CS4226 Internet Architecture

Probability

- **Sample space**, S: the set of all outcomes.
- **Event**, E: a subset of the sample space.
- **Probability function**, $P(E)$:
  1. $0 \le P(E) \le 1$.
  2. $P(S) = 1$.
  3. $P(\bigcup_{i=1}^{\infty} E_i) = \sum_{i=1}^{\infty} P(E_i)$ for any **mutually exclusive** events $E_1, E_2, \ldots$.
- **Conditional probability**: $P\{E_i | E_j\} := \frac{P\{E_i \cap E_j\}}{P\{E_j\}}$.
  - **Joint probability**: $P\{E_i \cap E_j\} = P\{E_i|E_j\}P\{E_j\}$.
  - **Bayes' rule**: $P\{E_i|E_j\} = \frac{P\{E_j|E_i\}P\{E_i\}}{P\{E_j\}}$.
- **Law of total probability**: let events $E_1, \ldots, E_n$ be a *partition* of the sample space, i.e., $\bigcup_i E_i = S$ and $\forall i, j,\ E_i \cap E_j = \varnothing$. For any event E,

$$P\{E\} = P\{E \cap S\} = P\left\{\bigcup_i (E \cap E_i)\right\}$$
$$= \sum_i P\{E \cap E_i\} = \sum_i P\{E|E_i\}P\{E_i\}.$$

- **Random variable**, X: a function that assigns a real value to each outcome $s \in S$.
  - For any set of real numbers $A \subseteq \mathcal{R}$, $P\{X \in A\} := P(X^{-1}(A))$.
  - **Distribution function**: $F(x) := P\{X \le x\} = P(X^{-1}((-\infty, x]))$.
  - An r.v. is continuous if there exists a **probability density function** s.t. $f(x) := \frac{d}{dx}F(x)$.
  - **Independence**: 2 rvs are independent if the realization of one does not affect the probability distribution of the other, $f_{X,Y}(x,y) = f_X(x)f_Y(y)$.
  - **Expectation/Mean** of a rv X: $E[X] := \begin{cases} \int_{-\infty}^{\infty} x f(x)dx & \text{(continuous rv)} \\ \sum_{x=-\infty}^{\infty} xP\{X = x\} & \text{(discrete rv).} \end{cases}$
- **Exponential distribution**: a continuous rv T follows an exp. distribution with parameter $\lambda > 0$ if for $x \ge 0$,

$$F(x) = P\{T \le x\} = 1 - e^{-\lambda x} \qquad (\text{or } \bar{F}(x) = P\{T > x\} = e^{-\lambda x})$$
$$f(x) = \frac{dF(x)}{dx} = \lambda e^{-\lambda x}$$
$$E[T] = \int_{-\infty}^{\infty} x f(x)dx = \frac{1}{\lambda}$$

  1. **Memoryless property**: $P\{T > s + t | T > s\} = P\{T > t\}$.
  2. **Merging property**: merging two arrival patterns gives rise to a process which is also Poisson, with $T = \min(T_1, T_2)$ and rate $\lambda = \lambda_1 + \lambda_2$.
  - Comparison between rvs: $P\{X > Y\} = \frac{\lambda_Y}{\lambda_X + \lambda_Y}$.
- **Geometric distribution**: discrete distribution with density function $P\{L = i\} = p(1-p)^i$.
  - Used to model the random number of coin flips of tails until you get a head.
  1. **Memoryless property**: $P\{L > s + t | L > s\} = P\{L > t\}$.

Performance

- **Link rate/capacity/bandwidth**: # bits that can be pushed onto a link per unit time.
- **Throughput**: # bits that can be communicated (i.e., transferred e2e) per unit time.
- **End-to-end delay**: related to
  a. Packet size.
  b. Queue size.
  c. Bandwidth.
  d. Distance.
  e. Propagation speed.

  Comprises of:
  1. **Processing delay**: time for processing data into packets, affected by [a].
  2. **Queuing delay**: affected by [b].
  3. **Transmission delay**: time taken to place data onto the link, affected by [a,c].
  4. **Propagation delay**: time taken by the last bit of the packet to reach the destination, affected by [d,e].
- **Little's law**: $L = \lambda W$.
  - **Average # customers in system**: $L = \lim_{t \to \infty} \frac{1}{t}\int_0^t L(s)ds$ (as a deterministic limit).
    - $L(t)$: # customers in the system at time t.
  - **Arrival rate**: $\lambda = \lim_{t \to \infty} \frac{N(t)}{t}$ (as a deterministic limit).
    - $N(t)$: # customers arrived till time t.
  - **Average sojourn time**: $W = \lim_{n \to \infty} \frac{1}{n}\sum_{j=1}^{n} W_j$ (as a deterministic limit).

Network Queueing Models

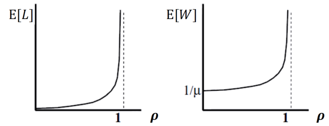
- **M/M/1 model**: **M**: Markovian arrival time, **M**: Markovian service time, **1**: single server.
  - Assumes a **queue of infinite size** (i.e., no packets are dropped).
  - **Arrival time**: Poisson with rate λ.
  - **Service time**: exponential i.i.d. with rate μ.
  - Arrival and service times are **independent**.
  - **FIFO** service discipline.
- Notations:
  1. Arrival:
     - **Arrival time** of i^{th} packet: t_i.
     - **Inter-arrival time**: $T_i = t_{i+1} - t_i$.
       - T_i's are **independent and identically distributed (i.i.d.) exponential** as T with mean $1/\lambda$.
       - The arrival pattern is a **Poisson process**.
     - **Arrival rate**: $\lambda = 1/E[T]$.
  2. Service:
     - **Service time**, S_i: follows i.i.d. rv S, with mean $1/\mu$.
     - **Service/Processing rate**: $\mu = 1/E[S]$.
- Results:
  1. **Utilization**, ρ: % time server is busy, or probability that a random observation finds the server busy,
$$\rho = \frac{\lambda}{\mu}.$$
     - For system stability, we require $\lambda < \mu$.
     - If $\lambda \ge \mu$, the system will be permanently utilized, and the queue will grow indefinitely.
  2. **Percentage of time that exactly** i **packets are in the system**, π_i:
$$\pi_i = P\{L = i\} = \rho^i(1-\rho).$$
     - $\pi = (\pi_0, \pi_1, \ldots)$.
     - $\sum \pi_i = \frac{1-\rho}{1-\rho} = 1$ ($\because$ sum of infinite geometric series, $S = \frac{a}{1-r}$).
     - π_i follows a **geometric distribution**.
- Relationship to Little's law ($L = \lambda W$):
  1. **Average # packets** in an M/M/1 queueing system: $E[L] = E[Q] + E[X]$ (as an rv).
     - $E[Q]$: avg # packets in queue.
     - $E[X]$: avg # packets in server.
       - $E[X] = \sum_{x=0}^{1} xP\{X = x\} = P\{X = 1\}$, since the M/M/1 model assumes a single server.

$$\begin{aligned} P\{X = 1\} &= P\{L - Q = 1\} && \text{\# in server = \# in system - \# in queue} \\ &= P\{L > 0\} && \text{see derivation below} \\ &= 1 - \pi_0 && \because \pi_0 \text{ rep. \% time system is idle} \\ &= \rho && \because \pi_0 = 1 - \rho \end{aligned}$$

       Thus, $E[X] = \rho$ also rep. the probability of the system being busy.
       - Derivation of $\{L - Q = 1\} = \{L > 0\}$:

$$\{L - Q = 1\} \subseteq \{L > 0\} \qquad \because L = 1 + Q \Rightarrow L > 0$$
$$\{L > 0\} \subseteq \{L - Q = 1\} \qquad \because \text{server cannot be idle when queue is nonempty} \quad \therefore \text{if system is nonempty, some pkt is in the server}$$
$$\therefore \{L - Q = 1\} = \{L > 0\}$$

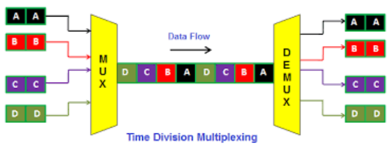
  2. **Arrival rate**: $\lambda = 1/E[T]$ (as a function of an rv).
  3. **Sojourn time**: $W_i = t_i^d - t_i$.
     - t_i^d: **departure time** of i^{th} packet.
     - In a queueing system, $W_i = D_i + S_i$ (as an rv).
       - D_i: sojourn time in queue (i.e., $E[D]$ = mean queueing delay).
       - S_i: sojourn time in server (i.e., $E[S]$ = mean service time).
- From the perspective of the system (i.e., queue + server), we have:
  - **Average # packets in system**: $E[L] = \frac{\rho}{1-\rho}$.
  - **Average sojourn time** for single entrance: $E[W] = \frac{1}{\mu - \lambda}$.
    - **Higher capacity** μ accommodates **higher throughput** λ and provides **lower delay** $E[W]$.



As $\rho = \lambda/\mu \to 1$, $\lambda = \mu \Rightarrow$ waiting time tends toward $\infty \Rightarrow$ unstable system.
- From the perspective of the queue, we have:
  - **Average # packets in queue**: $E[Q] = \frac{\rho^2}{1-\rho}$.
  - **Average queuing delay**: $E[D] = E[W] - \frac{1}{\mu}$.

Statistical multiplexing vs TDM

- Bandwidth:
  - **Physical link capacity**: the theoretical processing limit of the hardware.
  - **Effective bandwidth**: actual achievable throughput on the link, depends on utilization ρ and quality of service.
- **Packet-switching networks**:
  1. **Store-and-forward**: entire packet must arrive at the router before it can be forwarded.
  2. **Statistical multiplexing**: link bandwidth is shared on demand.
     - Distinct from **time-division multiplexing (TDM)** used in circuit-switching networks, where each host gets the same slot in a revolving TDM frame.



(+) **More users** are allowed to use the network, as compared to circuit-switching.
(+) Great for (asynchronous) bursty data.
(−) **Excessive congestion**: protocols are needed for reliable data transfer and congestion control.
- **Circuit-switching networks**:
  - In **TDM**, each Poisson stream has its own queue, with params $(\frac{\lambda}{k}, \frac{\mu}{k})$. Compared to statistical multiplexing,
    1. Both have the **same server utilization**: $\rho = \frac{\lambda}{\mu} = \frac{\lambda/k}{\mu/k}$.
    2. **Average sojourn time** is k **times larger** under TDM, since $E[W] = \frac{1}{\mu - \lambda} = \frac{1}{\mu}\frac{1}{1-\rho}$.
    3. **Average # queueing pkts** is k **times larger** under TDM, since $E[Q]$ is the same with SM for each sub-queue.
    4. **Total # pkts in system** is k **times larger** under TDM, since $E[L]$ is the same with SM for each sub-queue.

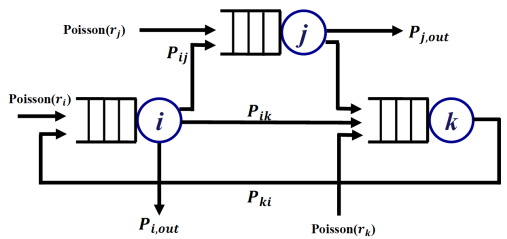
  $\therefore$ **SM is more efficient than TDM** in terms of resource utilization: (1) lower sojourn time, (2) lower average # packets in system and queue.

Tandem queues

Burke's theorem: if an M/M/1 system with arrival rate λ **starts in a steady state**,
1. the **departure process** is Poisson with rate λ, and
2. **# of customers in the system** at any time t is *independent* of the sequence of departure times prior to t.
- When queues are in tandem:
  - **Utilization** of each server i: $\rho_i = \frac{\lambda}{\mu_i}$.
  - **Joint probability** $P\{\ldots, L_k = l_k, \ldots\} = \prod_i P\{L_i = l_i\} = \prod_i \rho_i^{l_i}(1-\rho_i)$ by independence.
    - **Average # jobs** in system: $E[L] = \sum_i E[L_i]$.
- With probabilistic routing, different flows might take different paths.
  - **Utilization** of each server i: $\rho_i = \frac{\lambda_i}{\mu_i}$.
    - μ_i is known; the service rate of each subsystem i.
    - λ_i, the **effective arrival rate**, is given by:

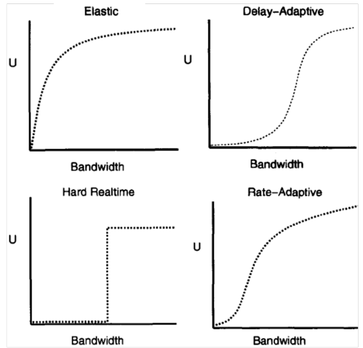
$$\lambda_i = r_i + \sum_{j=1}^{n} \lambda_j P_{ji} \qquad (\text{equivalently, } \lambda = r(I - P)^{-1} \text{ in matrix form})$$

      - r_i: rate of external arrival to server i.
      - λ_j: output rate of subsystem j (= effective arrival rate of j, assuming the system is in steady state).
      - P_{ji}: fraction of packets arriving from subsystem j to subsystem i.
- **Jackson network**:



Resource Allocation

- The Internet provides best-effort service (i.e., no guarantee for packet delivery).
  - Suitable for **elastic traffic**, but not for **real-time applications**.
  - Solution: allocate *different amount of resources* to different application flows.
    - U_i: utility function based on the type of application that is running.



- Notations:
  - $C = (C_1, \ldots, C_n)$: **resource capacity** of each link.
  - $d = (d_1, \ldots, d_m)$: **flow demand**.
  - $x = (x_1, \ldots, x_m)$: **feasible solution** satisfying:
    1. $\forall i, 0 \le x_i \le d_i$, and
    2. $\sum_{\text{each flow } j \text{ going through link } i} x_j \le C_i$.
  - $\phi = (\phi_1, \ldots, \phi_m)$: **weight vector** containing weights for each flow.
- **Fairness**:
  1. Users with small demand get all they want (and not more).
  2. Users with large demand evenly split the remaining resources.
- **Max-min fairness**: a feasible allocation is max-min fair iff *an increase of any rate* within the feasible domain results in a *decrease of a smaller or equal rate*.
  - i.e., x is max-min fair iff for any feasible y, if $y_i > x_i$, then $\exists j$ s.t. $y_j < x_j \le x_i$.
  - There always exists a *unique* max-min fair solution.
  - **Bottleneck**: we say that resource r is a bottleneck for flow i iff:
    1. Resource r is saturated (i.e., not possible to increase any x_i), and
    2. Flow i has the maximum rate x_i among all flows using resource r.
  - **Theorem**: when each flow has an *infinite demand* under a network system, a flow allocation is max-min fair iff *every flow has a bottleneck resource*.
  - **Water filling algorithm**:
    1. List the demands as empty buckets with width 1 and area d_i.
    2. Find the fair share at each bottleneck resource, and sort in ascending order.
    3. Starting from the lowest fair share, fill in water until a bottleneck is reached.
    4. For the remaining flows, repeat step 3 for each bottleneck resource.

- **Weighted max-min fair share**:
  1. Users receive a fair share based on their weights, $x_i = \frac{\phi_i}{\sum_j \phi_j} C$.
  2. Spillover resources (i.e., $\sum_i x_i - d_i$ for each i with $x_i > d_i$) are distributed to the remaining flows, proportional to the weights of each flow.
  3. Repeat step 2 until no spillover resources are left.
  - **Bottleneck**: we say that resource r is a bottleneck for flow i iff:
    1. Resource r is saturated (i.e., not possible to increase any x_i), and
    2. Flow i has the max *normalized* rate $\frac{x_i}{\phi_i}$ among all flows using resource r.
  - **Water filling algorithm** for weighted max-min fair share:
    1. List the demands as empty buckets with **width** ϕ_i and area d_i.
    2. Find the fair share at each bottleneck resource, and sort in ascending order.
    3. Starting from the lowest fair share, fill in water until a bottleneck is reached.
    4. For the remaining flows, repeat step 3 for each bottleneck resource.

Software Defined Networking (SDN)

- Approach to networking (i.e., network **implementation** + **management**) based on new fundamental principles.
  - Note: traditional networking is *protocol-defined*, not *software defined*.
- **Motivations**:
  1. While the Internet guarantees *best-effort* packet delivery and enables innovation in applications, change is only easy at the **edge** but not at the **core**.
     - **Network edge**: an hourglass IP model providing **layered service abstraction**.
       * Decomposes delivery into fundamental components, enabling independent innovation at each layer.
     - **Network core**: limitations include
       (a) **Closed equipment**: software are bundled with hardware, and interfaces are vendor-specific.
       (b) **Over-specification**: contributes to slow protocol standardization.
       (c) **Restricted innovation**: only vendors can improve the system $\rightarrow$ long delays in introducing new features.
       (d) **Difficulty in network management**: high cost of network operation + likelihood of buggy equipment.
  2. As compared to **computation** and **storage** (which have been virtualized), networks are lagging behind.
     - **Management is complex and primitive**, and rely heavily on network administrators.
     - **Routing algorithms change very slowly**.
     - New control requirements (e.g., isolation, traffic engineering, pkt processing + analysis, etc.) $\rightarrow$ **complexity**.
- **Routers** have 2 key functions:
  1. **Routing**: run routing algorithms/protocols (e.g., RIP, OSPF, BGP).
     - The **control plane** establishes the router state, and determines how/where packets are forwarded.
  2. **Forward datagrams** from incoming to outgoing links.
     - The **data plane** processes and delivers packets based on the router/endpoint state, and constructs the forwarding table based on the routing algorithm.

Principles

1. **Disaggregation**: separation of the data and control planes using an open interface.
   - Requires a **forwarding abstraction**: method to specify communications between the control and data planes.
     - E.g., **OpenFlow**: consists of *flow rules* (*Match-Action pairs*) which apply actions to packets with certain features.

   (+) Network operators able to **purchase** control & data planes **from different vendors**.
   (+) **Shifting of control from vendors to operators** that can build networks to meet their users' needs.
   (+) **Opportunities for fast innovation**: software control of the network can evolve independently of the hardware.
2. **Centralized control**: making the control plane *fully independent* of the data plane and *logically centralized*.

   (+) Centralized decisions are easier to make.
3. **Programmability**: enables performance optimizations and adaptability to protocol changes.
   - Data plane is implemented with a **forwarding pipeline**:
     - Each table focuses on a **subset of header fields** that might be involved in a given flow rule.
     - A **packet is processed by multiple tables** in sequence to determine how it is forwarded.

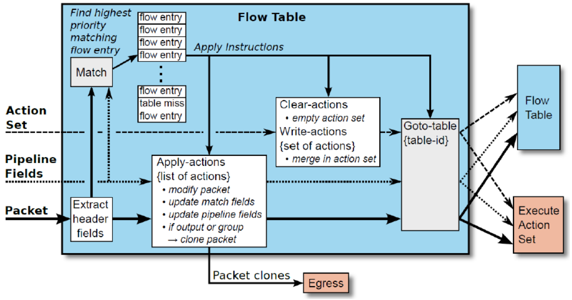
   (+) **Easy network management**: programmers are able to debug/check behaviour more easily.
   (+) **Rapid innovation**: each layer can evolve independently from one another.
   (+) **Control shift** from vendor to operators and users.
- These principles use abstractions to modularize the network control problem.
  1. **Specification abstraction**: allow control apps to express the desired network behaviour without implementing it (*northbound interface*).
  2. **Distribution abstraction**: shield apps from distributed states, making control logically centralized (*middle layer*).
  3. **Forwarding abstraction**: allow any forwarding behaviour desired by apps while hiding details of the underlying hardware (*southbound interface*).

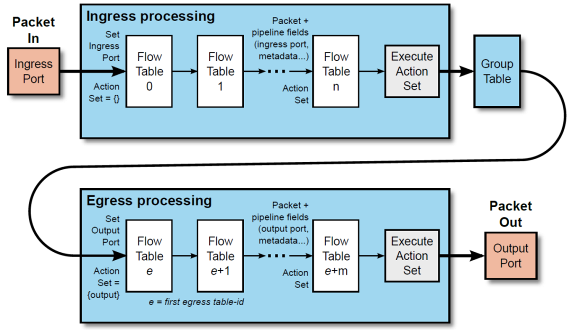
Use cases

- Users of SDN:
  - **Enterprises**: universities and private companies.
  - **Cloud providers**: e.g., Google, Meta, Microsoft.
  - **Network operators**: e.g., Comcast, AT&T, Singtel.
- Use cases of SDN:
  - **Network virtualization**: enable networks to be programmatically created, managed, and torn down.
    * **Single API entry point** to create, modify, and delete virtual networks.
    * Enables the virtual network to have its own private address space.
  - **Switching fabrics**: enable lower costs and new features for cloud data centers.
  - **Traffic engineering**: for wide-area links between data centers.
  - **Software-defined WANs**.

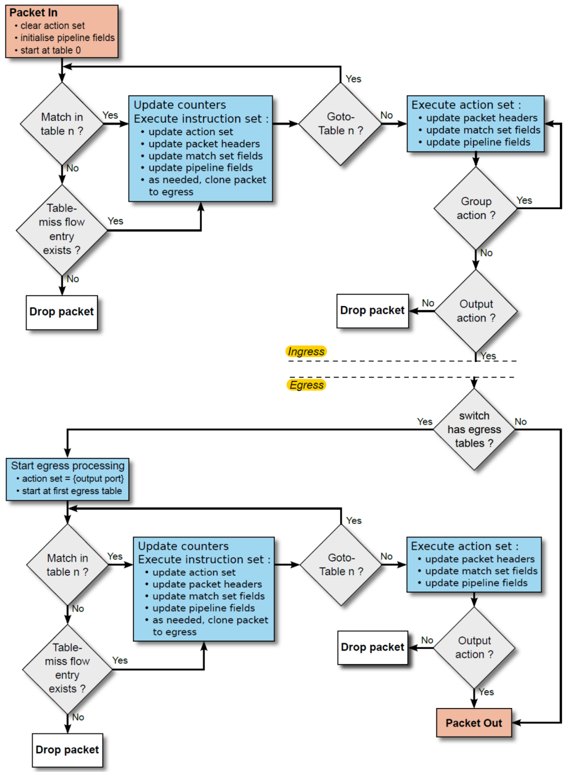
OpenFlow

- For each packet from a packet flow, it contains:
  1. **Header** and **header fields**.
  2. **Pipeline fields**: values attached to the packet during pipeline processing, e.g., ingress port and metadata.
  3. **Action**: an operation that acts on a packet, e.g., drop/forward/modify TTL.
  4. **Action set**: accumulated while processed by flow tables, and executed at the end of pipeline processing.
- A flow entry in a flow table contains:
  1. **Match field**: packets' header and pipeline fields are matched.
  2. **Priority**: used to choose from multiple matches.
  3. **Instruction set**:
     - Contains a list of actions to apply immediately.
     - Contains a set of actions to add to the action set (not applied immediately).
     - Enables modification of pipeline processing (e.g., going to another flow table, but backtracking to a previous table is not allowed).
  4. **Default entry**: flow entry for table-misses, by default packet will be sent to the control plane. If this is not present, the packet with a table-miss will be dropped.

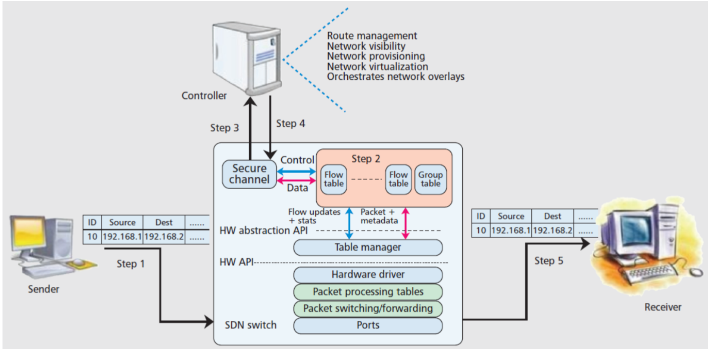




Packet flow through the processing pipeline

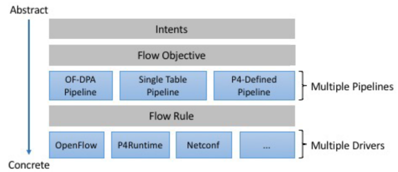


- **Operation** of SDN:
  1. Switch receives a packet from its input port.
  2. Switch tries to match packet against the flow entries in the flow table.
  3. If switch does not have any idea how to handle the packet, it forwards the packet to the controller.
  4. Controller writes new flow entry into the table.
  5. Packet is sent to receiver.



Open network operating system (ONOS)

- Similar to other horizontally scalable cloud applications:
  - Consists of a set of **loosely coupled subsystems** (microservice architecture).
  - Contains a scalable and highly available **key-value store**.
- **Architecture**:
  - **Northbound interface (NBI)**: for apps to stay informed about the network state and control the data plane.
  - **Distributed core**: manages network state and notifies apps about relevant changes in state.
  - **Southbound interface (SBI)**: constructed from plugins, e.g., shared protocol libraries and device-specific drivers.
- **Abstractions**:
  - **Intent**: abstract network-wide, topology-independent programming constructs, where users can specify what they want but not how to achieve it.
  - **Flow objective**: pipeline-independent; allows finer-grained control and device-centric programming constructs.
  - **Flow rule**: pipeline-specific; supports both fixed-function and programmability.



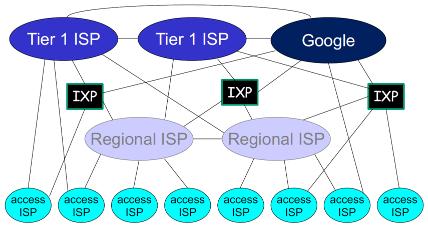
Interconnection

- Networks contain the following **basic units**:
  1. **Autonomous systems (AS)**: network of interconnected routers, each identified by a unique AS number (ASN).
     - ASNs are assigned by the **Internet Assigned Numbers Authority (IANA)**, consisting of 5 **regional internet registries (RIRs)**.
     - They are controlled by a single administrative domain (but a company can have several ASNs).
  2. **Internet peering**: voluntary interconnection of administratively separate Internet networks, for exchanging traffic between the downstream users of each network.
     - Responsible for **bilateral interconnection**.
     - Depends on business relationships, which include:
       (a) **Customer-provider relationships**:
         * Customer pays provider for access to the Internet, and reachability from anyone.
           · Provider provides **transit service** for the customer.
           · Most customers don't allow traffic through them (**nontransit AS**).
           · Some ASes provide transit between some providers (i.e., **selective transit**).
         * **Multi-homing**: a customer with ≥ 2 providers.
         * **Static routing** connects autonomous routing domains to the Internet $\rightarrow$ customers do not need ASNs.
       (b) **Peer-to-peer relationships** (i.e., *settlement-free peering*).
         * Peers provide transit between their respective customers.
         * Peers do not provide transit + often do not pay each other (i.e., **settlement-free** relationship).
         * Whether peering relationships are established depends on the following considerations:

To peer	Not to peer
Reduce upstream transit costs	Obtain and retain customers
Improve end-to-end performance	Working with your competition
Be the only way to connect customers to some part of the Internet	May require periodic renegotiation

  3. **Internet exchange point (IXP)**: an Ethernet switch where ASes freely interconnect to exchange traffic.
     - Responsible for **multilateral interconnection** (i.e., traffic exchange).
       * Ideally located at a **neutral**, **secure**, **accessible**, and **expandable** place (i.e., **carrier-neutral**).
       * Each member (1) brings a router or (2) runs fibre/wireless from their data center to the IXP location, and connects it to the IXP switch.
     - Enable cheap peering by:
       (a) Saving upstream transit costs.
       (b) Keeping local traffic local.
       (c) Enable better network performance, quality of service, and scalability.
     - Networks are allowed to join the IXP if they have: (1) **address space**, (2) **ASN**, and (3) **transit agreements**.
     - **Operation**: consortium model, representing all participants.
       * **Costs** are agreed and **covered equally** by all participants.
       * **Availability**: 24/7 cover provided by hosting location.

- **Structure of the Internet**: network of networks.



1. **Tier 1 commercial ISPs**: provide national and international coverage, e.g., Level 3, AT&T, NTT.
   – Have **access to the entire Internet**, only through its settlement-free peering links.
   – **Peer with other tier 1s** to form a full mesh.
   – Have **no upstream provider**, they are the top of the customer-provider hierarchy.
2. **Content providers**: private networks that connect their data centers to the Internet, often bypassing tier 1 and regional ISPs, e.g., Google.
3. **Lower layer providers**: typically have national or regional coverage, and include a few thousand ASes.
   – Provide **transit to downstream customers**, but need at least one provider of their own.
4. **Stub ASes**: do not provide transit service, merely connecting to upstream providers (e.g., consumers).

- The internet has a **valley-free property**, i.e., all valid AS paths are either:
1. **Single peak**: i.e., with a single tier-1 provider as root.
2. **Single flat top**: i.e., with a peering at the root.
3. Any subpaths of the above.

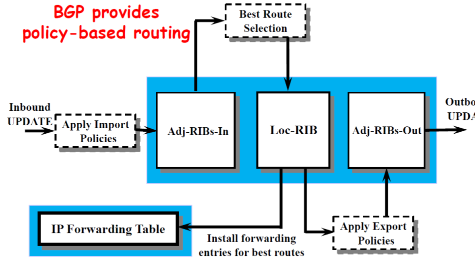
Inter-domain routing

- **Challenges**:
  – **Scale**: millions of routers and 200k+ prefixes.
  – **Privacy**: ASes do not want to expose their internal topologies and business relationships with neighbours.
  – **Policy**: no common notion of link cost + each AS needs control over its **traffic destination** and **peering**.
- **Routing algorithms**:

Link state algorithm	**Distance vector algorithm**
Assumes all routers have knowledge of *complete topology* and *link cost*	Assumes all routers know *connected neighbours* and their *link costs*
Global or centralized	Decentralized
Employs *Dijkstra's algorithm*	Employs *Bellman-Ford algorithm*, iteratively exchanging information with their neighbours
E.g., *Open Shortest Path First (OSPF)*	E.g., *Routing Information Protocol (RIP)*

1. **Link state routing**:
(−) Nodes **divulge sensitive information**.
(−) **High bandwidth and storage overhead**, since topology information is flooded.
(−) **High processing overhead** in large networks, since the entire path is computed locally per node.
(−) Works only if **policies are shared and uniform**, since a standardized notion of *total distance* is required.
2. **Distance vector routing**:
(+) **Hides details** of the network topology.
(−) Minimizes **some notion of total distance**.
(−) **Slow convergence**, certain parts of the network might have changed before the algorithm converges.
3. **Path vector routing**: advertise the entire path, instead of just the distance metric, for each destination (e.g., BGP).
(+) **Fast loop detection**: nodes can detect a loop by checking if the path contains itself + discard paths with loops.

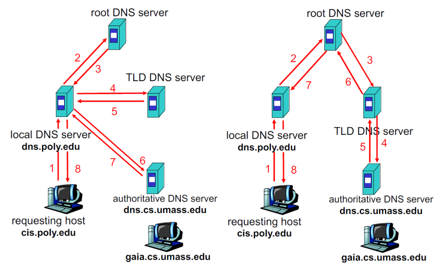
- **Border Gateway Protocol (BGP)**: the de facto inter-domain routing protocol.
  – Allows subnets to **advertise their existence** to the rest of the Internet.
  – Allows ASes to **determine "good" routes** to other networks, based on reachability information and policy.
  – Provides **policy-based routing**.
    ∗ **Import policy**:
      1. **Filter unwanted routes** from neighbour.
      2. Used to **rank customer routes over peer routes**.
      3. Manipulate attributes to **influence path selection** (e.g., assign local preference to favoured routes).
    ∗ **Export policy**:
      1. **Filter out routes** you do not want to tell your neighbour.
      2. Manipulate attributes to **control what neighbours see** (e.g., make paths look artificially longer).
    ∗ Used by commercial ISPs to:
      1. **Fulfil bilateral agreements** with other ISPs: define who will provide transit for what.
      2. **Minimize monetary costs**/maximize revenue.
      3. Ensure **good performance**.
  – **Types**:
    1. **Exterior BGP (eBGP) peering**: between BGP speakers in **different ASes**.
       ∗ These routers should be **directly connected**.
       ∗ When ASX advertises an IP prefix to ASY, ASX **promises** it will forward packets towards that prefix.
         · ASX can aggregate prefixes in its advertisement.
    2. **Interior BGP (iBGP) peering**: between BGP peers **within an AS**.
       ∗ Peers are **not required to be directly connected** (IGP e.g., RIP/OSPF, handles intra-AS peer connectivity).
       ∗ iBGP peers must be **(logically) fully meshed**.
         · Used to **originate connected networks**.
         · Used to **pass on prefixes learned** from outside the AS (those from other iBGP peers are not passed on).
  – BGP process:
    1. **Establish BGP session**.
    2. **Exchange all active routes**.
    3. Repeatedly exchange **incremental updates**.
  – **Getting entries into the forwarding table**:
    1. **Router becomes aware of prefix** via BGP route advertisements from other routers.
    2. **Determine router output port** for prefix:
       (a) Choose the **best inter-AS route** from BGP (i.e., shortest `AS-PATH`).
         ∗ BGP route = prefix + attributes = NLRI + path attributes.
         ∗ **Routing information bases (RIBs)** = Adj-RIBs-In + Loc-RIB + Adj-RIBs-Out: contains all routes.
           · **Adj-RIBs-In**: unprocessed routes from peers via inbound `UPDATE`.
           · **Loc-RIB**: selected local routes used by the router.
           · **Adj-RIBs-Out**: selected for advertisement to peers.



       (b) Use **OSPF** to find the best intra-AS route leading to the best inter-AS route.
       (c) Identify the **port number** for the best route.
    3. Add **prefix-port entry** to the forwarding table.
       ∗ **Hot potato routing**: if there is a tie, routers will try to differentiate these routes based on the distance that it needs to traverse within its network (by leveraging information in `NEXT-HOP`).
  – **Best route selection**:
    1. Calculate **degree of preference**:
       ∗ If the route is learned from an **internal peer**. use `LOCAL_PREF` attribute or the pre-configured policy.
       ∗ Else, use the pre-configured policy.
    2. **Route selection**: peers are recommended to use the highest **degree of `LOCAL_PREF`**, and tie breaking on:
       (a) Smallest **number of ASNs in `AS_PATH`**.
       (b) Lowest **origin number in `ORIGIN`**.
       (c) Most preferred **`MULTI_EXIT_DISC`**.
         ∗ For BGP peers to discriminate among entry points to a neighbouring AS → control inbound traffic.
       (d) Routes from **eBGP preferred over iBGP**.
       (e) Lowest **interior cost** based on **`NEXT_HOP`**.
  – **Messages**:
    1. `OPEN`: opens TCP connection to peer and authenticates sender.
    2. `UPDATE`: advertises new paths or withdraws old paths. Contains:
       ∗ **Withdrawn routes**: IP prefixes for the routes withdrawn.
       ∗ **Network layer reachability information (NLRI)**: IP prefixes that can be reached from the advertised route.
    3. `KEEPALIVE`: keeps connection alive in the absence of `UPDATE`s; also ACKs `OPEN` request.
    4. `NOTIFICATION`: reports errors in previous messages; also used to close connections.
  – **Path attributes**:
    1. **Well-known mandatory**: e.g.,
       ∗ `ORIGIN`: conveys the origin of the prefix.
       ∗ `AS-PATH`: contains ASes through which NLRI has passed.
       ∗ `NEXT-HOP`: indicates the IP address of the router in the next-hop AS.
    2. **Well-known discretionary**.
    3. **Optional transitive**: e.g.,
       ∗ `COMMUNITY`: used to group destinations; each destination can be a member of multiple communities.
    4. **Optional non-transitive**.
    ∗ Implementation rules:
      1. Must recognize all **well-known** attributes.
      2. **Mandatory** attributes must be included in `UPDATE` messages that contain NLRI.
      3. When BGP peers update **well-known** attributes, they must pass them to their peers.
  – BGP **prefix hijacking**: re-routing internet traffic by falsely announcing ownership of IP addresses.
    ∗ **Blackhole**: traffic is discarded.
    ∗ **Snooping**: traffic is inspected and then redirected.
    ∗ **Impersonation**: traffic is sent to bogus destinations.
    ∗ **Subprefix hijacking**: traffic is redirected to a more specific prefix (∵ traffic follows longest matching prefix).
    Prevention:
    ∗ Each AS **filters routes** announced by customers, **based on the prefixes that the customer owns**.
    (−) Difficult to filter routes initiated from far away.
    ∗ **Route origin authorisation (ROA)**.
    ∗ **Resource public key infrastructure (RPKI)**.

Peer-to-peer networks

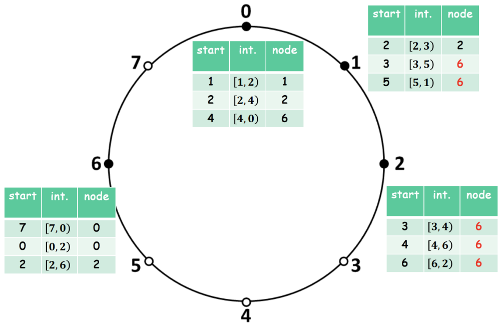
- **Centralized network models**:
  – **Client/server model**: asymmetric, traditional communication model with ad-hoc clients and dedicated servers.
  – **Delegation model**: first-hop server receives query as a delegation, forwards the query to other servers for processing, and eventually returns the result.
    ∗ Query can be served either **iteratively** (left) or **recursively** (right).



    ∗ Iterative delegation is used by the **domain name system (DNS)**: prevents overloading the root DNS server.
- **P2P network models**:
1. **No always-on server**/central entity.
2. **End systems directly communicate** with one another.
3. **No a-priori knowledge** of the structure.
4. **Flat architecture**/namespace.
(−) Peers are **intermittently connected**, and change IP addresses → **unreliable service providers**.
(−) Difficulty in **staying connected** and performing **resource lookup**.

- Applications:
1. **File sharing**:
   – **Napster**: based on a **central index server** which **knows all peers and files** in the network.
     ∗ **Delegation**: users register with the central server, which sends the list of files to be shared.
     ∗ **Keyword-based searching**: server returns a list of details w.r.t. each file and the peer sharing it.
   (+) Consistent view of network.
   (+) Fast and efficient search.
   (+) Accuracy of search results in guaranteed.
   (−) Single point of failure + vulnerable to attacks.
   (−) Heavy computation required to handle queries.
   (−) Unreliable content, downloads from a single peer only.
   – **Gnutella**: pure p2p network.
     ∗ An address of an active peer, obtained from an **out-of-band channel**, is required to join the network.
     ∗ Once joined, peers learn about others and the topology of the network.
     ∗ **Queries are flooded** into the network, and peers **download directly** from others.
   (+) Fully decentralized.
   (+) Robust against node failures + less susceptible to denial of service attacks.
   (+) Open protocol, can be written on any OS.
   (−) Inefficient query flooding, wastes network and peer resources.
   (−) Inefficient network management.
   – **KaZaA**: consists of two kinds of nodes forming a 2-tier hierarchy, with top level = SNs and lower level = ONs.
     (a) **Supernodes (SN)**: peers with more resources/responsibilities.
       ∗ **Exchange information between themselves**, may change connections with other SNs in minutes.
       ∗ Acts as a **hub for its children ONs**, keeping track of files in those ON peers.
       ∗ **Do not form a complete mesh**.
       ∗ **Do not cache information** from disconnected ONs.
     (b) **Ordinary nodes (ON)**: normal user/peer which sends requests through its parent SN.
       ∗ **Belongs to only 1 SN** at a time, and can change at will.
       ∗ **Not visible to other SN**.
       ∗ **Can be promoted** to SN if it has the desire and ability (sufficient resources: e.g., bandwidth, up time).
2. **VoIP** and **messaging**: e.g., **Skype**.
   – Similar architecture to KaZaA: hierarchical overlay with SNs.
   – Problem: **network address translation (NAT)** prevents an outside from initiating a call to inside peer.
     (a) Using Alice's and Bob's SNs, **relay** is chosen.
     (b) Each peer initiates session with relay.
     (c) Peers can now communicate through NAT via relay.
3. **Content distribution**: e.g., BitTorrent.
   – Builds a network (**swarm**) for *every file* that is being distributed.
   (+) Files can be sent to others as a link (i.e., `.torrent` file), which always refers to the same file.
     (a) For each file, one server (**seed**) initially hosts the original copy, before it is broken down into chunks.
     (b) Seed starts **tracker**: server keeping track of which seeds and peers are in the **swarm**.
     (c) Seed **creates torrent file** (with metadata: e.g., sizes + checksums of each chunk) and hosts it somewhere.
     (d) A new client **obtains the torrent file**.
     (e) Client **contacts tracker** and obtain the **torrent**: peers exchanging 256 kB chunks of the file.
     (f) Client **downloads/exchanges chunks** with peers.
       i. Peers joining the torrent have no chunks, but will accumulate them over time.
       ii. Peers register with tracker to get a list of peers, and connects to a subset of them.
       iii. Peers may leave after they have obtained the entire file.
       ∗ **Pulling chunks**:
         i. At any given time, peers have different subsets of file chunks.
         ii. Periodically, a peer may ask each neighbour for a list of chunks they have.
         iii. The peer sends requests for his/her missing chunks, and collects chunks via a **rarest first** strategy.
       ∗ **Pushing chunks**: tit-for-tat.
         i. A peer sends chunks to 4 neighbours currently sending her chunks over *at the highest rate*.
         ii. The top 4 will be re-evaluated every 10 seconds.
         iii. Every 30s, the peer will randomly select another peer, and start sending chunks over to them.
   (−) **No searching** possible: no name service, but websites with "link collections" exist.
4. Others: **Video streaming**, **p2p computation/storage**, **blockchains**.

- **Lookup services**:
  – **Addressing**: supported by **structured networks/systems**, which allow for deterministic routing. Network structure determines where peers belong in the net, and where objects are stored.
  (+) Object location can be made efficient.
  (+) Each object is uniquely identifiable.
  (−) Need to know unique names.
  (−) Need to maintain structure required for addressing.
  – **Searching**: supported by **unstructured networks/systems**, where peers are typically free to join anytime and anywhere, and objects can be stored anywhere.
  (+) No need for unique names; more user friendly.
  (−) Hard to make efficient.
  (−) Need to compare actual objects, since query term may match multiple results.
- **Distributed hash tables (DHTs)**:
1. Each node is responsible for ≥ 1 buckets.
2. Nodes communicate among themselves to find the responsible node.
3. Provide same abstraction supporting map operations, may differ in **namespace design** and **routing** in the overlay.
   – **Chord**:
     ∗ Uses the **same hash function for both objects and nodes**: SHA-1.
     ∗ Nodes are **organized in a ring**, keeping track of their predecessors and successors.
       · The **immediate successor** of each node is the closest node.
       · Each node is assigned an **identifier** in the range $[0, 2^m - 1]$.
       · Each node **maintains a finger table** with m shortcuts, with the i^{th} shortcut being $\geq 2^{i-1}$ nodes away.
       · **Node join**: finger table of other nodes are updated.
       · **Routing**: shortcuts are taken based on the entries in the finger table.
       · **Node leave**: each node periodically pings its 2 successors to see if they are still alive; if not, they will search for a replacement.



start	int.	node
1	[1, 2)	1
2	[2, 4)	2
4	[4, 0)	6

start	int.	node
2	[2, 3)	2
3	[3, 5)	6
5	[5, 1)	6

start	int.	node
3	[3, 4)	6
4	[4, 6)	6
6	[6, 2)	6

start	int.	node
7	[7, 0)	0
0	[0, 2)	0
2	[2, 6)	2

- **Scalable content-addressable network (CAN)**:
  * Namespace is a d-dimensional torus, nodes only keep track of neighbours without storing shortcuts.
    · **Node join**: suppose node B wants to insert a key-value pair (k, v).
      1. B chooses a coordinate $(a, b) = (h_i(k), h_j(k))$.
      2. The key-value pair (k, v) is routed to coordinate (a, b).
      3. If B is already a peer, the node who owns (a, b) stores $x = a, y = b$.
      4. Else if the coordinate is owned by another node A, A's zone is split by half and B owns half of the cell. Splitting is done along the x dimension first, before the y dimension.
    · **Routing**: routing is done by querying each node's 4 neighbours.
    · **Node leave**:
      1. The vacant zone is merged back to a neighbour, if possible.
      2. Else, a neighbour node will temporarily handle multiple zones.
  * With higher dimensions (i.e., $d > 2$), routing table size and # hash functions ↑, but path lengths are shorter.