1 Network Organisation

- Diameter: maximum length of the shortest path between any two vertices in a graph (affects latency)
- Bisection (Band)width: Min no. of links to cut to bisect the graph into two equal partitions (affects bandwidth)
- Cost: no. of links

Topology	Diameter	Bisection	Cost
Fully Connected	1	$\lfloor \frac{n}{2} \rfloor \times \lfloor \frac{n}{2} \rfloor$	$\frac{n \times (n-1)}{2}$
Ring	$\lfloor \frac{n}{2} \rfloor$	2	n
2-dimensional mesh	$2(\sqrt{n}-1)$	$\sqrt{n}$	$2(\sqrt{n})(\sqrt{n}-1)$
2-dimensional torus	$2\lfloor \frac{\sqrt{n}}{2} \rfloor$	$2\sqrt{n}$	2n

- Fat trees: Leaves are servers and nodes are switches. Link redundancy or oversubscribed topology.
- (Folded) Clos Networks: Multi-stage switching fabric, modular and scalable, link redundancy for resilience and fault tolerance.
- Content Delivery Network: Servers closer to users (collocated with ISP infrastructure) with temporal and physical locality.
- Control plane: Routing & management. Router software. ms time frame. Handles events (track changes in network topology, compute paths through network, reserve resources along path)
- Data plane: Forwarding. Linecard hardware. ns time frame. Handles packets (forward/drop/buffer, mark/shape/schedule)
- Routing: Every switch/router identifies the next hop, distributed coordination. (vs Source Routing)
- Intra-AS provides optimal routing while Inter-AS run by broader routers and can be affected by economic or political incentives
- Link State Algorithm: Router shares cost to neighbours with whole network. All routers have complete topology and link costs. All routers compute shortest path to other routers independently. Dijkstra's Algorithm. Converge faster. OSPF.
- Distance Vector Algorithm: Routers shares cost to whole network with neighbours. Routers participate in iterative process of computation and exchange information with neighbours only. Bellman-Ford's Algorithm. Use less bandwidth. RIP. Count to infinity.
- Principles: **Hierarchy** for scalability (Clos networks, ISP hierarchies, Routing algorithms), **Caching** for scalability and performance (CDN, Routing tables), **Redundancy** for fault-tolerance and performance (multi-path DC networks, multiple paths among ISP)

2 Software Defined Networks

- Distributed Approach: Hard to evolve (Many task-specific control mechanisms, no modularity, redundant functionality)
  Hard to reason about and debug (Unclear interaction between different mechanisms, no unifying abstractions).
  Hard and expensive to customise (Built in functionality on switches)
- Inter-domain Routing: Routing only on destination IP, influence only immediate neighbours, difficult to incorporate other info.
  Application-specific peering (route video traffic one way, non-video another), blocking DOS traffic (dropping unwanted traffic further upstream), Inbound traffic engineering (split over multiple peering links)
- Control Program: Direct control over switches, takes global network view and outputs configuration of each device. Not distributed (Network OS hides details of distributed state). Modular. Separation of concerns.
- Forwarding Abstraction: Standard way of defining forward state. Flexible (built from set of forwarding primitives, able to express different policies). Minimal (streamlined for speed and low power, control program not vendor specific).
- OpenFlow: Abstracts switch data plane as one big look-up table. When packet arrives, extract relevant headers, see if it matches any entries, execute corresponding action. If no match, forward packet to controller.
- OpenFlow Match: Input port, Ethernet header (src, dst, type), IP header (src, dst, proto), TCP header (src port, dst port)
- OpenFlow Action: Drop, Forward to port N, Send to controller, Modify value of field
- Switch to Controller: Connect, Disconnect, Status of ports, Packet (if matches no rules), traffic statistics
- Controller to switch: Add/Remove/Modify table entries, Packet, Request traffic statistics
- Limitation: Switches use SRAM/TCAM very fast but limited capacity for rules. Ask controller when no rule in cache.
- Trend: Rise of merchant switching silicon. Cloud/data centers facing management/cost problems. Compatibility with existing hardware.
- Virtualisation: Data center allows each tenant to specify topology, compiles into switch configurations and implemented simultaneously.

3 Programmable Dataplanes

- Multiple Match Tables: Pipeline of packet processing stages, each stage has its own tables, next stage can previous modifications.
- Controller to Switch: Packet parsing (Header X is from B1 to B2 in packet), Table configuration (T1 should use X for match and A1 and A2 for actions), Action configuration (Action A1 should modify field F1/keep state S2)
- Programmable Parser: Takes bits from a vector and outputs a Packet Header Vector (PHV).
- Protocol Independent Switch Architecture: Resources not shared between stages, computation model is constrained (feed forward, limited memory access, limited ALU), only can run if fits within constraints. If it fits it runs at line rate. Protocol Adaptability.
- P4 Parsing Headers: **extract** takes bits out of packets and put them in header instances, **select** picks which state to **transition** to and the final stages are **accept** and **reject**.
- P4 Tables: **key** specifies the set of fields for matching, **actions** specifies the set of possible actions, **default_action** specifies default.
- P4 Actions: Modify metadata (standard_metadata), packet fields, local state
- P4 Metadata: Variables that accompany the packet as it is processed in the switch. Read and write from different parts of the P4 program. Switch makes some metadata available, rest has to be defined by us.
- P4 Runtime: P4 does not specify how rules are added/modified/removed from tables as packets are processed by switch. Controller has to populate P4 tables, get statistics, and send/receive packets to/from data plane. Cannot use OpenFlow out of box as table definitions change between programs. P4 Runtime is a controller platform for targets whose behavior is described by P4 programs.

```
state parse_ethernet {packet.extract(hdr.ethernet);
    transition select(hdr.ethernet.etherType) {TYPE_IPV4: parse_ipv4; default: accept;}}
apply {if (hdr.ipv4.isValid()) {ipv4_forward.apply();}}
table ipv4_forward {
    key = {hdr.ipv4.dstAddr: exact;} actions = {forward; drop; NoAction;} default_action = drop();}
action forward(macAddr_t dstAddr, egressSpec_t port) {
    standard_metadata.egress_spec = port; hdr.ethernet.srcAddr = hdr.ethernet.dstAddr;
    hdr.ethernet.dstAddr = dstAddr; hdr.ipv4.ttl = hdr.ipv4.ttl -= 1;}
```

4 Quality of Service

- No best scheduling policy: Fairness/QoS/Application requirements/Tenant requirements
- Queuing: Persistent vs Transient. Cons: Long latency, Packet loss, Unpredictable performance, Interference
- Link Scheduling: Round Robin, Deficit Round Robin (equal bytes), Weighted Round Robin, Strict Priority
- Priorities: Ethernet IEEE 802.1Q/802.1p 8 prioirty levels / IP Differentiated Services Code Point & ECN
- Drop Packets (Policing): Too aggressive (interfere with congestion control - slow packet recovery) and too late
- Explicit Congestion Notification (ECN): Switch mark IP header when buffering is high instead of dropping, receiver echos back in TCP header. When a marked TCP packet is received, reduce sending rate. Layering violation (assumes underneath layers).
- Active Queue Management: Drop/mark packets inside a buffer before it's full. RED or Controlled Delay.
- Random Early Detection (RED): Drop/mark packets between MinThres and MaxThres probabilistically.
- Traffic shaping: Force traffic to conform to profile, implemented in endhosts and in network. But increase buffering and latency.
- Token Bucket Algorithm: Tokens added to bucket at fixed rate up to a max d, send packet if there are tokens. Forwards burst traffic.
- Leaky Bucket Algorithm: Packets allowed at constant rate, drops new packets if bucket full. Absorbs bursts and smooths traffic.
- Metering: Hardware primitive to monitor rates and burst allowances. No buffering, counters instead.
- 2 Rate 3 Colour Marker: Red = drop. Yellow = best effort. Green = low drop prob. 2 rates & 2 burst sizes: Committed & Peak.

5 Server-based Packet Processing

- Network Functions: Deployment flexibility, more complicated functionality, CPUs can still keep up with IO speeds.
- Virtual Networking & Cloud: Multiple collocated VM on same host, isolation requirements. Virtual switch run on host.
- Kernel Modification: Not easy to implement new functionality, one bug can crash the machine.
- Netfliter Modules: Extend linux kernel using special purpose kernel modules that get hooked at different stages of packet processing pipeline. Becomes part of kernel, one bug can still crash machine.
- eBPF: Lightweight VM inside Linux kernel, run programs in sandbox, kernel unchanged. Attached to hooks (XDP/TC/Socket) inside kernel through bpf() syscall, only run when execution goes through hook (event-driven approach). Easier to use but restrictions to ensure safe programs (bounded loops, predicatable memory access, limited size, data structures, libraries).
- Userspace Packet Processing: Bypass kernel and receive incoming frames inside userspace application. Flexible and better performance. Implementation and deployment complexity and dedicated interface needed. Intel DPDK.

- Software Limitations: Line rate increasing (400Gbps), impossible for a core to process a packet every 2ns.
- SmartNIC: NIC with programmable domain-specific hardware to accelerate/offload packet processing. FPGA/SoC/ASIC.
- SmartNIC Architecture: On-path vs Off-path. Traffic can be dropped/forwarded to host/generated from SmartNIC.
- SmartNIC Limitations: Limited programming model (cannot offload complex protocols like TCP), limited resources (limit number of flows where state is stored), no shared memory with host cores (consistency problems / expensive PCIe transitions for computations).

6 Congestion & Flow Control

- Types: Core/Downlink (most common in DCs - incast problem - multiple senders send to same receiver simultaneously)/Sender
- Flow Control: Avoid overwhelming receiver by adjusting sending rate to the receiving rate
- Congestion Control: Avoid overwhelming network by adjusting sending rate to appropriate share not network bandwidth
- End-to-end principle: Core should remain simple. Intelligence and complexity pushed to end points. Functions at low levels may be redundant/little value when compared to cost of providing them at that level and function can only be implemented with the knowledge and help of application at the end points of the system.
- TCP Sliding Window: Sender window holds sent but unacked data. Receiver window holds received but unprocessed data.
- Flow Control Mechanism: Receiver uses Advertised Window (RWND) to prevent sender overflowing its window. Sender adjusts window so bytes in flight <= RWND.
- Congestion Signals: Packet loss (Reno), Delays and RTT (BBR), Switch notifications (ECN, ICMP Source Quench)
- Reaction Mechanism: Window Based (adjust sender window to minof RWND CWND) or Rate Based (traffic shaping mechanisms)
- Reaction Policy: Efficiency (high utilization) and Fairness (each flow gets equal share)
- Mechanism should be fast/efficient and allow for different policies. Policies should be specific to problem.
- TCP Congestion Control: AIMD - Sawtooth behaviour to probe for bandwidth, only one to converge to optimal sharing point
- Datacenter TCP: Interpret ECN as bitstream. Marked estimate $\alpha \leftarrow (1-g)*a+g\times F$ where F is the fraction of marked packets and g is weight for new samples. Multiplicative decrease: $cwnd \leftarrow cwnd*(1-\alpha/2)$. Faster feedback.
- Congestion Control Classification: Sender driven - sender react to congestion signals. Most common.
  Receiver driven - receivers explicitly ask senders to transmit. Data centers. deal with incast problem better and allows different packet scheduling policies at bottleneck link, does not deal with core congestion
  Centralised scheme - senders get permission from central arbiter. Zero queuing in network. Central arbiter bottleneck, low utilization.

7 The Design Philosophy of the DARPA Internet Protocols

The paper describes the design of the TCP/IP protocol stack and the main principles behind it. The fundamental goal of the design was to connect existing networks. An ordered set of secondary goals, such as fault tolerance and support for multiple services, also affected the said design and exposed a set of trade-offs. Some of the key ideas discussed in the paper are the need for network simplicity which leads to placing key functionality at the endpoints, layering triggered by the need for multiple different services, and fault-tolerance given the potential military usecases.

- Top Goal: Develop an effective technique for multiplexed utilization of existing interconnected networks
- Continue despite loss of networks or gateways. Simple network design where state is kept at the end-points. “Fate-sharing” approach.
- Support multiple types of communications service. Different needs require different abstractions. Layering. Forward-looking design.
- Accommodate a variety of networks. Decentralized design. Layering for lower parts of the stack.
- Permit distributed management of its resources. The design allowed for interoperability.
- Be cost effective. Header overheads & retransmissions. Trade-off since reliability needs to be implemented by the endhosts.
- The Internet architecture must permit host attachment with a low level of effort. Outdated view.
- The resources used in the Internet architecture must be accountable. Net neutrality. Service Level Agreements (SLA).

8 A Decade of Clos Topologies and Centralized Control in Google’s Datacenter Network

The paper describes the evolution of Google’s datacenter networks from a topology and control plane perspective. Given their special needs for scale to Pbps of bisection bandwidth and the single administrative domain of the datacenter, the authors describe their modular approach to building a Clos network based on off-the-shelf line cards. Although they maintain BGP compatibility for external communication, the authors design a custom centralized IGP inspired from big storage systems, such as GFS.

- MTTF: Mean Time To Failure. MTBP: Mean Time Between Failures. MTTP: Mean Time To Repair. MTBF: MTTF + MTTR.
- Avaliability = MTTF/(MTTF+MTTR)

9 A Clean Slate 4D Approach to Network Control and Management

The authors make the observation that the Internet architecture bundles control logic and packet handling into the individual routers and switches. Thus, they propose a clean slate design for network management, called 4D, that decouples the logic into reusable planes. 4D manages the network through network-level objectives for performance and reachability based on an up-to- date network view. The authors describe the alternatives and challenges in building and deploying a 4D network.

- Decision Plane (Control Program), Dissemination Plane (Network OS), Discovery Plane (Network OS), Data Plane
- Discovery is only half the width as we want direct control without having to go through discovery
- Network-level Objectives: Each network should be configured via specification of the requirements and goals for its performance and reachability
- Network-wide Views: The configuration should happen based on a coherent snapshot of the state of each network component.
- Direct Control: The control and management planes have the ability and the sole responsibility for setting all state in the data plane.

10 Fast Programmable Match-Action Processing in Hardware for SDN

The paper focuses on the limitations of OpenFlow and SDN, i.e. the limited match fields and actions defined by the paradigm. The authors propose Reconfigurable Match Tables (RMT), a new architecture that implements a programmable packet processing pipeline. This pipeline is split in stages each consisting of a reconfigurable match table that allows to match and modify arbitrary packet fields. The authors design such a chip consisting of 32 stages and running at 1Ghz with only a 12% power overhead and a 14% area overhead over a commodity off-the- shelf switch.

- Field definitions can be altered and new fields added; The number, topology, widths, and depths of match tables can be specified; New actions may be defined; Arbitrarily modified packets can be placed in specified queue(s)
- Factoring State: physical pipeline is a natural choice given the usecase requirements
- Flexible resource allocation: One logical stage can take multiple physical stages given the required resources
- Layout optimality: Having dedicated resources per stage performs better
- Restrictions: Match stages (no of stages), Packet header limits (size of PHV), Table size (SRAM & TCAM capacity), Action restrictions (no of instructions & complexity per stage)

11 Nimble: Scalable TCP-Friendly Programmable In-Network Rate-Limiting

The paper focuses on the problem of in-network rate limiting. The authors identify that previous solutions are not scalable and lead to poor TCP performance. They introduce Nimble, a system that can implement network-wide policies based on logical meters, a novel way to implement metering on programmable switches, while being TCP- friendly, since it depends on ECN marking rather than packet drops. Nimble’s control plane is scalable based on the observation that it only has to apply rate limiting on the most bottleneck switch for every traffic class. The authors implement Nimble in P4 and on a network simulator and show that Nimble can support up to 100k rate limiters per switch, while the control plane allows for up to 24% better network utilization with up to 24x less rate limiter updates.

- In-network Rate-Limiting more precise and higher network utilization than edge-based rate limiters.
- Local Enforcement can fail to enforce global network sharing policy.
- Logical Meter = Logical Meter + PacketLength - DrainB, DrainB = Rate * Time

12 The eXpress Data Path: Fast Programmable Packet Processing in the Operating System Kernel

The paper introduces the eXpress Data Path (XDP), a framework to implement high-performance packet processing applications in the kernel. To achieve so, XDP depends on eBPF, a virtual machine running eBPF code. To guarantee safety, the kernel verifies the eBPF programs before loading them. The authors show that XDP can achieve up to 24 Mpps on a single core, while being flexible to express different packet processing applications.

13 BBR: Congestion-Based Congestion Control

The paper focuses on Internet congestion control. The authors identify that existing loss-based schemes lead to high latency and low bandwidth. They introduce BBR, a new scheme that identifies the optimal operation point for each connection by estimating the RTT and bandwidth bottleneck, and maintaining the number of inflight bytes at the BDP. The authors also, share their experience in deploying BBR in Google’s infrastructure.

- Bandwidth-delay product = Bottleneck Bandwidth (BltBw) * Rtprop (Round trip propagation time)