CSC 458/2209 – Computer Networking Systems

Handout # 24: Data Center Networks & Networks for Machine Learning

Professor Yashar Ganjali
Department of Computer Science
University of Toronto

ganjali7@cs.toronto.edu
http://www.cs.toronto.edu/~yganjali

Announcements

- Programming Assignment 2: Simple Router
  - Due Friday March 28th at 5pm.

- This week's tutorial:
  - Programming Assignment 2 Q&A

- Next week:
  - Sample final exam review
  - Sample final and solutions posted on class website

- Final Exam
  - Time: April 26th, 2pm – 4pm
  - Location: MS 3153
    - Please double check before the exam

The Story So Far

- Layering
  - Link layer
  - Network layer
  - Transport layer

- Queueing Mechanisms, Middleboxes,
- Software-Defined Networking

- **Today**: Data Center Networks & Networks for Machine Learning

Machine Learning and Computer Networks

- Machine Learning (ML):
  - Significant growth in the recent years
  - Lots of attention and impact

- How does it affect computer networking?
  - **Networks for ML**: can traditional networks handle ML requirements?
    - Our focus is on data center networks here.
  - **ML for Networks**: how can ML be used to enhance computer networks?
    - Networks in general.

Machine Learning and Computer Networks

- Machine Learning (ML):
  - Significant growth in the recent years
  - Lots of attention and impact

- How does it affect computer networking?
  - **Networks for ML**: can traditional networks handle ML requirements?
    - Our focus is on data center networks here.
  - **ML for Networks**: how can ML be used to enhance computer networks?
    - Networks in general.

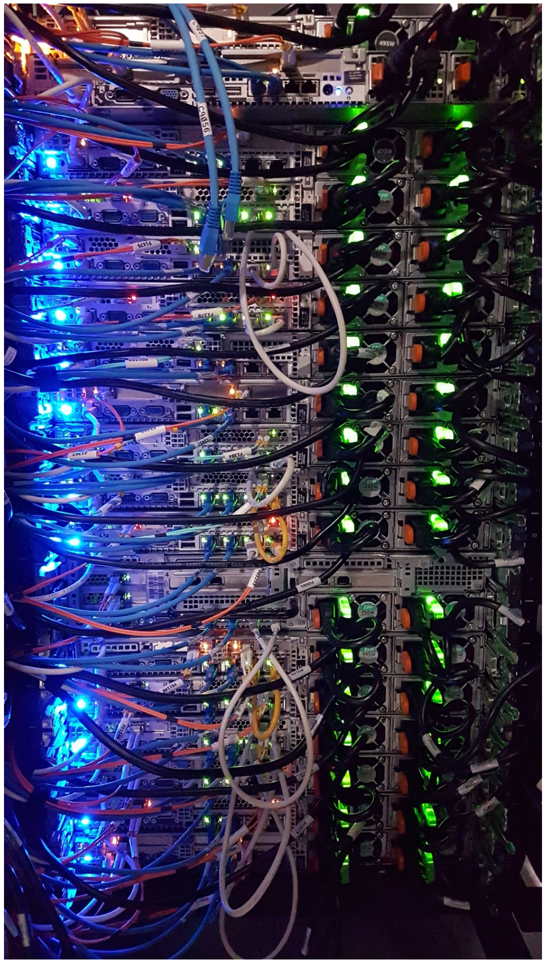
Today's Lecture

- Data Center Networking
  - Design Decision
  - Topology
  - Transport
- Networks for ML
- ML for Networks

Data Center Network (DCN)

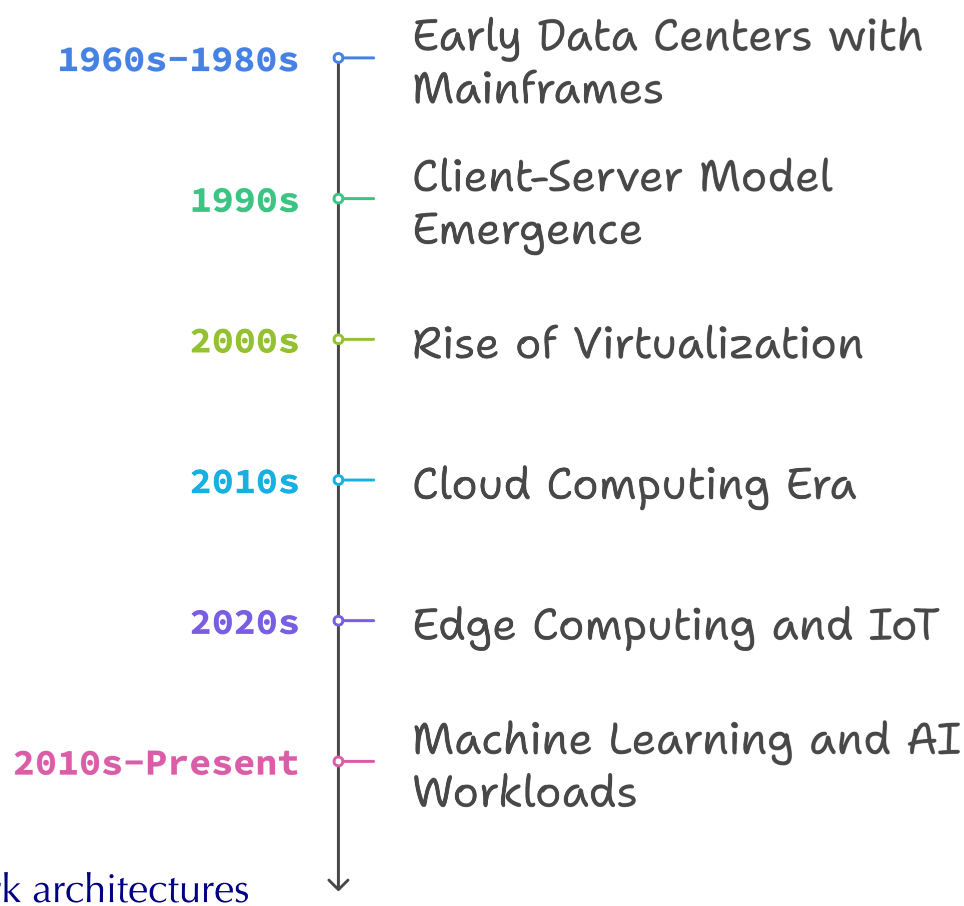
- A network of computing and storage resources
  - Proximity of components (within the data center) facilitates communication, i.e., high-performance
  - Can lead to reduced cost and overheads
    - Major cost upfront but less cost in long run.

- Functions
  - Data Storage and Management
    - Security, efficiency, reliability
  - Application Hosting
    - End-users and business applications, cloud computing and SaaS models
  - Data Processing
    - Large volumes of data for analytics and processing, big data and AI workloads
  - And more …

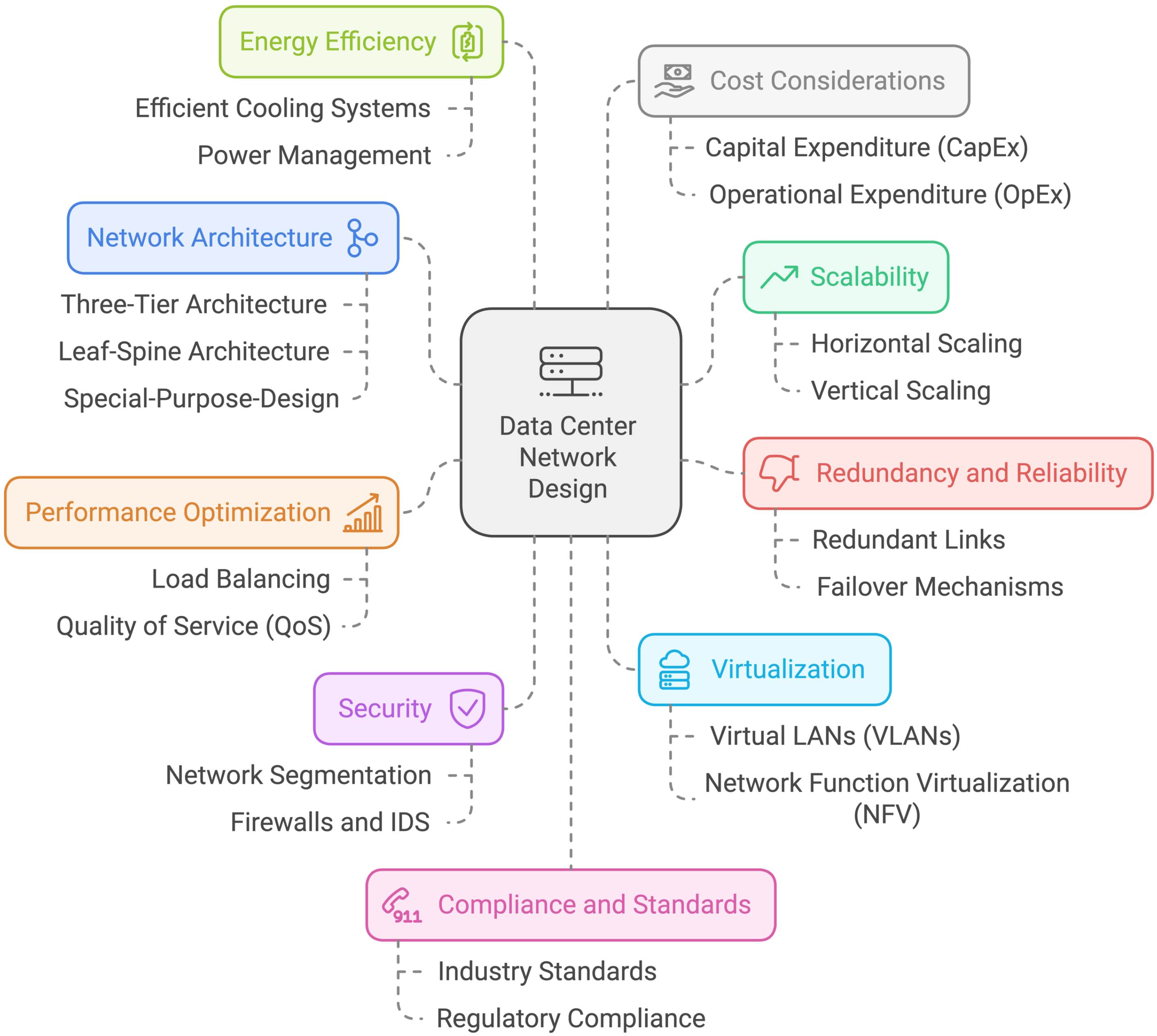


Evolution of DCN

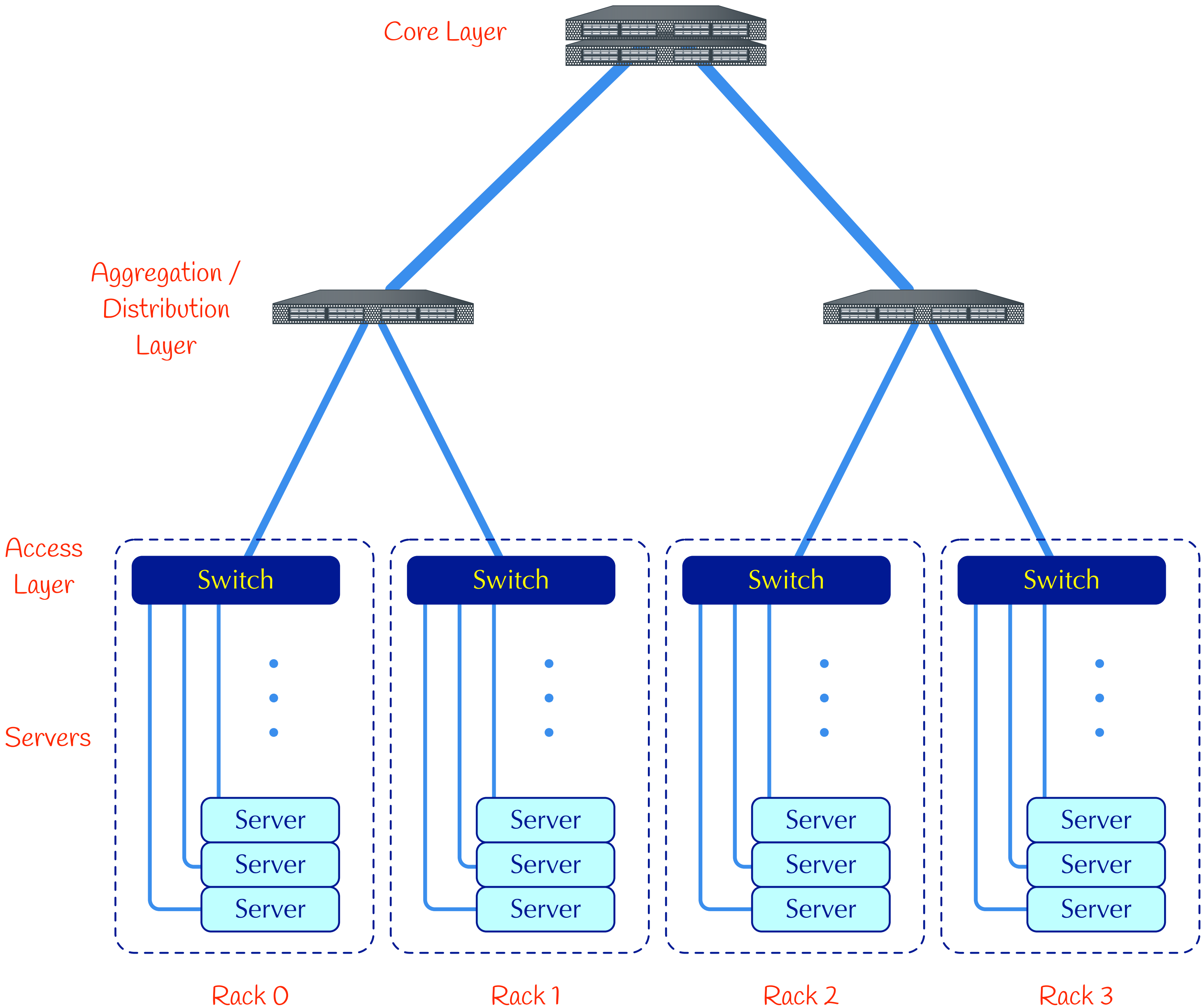
- Early Data Centers (1960s-1980s)
  - Mainframes
  - Centralized computing: limited networking
  - Point-to-point and proprietary connections
- Client-Server Model (1990s)
  - Distributed computing
  - Ethernet, TCP/IP
- Rise of Virtualization (2000s)
  - Virtual machines
  - Efficiency and scalability
  - VLANs, network segmentation
- Cloud Computing Era (2010s)
  - Cloud services
  - SDN: Scalability and automation
- Edge Computing and IoT (2020s)
  - Demand for low-latency, distributed network architectures
  - Micro data centers closer to data sources
- Machine Learning and AI (2010s-Present)
  - Exascale high-performance computing
  - Network as the bottleneck: stringent performance requirements



DCN Design Dimensions



Three-Tier Architecture



Hierarchical tree network topology

- Commonly used in traditional DCNs

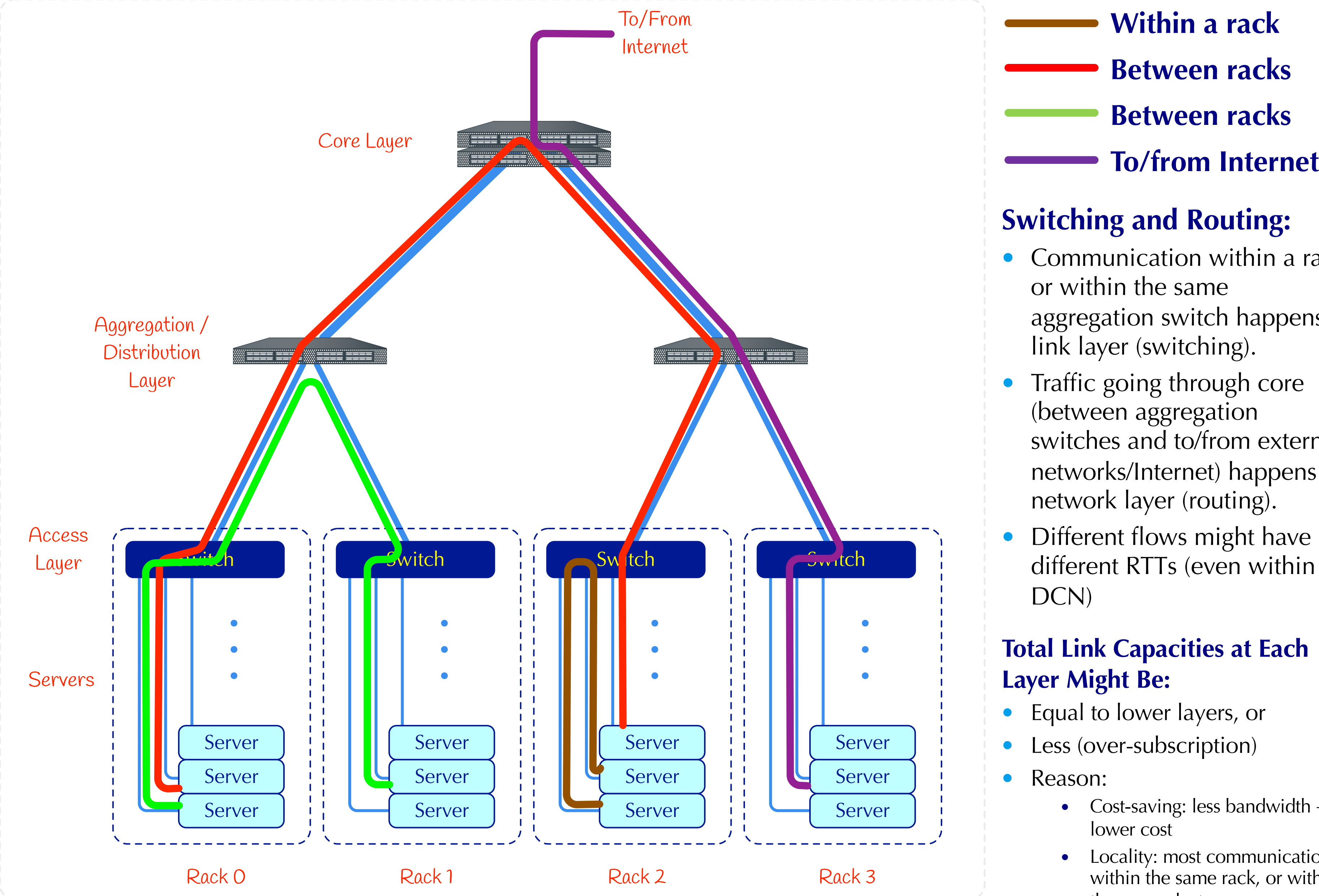
Three layers:

- Core:
  - Layer 3 (network layer)
  - Fully connected high-speed mesh of multiple routers
  - Connect to the external networks
- Aggregation:
  - Layer 3 and 2 (network and link layers)
  - Connect to core with few high-speed links (e.g., 100 Gb/s links), to access with many low-speed links (e.g., 10Gb/s) → simplify cabling
  - Middleboxes sit here (firewall, load balancer, …)
- Access: layer 2 (link)
  - Connect to each server in the rack (e.g., through one or two 10Gb/s links)
  - VLANs used to limit broadcast

Modular design

- Easy to expand

Traffic Direction in 3-Tier Architecture



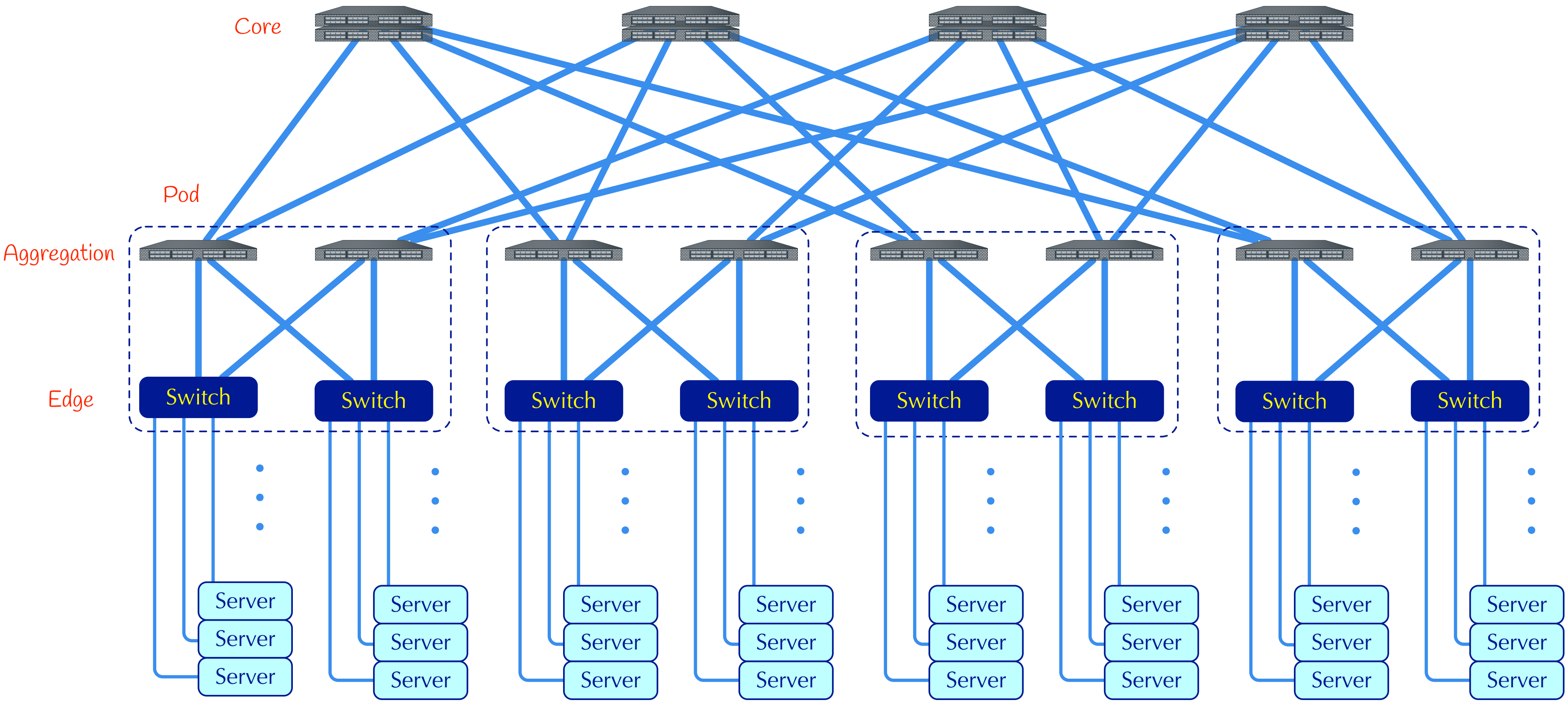
Switching and Routing:

- Communication within a rack or within the same aggregation switch happens at link layer (switching).
- Traffic going through core (between aggregation switches and to/from external networks/Internet) happens at network layer (routing).
- Different flows might have different RTTs (even within DCN)

Total Link Capacities at Each Layer Might Be:

- Equal to lower layers, or
- Less (over-subscription)
- Reason:
  - Cost-saving: less bandwidth → lower cost
  - Locality: most communication within the same rack, or within the same cluster.

Fat-Tree Architecture



Properties of Fat-Tree Architecture:

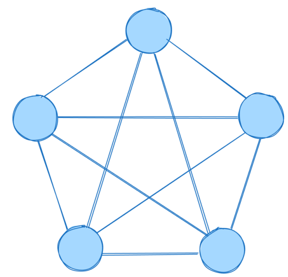
- Scalable: easily expandable
- Low latency, high throughput
- Cost-effective
  - Commodity hardware
  - Lower operational costs

Reliability and Improved Performance:

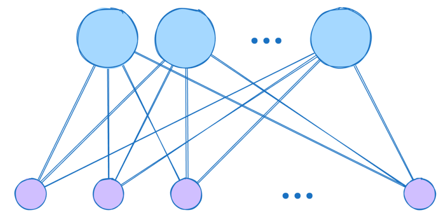
- Edge and aggregation switches are grouped into "pods".
  - Multiple-paths (choice of aggregation and core)
  - Automatic failover
- Leads to better load balance, redundancy, and thus
  - Reliability and high availability, and
  - Improved performance

Other DCN Architectures/Topologies

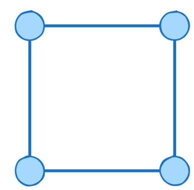
- Mesh: every node is connected to every other node
  - Direct communication, costly, but high performance
- Leaf-Spine topology: two-tier structure, servers and storage node connect directly to leaf switches
  - Switched environment, VLANs to limit broadcasts
- Hyper-cube: multi-dimensional cube structure
  - Used in high-performance computing and ML solutions
- Hybrid: combine two or more topologies
  - Tailored to specific requirements of DCNs
- And many more …
- Question: what are the properties of each of these topologies?
  - End-to-end latency?
  - Simplicity?
  - Scalability?
  - …



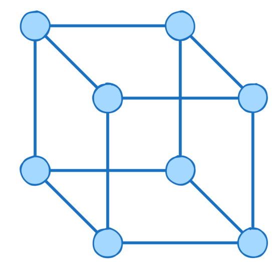
Mesh Topolog



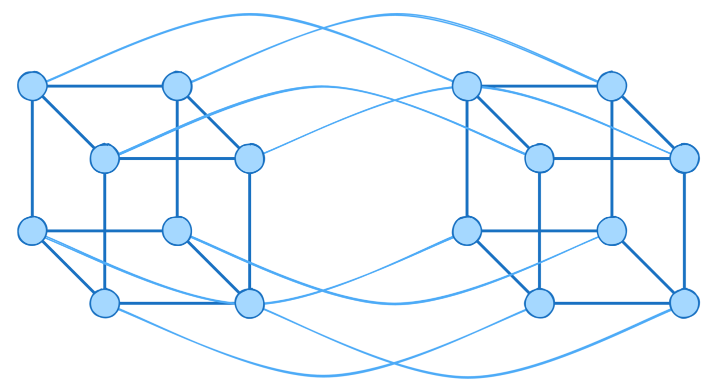
Leaf-Spine Topology



Two Dimensional Hypercube



Three Dimensional Hypercube



Four Dimensional Hypercube

Transport in Data Center Networks

- Data center network properties:
  - Extremely short RTTs
  - Extremely high bandwidth links
  - Extremely large transfer
  - Single authority (typically)

- Leads to new challenges and opportunities
  - Can you think of any challenges for providing good transport solutions?
  - How about opportunities?

ECN + DCTCP

- Easier to ensure ECN is enabled on all devices in DCN
  - Single authority

- DCTCP
  - A variant of TCP
  - Uses ECN as congestion signal Use → reduced packet loss

- How does DCTCP work?
  - Congestion measured based on fraction of packets marked with ECN (called α).
    - α is the moving average of the observed fractions (like estimatedRTT)
  - Adjust congestion window based on the extent of congestion: $cwnd \leftarrow cwnd \times (1-\alpha/2)$
    - Instead of halving the window in case of congestion.

- DCTCP Properties:
  - More responsive and less aggressive to network conditions compared to traditional TCP
  - Keeps queue lengths shorter (as reacts faster) → low latency

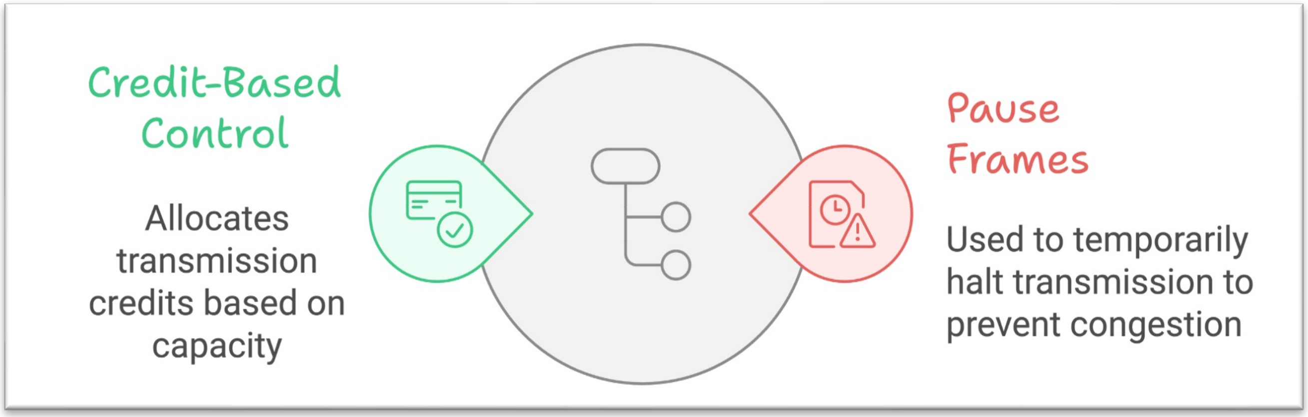
Link Layer Flow Control

Flow control mechanism used to create lossless networks.

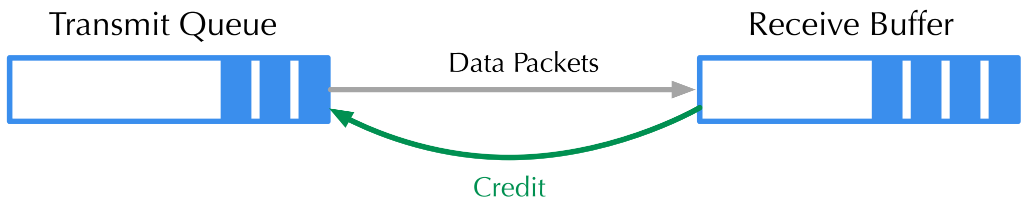
- Not to be confused with flow control in transport layer which is end-to-end.
- Setup: two nodes (end-hosts or switches) connected via a link.
  - Both have buffer (transmitter queue and receive buffer).
- Goal: ensure the receiver can handle the traffic injected on the link → no packet loss

Two prominent techniques:

- Credit-based
- Pause-based



Credit-Based Link-Level Flow Control

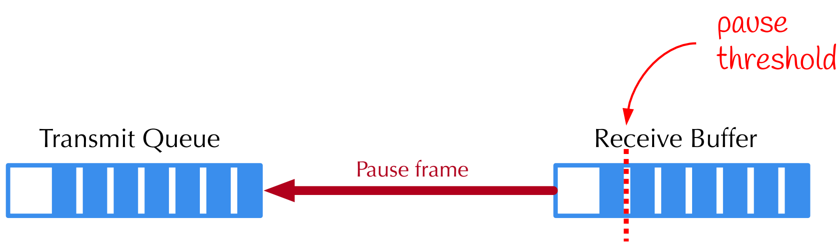


- Receiver provides credits to the sender when it has room
  - Credit unit: bytes or packets

- Credit allocation:
  - At the beginning certain (fixed) credit is allocated.
    - Question: how much credit should be allocated initially?
  - Transmitter uses credit when transmitting.
    - Pauses if there is no more credit available.
  - Receiver replenishes transmitter credits as receiver buffer becomes available.

- Note: we also can have credit-based congestion control. This is not what we are covering here. The concepts are similar, but at different layers.

Pause-based Link-Level Flow Control



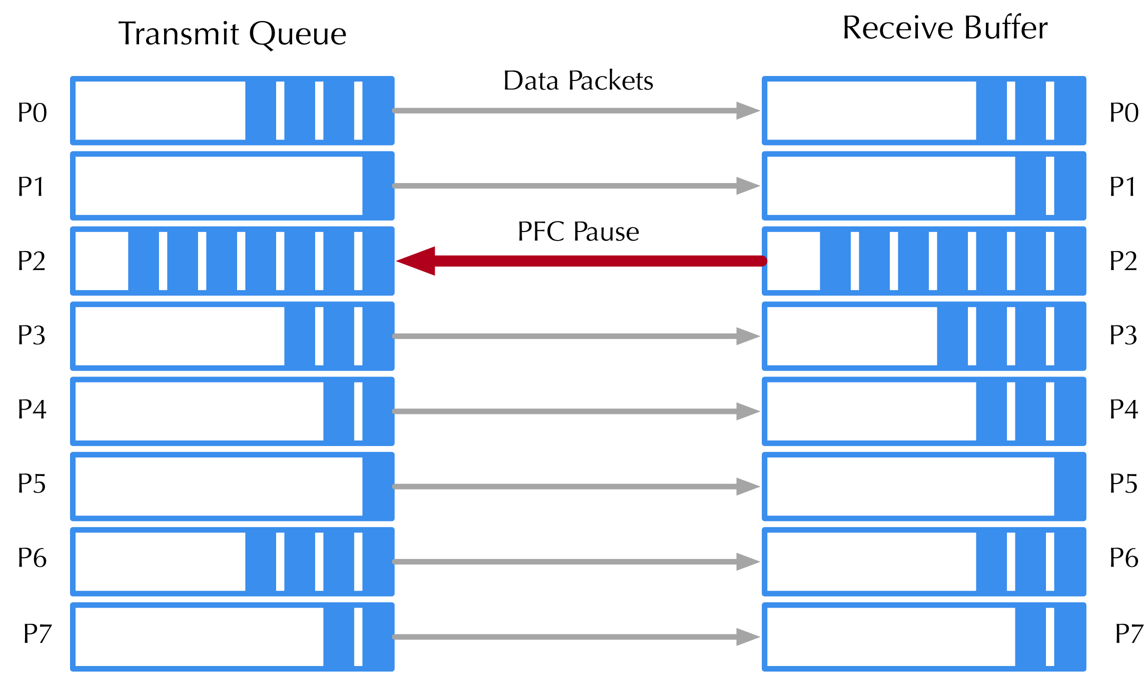
- Transmitter does not need permission to start.

- Receiver signals the transmitter when it is running out of buffer space.
  - When buffer occupancy goes above a fixed pause-threshold.
  - Pause-frame sent to transmitter
  - Transmitter halts transmission

- Once the receiver buffer has sufficient space …
  - Receiver sends a resume frame to the transmitter
  - The transmitter can resume sending.

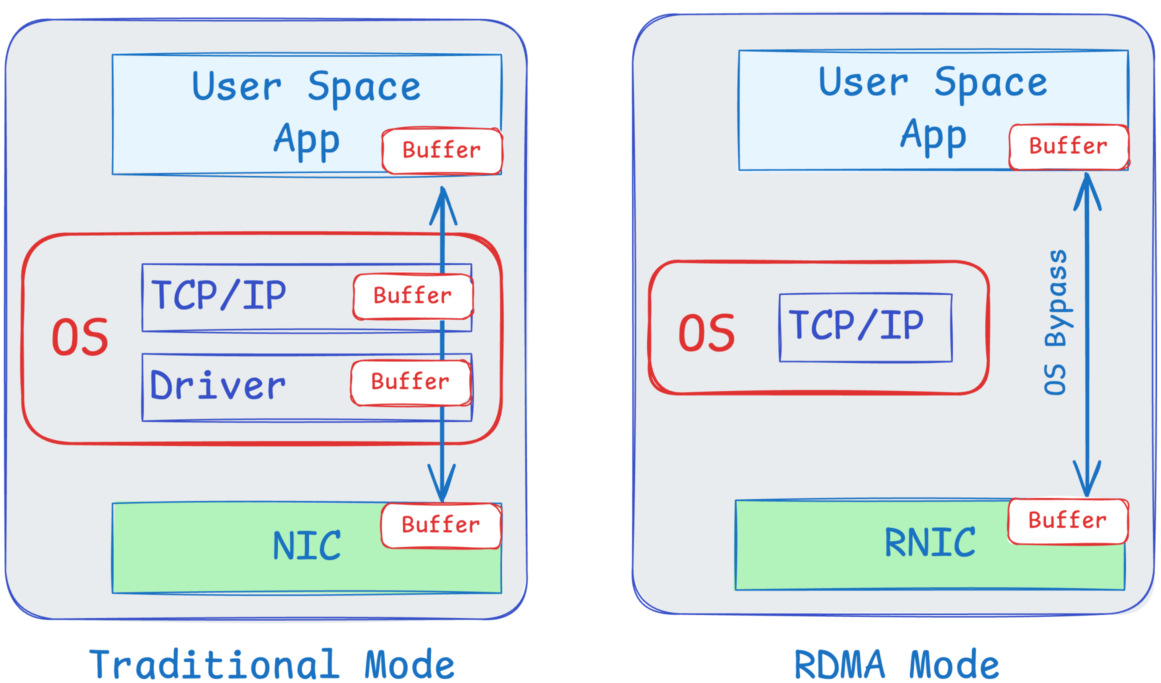
Priority-based Flow Control (PFC)

- Allows multiple priority-queues.
- Pause individual queues not all traffic
  - Allows other priority queues to continue transmitting even if a single queue is paused.
- Improves impact of pause on non-congested traffic to some extent
- Still, we might pause non-congested flows
  - Why?
  - Is there an easy way to solve this problem?
- Known issues: head-of-line blocking (deadlock), PFC storm



Remote Direct Memory Access

- Directly write to remote server's memory
  - Both sides register *memory regions* to give RDMA direct access permission and mapping
  - Need trust/cooperation between two ends
- RDMA-capable NIC (RNIC) handles data transfer entirely in hardware
  - No need to involve CPU for transfer
- Queue Pairs (QPs): a send queue and a receive queue.
  - Supports various operations like send, receive, read, and write.



Benefits of RDMA

- Low Latency
  - Minimizes delays by avoiding CPU intervention.
  - Ideal for applications requiring real-time data processing.

- High Throughput
  - Enables faster data transfer rates.
  - Suitable for high-performance computing and large data sets.

- Reduced CPU Load
  - Frees up CPU resources for other tasks.
  - Improves overall system efficiency.

- Zero-copy data transfer.
  - Eliminate (or minimize data copies)

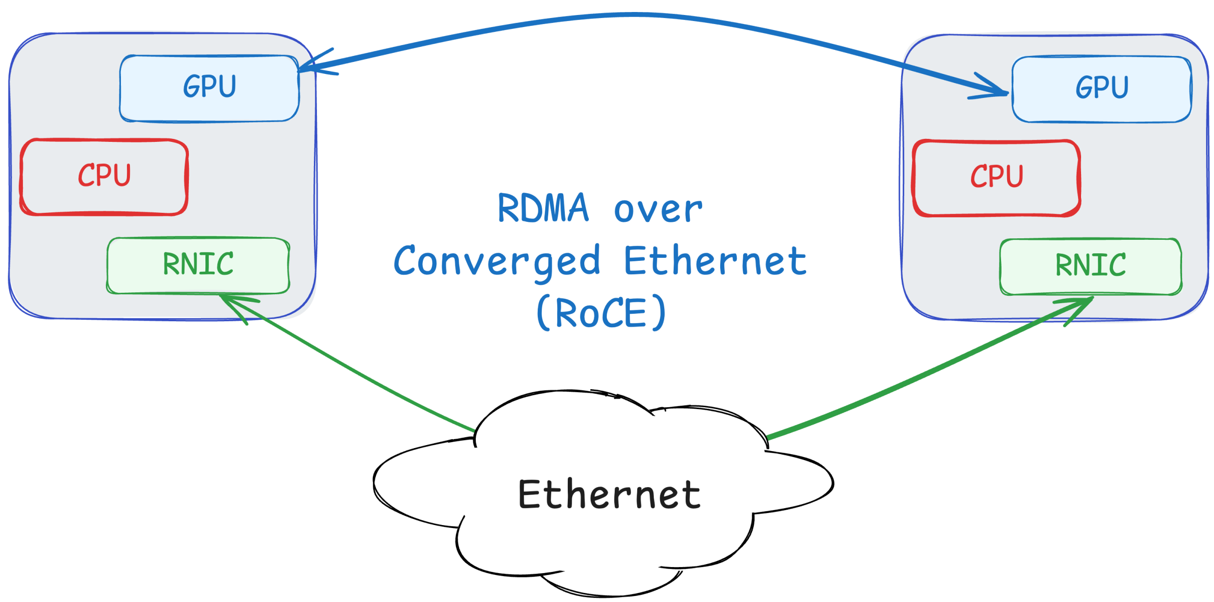
- Applications: High-Performance Computing (HPC), Storage, …

From Proprietary to Commodity

- Original technology InfiniBand
  - Touching physical layer, link layer, and transport layer in the stack.
  - Small number of vendors

- Later RDMA enabled over Ethernet
  - RoCE: RDMA over Converged Ethernet
  - With and without PFC support (RoCE v1 vs. v2)

- And even in WAN
  - iWARP (Internet Wide-Area RDMA Protocol)
  - Implemented over TCP/IP, no need for lossless network
  - Significant challenges here, especially over long distances



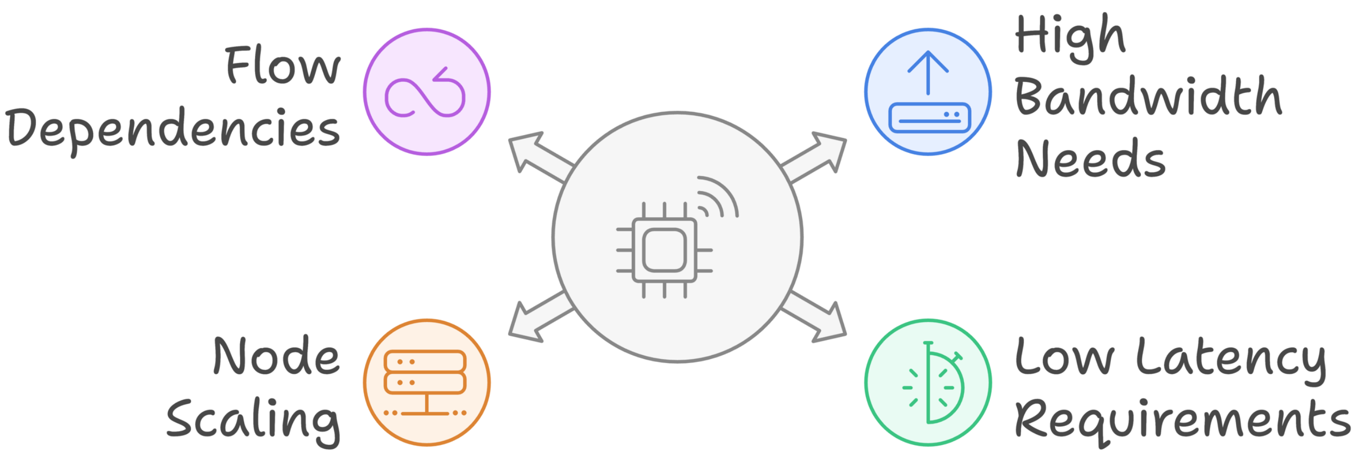
Networks for Machine Learning

- Data Center Networks have evolved significantly
  - To accommodate demands for modern applications.
  - Example: many novel congestion control algorithms in recent years:
    - Swift, timely, HPCC, DCQCN, …
    - Enablers: more accurate information from network (exact queue occupancy), assumptions about start rate (start at line rate), etc.

- Machine learning applications have grown significantly as well.
  - Used more in various domains, solving a wide range of problems.
  - At the same time, ML applications have higher demands from the underlying network

- Question: are existing DCN solutions enough?
  - I.e., can they provide the high-performance connectivity needed for ML applications?

Challenges: ML Workloads



- ML workloads can be extremely large:
  - E.g., Training of Large-Language Models (LLMs)
- Need various forms of parallelism
  - Data parallelism
  - Model parallelism
  - Hybrid
- ML workloads have extremely high requirements from the network:
  - High bandwidth
  - Low latency
  - Low jitter (variations in delay)
- Goal: Keep expensive GPUs busy

Challenges: End of Moore's Law

- Moore's Law: the number of transistors on a microchip will double approximately every two years.
  - For years, Moore's law meant we could easily grow compute power according to growth in demand.
- End of Moore's Law: recently, we have hit a wall and cannot continue growing compute per node as predicted by the Moore's Law.
  - However, the demand keeps growing …
  - For ML even faster than what Moore's law could handle.
- We need to add more nodes to scale to the demands of ML means.
  - More nodes → more communication → increased need for network.

All of this leads to significant pressure on the network (throughput, latency, reliability, …)

Challenges: Flow Dependencies

- Handling many independent flows in a network makes many network problems easy (easier) to solve
  - Random arrivals, each flow has a small share of bandwidth
  - Why?

- In ML, we have few flows
  - Having few flows means each flow can have a large fraction of link bandwidth
    - ⇒ Any interaction between flows can lead to significant performance degradation

- In ML flows have direct/indirect dependencies
  - Dependence between compute and communication
    - ⇒ flows directly or indirectly depend on each other
    - ⇒ Performance degradation in one flow can impact the performance of the entire job

- Providing high-performance connectivity for ML workloads is extremely challenging.
  - Due to scale, high-performance requirements (bandwidth, latency, …), larger flows, dependencies, …

- Example: load balancing in ML
  - Even two flows sharing a path can significantly reduce the overall performance.

What Makes ML Different: Opportunities

- ML workloads are more more predictable
  - Repeating patterns of communication
  - Collective Communications: scatter, gather, all-reduce, …

- Knowing communication patterns ⇒ opportunities for …
  - Building specialized hardware
    - E.g., topology that matches the flow requirements
      - Today's most successful ML networking solutions
  - Build network solutions that adapt based on application requirements
    - Application-aware scheduling
    - Reconfigurable topology
    - Adaptive routing, …

- Even without prediction, access to "Collective Communication Libraries" can provide significant opportunities.
  - Examples to come.

Network Topology

- DCN topology: fat-tree, leaf-spine, …
  - Static and uniform
  - Needs to work with a wide range of workloads
    - Tuned for average workload

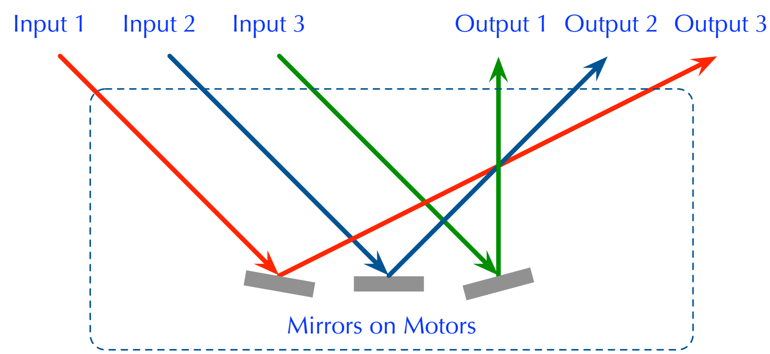
- Distributed ML application have high demand which is not necessarily uniform
  - More demand for certain paths

- Two options to deal with extra traffic
  1. Add extra capacity and over-provision; or
  2. Use existing capacity better:
     - Measurement studies show there is extra capacity available, it just needs to be used effectively

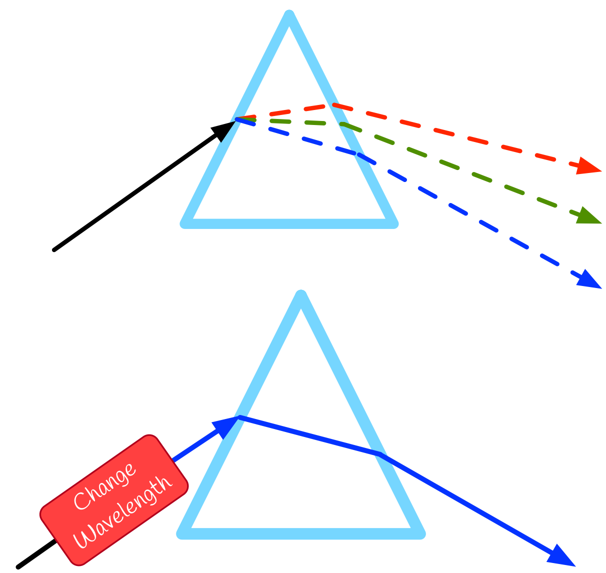
- How can we adapt the topology to match demand?
  1. Customized the topology for specific classes of traffic
     - Special-purpose design
     - Can be optimal, but very expensive
     - What happens if new communication patterns emerge?
  2. Dynamically rearrange the topology
     - How?

Reconfigurable DCN Topology

- Needs to adapt topology to workload
  - Even more important when you share the infrastructure:
    - Cloud-based ML training, or various ML jobs sharing the network
- Optical Circuit Switching:
  - Reconfigure input/output connectivity of switch ports
  - Avoids electronic-optic-electronic conversion
  - Various technologies:
    - MEMS-based Switching: mechanically rearrange mirrors to change connectivity of ports
    - Arrayed Wave Guide (AWG) Switching: change wavelength to connect to different output ports
- Benefit: shifting capacity to where it is needed on demand
- Challenge: reconfiguration might take some time
  - Not ideal for typical packet switching scenarios
- What if we know the demand and it is fairly stable?
  - Google's Jupiter: estimate demand matrix, adapt topology using a fast control plane
- ML Workloads are predictable ⇒ opportunity to change the switch connectivity to meet demand in real time
- RDCN performance improvements:
  - High bandwidth (1.6-4x increased in available bandwidth), 70-80% lower power, micro to nano-second scale delay



MEMS-based Optical Circuit Switch



Adapting Other Network Functions

- Topology is only one dimension
- What about: routing, prioritization, scheduling, …?
- How can we optimize network behavior based on application requirements?

Application-Aware Networking

- To optimize network behavior, we need to provide information about application requirements.
  - Application-Aware Networking
  - Today's networks lack this information

- Main Question: how can we provide information from applications to the network?

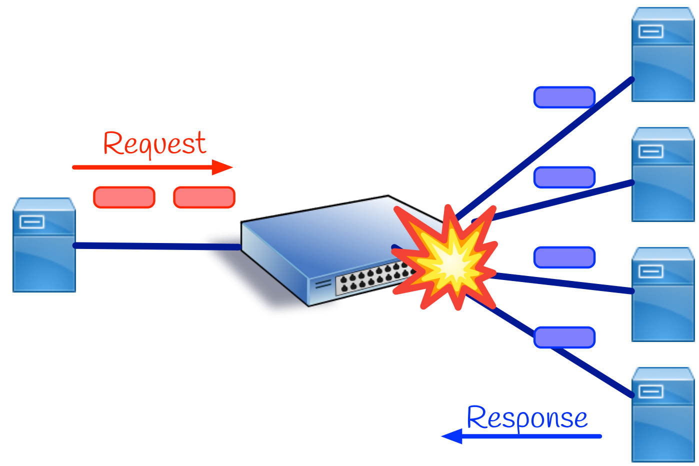
- Naïve approach:
  - Create new interfaces between network and applications.
  - Allow application developers provide more information about the requirements
    - E.g., I need 10Gb/s bandwidth for 2 seconds.

- This is not very practical
  - Putting the burden on application developer
  - She/he might not even know the requirements

- Alternative: create tools/mechanisms to automatically generate signals that can help network adjust itself based on application requirements, or state.

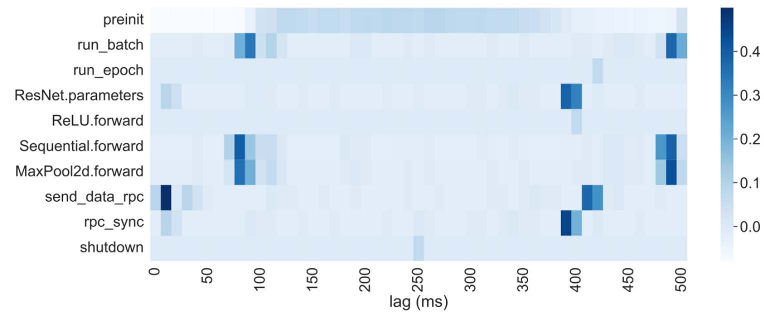
Example: Incast Problem

- Consider a simple multi-get request from a distributed storage.
  - Get some content from several servers
- The request can trigger large number of flows.
  - Quickly fill up the buffer at switch → lots of packet drops
  - This is called the incast problem
  - Very challenging problem in DCNs
- Network does not know when incast will happen
  - Cannot react in a timely manner.
  - Over-provisioning seams to be the only viable solution.
- Applications, however, have information that can be used to predict incast.
  - E.g., multi-get request can be seen as a hint.

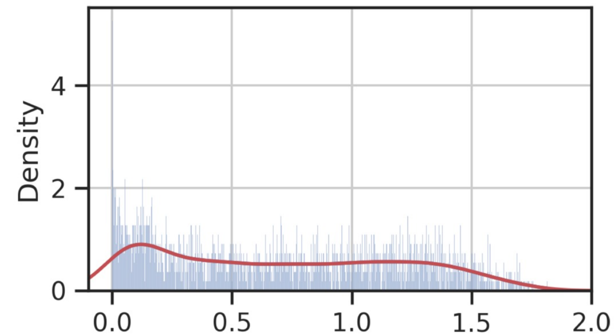


Networks and Applications*

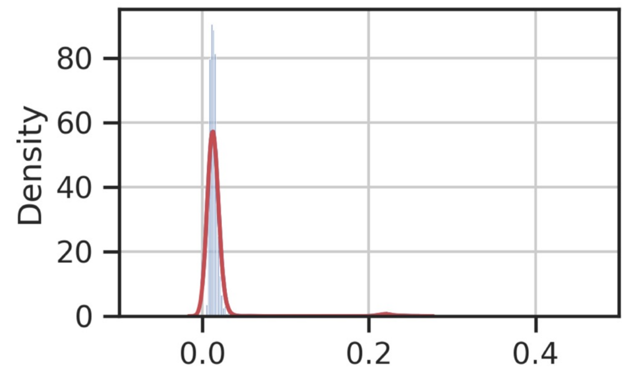
- Imagine, we have a system that observes *network events*
  - E.g., incast in switches
- Also, it can observe certain events in applications
  - E.g., when each function is called
- We correlate these events find out triggers on the end-host side for network events
  - I.e., which application events lead to network problems
  - E.g., each incast event in the network is preceded by a multi-get request 1 RTT before



Correlation between selected function calls and micro-bursts for a distributed ML application.



Probability of Incast in Future

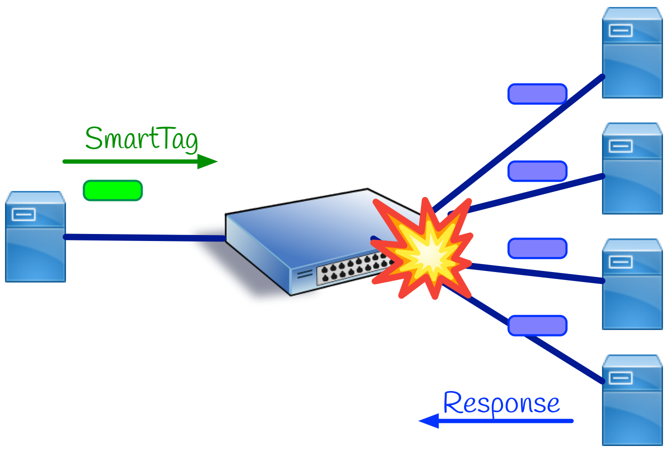


Incast Probability with SmartTag

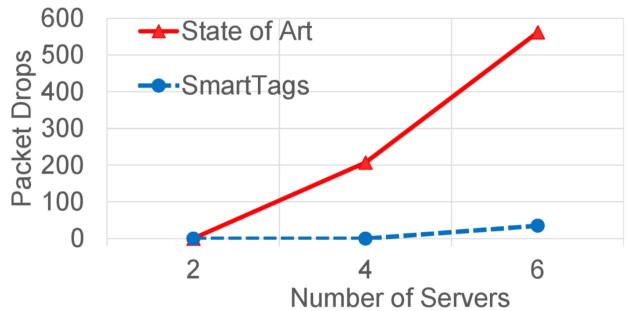
* Mortazavi, S.H., Munir, A., Bahnasy, M.M., Dong, H., Wang, S. and Ganjali, Y. 2022. EarlyBird: automating application signalling for network application integration in datacenters. Proceedings of the ACM SIGCOMM Workshop on Network-Application Integration (New York, NY, USA, Aug. 2022), 40–45.

SmartTags*

- We can use this information to generate messages on the end-host to notify the network of future network events.
  - We call these SmartTags.
- We can use SmartTags to change the behavior of the network
  - Example. To alleviate incase we can reroute traffic, delay certain flows, …
- Can lead to significant improvements in network behavior.
  - Improvements that are not possible in today's networks.
- Can be very effective for ML applications.
  - Automatically predict flow arrivals to change topology, reroute traffic, adjust flow priorities, etc.
  - Many more opportunities …



SmartTags are indicators of future events: opportunity to prepare network.



Packet Drops with and without SmartTags

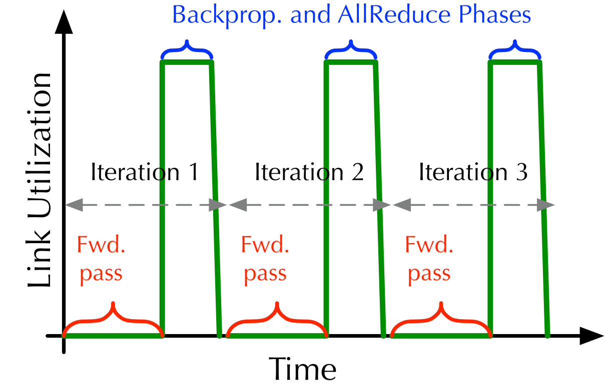
* Munir, A., Mortazavi, S.H., Bahnasy, M.M., Baniamerian, A., Wang, S., Guan, S. and Ganjali, Y. 2022. SmartTags: bridging applications and network for proactive performance management. Proceedings of the ACM SIGCOMM Workshop on Network-Application Integration (New York, NY, USA, Aug. 2022), 46–52.

Network Aware Applications

- Application-Aware Networking:
  - Adapt network based on application requirements, signals, …
- Given tighter integration of applications and network why not use network information to help applications?
  - Adapt application to network state.
- Example: scheduling ML jobs based on network state
  - Each application is aware of its own requirements at best.
    - But not network state (e.g., available bandwidth): application must estimate network state by probing
  - Network can provide information about its state
    - Help application-level scheduling
    - As well as state of other applications

Cassini: Network-Aware Job Scheduling*

- CASSINI is a network-aware job scheduler for ML clusters
  - Goal: schedule ML jobs based on network state and other jobs in the system to ensure smooth operation
- Step 1. Profile individual jobs
  - Identify communication patterns
  - Why can we do this here?
- Step 2. Convert to a geometric abstraction that represents the network demand
  - Perimeter of the circle: job's iteration time
  - Arcs of the circle: job's up and down phases.
- Question. How can we use this geometric abstraction to find good job schedules?
  - Rotating the circle is equivalent to shifting the job in time.



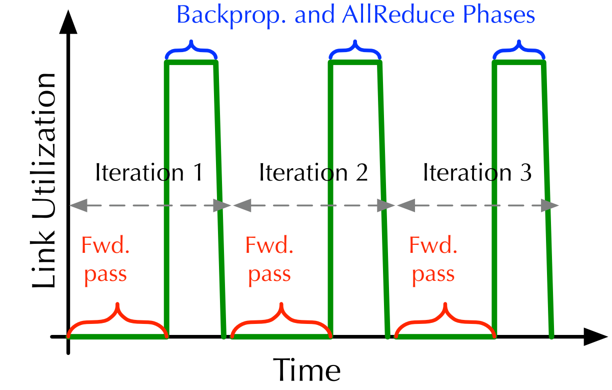
Data Parallelism Communication Patterns

Similar, but more complex, patterns for other type of parallelism

* Rajasekaran, S., Ghobadi, M. and Akella, A. 2024. CASSINI: Network-Aware Job Scheduling in Machine Learning Clusters. (2024), 1403–1420.

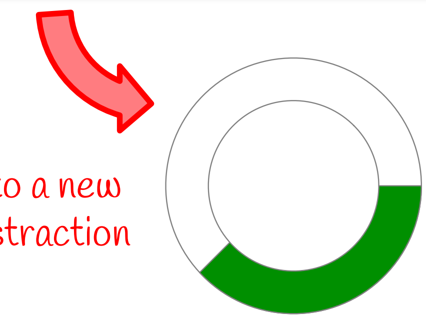
Cassini: Network-Aware Job Scheduling

- CASSINI is a network-aware job scheduler for ML clusters
  - Goal: schedule ML jobs based on network state and other jobs in the system to ensure smooth operation
- Step 1. Profile individual jobs
  - Identify communication patterns
  - Why can we do this here?
- Step 2. Convert to a geometric abstraction that represents the network demand
  - Perimeter of the circle: job's iteration time
  - Arcs of the circle: job's up and down phases.
- Question. How can we use this geometric abstraction to find good job schedules?
  - Rotating the circle is equivalent to shifting the job in time.

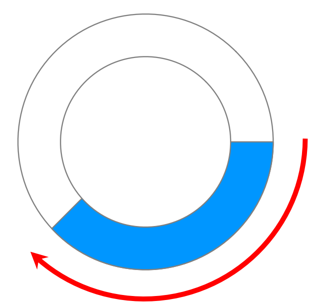


Data Parallelism Communication Patterns

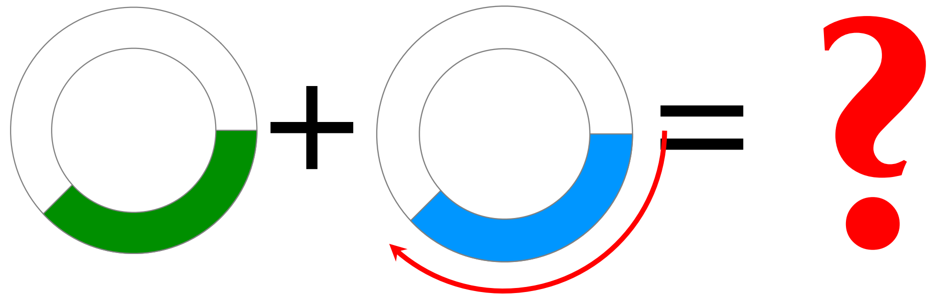
Convert to a new geometric abstraction



How much should we rotate each circle?



Cassini: Network-Aware Job Scheduling

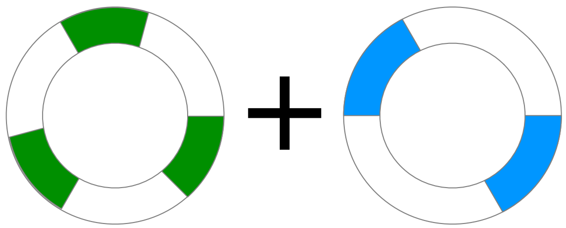


- Step 3. For each link in the network:
  - Overlays the circles of jobs going through the link and rotates them
  - Find a configuration that minimizes the total bandwidth demand
  - This gives us a set of relative shifts in time

- Step 4. Extend link-level compatibility to cluster level
  - Start with job 1, set time to 0.
  - For each job that shares a bottleneck, use the method above to find the required shift in time.

- If there is no loop the output would be a job schedule
  - I.e., when each job should start

- Question 1: what if jobs have different iteration periods?
- Question 2: can we have a loop in the graph above?



ML for Networks

- So far, we have focus on how to enhance networks to meet the stringent requirements of ML applications.

- Given the advances in ML, a natural question is: *how can we use ML to enhance computer networks?*

- Any suggestions?

ML for Networks Examples - Part 1

Traffic Prediction and Forecasting:

- We saw how knowing the pattern of traffic in ML workloads can help us make the network better.
- How about using ML to predict network traffic patterns
  - To optimize resource allocation, …
- We can use time-series models (e.g., ARIMA, LSTMs, etc.).
  - Analyze historical traffic data to anticipate congestion.

Congestion Control:

- We have seen some examples of how congestion control can be enhanced for certain environments (DCN, wireless, …) and workloads
- We can use ML to dynamically adjust flow rates
  - Minimize packet loss, delay, …
  - Also, use ML-based prediction of queue lengths or latency
- Train reinforcement learning (RL) models for real-time congestion-mitigation
- Examples: Orca, PCC-Vivace, …

* Abbasloo, S., Yen, C.-Y. and Chao, H.J. 2020. Classic Meets Modern: a Pragmatic Learning-Based Congestion Control for the Internet. *Proceedings of the Annual conference of the ACM Special Interest Group on Data Communication on the applications, technologies, architectures, and protocols for computer communication* (Virtual Event USA, Jul. 2020), 632–647.
** Mo Dong, Tong Meng, Doron Zarchy, Engin Arslan, Yossi Gilad, Brighten Godfrey, and Michael Schapira. 2018. PCC -Vivace: Online-Learning Congestion Control. In 15th USENIX Symposium on Networked Systems Design and Implementation (NSDI'18). 343–356.

ML for Networks Examples – Part 2

Network Routing and Load Balancing:

- Typically, rely on static routes
  - Shortest path based on fixed costs and randomized load balancing
- If information about link loads are available, we can use reinforcement learning for adaptive path selection.
  - Distribute load evenly across servers in data centers.
- E.g., apply graph neural networks (GNNs) to analyze network topologies, and find optimal routers.

Network Configuration Automation:

- Configuring switch/routers a tedious task, typically done manually
- We can use ML to automate switch/router configurations.
  - Use natural language processing (NLP) to translate operator requirements to device configuration.
  - Train models on historical configuration logs to predict optimal settings.
- Avoid misconfigurations (that can lead to outages) and optimize network performance.

ML for Networks Examples – Part 3

Network Management and Troubleshooting:

- Automatic configuration is done when the network is setup.
  - We can also think of more dynamic scenarios.
- How can we manage the network?
  - Ensure optimized behavior
- We have talked about SDN control applications.
  - Routing, access control, load balancing, …
- A management layer above can help make high-level decisions on resource allocations, adapting control applications, etc.
  - Ideally, use natural language to describe the intent of network operators, called intent-based networking
- ML can provide the tools needed to convert operator intent to rules, and policies
  - Push to the network through SDN control and management plane.
  - Also, optimize behavior based on operator intent.
- ML can also provide mechanisms for automated interaction with customers
  - E.g., troubleshooting customer systems in real-time.