Datacenter Networks

CPSC 433/533, Spring 2021
Anurag Khandelwal

Recap

- What is a data center network?
  - Scale, service-model, application characteristics
- What makes it different?
  - Characteristics, goals (w.r.t. internet), degrees of freedom
- How do we achieve goals by exploiting freedom?
  - Topology redesign, L2/L3 redesign, L4 redesign

Recap

- What is a data center network?
  - Scale, service-model, application characteristics
- What makes it different?
  - Characteristics, goals (w.r.t. internet), degrees of freedom
- How do we achieve goals by exploiting freedom?
  - Topology redesign, L2/L3 redesign, L4 redesign
  - We will look at some approaches, not all

How do we achieve DCN goals?

- Network architecture design [Done]:
  - rearchitect network topology to achieve full-bisection b/w
  - DC as a giant switch => Clos/fat-tree topologies

- L2/L3 design:
  - Via modifications to LS/DV + ECMP
  - New addressing / routing / forwarding for new topology

- L4 design **[Today]**:
  - transport protocol design to meet DCN goals

How do we achieve DCN goals?

- L4 design:
  - Transport protocol design (w/ Fat-Tree)

Many slides courtesy of Mohammad Alizadeh, MIT

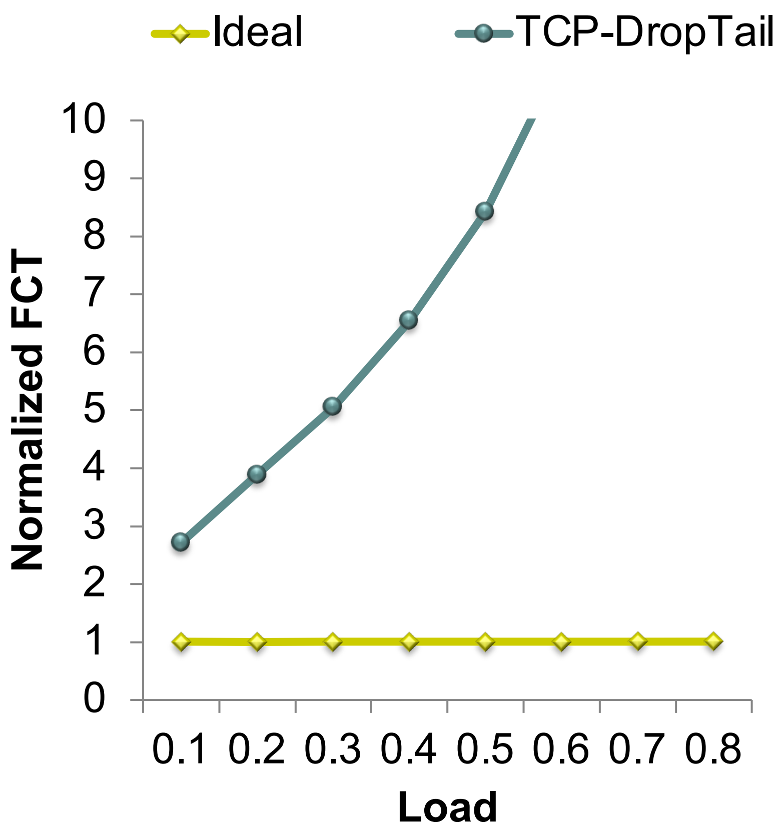
What’s “ideal” ?

- What is the best measure of performance for a data center transport protocol?
  - Latency of each packet in the flow?
  - Number of packet drops?
  - Link utilization?
  - Average queue length at switches?
  - When the flow is completely transferred?
- What does the application/user care about?

Flow Completion Time (FCT)

- Time from when flow started at the sender, to when all packets in the flow were received at the receiver

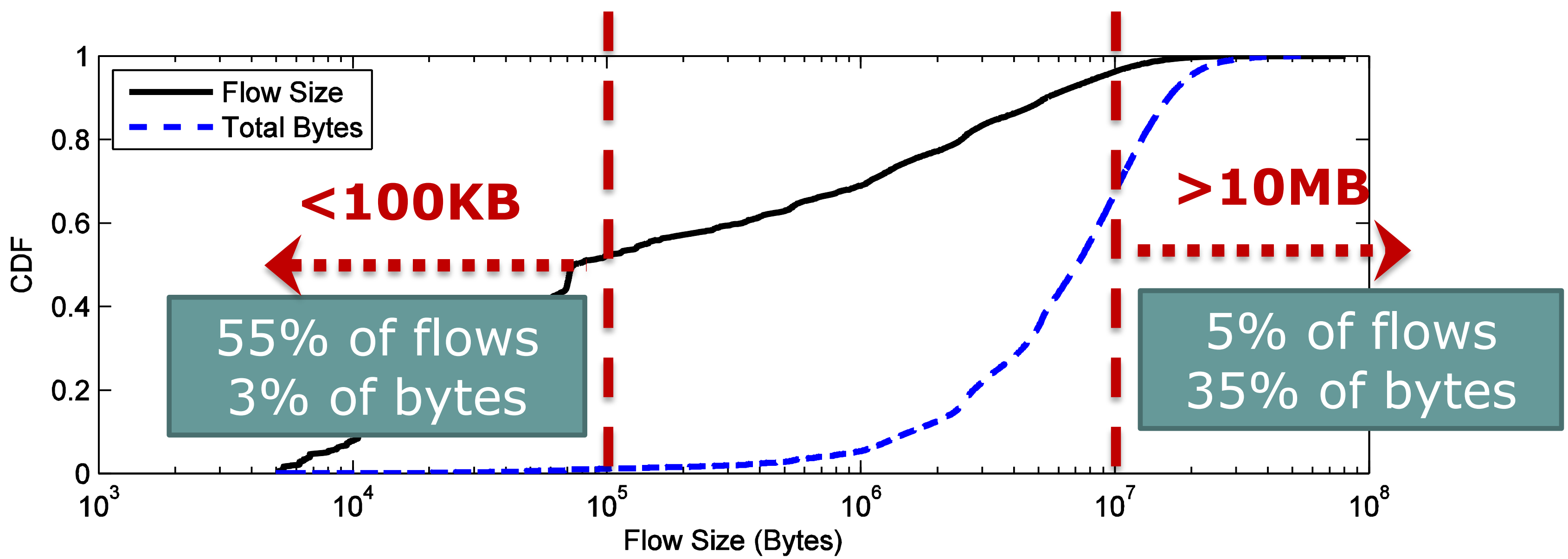
FCT with TCP



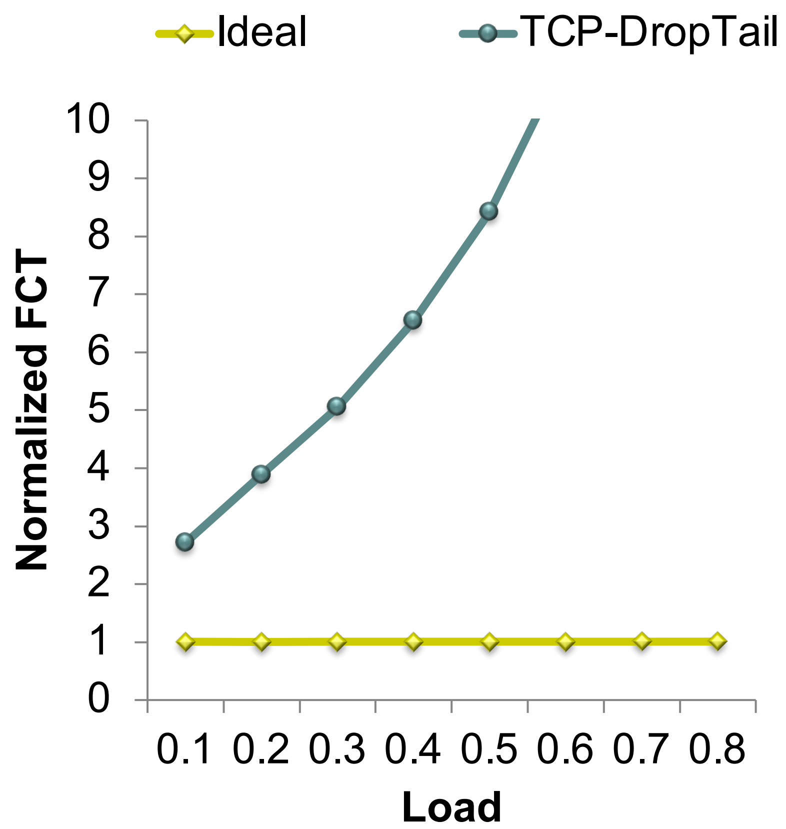
TCP-DropTail: TCP with “drop-tail” queues at switches (drops packets at the tail of the queue)

Recall: “Elephants” and “Mice”

- Microsoft [Alizadeh 2010]
  - Web search (north-south), data mining (east-west)



FCT with TCP

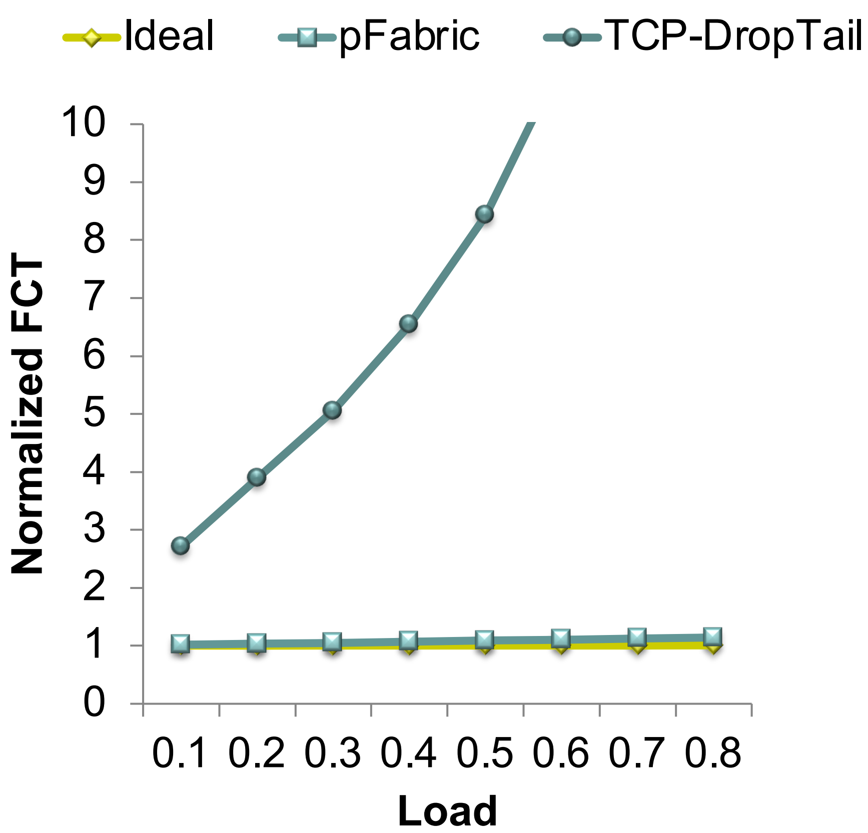


Problem: the mice are delayed by the elephants

Solution: use priorities! *[pFabric, Sigcomm 2013]*

- Packets carry a single priority number
  - priority = remaining flow size (e.g., #bytes un-ACKed)
- Switches
  - very small queues (e.g., 10 packets)
  - send highest priority / drop lowest priority packet
- Servers
  - Transmit/retransmit aggressively (at full link rate)
  - Drop transmission rate only under extreme loss (timeouts)

FCT


Ideal
pFabric
TCP-DropTail
Normalized FCT
10
9
8
7
6
5
4
3
2
1
0
0.1
0.2
0.3
0.4
0.5
0.6
0.7
0.8
Load

Why does pFabric work?

- Consider problem of scheduling N jobs at a queue
  - $J_{1,}$ J_2, … , J_n with duration T_1, T_2, …T_n respectively
- How do you minimize average job completion time?
- "Shortest Job First" (SJF) scheduling
  - Pick job with minimum T_i ; de-queue and run ; repeat
  - I.e., job that requires minimum runtime has max priority
- Solution for a network of queues is NP-hard
- Setting priority = remaining flow size is a heuristic to approximate SJF

Questions?

Network Management & Software Defined Networks

Slides thanks to Scott Shenker, one of the pioneers of SDN

Goal for rest of the discussion

- Provide the “why” of software-defined networking
  - Deeper understanding of the problem
  - An exercise in architectural thinking
  - To build a principled approach towards a solution

- Only a high level of the “what”
  - Enough that some of you will want to know more
  - Take advanced networking course or do research!

What is Network Management?

- Recall the two “planes” of networking

- **Data plane**: forwarding packets
  - Based on local forwarding state

- **Control plane**: computing that forwarding state
  - Involves coordination with rest of system

- Broad definition of “network management”:
  - ***Everything having to do with the control plane***

Original goals for the control plane

- **Basic connectivity**: route packets to destination
  - Local state computed by routing protocols
  - Globally distributed algorithms
- **Inter-domain policy**: find policy-compliant paths
  - Done by globally distributed BGP
- For long time, these were the only relevant goals!
  - *What other goals are there in running a network?*

Isolation

- L2 broadcast protocols often used for discovery
  - Useful but unscalable and invasive
- Want multiple logical LANs on a physical network
  - Retain usefulness, cope with scaling, provide isolation
- Use VLANs (virtual LANs) tags in L2 headers
  - Controls where broadcast packets go
  - Gives support for logical L2 networks
  - Routers connect these logical L2 networks

Access Control

- Operators want to limit access to various hosts
  - “Don’t let laptops access backend database machines”
- This can be imposed by routers using ACLs
  - ACL: Access Control List
- Example entry in ACL: <header template; drop>
  - If not port 80, drop
  - If source address = X, drop

Traffic Engineering

- Want to avoid persistent overloads on links
- Choose routes to spread traffic load across links
- Example:
  - Adjusting weights in OSPF
- Done with centralized computation
  - Take snapshot of topology and load
  - Compute appropriate OSPF state
  - Send to network

Network management has many goals

- Achieving these goals is job of the control plane…
- …which currently involves many mechanisms

- **Globally distributed:** routing algorithms
- **Manual/scripted configuration:** ACLs, VLANs
- **Centralized computation:** Traffic engineering

Bottom Line

- Many different control plane mechanisms
- Each designed from scratch for their intended goal
- Encompassing a wide variety of implementations
  - Distributed, manual, centralized,…
- **Network control plane is a complicated mess!**

Questions?

Making Network Operators Cry:
A Two Step Process

Step 1: Large datacenters

- 100,000s machines; 10,000s switches
- This is pushing the limits of what we can handle...

Step 2: Multiple tenancy

- Large datacenters host many customers
- Each customer gets their own logical network
  - Customer should be able to set policies on this network
  - ACLs, VLANs, etc.
- If there are 1000 customers, that adds 3 orders of magnitude
- This goes *way* beyond what we can handle

Network Operators Are Now Weeping

- Ad hoc control for millions of networked entities
  - And something goes wrong
  - Try debugging that…
- What is the key problem here?
  - ***Complexity***
- We need a simpler, more systematic design
  - ***How do we achieve this?***

An Example Transition: Programming

- Machine languages: no abstractions
  - Had to deal with low-level details
  - Mastering complexity was crucial

- Higher-level languages: OS and other abstractions
  - File system, virtual memory, abstract data types, ...

- Modern languages: even more abstractions
  - Object orientation, garbage collection,...

Abstractions key to extracting simplicity

What About Network Abstractions?

- Consider the data and control planes separately
- Different tasks, so naturally different abstractions

Abstractions for Data Plane: Layers

Applications

…built on…

Reliable (or unreliable) transport

…built on…

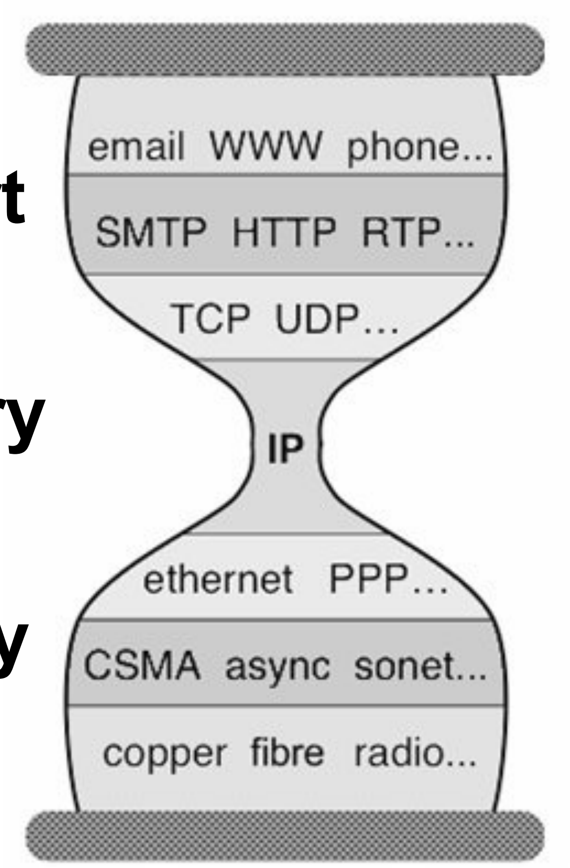
Best-effort global packet delivery

…built on…

Best-effort local packet delivery

…built on…

Physical transfer of bits



The Importance of Layering

- Decomposed delivery into basic components
- Independent, compatible innovation at each layer
  - Clean "separation of concerns"
  - Leaving each layer to solve a tractable problem
- Responsible for the success of the Internet!
  - Rich ecosystem of independent innovation
- Think about it….
  - Original architecture has handled many order-of-magnitude changes in speed, size, scope, diversity

Control Plane Abstractions

?

Many Control Plane Mechanisms

- Variety of goals, no modularity:
  - **Routing:** distributed routing algorithms
  - **Isolation**: ACLs, VLANs, Firewalls,…
  - **Traffic engineering**: adjusting weights, MPLS,…

- **Control Plane: mechanism without abstraction**
  - ***Too many mechanisms, not enough functionality***

Questions?

Finding Control Plane Abstractions

How do you find abstractions?

- You first decompose the problem….
- …and define abstractions for each sub-problem
- **So what is the control plane problem?**

Computing forwarding state

- Consistent with low-level hardware/software
  - Which might depend on vendor

- Based on entire network topology
  - Because many control decisions depend on topology

- For all routers/switches in network
  - Every router/switch needs forwarding state

Our current approach

- Design one-off mechanisms that solve all three
- Introduces a lot of complexity
  - Think back to the DCN operator

Separate Concerns with Abstractions

- Be compatible with low-level hardware/software
  - Need an abstraction for general **forwarding model**
- Make decisions based on entire network
  - Need an abstraction for **network state**
- Compute configuration of each physical device
  - Need an abstraction that **simplifies configuration**

Separate Concerns with Abstractions

- Be compatible with low-level hardware/software
  - Need an abstraction for general **forwarding model**
- Make decisions based on entire network
  - Need an abstraction for **network state**
- Compute configuration of each physical device
  - Need an abstraction that **simplifies configuration**

Abs#1: Forwarding Abstraction

- Express intent **independent of implementation**
  - Don't want to deal with proprietary HW and SW
- OpenFlow is current proposal for forwarding
  - **Standardized** interface to switch
  - Configuration in terms of flow entries: <header, action>
- Design details concern exact nature of:
  - Header matching
  - Allowed actions

Two Important Facets to OpenFlow

- Switches accept external control messages
  - Not closed, proprietary boxes
- Standardized flow entry format
  - So switches are interchangable

Separate Concerns with Abstractions

- Be compatible with low-level hardware/software
  - Need an abstraction for general **forwarding model**
- Make decisions based on entire network
  - Need an abstraction for network state
- Compute configuration of each physical device
  - Need an abstraction that simplifies configuration

Abs#2: Network State Abstraction

- Abstract away various distributed mechanisms

- Abstraction: **global network view**
  - Annotated network graph provided through an API

- Implementation: “Network Operating System”
  - Runs on servers in network (“controllers”)

- Information flows both ways
  - Information *<u>from</u>* routers/switches to form “view”
  - Configurations *<u>to</u>* routers/switches to control forwarding

Network Operating System

- Think of it as a centralized link-state algorithm
- Switches send connectivity info to controller
- Controller computes forwarding state
  - Via some control program that uses the topology as input
- Controller sends forwarding state to switches
- Controller is replicated for resilience
  - System is only “logically centralized”

Software Defined Network (SDN)

Network of Switches and/or Routers

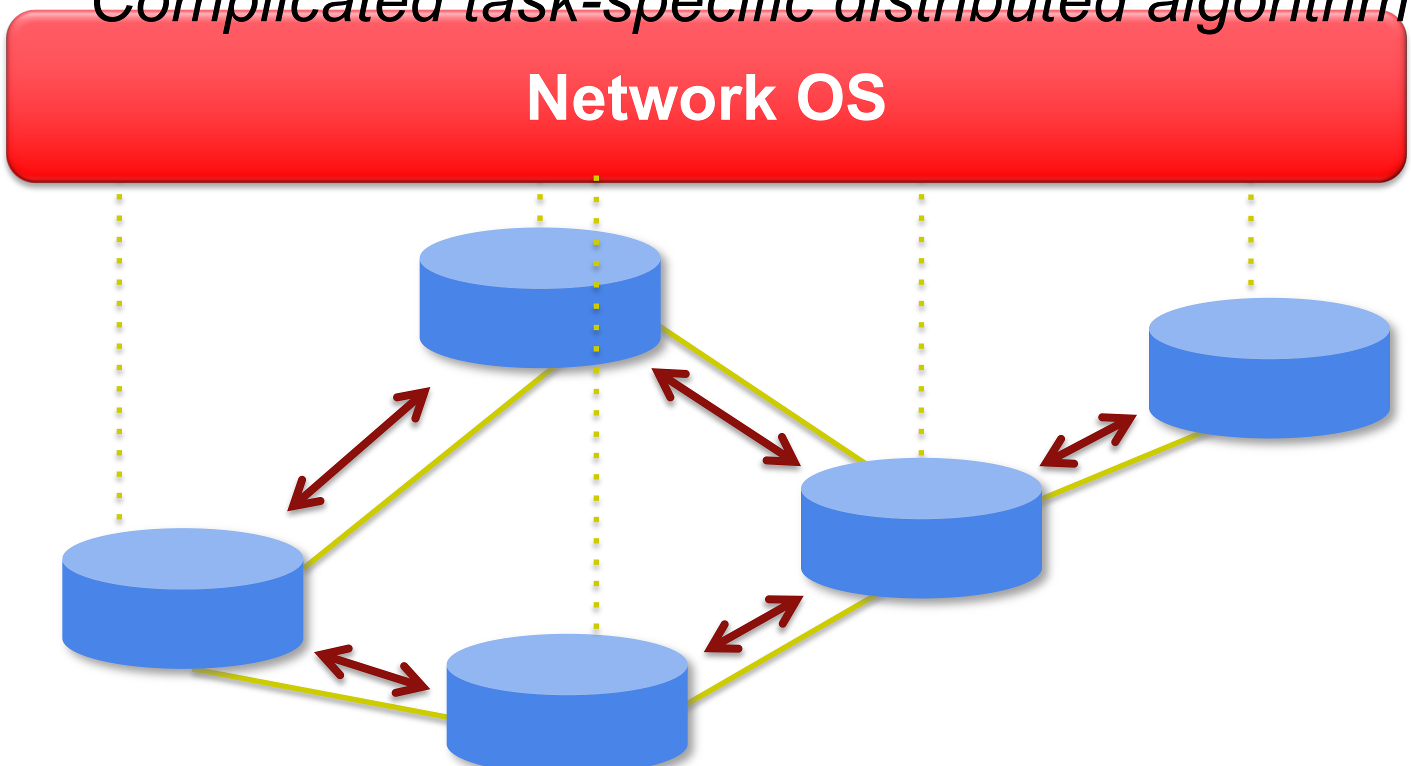
routing, access control, etc.

Control Program

Distributed algorithm running between neighbors

Global Network View

Complicated task-specific distributed algorithm



Major Change in Paradigm

- Control program:
  - Network configuration is a function of the global view
- Control mechanism now program using NOS API
- Not a distributed protocol, just a graph algorithm

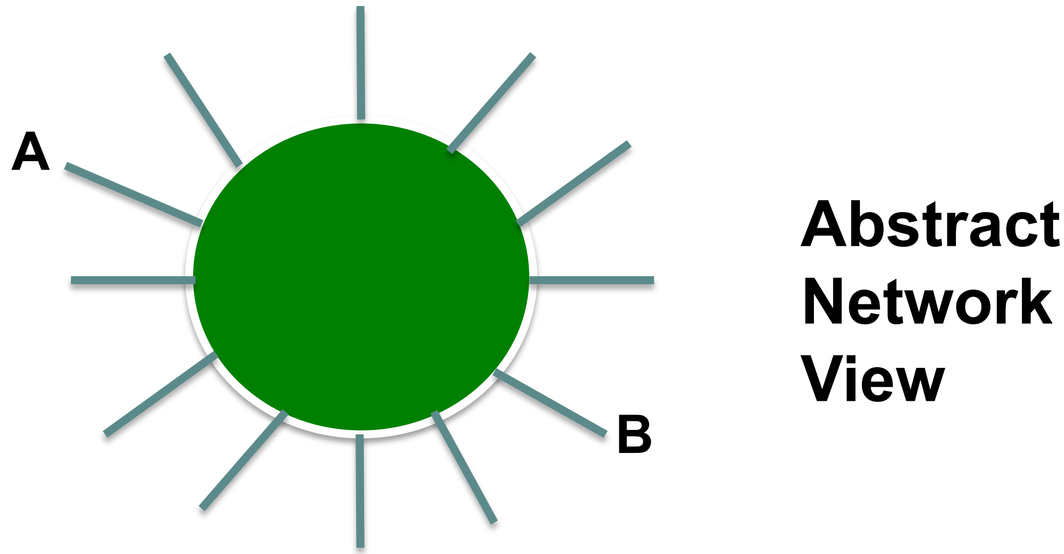
Separate Concerns with Abstractions

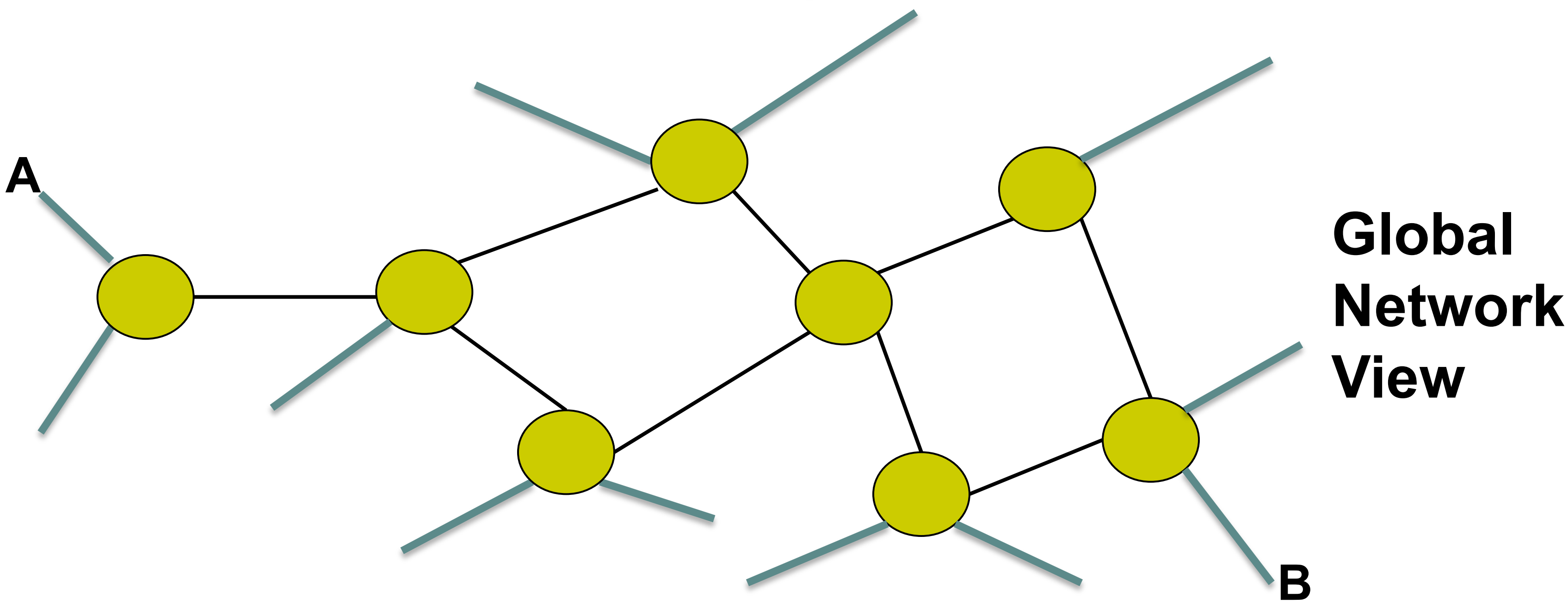
- Be compatible with low-level hardware/software
  - Need an abstraction for general forwarding model
- Make decisions based on entire network
  - Need an abstraction for network state
- Compute configuration of each physical device
  - Need an abstraction that simplifies configuration

Abs#3: Specification Abstraction

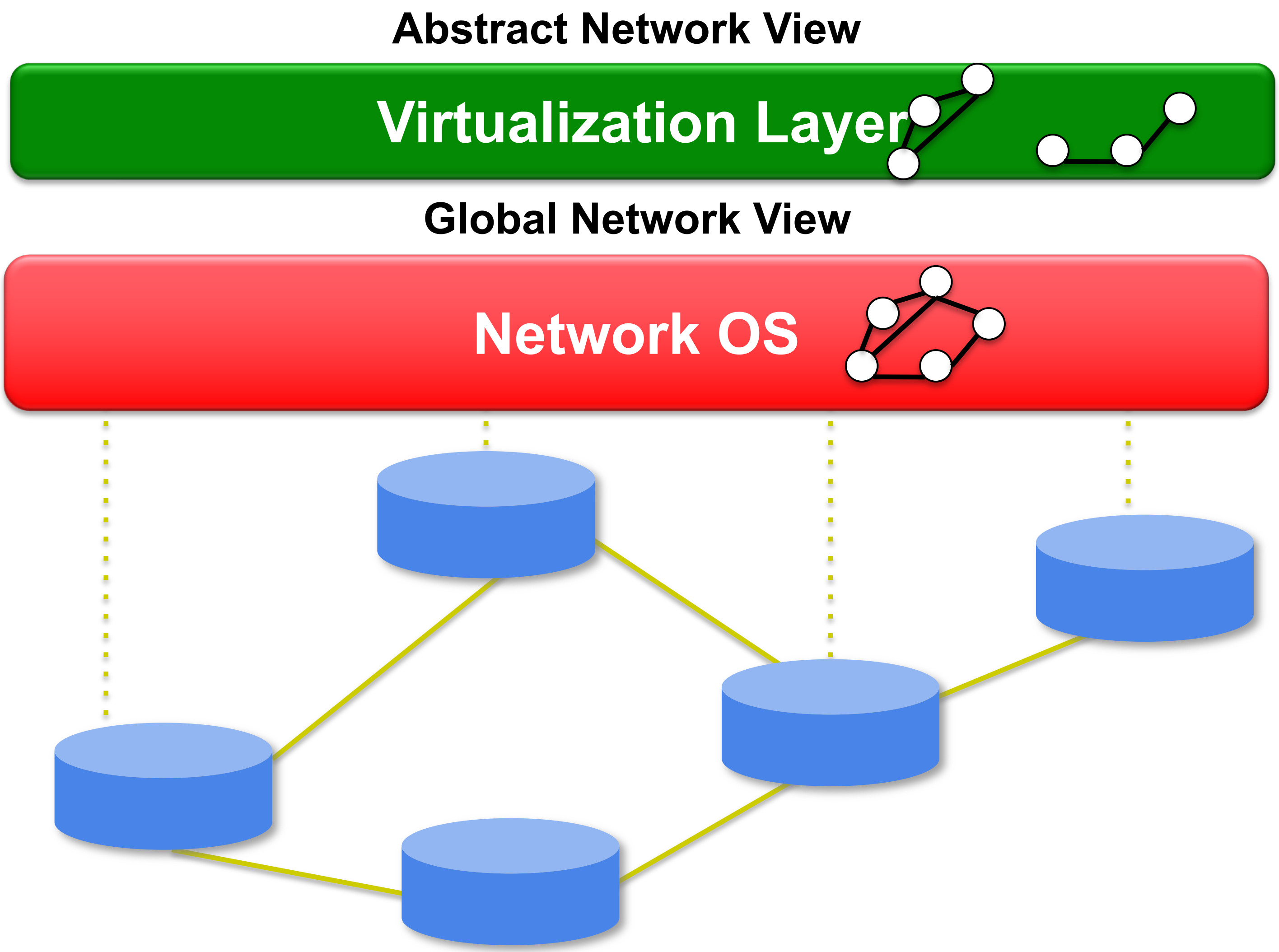
- Control mechanism expresses desired behavior
  - Whether it be isolation, access control, or QoS
- It should not be responsible for ***implementing*** that behavior on physical network infrastructure
  - Requires configuring the forwarding tables in each switch
- Proposed abstraction: **abstract view** of network
  - Abstract view models only enough detail to <u>*specify goals*</u>
  - Will depend on task semantics

Simple Example: Access Control





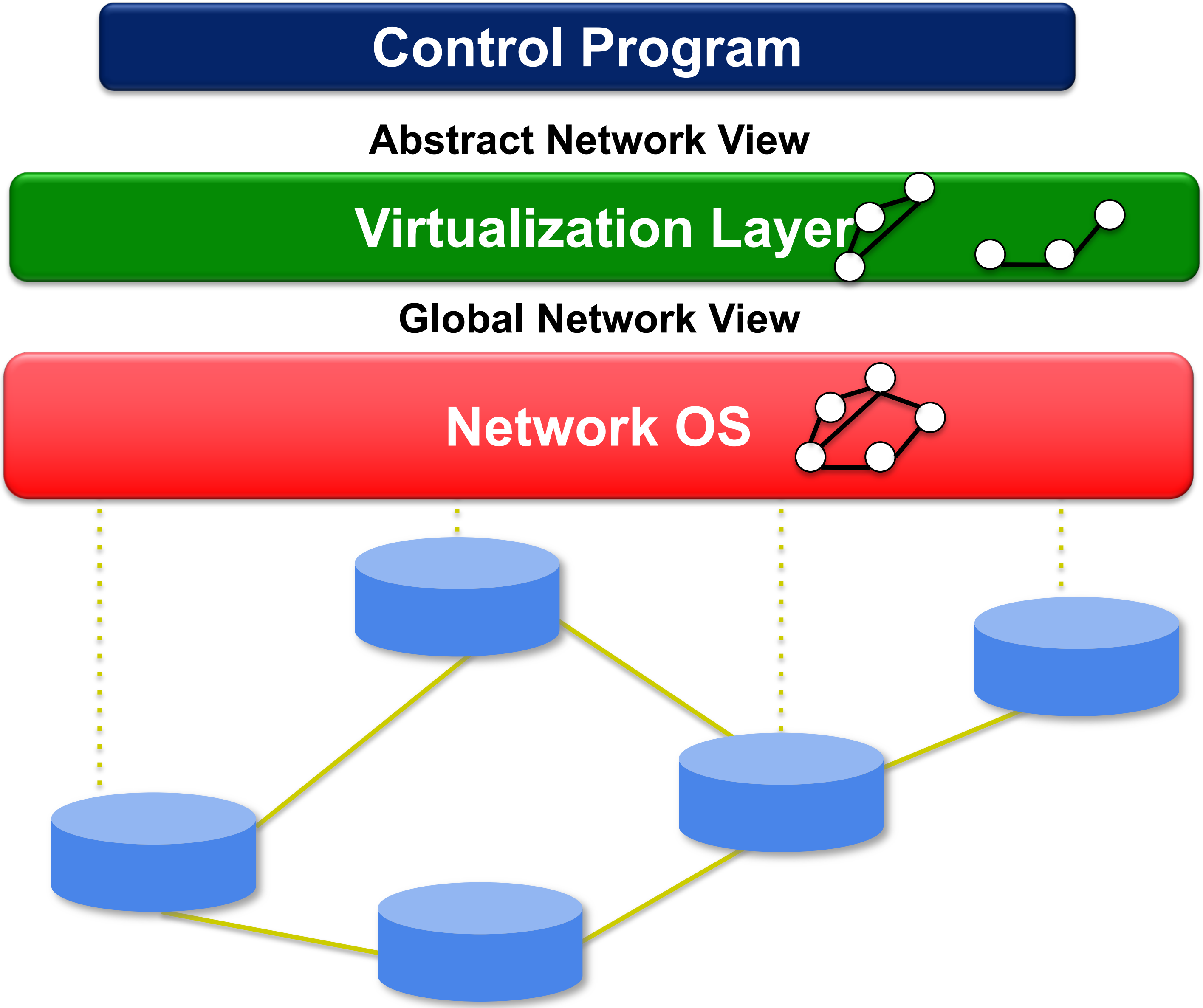
Software Defined Network



Clean Separation of Concerns

- **Control program:** express goals on abstract view
  - Driven by **Operator Requirements**
- **Virt. Layer:** abstract view **←→** global view
  - Driven by **Specification Abstraction** for particular task
- **NOS:** global view **←→** physical switches
  - Controller API: driven by **Network State Abstraction**
  - Switch interface: driven by **Forwarding Abstraction**

SDN: *Layers* for the Control Plane



Access Control Application

- Control program decides who can talk to who
- Pass this information to SDN platform
- Appropriate ACL flow entries are added to network
  - In the right places (based on the topology)
- *The control program that decides who can talk to whom doesn’t care what the network looks like!*

Takeaway

- SDN is not a revolutionary technology…
  - …just a way of organizing network functionality

- But that's all the Internet architecture is….
  - The Internet architecture isn't clever, but it is deeply wise
  - *SDN isn't clever, but hopefully it is wise….*

Future of Networking: A Systems Perspective

Goals of this Discussion

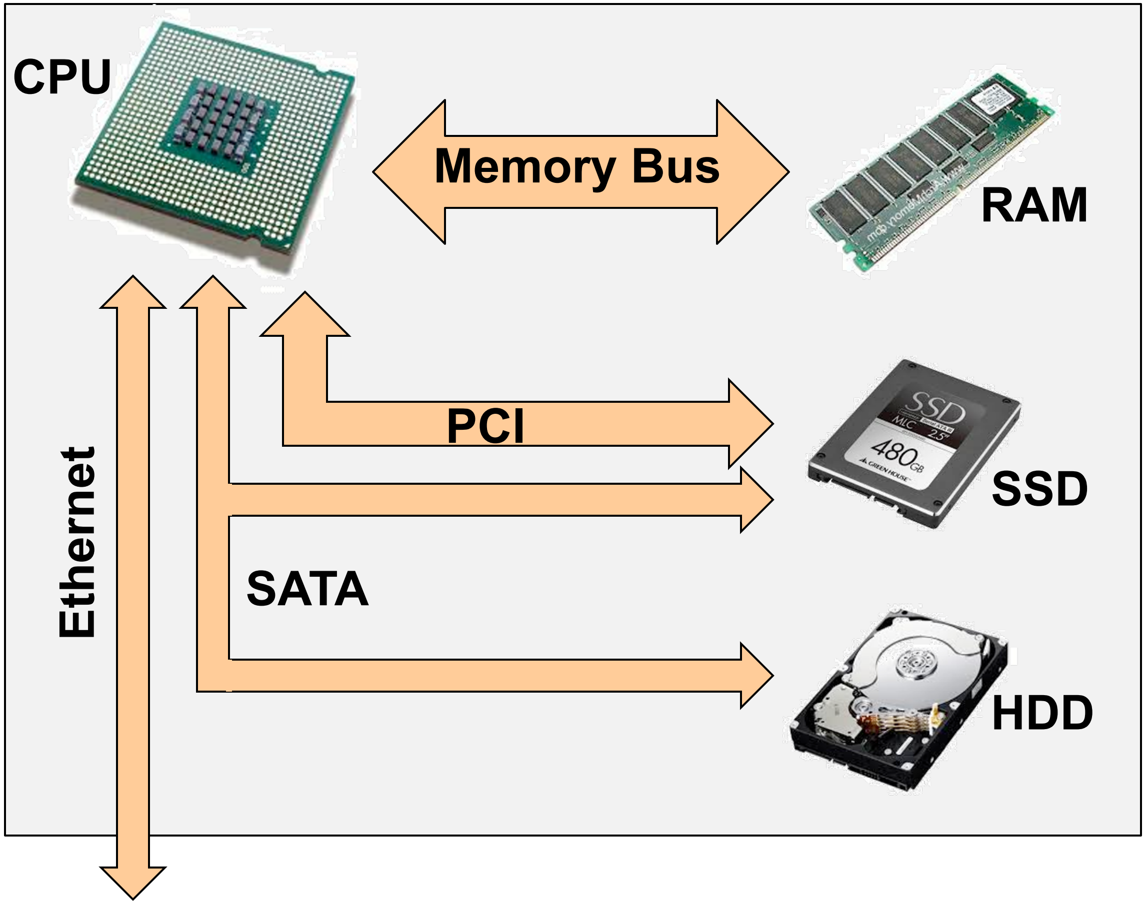
- We have already seen some of the recent directions
  - Datacenter networks, with protocol innovations across L2-L5
  - Software defined networks, rearchitecting the control plane
- The remainder of this discussion:
  - How a systems researcher sees the future of networking
- Don’t worry, this won’t be on the exam ☺

“Skate where the puck's going, not where it's been”

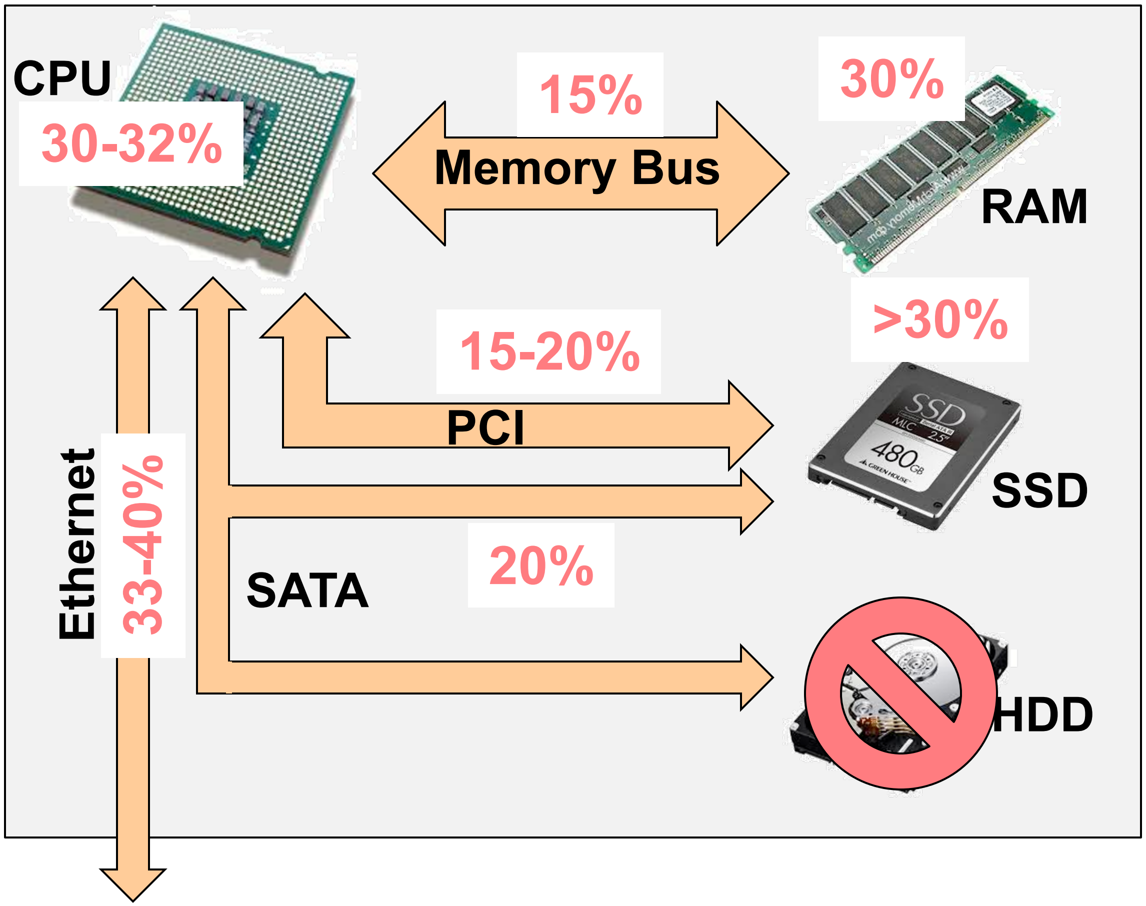
– *Walter Gretzky*

So where is the puck going?

Typical Server Node



Typical Server Node: Yearly Improvements



So what does this mean?

Growth Rates

- Network b/w growth (33-40% per year) **outpacing**:
  - CPU (30-32%)
  - memory bus (15% per year)
  - PCIe (15-20% per year)
  - SATA (20% per year)
- Current speeds:
  - Network b/w: 12.5GB/s
  - Memory bus: 80GB/s
  - PCIe: 16 GB/s
  - SATA: 600 MB/s

A Couple of Implications

- Network stack processing will become a bottleneck
  - CPU speed growth vs. network speed growth
- Accessing SSD locally will be slower than accessing remote memory
  - PCIe bus speed growth vs. network speed growth

Network Stack Processing

- Existing OS network stacks designed for 1 Gbps
- Example:
  - Typical TCP processing: ~3.2 Gbps per core
  - With low-level optimizations: ~9-12 Gbps per core
  - TCP on a 40 Gbps link: > 3 cores per server
  - TCP on a 100 Gbps link: > 8 cores per server
- This is not economically sustainable…
  - A core used for n/w stack is a core stolen from "useful" processing by applications/customers

How do we fix this?

- Research trend: *kernel bypass*
  - Packet rates: ~90k pkts/s (1 Gbps), ~9M pkts/s (100 Gbps)
  - Traversing the kernel stack is too expensive
  - Bypass the kernel completely
  - Where do the protocol implementations go?
    - User-space for lightweight applications, or…
- Research trend: *hardware offload*
  - Implement TCP (+other protocols) on specialized h/w
  - Lots of interesting challenges

Remote memory faster than SSDs

- Consider 4kB page, 100Gbps link, 1 switch hop
  - Zero queueing delay
- Baseline: 4.78 us
  - OS: 1.9us, Data copy: 2us, Switching: 0.48us, Propagation Delay: 0.08us, Transmission Delay: 0.32us
- How can we avoid OS overhead?
  - Research Trend: RDMA

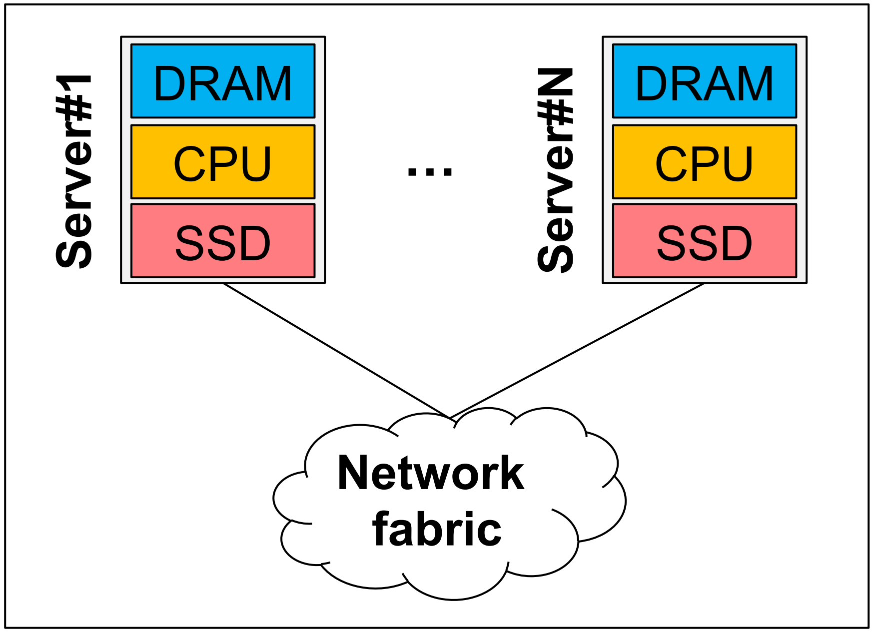
Remote memory faster than SSDs

- Consider 4kB page, 100Gbps link, 1 switch hop
  - Zero queueing delay
- Baseline: 4.78 us
  - OS: 1.9us, Data copy: 2us, Switching: 0.48us, Propagation Delay: 0.08us, Transmission Delay: 0.32us
- With RDMA: 2.88us
  - Can reduce data copy overheads with NIC support
  - Brings latency down to 1.88us!
  - ~10x of local memory, 1/10x of local SSD

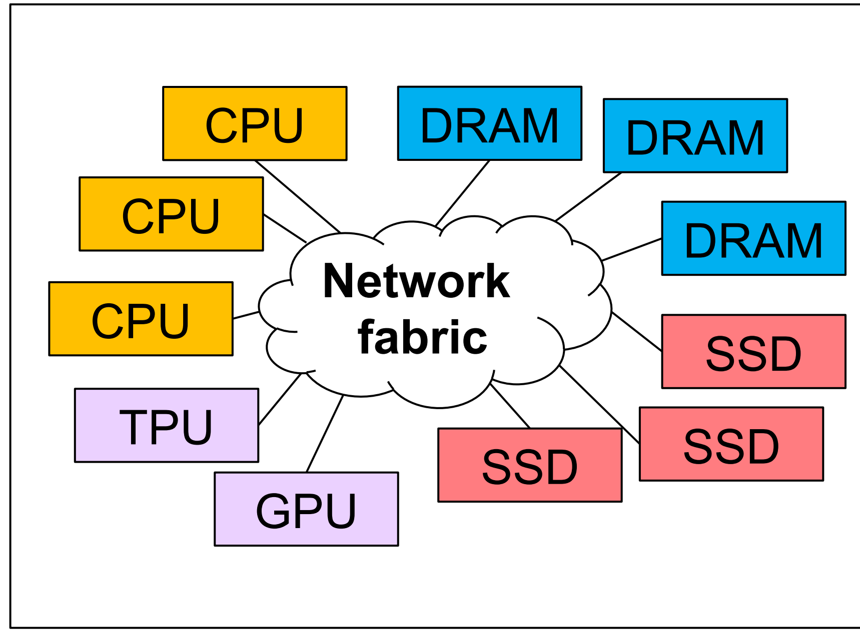
What are the challenges?

- This assumes zero queueing delay…
  - Transport protocol design to minimize queueing delay
  - Recall pFabric; Other protocols: DCTCP, pHost, …
- RDMA requires <u>lossless</u> ethernet
  - Via ethernet flow control or priority flow control (PFC)
  - Lossless networks: very active area of research

Taking it one step further: Resource Disaggregation



Traditional Datacenter



Datacenter with Resource Disaggregation

- Disaggregated Computer
  - Instead of a network of computers
- Benefits?
  - Resource utilization, upgradeability, fault-tolerance

My Current Research Focus

- How can we design OS for disaggregated computer?
  - Where does the OS logic run?
- Idea: Put key OS functionalities in network fabric
  - Why?
  - Network has all the visibility…
- SDN design to modularize & layer OS functionality
  - e.g., Address translation = *data plane*; memory allocation = *control plane*, etc.
- Remind you of something?
  - Forwarding = *data plane*; route computation = *control plane*

Questions?