loss (checksum errors), reordered pkts are complications
Dup ACKs - isolated loss
Timeout - possible disaster
AIMD - converges to fairness
- embodies gentle increase, rapid decrease
AIAD, MIMD - retains unfairness
MIAD - max unfair
AIMD
$CWND \to CWND + \frac{1}{CWND}$ after 1 RTT, CWN increases by 1 MSS
3rd dupack: $CWND \to \frac{CWND}{2}$
Slow Start: start at 1 MSS increase CWND by 1 on each valid ACK
- start slow but ramp up quickly
→ double CWND per RTT
∀ ACK, CWND += MSS ⇒ CWND = 2 · CWND per RTT
Switch from slow start to AIMD when CWND > ssthresh
Timeout:
· ssthresh ← CWND/2
· CWND ← 1
· retransmit first lost packet
· SS until CWND > SSTHRESH
ACK:
if in slow start: } double cwnd per RTT
CWND += 1 (MSS)
else:
$CWND = CWND + \frac{1}{CWND}$ MSS AIMD w/ increment CWND per RTT
Reset dupACK
dup ACK
dupACK ++
if dupACKCount = 3:
ssthresh = $\lfloor CWND/2 \rfloor$
CWND = $\lfloor CWND/2 \rfloor$
retx!
Fast Recovery solves problem of having to wait a long time before sending new packets in case of packet drop
if dupACKCount = 3:
ssthresh = cwnd/2
cwnd = ssthresh + 3
retx
while in fast recovery:
cwnd = cwnd + 1 ∀ additional dup ACK
receive new ACK ⇒ exit fast recovery
cwnd = ssthresh



$Thruput = \frac{MSS}{RTT} \cdot \sqrt{\frac{3}{2p}}$,
p = packet drop rate
$\frac{\text{Average \# pkts in flight}}{RTT}$
ASes
· stub: AS that sends/receives packets on behalf of its users
· transit - carries packets for other ASes
Tier 1, 2, 3
Require policy support, autonomy, policy
Relationships:
- customer:provider
AS B carries A's traffic for a fee
- peer
B, A carry each other's traffic for free
1) don't carry traffic if not being paid
2) save/make money when sending traffic
☆ AS topology reflects business relationships between ASes
BGP: Border Gateway Protocol
Concurrent requests ⇒ no proxy caching

BGP implemented by AS border routers

→ per dst route advertisements

BGP: • best route based on policy, not shortest distance
- path vector routing used to avoid paths, allow flexible policies
- selective route advertisement ⇒ reachability not guaranteed even if graph connected
- BGP may aggregate routes for scaling

Selection: which path to use
- how traffic leaves nw

Export: which path to advertise
- how traffic enters nw

Favor customer > peer > provider

Prefix Advertised By	Export to:
Customer	everyone
Peer	Customers
Provider	Customers

Advertise all paths, to customers. To others, advertise only paths to customers

Gao-Rexford
- graph of customer-provider relationships are acyclic
  ⇒ can arrange providers in a hierarchy
- top of provider chain is a tier 1 provider
- selection never causes unreachability
- export CAN
- border routers speak BGP

eBGP - BGP sessions between border routers in different ASes

iBGP - BGP sessions between routers in same AS
- distribute externally learned routes internally

IGP - intradomain routing protocol
- provide internal reachability

Each router has 2 routing tasks
- IGP for internal locations
- iBGP for egress router

Messages:

Open
- establish BGP connection (uses TCP)

Notification
- report unusual conditions

Update
- inform neighbor of new routes
- inform neighbor of old routes that become inactive

Keep Alive
- inform neighbor connection is still viable

<IP prefix: route attributes>

→ announcements: new/update

→ withdrawal: remove routes

Attributes:

ASPATH
- lists all ASes a route advertisement has traversed (in reverse order)

Local Pref
- value that represents preffered paths
- local to AS, carried only in iBGP

MED - multi exit discriminator
- for ASes connected via 2 or more links
- specify how close a prefix is to link it's announced on

IGP cost
- used for hot potato routing
- each router selects closest egress point based on path cost

∃ asymmetric routes bc MED ≠ IGP

Route selection priority:

1) LOCAL PREF　　4) eBGP > iBGP
2) ASPATH　　5) iBGP path
3) MED　　6) router ID

Issues
- security
  - AS can claim to serve a prefix it doesn't → "blackhole"
- convergence (if not Gao-Rexford)
- policy oscillation
  - not in accordance w Gao Rexford
- many more paths must be calculated in BGP
  → slow response/convergence
- misconfigurations
  - each router is configured individually
- policies are changing

Traffic Engineering: way of distributing load on the network
- define traffic aggregates: set of flows that follow same path
  e.g. pipes, tunnels
- traditional routing get packets to/from pipe

Multiprotocol Label Switching
- way of establishing pipes
- insert label before IP header
- route on larger aggregate, or finer granularity

MPLS header between L2, L3
- edge routers inspect IP header, insert MPLS label
- core routers route based on MPLS label

edge: all intelligence/functionality

core: dumb plumbing providing connectivity

Multicast: send single packet that is delivered to multiple locations
- no link sees more than one copy of packet
- allow receivers to come + go w/out source needing to keep track

Anycast: sending packet to one of a set of possible locations
- only one copy of packet delivered

Broadcast: sending packet to all hosts
- many copies of packet delivered

Multicast: sending packet to multiple recvrs (those that want it)
- join multicast group w address G
- can implement at different layers

Link　nonmembers can send
- NIC listens for packet sent to multicast address G (MAC)
- send: like broadcast
  → just over single subnet

IP　DVMRP - different tree for each source
- uses L2 multicast
- portion of IP addr space used for multicast
- open group membership
  - send IGMP message
    internet group management protocol
- use delivery tree for multicast

1) use reverse paths
- arriving packet has dst, src
  if ∃ route to src thru port P, shortest forward out other ports
  else
  drop　state: MMR

2) prune tree when there are subtrees w no group members
- router knows whether it has local members
- if all children of router send non-membership report, prune
- propagate...

Core Based Trees - members send on the non send to core
- pick rendezvous point for group
- build tree from all member to core using fwd path unicast routing
- no scalability/flooding issues

Middleboxes - hit a router that's part of tree is top
- more complex/diverse functionality
- keep state
- smarter traffic processing

Network Function Virtualization
- deploy network functions as VMs on racks
- spin up as needed

Challenges: speed, scaling, ease of writing

Enables edge computing.

Programmable Forward Chips allow for defining the parsing/control flow for pkts in high level language

Edge computing/programmable switches debate

Original Goals for Control Plane
- basic connectivity: routing
- interdomain policy: policy compliant

VLANS - virtual LANs, tags in header

Access Control Lists allow routers to limit access to hosts

SDN motivation: control plane is a mess

Networking has yet to extract simplicity

Need SDN because:
- large datacenter
- multi-tenancy, i.e. each customer gets own network

Data Plane has layers

Control plane has no layers

SDN - abstractions for subtasks

Forwarding Abstraction

OpenFlow standardized interface to switch
→ current proposal for forwarding
1) switches accept external control messages
2) standardized flow entry format

Network State Abstraction

Global Network View
- implemented using Network OS
- info from routers/switches to form "view"
- config to routers/switches from controller to control forwarding
- centralized controller using NOS API

Specification Abstraction
- specify goals, not implementation

Spec Abs

Control: express goals on abstract view

Virt Layer: implement goals on network state

NOS: API driven by NW state abstraction

SDN localizes complexity
- simplifies interface for control program

Routing
- given graph of network, compute routes
  ↳ pass to switches

Access Control
- control program decides who can talk to who
- SDN platform adds appropriate ACL flow entries

Given virtual switch, can enable SDN first hop switch for all VMs
- hypervisors control multiple VMs
- NOS on servers

Layering allowed:
- simultaneous innovation at different layers
- allowed internet to scale
- made internet a recursive overlay

Connectivity between domains done through BGP
- coupled inter/intra domain (IP)
- domains could have made change easier
  ↳ first internet source of management

"L3.5" - interdomain layer on edge routers
- decouple L3, L3.5
- different domains can use different L3s

⇒ multiple L3.5 designs
- domains decide when to support L3.5
- can arrange remote peering with others
- different designs allow for evolution

Trotsky

Logical Pipe:
- next header field that identifies L3.5 design
- connect every host to edge w pipe

Bootstrap Mechanism
- tells hosts which L3.5 supported
- learn logical pipe technology

netAPI
- OS provides API that allows applications to request any L3.5 design

Deployment
- domain adds edge support
- OS vendors add OS support
- app vendors modify nw calls to use new L3.5
- hardest step is to deploy L3.5 designs at edge

NFV creates edge to edge principle, restores simple core
↳ moves functionality out of hardware into software at edge
- deploy INs as VMs at domain edge

Trotsky enables new architectures, designs

Edge Computing - computing at edge for application-specific tasks

Open Carrier Interface - allow third parties to provide edge services on demand

2 step process for permanent evolution
1) adopt Trotsky
2) ongoing development of new architecture
- move from client-server to client-edge-server

Availability - will nw deliver data?

Authentication - who's sending?

Integrity - do msgs arrive in og form

Provenance - who is responsible for/created this data

Public Key Crypto:
- auth: signature
- privacy: encryption
- integrity: hash function
- provenance: signature

Harming Availability:
- natural outages
- external: prevent from fn-ing
- internal: compromise routers
- DoS

↳ Defense:
- fight fire w fire
  - spam good packets
- permission to send
  ↳ huge change
- shut up packets (NIC)
  ↳ tell host not to send for period of time

IP header length includes TCP/IP header lengths

headers added to inner part of packet
- L2 header
- L3 header
- L4 header