CS4226 Cheatsheet AY24/25 —— @JasonYapzx

1. Performance

Network Performance Metrics

- **Link Rate/Bandwidth/Capacity:**
  - Cable - 1 100 Mbps, Fiber optics - 1 100 Gbps
  - Ethernet - 3 Mbps to 100 Gbps
  - How many bits can be 'pushed' onto a link per unit time
  - How fast we can place bits onto the physical medium delivering
- **Throughput**
  - Throughput of a TCP/UDP flow
  - How many bits can be communicated per unit time
  - How fast the bits get delivered through the route (regardless of the medium)
- **End-to-end delay**
  - Components
    - Processing + Queueing Delay + Transmission + Propagation
  - Related: bandwidth, packet and queue size, distance, prop. speed
  - Source → Destination
- **Response Time — Round Trip Time**
  - Source ⇄ Destination
- **Others**
  - Dropping probability
  - Utilization (percentage of time link is busy)

Packet Switching

Compared to *circuit-switching*, it allows more users to use the network at once.

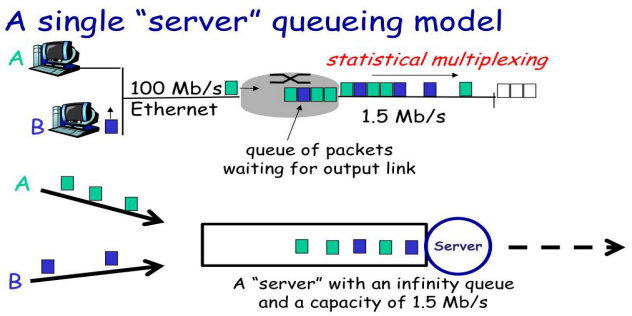
- Great for async bursty data
  - resource sharing — simpler, no call setup
- **excessive congestionn:** packet delay and loss
  - protocols needed for reliable data transfer, congestion control
- reserved resources (circuit) vs on-demand allocation (packet)

Packet-switching store-and-forward

Entire packet must arrive at router before it can be transmitted onto the next link $Delay = \frac{L}{R}$, if L = packet size and link rate R

Statistical Multiplexing

- Sequence of A & B has no fixed timing pattern
  - Bandwidth shared on demand: *statistical multiplexing*
- Time Divison M.: each host gets same slot in revolving TDM frame



Little's Law: $L = \lambda \times W$

- Arrival rate into the system λ:
  - $\lambda \overset{def}{=} \lim_{t\to\infty} \frac{N(t)}{t}$
- Average sojourn time (time W_i for i^{th} customer spent in) W:
  - $W \overset{def}{=} \lim_{n\to\infty} \frac{1}{n} \sum_{j=1}^{1} W_j$
- Time average# of customers in system L:
  - $L \overset{def}{=} \lim_{t\to\infty} \frac{1}{t} \int_0^t L(s)ds$
- Avg # of customers in system = arrival rate × avg. sojourn time

2. Network Queueing Models

Model packet flow: Arrival Pattern

- T_i are independent and identically distributed **random variable**
- Inter-arrival time: $T_i \triangleq t_{i+1} - t_i$

Random Experiment

Consider a random experiment whose outcome cannot be determined in advance

- **Sample space** S: set of all outcomes
- **Event** E: $\subset$ of sample space, event E occured if outcome $s \in E$
- **Probability** function $P(E)$
  - $0 \le P(E) \le 1, P(S) = 1 \Longrightarrow P(\bigcup_{i=1}^{\infty} E_i) = \sum_{i=1}^{\infty} P(E_i)$
  - For any sequence of events $E_1, E_2 \ldots$ that are *mutually exclusive*

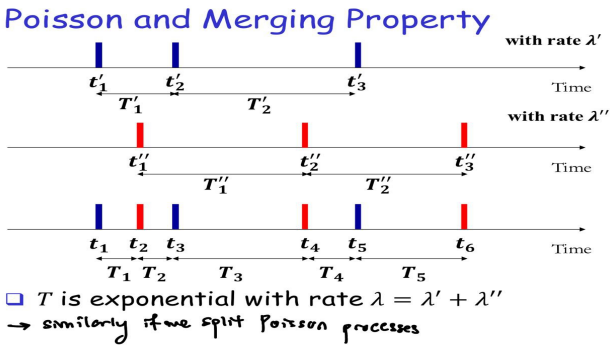
Random Variable

- **Random variable**: function assigning real value to each outcome $s \in S$: For any set of $A \subseteq \mathbb{R}$: $P\{X \in A\} \overset{def}{=} P(X^{-1}(A))$
- **Distribution function** F of r.v. X is defined on any *real number* x by: $F(x) \overset{def}{=} P\{X \le x\} = P(X^{-1}((-\infty, x]))$
- A random variable is *continuous* if there exists a **probability density function** (pdf) $f(x)$ such that: $f(s) \overset{def}{=} \frac{d}{dx} F(x)$
- 2 random variables are *independent* if realization of 1 does not affect probability distribution of other: $f_{X,Y}(x, y) = f_X(x) f_Y(x)$
- **Expectation or mean** of random variable X:
  $E[X] \overset{def}{=} \int_{-\infty}^{+\infty} x f(x) dx$ **or** $E[X] \overset{def}{=} \sum_{x=-\infty}^{+\infty} x P\{X = x\}$

Exponential Distribution

Continuous random variable T follows/has an **exponential distribution** with parameter $\lambda > 0$ if for $x \ge 0$, where $\frac{1}{\lambda}$ is the frequency

- $F(x) = P\{T \le x\} = 1 - e^{-\lambda x}$ or $\bar{F}(x) = P\{T > x\} = e^{-\lambda x}$
- $f(x) = \frac{dF(x)}{dx} = \lambda e^{-\lambda x}$
- Average: $E[T] = \int_{-\infty}^{\infty} x f(x) dx = \frac{1}{\lambda} \Longrightarrow$ inter-arrival rate
- **Memoryless property:** $P\{T > s + t | T > s\} = P\{T > t\}$

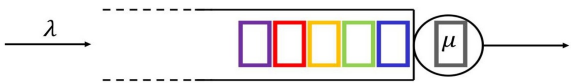


Model packet flow: Service Time

- Packets have varying length
  - takes different amt of time to process, same packet given different link capacity/rate OR packets with different lengths under fixed links
- Service time S_i
  - processing time of packet i under fixed link rate
  - follows i.i.d r.v. S with mean $E[S] = \frac{1}{\mu}$

M/M/1 Model

- single server with queue of ∞ size
- poisson arrival with rate λ
- exponential i.i.d service time with rate μ
- arrival/service times independent FIFO service discipline



Result

- Utilization ρ: percentage of time that *server* is busy OR
- probability of random observation finds server busy
- general result (applies but not limited to M/M/1 Model)
- $\rho = \frac{\lambda}{\mu}$ need condition for $\lambda < \mu$ for system stability

Main Result (without proof)

- π_i = % of time exactly i packets or customers in system → server + queue
- $P\{L = i\}$ that random observation finds i packets in the system
- For M/M/1 system, we have $\pi_i = P\{L = i\} = p^i(1-p)$
  - $\pi_0 = \rho^0(1-\rho) \Rightarrow 1 - \rho \to$ % of time server idle/no packets in system
  - Discrete distribution → density function $= P\{L = i\} = \rho(1-\rho)^i$
  - Follows a geometric distribution
- **System + Queue**
  - Average no. of packets in system $E[L] = \frac{\rho}{1-\rho}$
  - Average sojourn time of packets $E[W] = \frac{1}{\mu - \lambda}$
- **Queue only**
  - Average no. of packets in the queue $E[Q] = \frac{\rho^2}{1-\rho}$
  - Average queueing delay of packets $E[D] = E[W] - \frac{1}{\mu}$

When $\lambda = \mu$ it is unstable, $E[L], E[W] \to \infty$, because of the randomness.

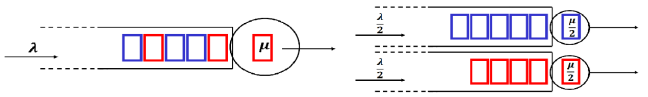
Throughput of system

- General system, throughput is $\min(\mu, \lambda)$
- Stable system $\mu < \lambda$: accept more packets into the system then service them, thus it will be bounded by λ
- Throughput & queueing delay **positively correlated** if λ constant

Effective Bandwidth

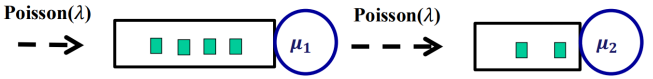
- **Phyiscal link capacity:** theoretical processing limit of hardware
- **Effective bandwidth of the link:** actual throughput that can be achieve / quality of service

Statistical Multiplexing vs TDM



- Each Poisson stream its own queue $(\frac{\lambda}{k}, \frac{\mu}{k})$ or shared queue (λ, μ)
- $\rho = \frac{\lambda}{\mu} = \frac{\lambda/k}{\mu/k} \Longrightarrow$ same server utilization
- $E[W] = \frac{1}{\mu} \frac{1}{1-\rho} \Longrightarrow$ avg. sojourn time for each user ↑ by $\times k$ under TDM
- $E[Q] = \frac{\rho^2}{1-\rho}, E[L] = \frac{\rho}{1-\rho} \Longrightarrow$ avg. no. of waiting + total users are also k times larger

Burke's Theorem



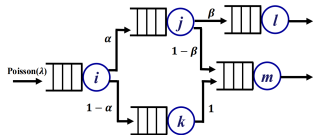
If M/M/1 system with arrival rate λ starts in steady state

- departure process is *Poisson* with rate λ
- number of customers in system at any time t is *independent of sequence* of departure times prior to time t

Tandem Queues

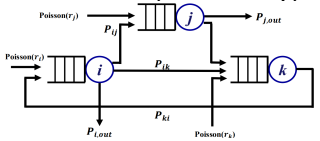
- Utilization of each server i becomes $p_i = \frac{\rho}{\mu_i}$
- By independence, joint probability:
  $P\{L_1 = j, L_2 = k\} = P\{L_1 = j\} \cdot P\{L_2 = k\} = \rho_1^j(1-\rho_1)\rho_2^k(1-\rho_2)$

Acyclic Network with Probabilistic Routing



- Diff. flows may take different paths.
- Each link shared by different flows

Jackson Network (Feedback Loop)



- P_{ij}: join queue that leaves system
- P_{ik}: Probability to join another system that will be a feedback loop to rejoin system

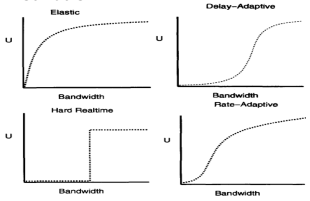
Solving the Jackson Network

- Effective arrival rate λ_i:
- outside arrival directly to server i, i.e. r_i + feedback arrivals (other servers)
- (stable system, leaving system rate = arrival rate $\approx lambda_j$)
- $\lambda_i = r_i + \sum_{j=1}^{n} \lambda_j P_{ji}$
- **Matrix form:** $\lambda = r + \lambda P \Rightarrow \lambda = r(I - P)^{-1}$

3. Resource Allocation

- flow 1 rate λ_1, flow 2 λ_2, link capacity μ packets/second respectively
- divide capacity into $\mu = \mu_1 + \mu_2$, serving 2 flows separately
- packet flows might have different *average delay*
- given delay gurantee, acheived throughput: $\lambda = \mu - \frac{1}{E[W]}$

Motivation



- internet is free, suitable for elastic services: `Telnet`, `FTP`, `DNSm SMTP`
- not suitable for real-time applications: VoIP, Skype, Zoom
- allocate different amt of resources to different application flows
- U_i (user's happiness) as a function of performance metrics

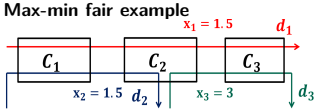
Fairness

- equal share of resources: split allocation equally based on available resources
  - Equal share amongst all: Demand $\boldsymbol{d} = (d_1, d_2, d_3, d_4) = (3, 4, 5, 6)$ Allocate $\boldsymbol{x} = (x_1, x_2, x_3, x_4) = (2.5, 2.5, 2.5, 2.5)$
  - ***Fulfill small demands completely (equal share higher demands):*** Demand $d = (1, 4, 5, 6)$ Allocate $x = (1, 3, 3, 3)$ (d_1 fulfilled)
- **feasible solution**
  - for links with capacity C_1, C_2, C_3, and allocation $x = (x_1, x_2, x_3)$
  - $0 \le x_i \le d_i$ for all i, where flow demand $= d_i$
  - $x_1 + x_2 \le C_1$; etc. (the flows which go through the links should have their total allocation less than link capacity)

Max-Min Fairness

- feasible allocation is max-min fair *iff.* increase of any rate within feasible domain **must be at a cost** of decrease of an already smaller/equal rate
- $\boldsymbol{x}$ is max-min fair if for any feasible $\boldsymbol{y}$, if $y_i > x_i$ then $\exists j$ s.t. $y_i < x_j \le x_i$
- **always exists** a max-min fair solution, and **always unique**

Max-min fair example



- $C = (3, 6, 7)$ $d = (2, 4, 4)$

- Resource C_3 not bottleneck
- C_2 Fairshare: $x_1 = x_2 = x_3 = 2$
- C_1 Fairshare: $x_1 = x_2 = 1.5$
- $x_1 = x_2 = 1.5$
- $x_3 = C_2 - x_1 - x_2 = 3$

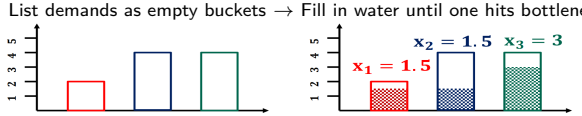
Bottleneck Resource

- Assume allocation $\boldsymbol{x}$, in network case, resource $\boldsymbol{r}$ is bottleneck for flow $\boldsymbol{i}$ *iff.*
- resource r is *saturated* (fully allocated to all flows)
- flow i has the *maximum* rate amongst all flows using resource r
- flow i CANNOT get more resources from link r if allocation is fair (otherwise hurts flows with lower rates)

⋆ Theorem: when each flow has ∞ demand in a network system, a flow allocation is max-min fair *iff* every flow has a bottleneck resource

Waterfilling Algorithm

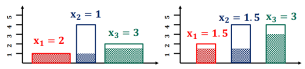
1. List demands as empty buckets → Fill in water until one hits bottleneck



Weighted Max-Min Fair Share

- Extension weight vector: $\boldsymbol{\phi} = (\phi_1, \phi_2 \ldots)$
  - no customer receives more than demanded
  - unsatisfied demand split resource proportional to their weights
- $C = 16, \boldsymbol{d} = (4, 2, 10, 4)\phi = (2.5, 4, 0.5, 1)$
  - fair share $x_i = \frac{\phi_i}{\Sigma_i \phi_j} C \Rightarrow \boldsymbol{x} = (5, 8, 1, 2)$
  - given the upper bound on demand $\Rightarrow x_1 = 4, x_2 = 2$, distribute the rest 10 proportionally $(3.33, 6.66) \Rightarrow x_4 = 4, x_3 = 6$

❒ $\boldsymbol{C} = (3, 6, 7)$, $\boldsymbol{d} = (2, 4, 4)$ and $\boldsymbol{\phi} = (2, 1, 2)$



- buckets has a width of ϕ_i and volume of d_i
- Max-min solution (same example as before) for $\phi = (1, 1, 1)$

4. Software Defined Networking

Implement and manage networks based on new fundemental principles

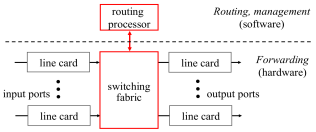
Networks vs Other Systems

- computation and storage have been virtualized → flexible + managable infra
- networks still hard to manage → heavily rely on network admins
- hard to evolve due to new innovations in systems software → OS, languages
- routing algorithms change very slowly, network mgmt extremely primitive

Network Control Problem

- Compute the configuration of each phyiscal device
- Operate w/o communication gurantees & network-level protocol (`RIP`, `OSPF`)

Router/Switch at the Core



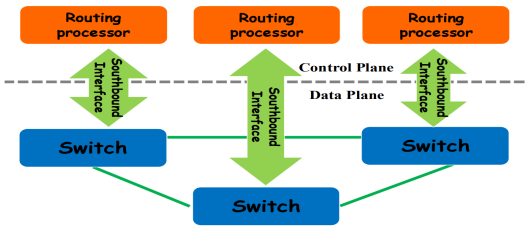
- run *routing algorithm* protocols (RIP, OSPF, BGP)
- *forwarding datagrams* from incoming to outgoing link

Control Plane vs Data Plane

- **Control:** establish router state
  - determine how/where packets are forwarded
  - routing, traffic engineering, firewall
  - slow time-scales (per control event)
- **Data:** process/deliver packets
  - based on state in routers and endpoints
  - e.g. IP, TCP, Ethernet etc
  - fast timescales (per packet)
- **Control Plane:** Routing Table (RIB) → **Data Plane:** Forwarding Table (FIB)

Principle 1: Disaagregation

- Separation of Control/Data plane → should be codified in an open interface



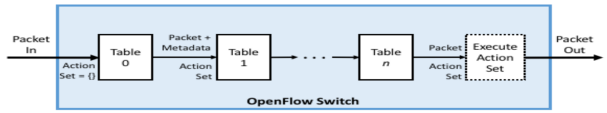
- **Implications**
  - Network operators able to purchase control/data planes from vendors X/Y
  - Data plane consists of cheaper commodity forwarding devices (bare-metal switches) BUT Needs to define a forwarding abstraction
    - ∗ general purpose way for control plane to tell data plane to forward pkts in particular way → eg. OpenFlow's Flow rules
    - ∗ Flow rule is match-action pair, any packet matched have the *associated action* applied to it
- **Benefits**
  - New market landscape and value shift → shift control from vendors to operators that build networks to satisfy users' needs
  - Opportunities for fast innovations → independent evoluation/developmenet, software control of network can evolve independently of hardware

Control vs Configuration

- **Control:** making real-time decisions about how to respond to link/switch failures. Learn about failure and provide remedy in ms.
- **Configuration:** Operators need ot configure switches and routers, using CLI to update RIB. Interface is capable of installing new routes, which on surface seems equivalent to isntalling new flow rule.
  1. run software that implements control plane *on-switch*
     - implies switches operate autonomously, communicate with peer switches throughout network to construct local routing tables
  2. make control plane physically decoupled from data plane
     - implies control plane implemented *off-switch*, possible to make it logically centralised

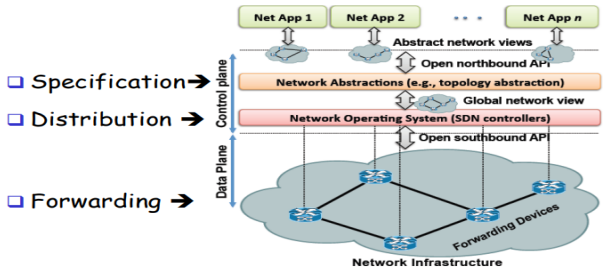
Principle 2: Centralised Control

- Centralised decisions easier to make
- From FIB to Forwarding pipeline
  - each table focuses on subset of header fields(may be involved in flow rule)
  - pkt processed by mulitple tables sequentially, determine how its forwarded



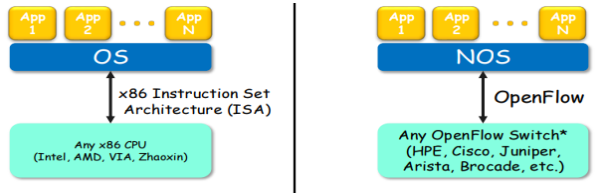
- How to implement data plane?
  - fixed-function data plane
    - ∗ as header fields are well-known + easy to compute offsets in every pkt
    - ∗ inital idea was purposely data plane agnostic-SDN, focused on opening control plane to ***programmability***
  - programmable data plane
    - ∗ performance optimisation, potential changes to protocols
    - ∗ **Easy network management:** management goals as policies, debug/check behaviour easily
    - ∗ **Rapid innovation and fast evolution:** enable new services and better performance, detailed configurations are done by the controller
    - ∗ **Control shift:** vendor → operators → users

Three layers of abstractions

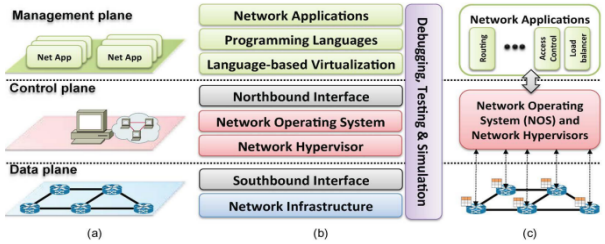


- **Specification abstraction (northbound API)** → Allow control app to express desired network behaviour without implementation
- **Distribution abstraction** (internal to control plane) → shield SDN apps from distributed states, making distributed control logically centralized
- **Forwarding abstraction** (southbound to open interface) → Allow any forwarding behaviour desired by apps + hiding details of underlying hardware

Networking Operating System (NOS)



OS provides high-level abstractions that make it easier to write applications, an NOS makes it easier to implement control functionality



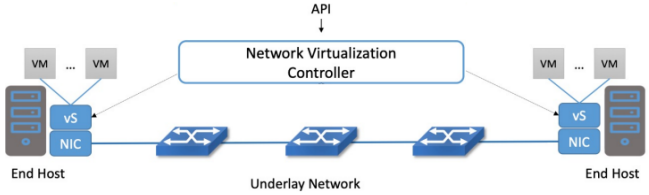
5. Use Cases, OpenFlow and ONOS

Users of SDN:

1. Cloud Providers: Google, Facebook, Microsoft, Opensource Components
2. Network Operatros: Comcast, AT&T, NTT
3. Enterprises: universities/private companies → managed edge services/SDN

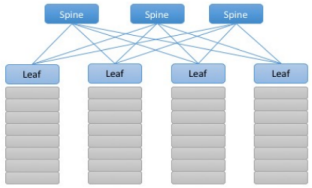
Case 1: Network Virtulization

- Exisiting virtualization solutions
  - compute virtualization: VMs, containers — networks: VPNs/VLANs
  - limted scope: virtualizing the address space
- Insight: need for modern cloud: networks to be programmatically created/managed (*without manual configuration*)
  - Disagregation: single API entry point to create/modify/delete VNs
  - Virtual Networks (VNs) has its own private address spaces



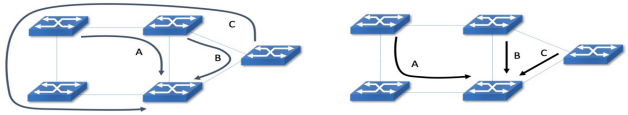
Case 2: Switching Fabrics

- **Cloud Datacenters:** lower costs, newer features
- **Leaf-spine topology:**
  - 2-hop multi-path for rack to rack
  - 2-hop for intra-rack server-to-server path
  - 4-hop for inter-rack server-to-server path



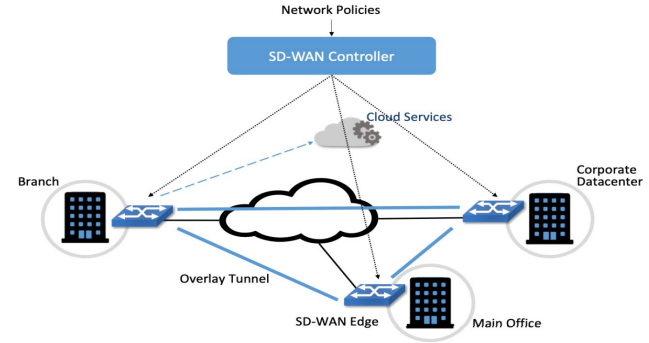
Case 3: Traffic Engineering

- For wide-area links between datacenters



- Traffic classes with priorities → delay tolerance vs availability requirements

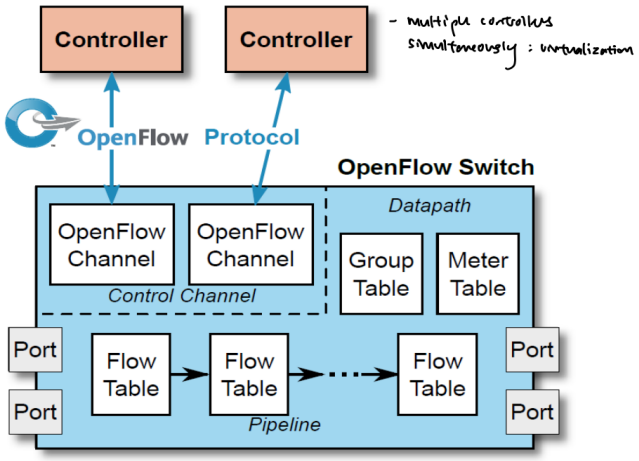
Case 4: Software-Defined WANs



Use Cases

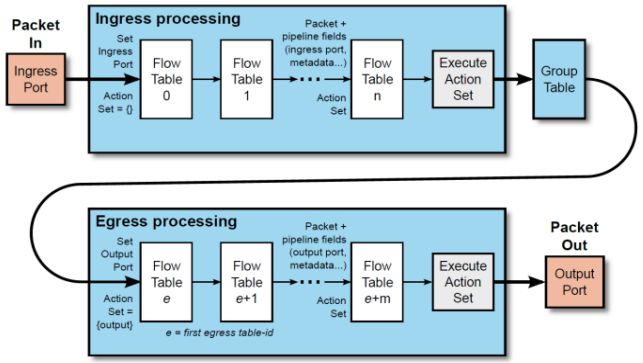
- Killer applications: network virtualization, datacenter/cloud computing
- More applications like: (1) last-mile access networks, (2) software-defined internet exchange (SDX), (3) mobility and wireless, (4) security, measurement and monitoring

OpenFlow Protocol and Switch



Main components of an OpenFlow switch

- **OF protocol:** Open southbound API, **OF Switch:** forwarding abstraction



Packet flow through the processing pipeline

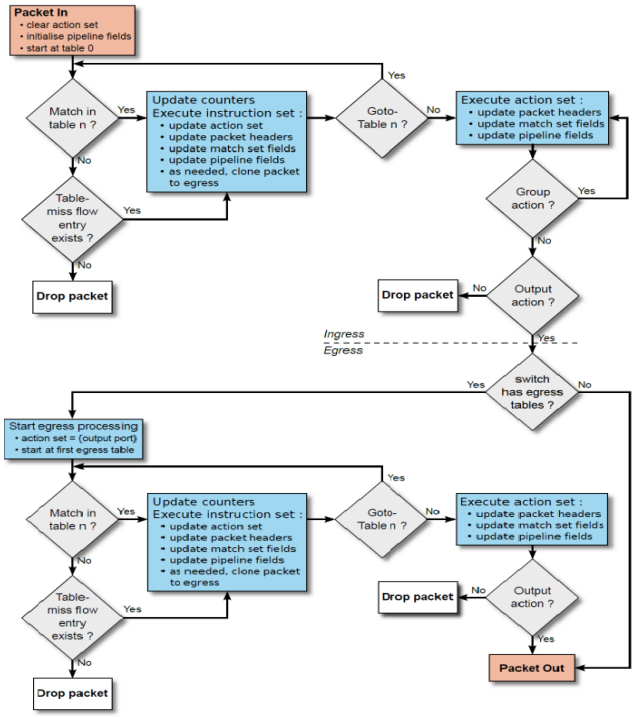
Terminology for each packet from packet flow

- **Header and header field**
- **Pipeline fields:** values attached to packet during pipeline processing → e.g. ingress port and metadata
- **Action:** operation that acts on packet: drop/forward to port/modify TTL
- **Action set:** acumulated while processed by flow tables, executed at the end of the pipeline processing

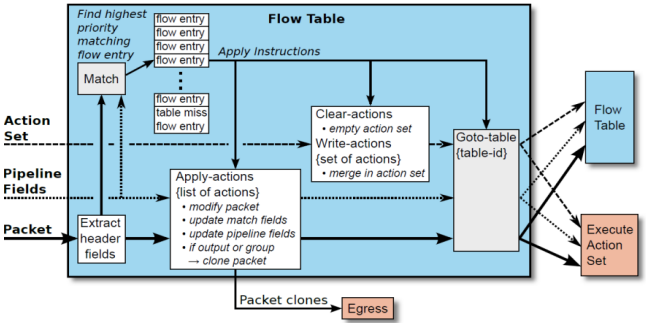
OpenFlow flow entry

- Flow entry in flow table looks like:
  - `Match Fields` | `Priority` | `Counters` | `Instructions` | `Timeouts`
- Match field: packets are matched against
  - Header fields and pipeline fields, may be wildcarded (any) or bitmasked (subset of bits)
- Priority: used to choose from multiple matches
- Instruction Set
  - contains list of actions to apply immediately, and a set of actions to add to the action set
  - Modify pipeline processing (go to another flow table)

- Default entry: table-miss flow entry → send packet to control plane OR drop the packet

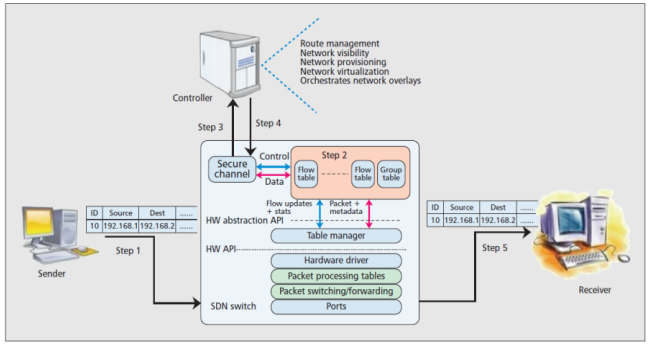


Matching and instruction execution

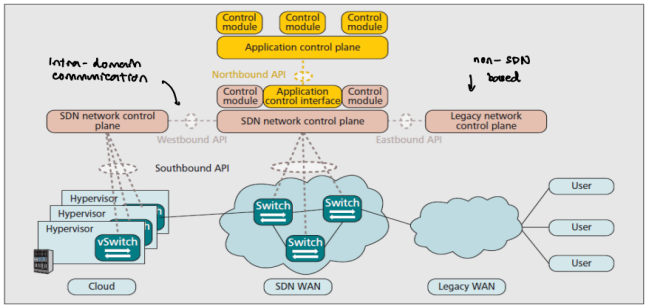


Operation of SDN (controller-switch)

Reactive paradigm: relying on dataplane → control plane



Interfaces of an SDN



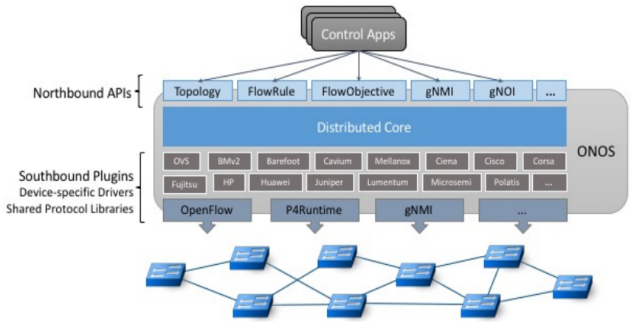
ONOS Control Plane

- ONOS model: Open Northbound API, specification abstraction
- ONOS system: Distribution abstraction
- ONOS abstractions:
  - **Intent:** high-level intents, network-wide, topology-independent programming constructs → e.g. intent to custom IP reach target IP
  - **Flow Objective:** finer-grained control, device-centric, programming constructs and also *pipeline-independent*
  - **Flow Rule:** use to control various pipelines, fix-function/programmable

Network Operating System (NOS)

- Like any other horizontally scalable cloud application
- Consists of a set of loosely coupled subsytems
  - Each for an aspect e.g. topology, host tracking
  - maintains its own service abstraction
- often associated with a micro-service architecture
- includes a scalable and highly available key/value store

Architecture of ONOS



- **Northbound Interfaces(NBI)**
  - apps use to stay informed about *network state* (topology, intercept packets) AND to *control network* data plane
- **Distributed Core**
  - responsible for managing network state, notifying apps about state changes
  - internal: scalable key/value store (Atomix)
- **Southbound Interface (SBI)**: constructed from a set of plugins including shared protocol libs/device-specific drivers

6. P4 SDN

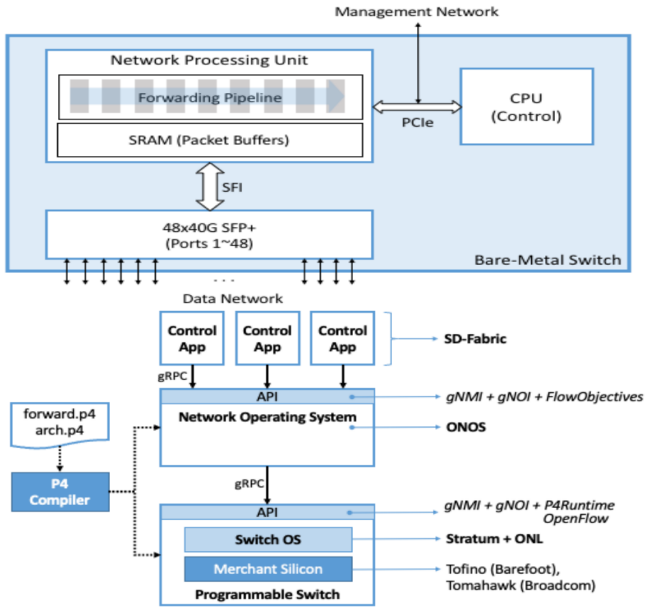
Issues with OpenFlow

- Data-plane protocol evolution requires frequent changes to standards ($12 \rightarrow 40$ OpenFlow match fields now)
- Limited interoperability between vendors (OpenFlow / netconf / JSON / XML variants) & Limited programmability

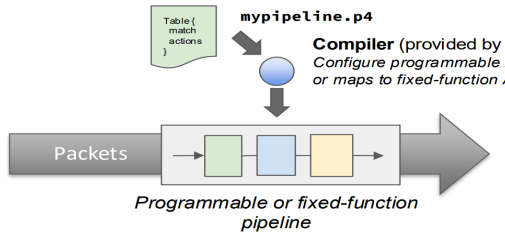
Software Stack of an SDN + Bare-metal switches

Bottom is a bare-metal switch, with Switch OS on top of the Merchant Silicon

- Each packet is processed by the NPU through:
  - Multi-stage Forwarding pipeline, stages are fixed-function/programmable



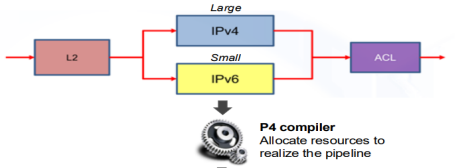
P4

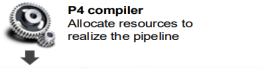


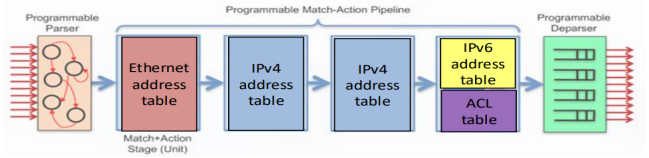
- **Domain-specific language** to formally defined dataplane pipeline
  - Describe protocol headers, lookup tables actions, counters
  - Can describe fast (e.g. ASIC, FPGA) / slow pipelines (SW switch)
- **Good for programmable switches/fixed-function ones**
  - Defines 'contract' between control plane/dataplane for runtime control

Protocol Independent Switch Architecture

- **Parser:** declare headers that should be recognized + their order in the packet
- **Match-Action Pipeline:** define tables and the exact processing algorithm
- **Deparser:** declare how output packet will look on the wire
- Process:
  1. Packet parsed into individual headers (parsed representation)
  2. Headers and intermediate results can be used for matching/actions, can be modified, added or removed
  3. Packet is deparsed (serialized) after the pipeline



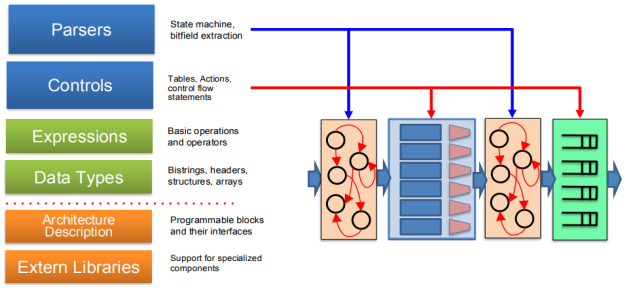




From logical to physical

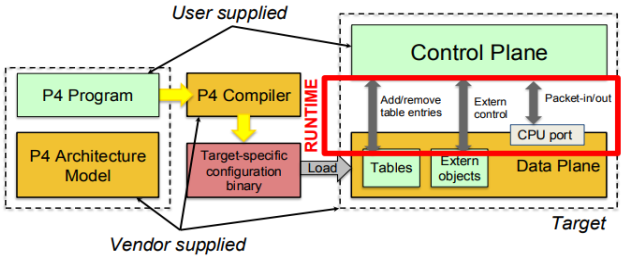
- P4 needs to account for different chips with different physical pipelines
- If only one logical pipeline, P4 compiler's job is easy, but marketplace not converged, many vendors for high-speed NPUs
- Allow vendor specific logical pipeline, through different architectures (`arch.p4` provided by vendors) $P4_{14} \rightarrow P4_{16}$

$P4_{16}$ Language Elements



P4_16 approach

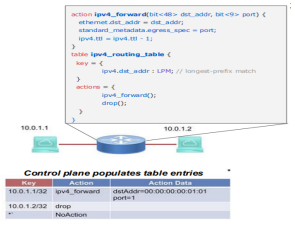
Programming a P4 Target



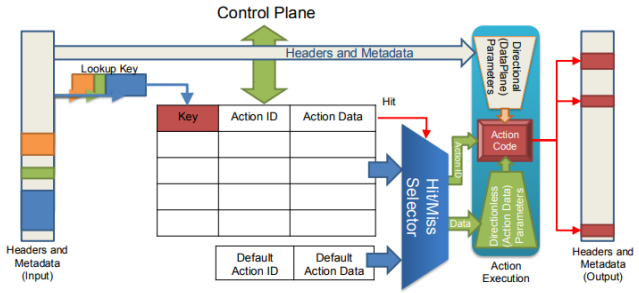
- **P4 Target:** An embodiment of a specific hardware implementation
  - specific device/platformn (sw/hw), refers to actual entity processing network packets according to P4 code
  - *purpose:* provides concrete execution env where P4 program deployed
  - *characteristics:* has own hw capabilities, vendor-specific optimization/constraints affect how P4 code is mapped/executed
- **P4 Architecture**: Provides an interface to program a target via some set of P4-programmable components, externs, fixed components
  - high-level abstraction define how packets flow through programmable pipeline, describes logical pipeline as general strucure and processing behaviour of P4 programs

Behavioral Model (bmv2)

- `bmv2` allows developers implement their own P4-programmable architecture as a software switch
- think of `bmv2` as the hardware of the switch, `psa_switch` target supports PSA architecture, `simple_switch` supports V1Model
- **simple_router.p4**
  - *Data plane program (P4):* Defines match-action tables, performs lookup and executes chosen runtime
  - *Control plane (runtime):* Populates table entries with specific information, based on config, automatic discovery and protocol calculations



Tables: Match-Action Processing

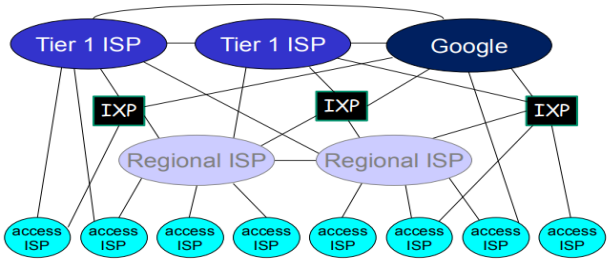


7. Internet Interconnection

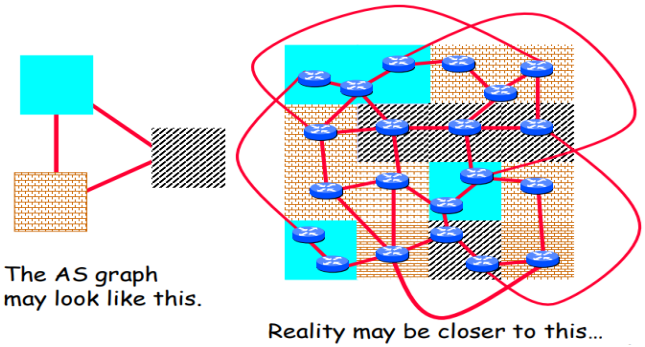
Autonomous System

- Network of interconnected routers, identified by unique AS number (ASN)
- Control by a single administrative domain (company can have multiple ASN)
- Use common routing protocol and policies
- *How to obtain AS number?*
  - IANA: Internet Assigned Numbers Authority → consists of 5 regional internet registries OR AS Databases

Internet structure: Network of Networks



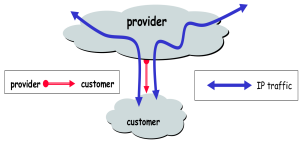
- at center: small # of well-connected large networks
  - tier-1 commercial ISPs: (e.g., Level 3, Sprint, AT&T, NTT), national & international coverage
  - content provider: (e.g., Google): private network that connects it data centers to Internet, often bypassing tier-1, regional ISPs



Internet Peering — How are they connected?

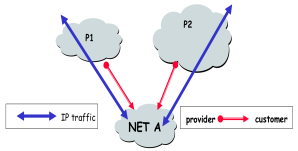
- *peering:* voluntary interconnection of separate internet networks: purpose of exchanging traffic between 'down-stream' users of each network
- bilateral agreement between neighbor ASes
- depends on business relationships: (1) customer-provider r/s or (2) peer-to-peer (settlement-free peering) r/s

Customer and Providers



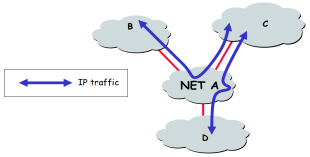
- Customer pays provider for access to internet, reachable from anyone
- Provider provides *transit service* for the customer

Nontransit vs Transit ASes



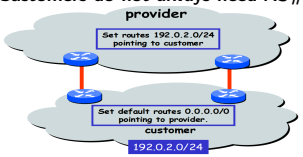
- customer does not allow traffic to go through it
- NET A has 2 providers: **multi-honing**
- traffic should NEVER flow from P1 through NET A to P2
- non-transit AS might be a corporate/campus network/'content provider'

Selective Transit



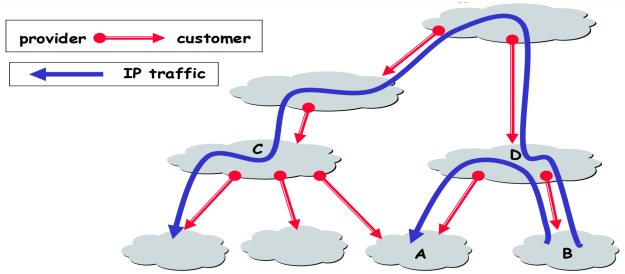
- NET A provides transit between B & C and C & D
- Net A DOES NOT provide transit between D & B
- Most transit networks transit in a selective manner

Customers do not always need AS#



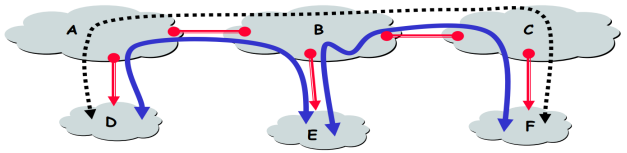
- Static routing is the most common way of connecting autonomous routing domain to the internet.
- If not helping other networks exchange traffic → not needed to have any AS#

Customer-Provider Hierarchy



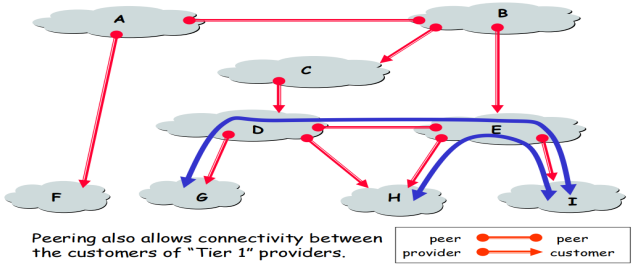
❑ A multi-home with C and D, one of which is a backup

Peer-to-peer Relationship



❑ Peers provide transit between their respective customers
❑ don't provide transit between peers
❑ often don't pay each other (the relationship is settlement-free)

Peering Provides Shortcuts



- At top there are Tier 1 ASP
  - These AS does not have any upstream providers, do not pay other ASP for transit services → any pair of them must have a peering connection

Peering Dilemma

- **To Peer**
  - Reduce upstream transit costs, improve end-to-end performance
  - Be the only way to connect sutomers to some part of internet (tier-1)
- **Not Peer**
  - Rather have customers than peers, usually peers are competition
  - Peering relationships may require periodic renegotiation

Tier 1 ASes/ISPs

- Have access to entire Internet through settlement-free peering links
- Top of customer-provider hierarchy, typically large (inter)national backbones
- Have no upstream provider, peer with one another to form a full-mesh ($\approx 10-12$ ASes → AT&T, Sprint, Level3)

Other ASes

- Lower layer providers (tier-2)
  - provide transit to downstream customers, need ≥ 1 provider of their own
  - typically have national or regional scope → include $1000+$ ASes
- Stub ASes:
  - Do not provide transit service, but connect to upstream provider(s)
  - Most ASes are like these ($\approx 85-90\%$) e.g. NUS

Valley-free Property

- **Valid AS paths (from routing tables)**
  - single peak (uphill + downhill)
  - single flat top (uphill + 1 peering + downhill)
  - any sub-paths of above are valid
- **Invalid patterns**
  - provider → customer → peering
  - provider → customer → provider
  - peering → peering
  - peering → provider

8. Inter-Domain Routing and Policy

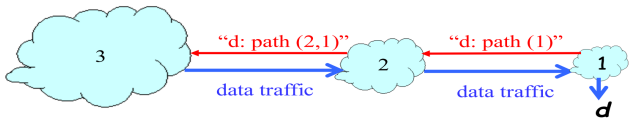
Challenges for Inter-domain Routing

- **Scale**
  - millions of routers, $200,000+$ prefixes
  - $35K+$ self-operated networks, $50K+$ ASes
- **Privacy:** ASes don't want to expose internal topologies/business relationships with neighbours
- **Policy**
  - No Internet-wide notion of a link cost metricc
  - Each AS needs to control over where it sends traffic and who can send traffic through it

Routing Algorithms

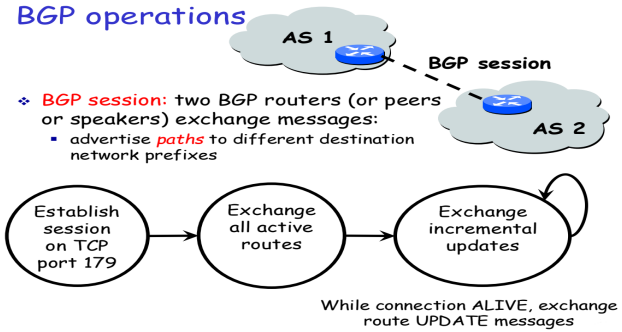
- **Link state algorithm:** all routers have complete topology, link cost info
  - Global/centralized: Dijsktra's — Open Shortest Path First (OSPF)
  - Limitation:
    - Topology information flooded → high bandwidth/storage overhead, nodes divulge sensitive information
    - Entire path computed locally per node → high processing overhead
    - Minimize notion of total distance → only if policy shared/uniform
- **Distance vector algorithm:** router knows connected neighbours' link costs
  - Decentralized algorithm: Bellman-Ford, iterative process of computation, exchange info with neighbours — Routing Information Protocol (RIP)
  - Advantages: hide details of network topology, next hop determined/node
  - Disadvantages: minimize some notion of total distance, slow convergence

Path-Vector Routing



- Extension of distance-vector routing, support flexible routing policies
- advertise entire path, *DV* send distance metric per destination d
- *PV* send entire path for each destination d
- Faster loop detection:
  - Node can easily detect loops, check if itself is in path
  - Node can then discard paths with loops

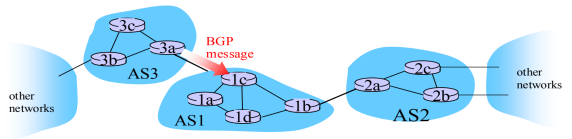
Border Gateway Protocol (BGP)



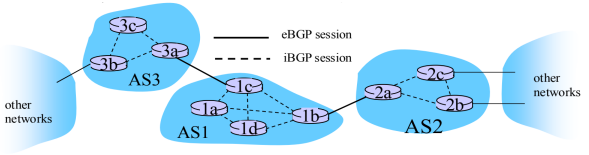
- Allows subnet to advertise its existence to rest of internet: 'I am here'
- Allows ASes to determine good routes from other networks based on reachability info/policy

BGP/IGP model used in ISPs

- **eBGP:** exchange reachability info from neighbor ASes



  - exterior BGP peering (eBGP), between BGP speakers in different ASes → should be directly connected, no intermediate hop
  - AS3 advertises an IP prefix to AS1:
    - AS3 promises it will forward packets towards that prefix
    - AS3 can aggregate prefixes in its advertisement
- **iBGP:** propagate reachability info across backbone; carry ISP's own customer prefixes



  - peers within an AS; not required to be directly connected ⇒ IGP (RIP or OSPF) takes care of inter-BGP speaker connectivity
  - iBGP peers must be (logically) fully meshed
    - used to originate connected local networks
    - used to pass on prefixes learned from outside the AS
    - do not pass on prefixes learned form other iBGP speakers
  - 1c can use iBGP to distribute prefix info to all routers in AS1; 1b can re-advertise info to AS2 over eBGP

BGP Messages

- `OPEN`: opens TCP connection to peer and authenticates sender
- `UPDATE`: advertises new paths (or withdraws old paths)
- `KEEPALIVE`: keeps conn. alive in absence of `UPDATES`/ACKs OPEN req
- `NOTIFICATION`: reports err. in previous mssages/used to close conn

UPDATE Message Format

Marker (16)	
Length (2)	Type (1)
Withdrawn Routes Length (2)	Withdrawn Routes (variable)
Path Attribute Length (2)	Path Attributes (variable)
Network Layer Reachability Information (variable)	

- Withdrawn Routes: IP prefixes for the routes withdrawn
  - Can withdraw multiple routes in an UPDATE message
  - Network Layer Reachability Information (NLRI): IP
- prefixes that can be reached from the advertised route
  - Can only advertise one feasible route for the NLRI
  - IP prefixes are coded more compactly (refer to RFC)

Withdrawn Routes

- No expiration timer for the routes like RIP
- Invalid routes are actively withdrawn by the original advertiser
- Or use UPDATE message to replace the existing routes
- All routes from a peer become invalid when the peer goes down

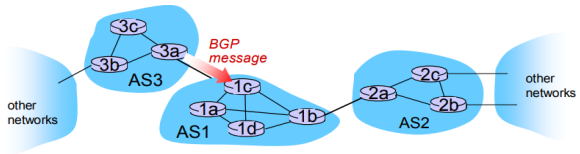
BGP Path Attributes

1. well-known mandatory: every single route update MUST HAVE
   - `ORIGIN`: conveys the origin of the prefix, learned from
     - i (IGP): an interior gateway protocol such as RIP/OSPF
     - ? (INCOMPLETE): unknown source
     - e (EGP): an exterior gateway protocol → BGP (used to be EGP)
   - `AS_PATH`:
     - contains ASes through which NLRI has passed
     - expressed as a sequence e.g. AS 79, AS 11 or a set
   - `NEXT_HOP`: indicates IP address of the router in the next-hop AS
2. well-known discretionary
   - `LOCAL_PREF`, `ATOMIC_AGGREGATE`
3. optional transitive: have to be propagated along routing path
   - `AGGREGATOR`, `COMMUNITY`
4. optional non-transitive
   - `MULTI_EXIT_DISC (MED)`
5. Some implementation rules:
   - must recognize all well-known attributes
   - mandatory attributes must be included in `UPDATE` msg that contain NLRI
   - once a BGP peer updates well-known attributes, it must pass to its peers

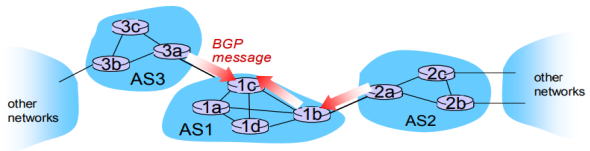
How do entries get in forwarding table

Ties together hierarchical routing with BGP/OSPF, provides overview of BGP

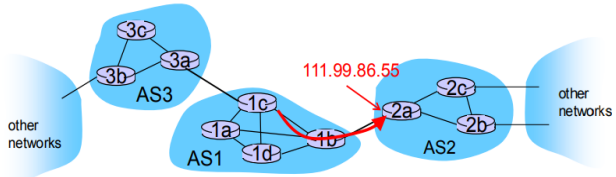
- Router becomes aware of IP prefix



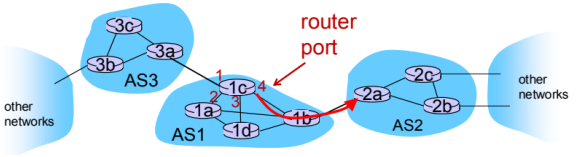
  - BGP message contains "routes"
  - route = prefix + attributes: AS-PATH, NEXT-HOP,... Example: route:
    - `Prefix`: 138.16.64.29/22; `AS-PATH`: AS3 AS131;
    - `NEXT-HOP`: 201.44.13.125



  - Router may receive multiple routes for the same destination prefix
  - Router has to select **one** route
- Router determines output port for IP prefix
  - Select best BGP route to prefix: based on shortest AS-PATH
  - Use selected route's `NEXT-HOP` attribute

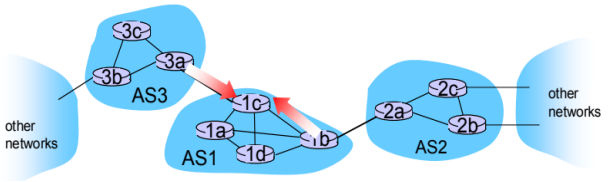


    - Route's `NEXT-HOP` attribute is the IP address of router interface that begins the `AS PATH`
    - e.g. `AS-PATH`: AS2 ··· AS98; `NEXT-HOP`: 111.99.86.55
    - Router uses OSPF to find shortest path from 1c to 111.99.86.55
- Router enteres the prefix-port in forwarding table



  - Identifies port along the OSPF shortest path
  - Adds prefix-port entry to its forwarding table → (138.16.64/22, port 4)

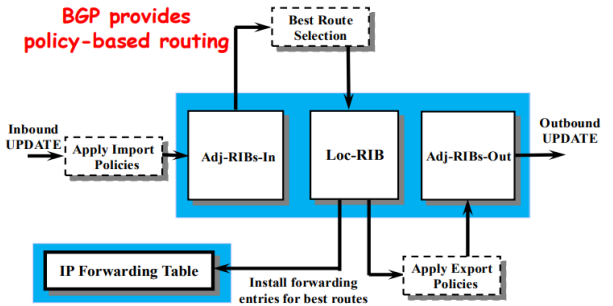
Hot Potato Routing



- If exists multiple best inter-domain routes
- Then choose route with closest NEXT-HOP
  - use OSPF to determine which gateway is closest (as interdomain routing highest cost)
  - Q: From 1c, chose AS3 ··· AS98 or AS2 ··· AS98?
  - A: route AS3 ··· AS98 since it is closer

BGP Routing Information Bases

- Route = prefix + attributes = NLRI + path attributes
- All routes in a BGP speaker:
  - Routing information Bases (RIBs)
  - RIBs = Adj-RIBs-In + Loc-RIB + Adj-RIBs-Out
  - **Adj-RIBs-In:** unprocessed routes from peers via inbound UPDATE; input for decision making
  - **Loc-RIB:** selected local routes used by the router
  - **Adj-RIBs-Out:** selected for advertisement to peers



BGP applying policy to routes

- **Import policy:**
  - filter unwanted routes from neighbor → prefix your customer doesnt own
  - used to rank customer routes over peer routes

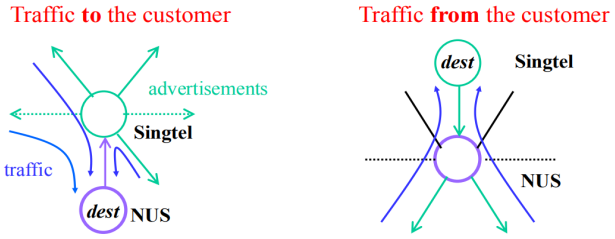
- manipulate attributes to influence path selection → assign local preference to favored routes
- **Export policy:**
  - filter routes you don't want to tell your neighbor → export only customer routes to peers & providers
  - manipulate attribute to control what they see → make paths look artificially longer (AS prepending)

BGP how is policy used

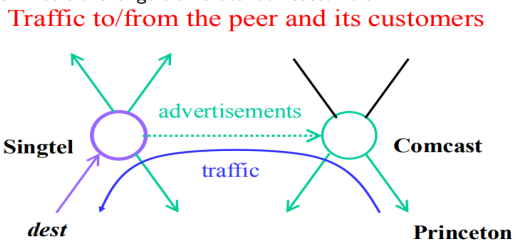
- Objectives: used by commercial ISPs to
  - fulfill bilateral agreements with other ISPs, minimize monetary costs (or maximize revenue) AND ensure good performance for customers
- Bilateral agreement between neighbor ISPs
  - defines who will provide transit for what
  - depends on business relationships: Customer-provider/Peer-to-Peer r/s

Business Relationships

- **Customer-Provider:** Customer pays provider for access to Internet

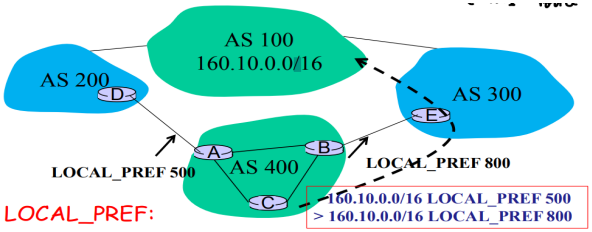


  - provider exports customer's routes to everybody
  - customer exports provider's routes to customers
- **Peer-to-Peer:** Peers exchange traffic between customers



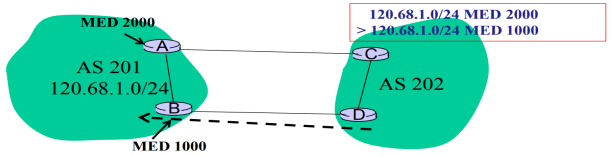
  - AS exports only customer routes to a peer
  - AS exports a peer's routes only to its customers

`LOCAL_PREP` attribute



- 4-byte unsigned integer (default value 100)
- for a BGP speaker to inform its other internal peers of its degree of preference for a route
- should include in UPDATE messages that are sent to internal peers; should not send to external peers
- used to coordinate within its *OWN* AS, not transitive, larger value, more this route is preferred

`MULTI_EXIT_DISC` attribute



- 4-byte unsigned integer (default value 0)
- for a BGP speaker to discriminate among multiple entry points to a neighboring AS to control inbound traffic
- if received over eBGP, may be propagated over iBGP, but must not be further propagated to neighboring ASes

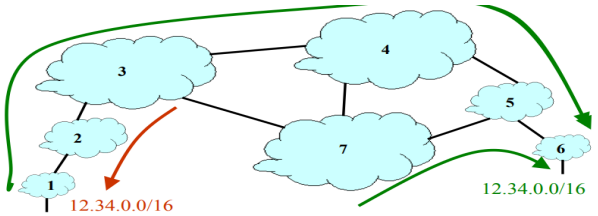
`COMMUNITY` attribute

- Described in RFC 1997 → 4-byte integer value
- Used to group destinations → each destination could be member of multiple communities
- Very useful in applying policies within and between ASes
- import and export policies based on the `COMMUNITY` attributes

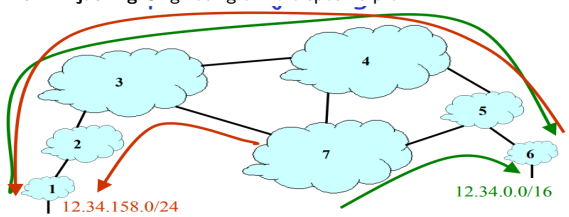
BGP Best Route selection

1. Calculation of degree of preference
   - If the route is learned from an internal peer, use `LOCAL_PREF` attribute or preconfigured policy
   - Otherwise, use preconfigured policy
2. Route selection (recommended process)
   - Highest degree of `LOCAL_PREF` (or the only route to the destination), and then tie breaking conditions on:
   - Smallest number of AS numbers in `AS_PATH` attribute
   - Lowest origin number in `ORIGIN` attribute
   - Most preferred `MULTI_EXIT_DISC` attribute
   - Routes from eBGP are preferred (over iBGP)
   - Lowest interior cost based on `NEXT_HOP` attribute

BGP Prefix Hijacking



- Blackhole: data traffic is discarded
- Snooping: data traffic is inspected, and then redirected
- Impersonation: data traffic is sent to bogus destinations
  - **SubPrefix Hijacking** Originating a more-specific prefix



  - Every AS picks the bogus route for that prefix
  - Traffic follows the longest matching prefix
- **Preventing SubPrefix Hijacking**
  - Best common practice for route filtering
    - each AS filters routes announced by customers
    - e.g., based on the prefixes the customer owns
  - But not everyone applies these practices
    - hard to filter routes initiated from far away
    - so, BGP remains very vulnerable to hijacks
  - Industrial trends
    - Route Origin Authorisation (ROA)
    - Resource Public Key Infrastructure (RPKI)