Project 3b

Stateless, Passive Firewall (3a)

Stateful, Active Firewall (3b)

What stays…

- Same framework
- Same VM
- Same test harness
- Same tools
- Same basic firewall behavior

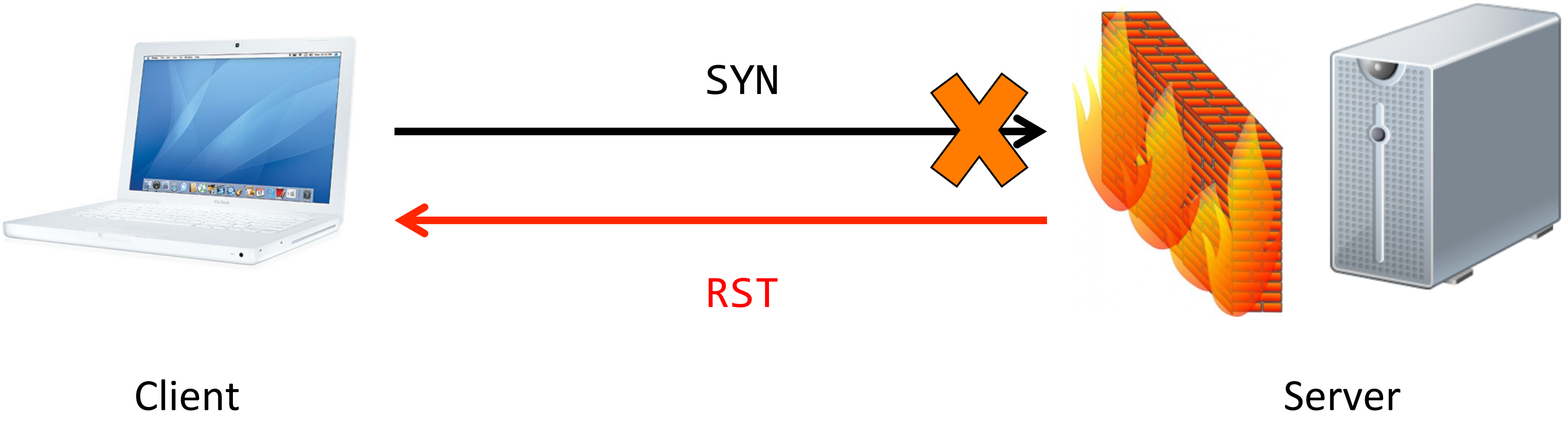
What’s new?

- Three new rules:

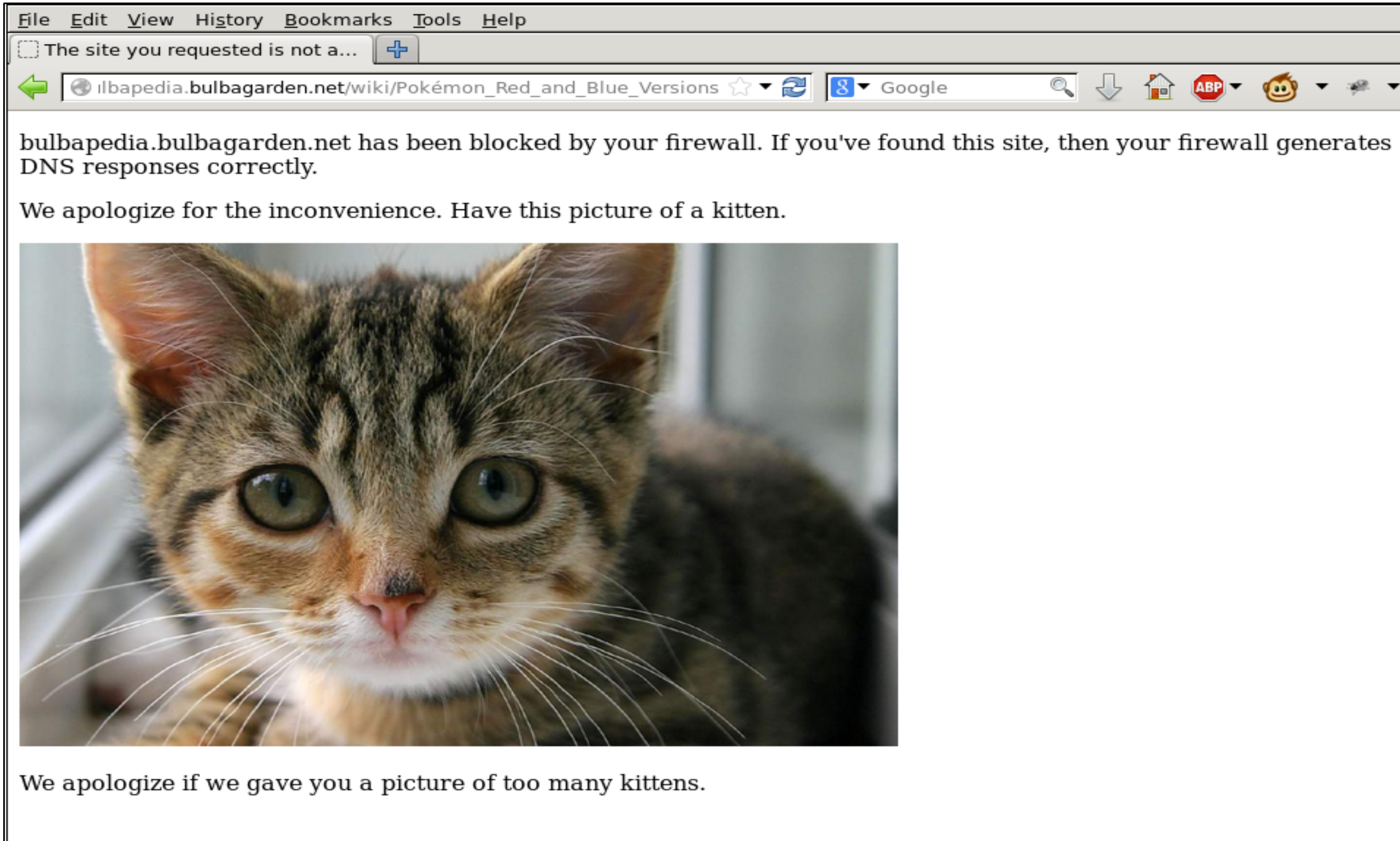
```
1.deny tcp <IP address> <port>
2.deny dns <domain name>
3.log http <host name>
```

Active!

- `deny tcp <IP address> <port>`



Even more active!



Stateful!

- `log http <host name>`

- Log HTTP *transactions*
  - Request/Response pair

```
log http *.berkeley.edu

www-inst.eecs.berkeley.edu GET /~cs168/fa14/ HTTP/1.1 301 254
www-inst.eecs.berkeley.edu GET /~cs168/fa14/ HTTP/1.1 200 273
www-inst.eecs.berkeley.edu GET /favicon.ico HTTP/1.1 200 0
...
```

Stateful!

- Handle:
  - segmented TCP packets
  - headers spanning multiple packets
  - packet drops/re-orderings
  - and more… (read project spec!)

- My advice?
  - Start early

What we won't check…

- Firewall rules from 3a
- Country-based matching

Takeaway?

- Don't worry too much if your solution for 3a wasn't complete
  - But make sure the solution to this one is!

NO CHEATING

WE RUN COPY CHECKER

Logistics

- GSIs:
  - Anurag, Shoumik and Sangjin
- Additional OH:
  - Will be announced on Piazza
- Slides, specs (code framework same as 3a):
  - Online midnight today
- Due:
  - At noon on Dec 3rd

Datacenters (part 2)

CS 168, Fall 2014
Sylvia Ratnasamy
http://inst.eecs.berkeley.edu/~cs168/

What you need to know

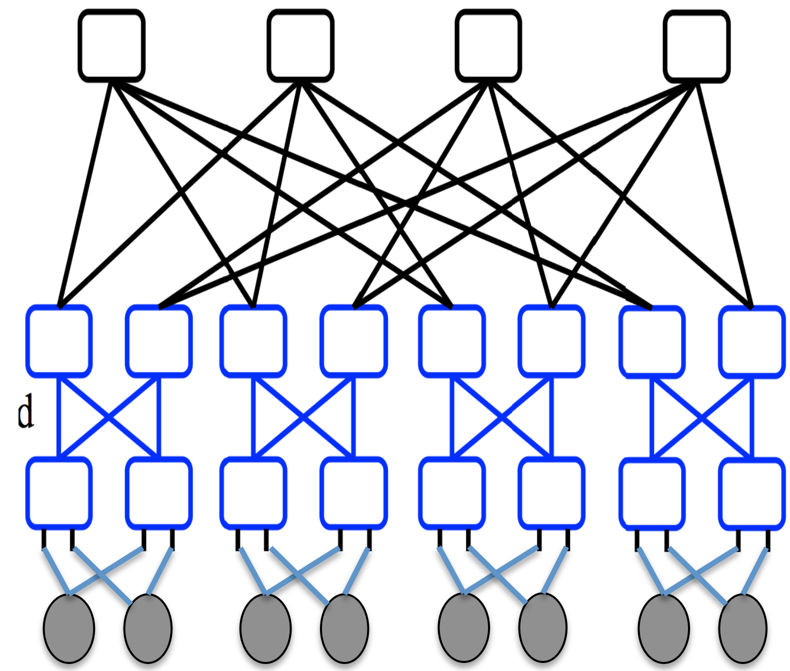
- Characteristics of a datacenter environment
  - goals, constraints, workloads, *etc.*
- How and why DC networks are different (*vs.* WAN)
  - e.g., latency, geo, autonomy, …
- How traditional solutions fare in this environment
  - E.g., IP, Ethernet, TCP, DHCP, ARP
- Specific design approaches

Recap: Last Lecture

- Key requirements
  - High “bisection bandwidth”
  - Low latency, even in the worst-case
  - Large scale
  - Low cost

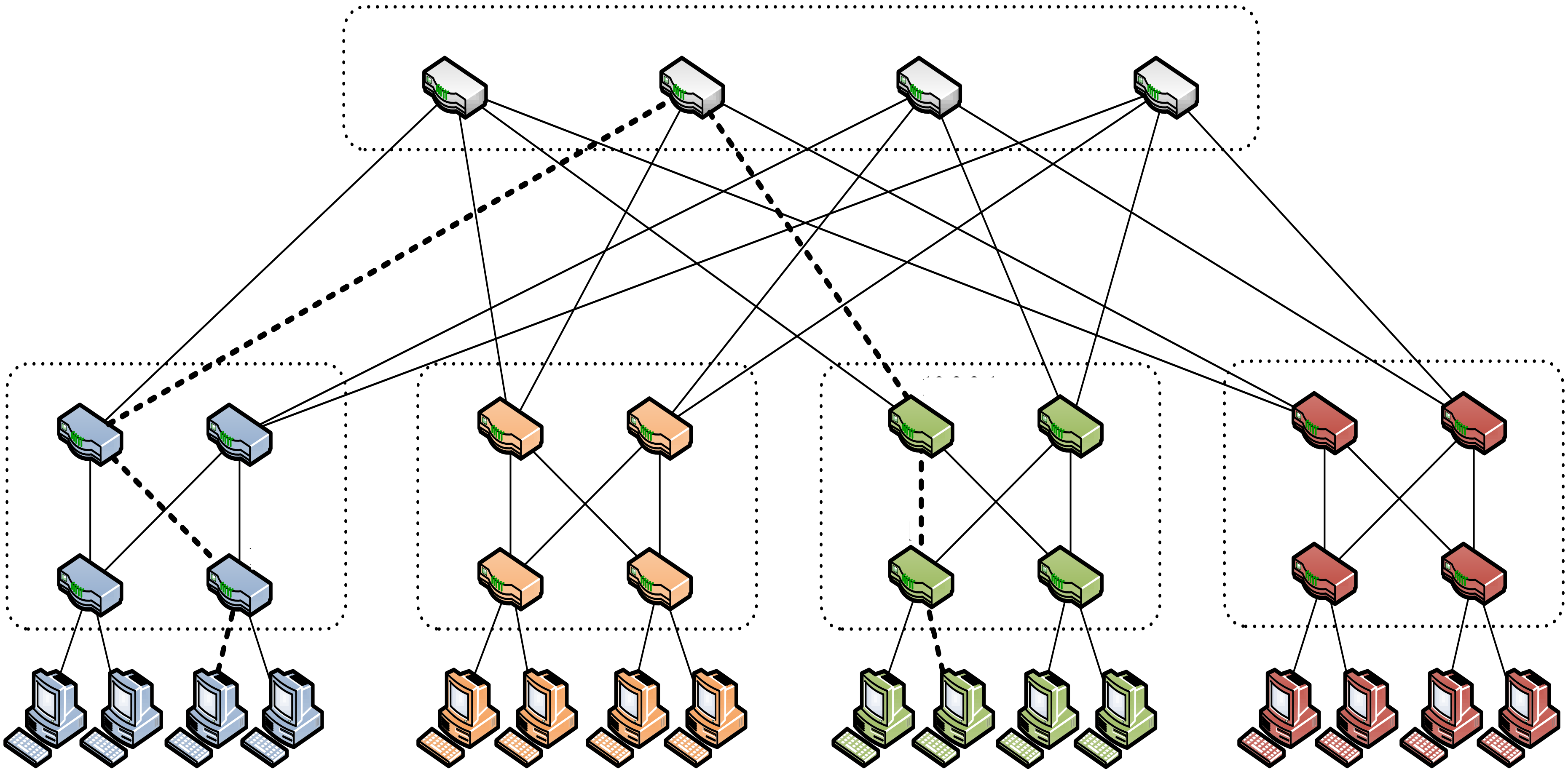
Recap: High bisection BW topology

- E.g., `Clos' topology
  - Multi-stage network
  - All switches have k ports
  - k/2 ports up, k/2 down
- All links have same speed



“scale out” approach

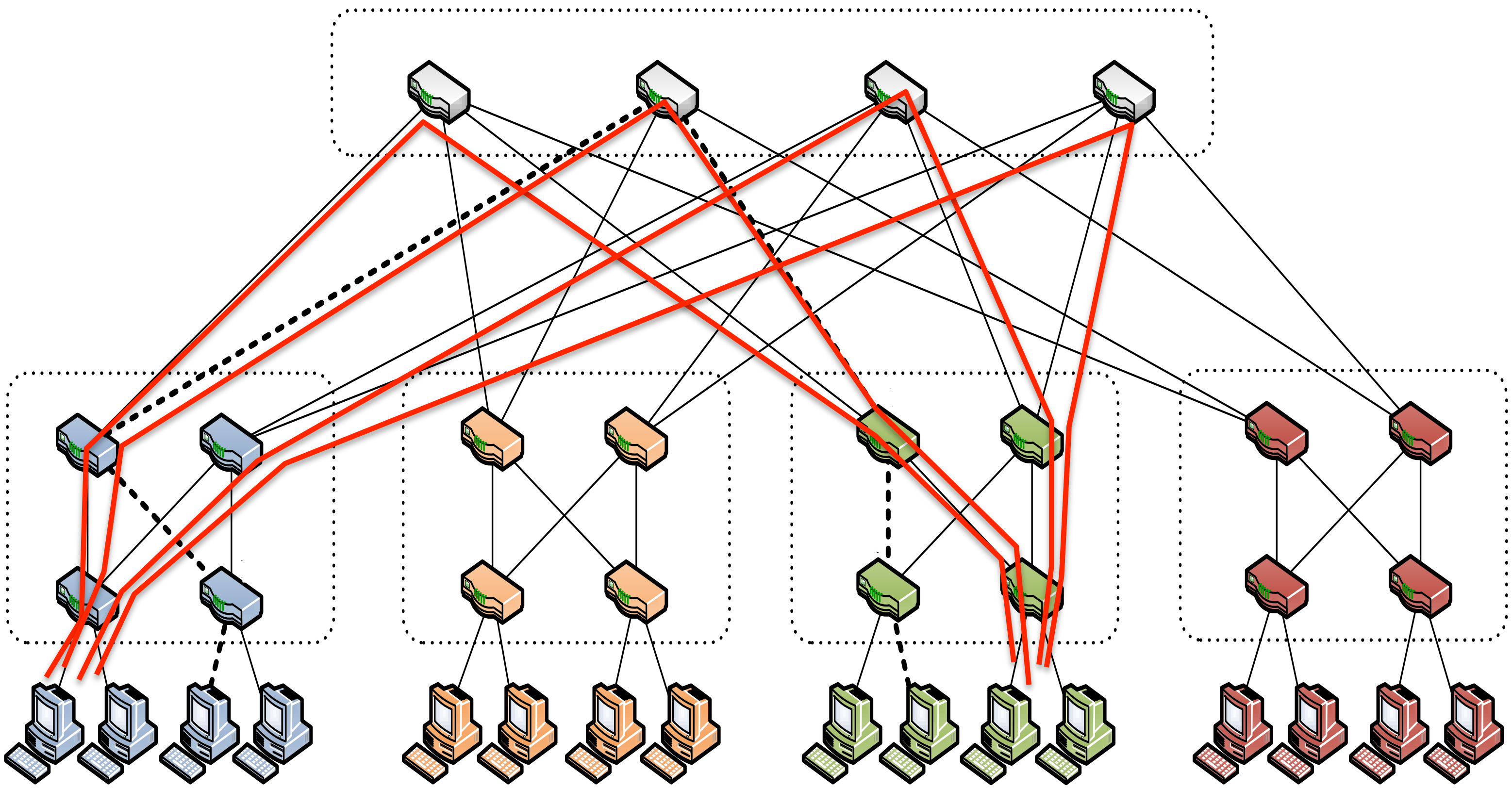
E.g., “Fat Tree” Topology [Sigcomm’08]



Questions for today

- L2/L3 design:
  - addressing / routing / forwarding in the Fat-Tree
- L4 design:
  - Transport protocol design (w/ Fat-Tree)

Using multiple paths well



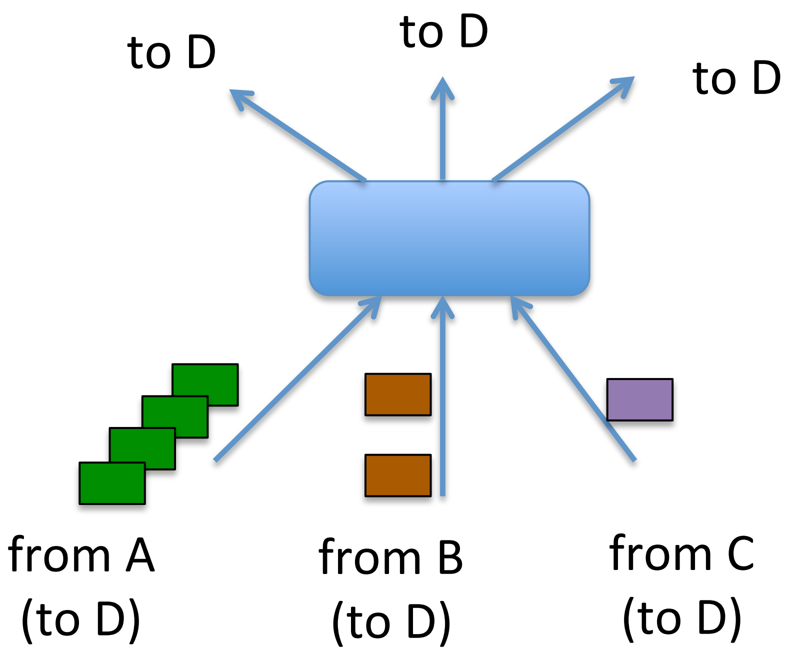
Questions for today

- L2/L3 design:
  - addressing / routing / forwarding in the Fat-Tree
- Goals:
  - Routing protocol must expose all available paths
  - Forwarding must spread traffic evenly over all paths

Extend DV / LS ?

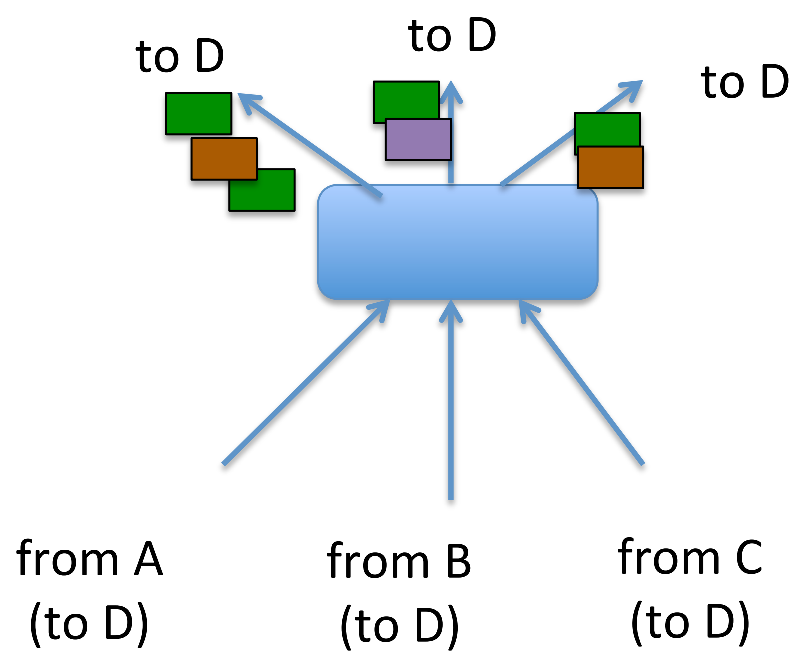
- Routing
  - Distance-Vector: Remember *all* next-hops that advertise equal cost to a destination
  - Link-State: Extend Dijkstra's to compute *all* equal cost shortest paths to each destination
- Forwarding: how to spread traffic across next hops?

Forwarding



- Per-packet load balancing

Forwarding



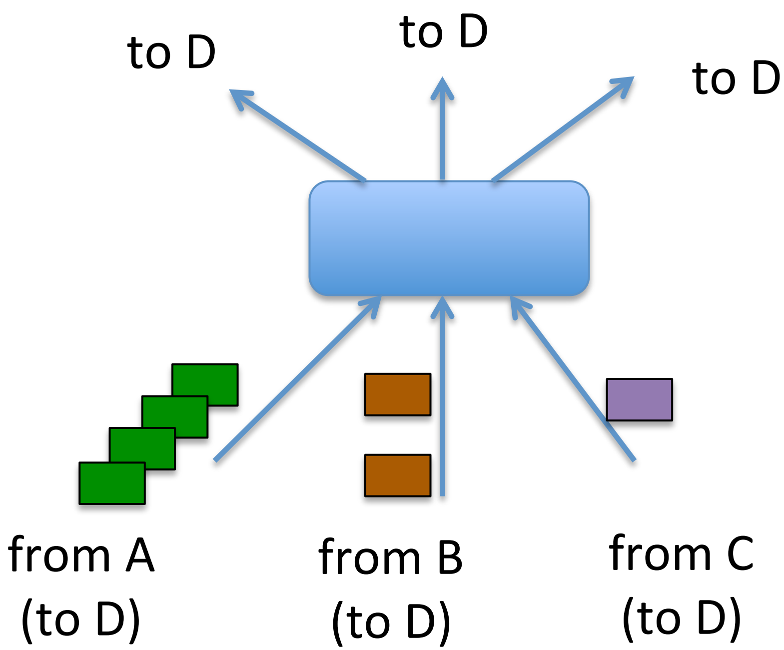
- Per-packet load balancing
  + traffic well spread (even w/ elephant flows)
  – Interacts badly w/ TCP

TCP w/ per-packet load balancing

- Consider:
  - sender sends seq#: 1,2,3,4,5
  - receiver receives: 5,4,3,2,1
  - sender will enter fast retx, reduce CWND, retx #1, …
  - repeatedly!
- Also: one RTT and timeout estimator for multiple paths
- Also: CWND halved when a packet is dropped on *any* path

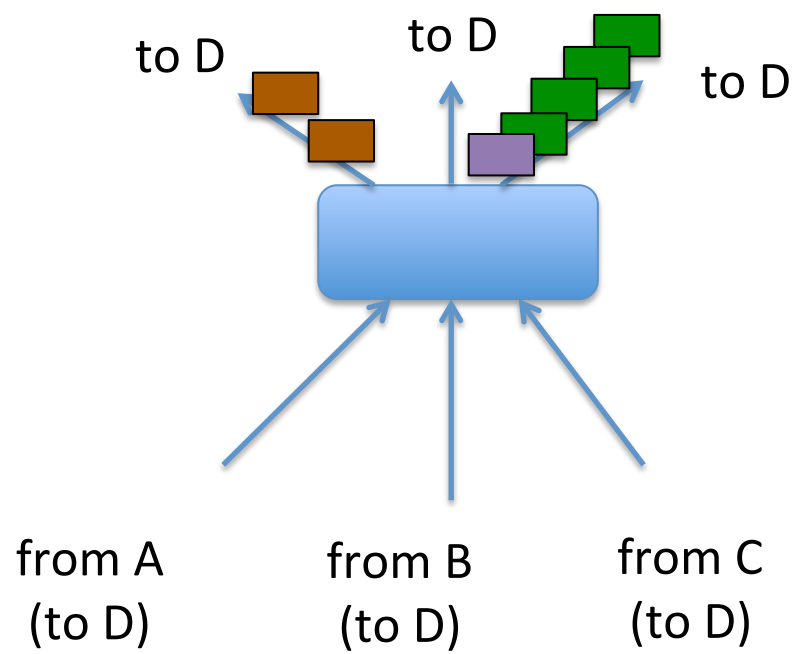
- Multipath TCP (MP-TCP) is an ongoing effort to extend TCP to coexist with multipath routing
  - Value beyond datacenters (e.g., spread traffic across wifi and 4G access)

Forwarding



- Per-flow load balancing *(ECMP, "Equal Cost Multi Path")*
  - E.g., based on HASH(source-addr, dst-addr, source-port, dst-port)

Forwarding



- Per-flow load balancing *(ECMP, "Equal Cost Multi Path")*
  - E.g., based on HASH(source-addr, dst-addr, source-port, dst-port)
  - Pro: a flow follows a single path (→ TCP is happy)
  - Con: non optimal load-balancing; elephants are a problem

Extend DV / LS ?

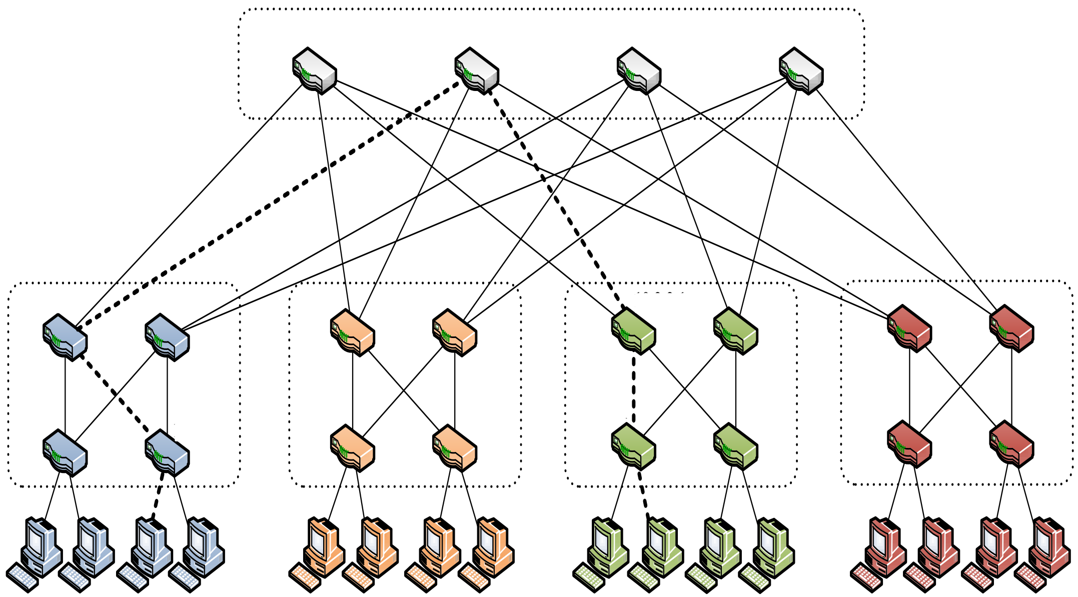
- How:
  - Simple extensions to DV/LS
  - ECMP for load balancing

- Benefits
  + Simple; reuses existing solutions

- Problem: scales poorly
  - With N destinations, O(N) routing entries and messages
  - N now in the millions!

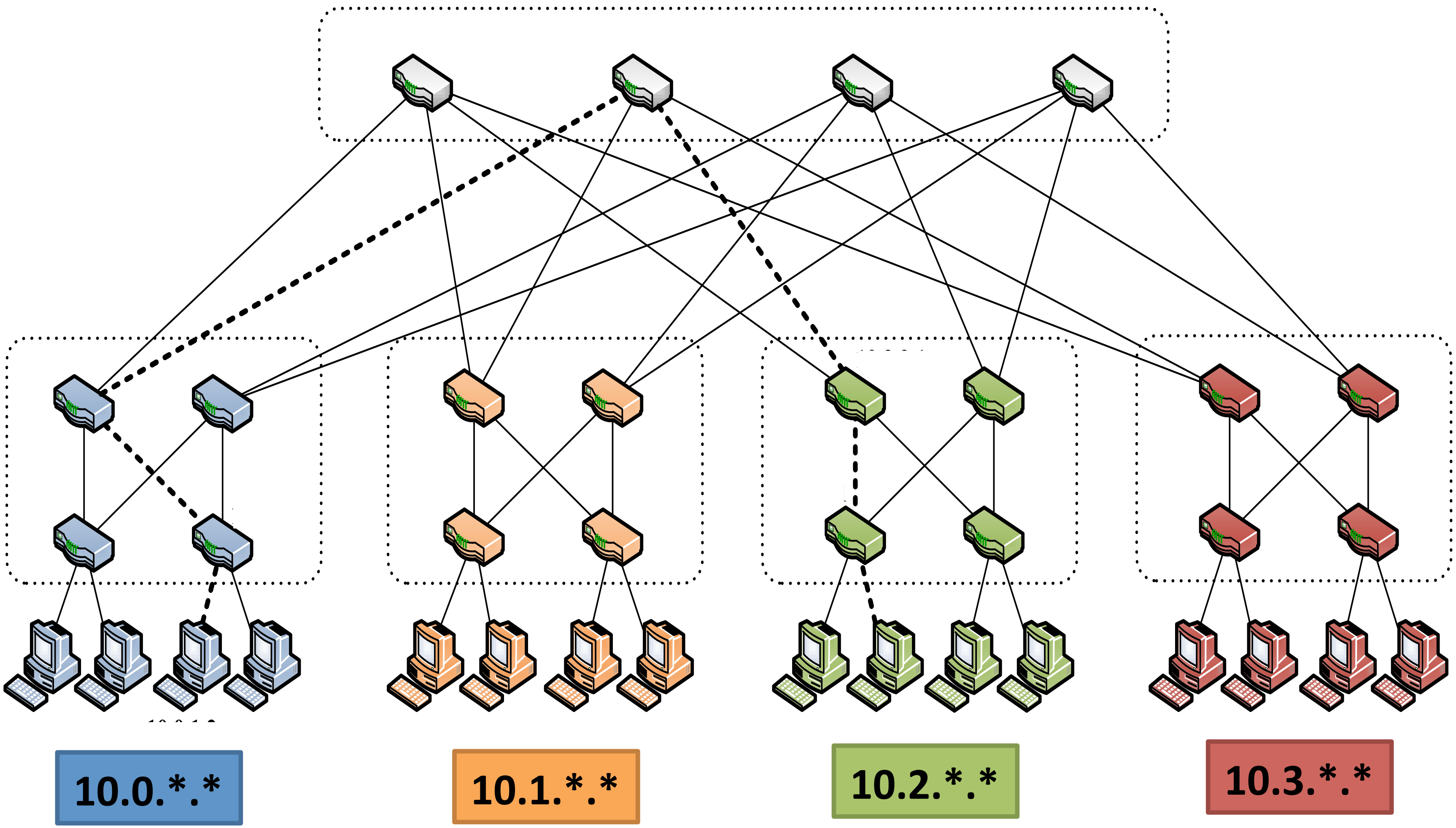
You: design a scalable routing solution

- Design for this specific topology

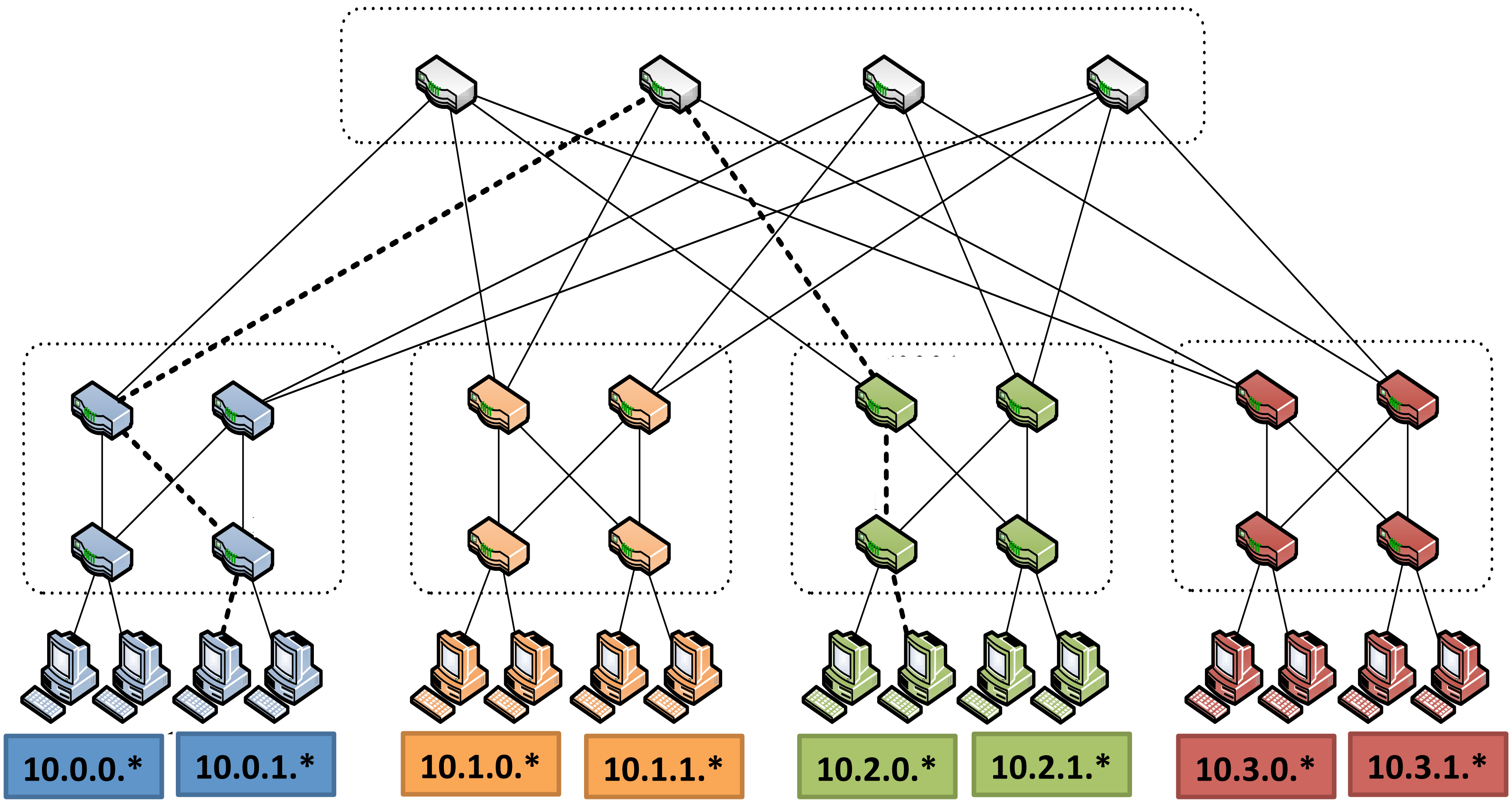


- Take 5 minutes, then tell me:
  - #routing entries per switch your solution needs
  - many solutions possible; remember: you're now free from backward compatibility (can redesign IP, routing algorithms, switch designs, etc.)

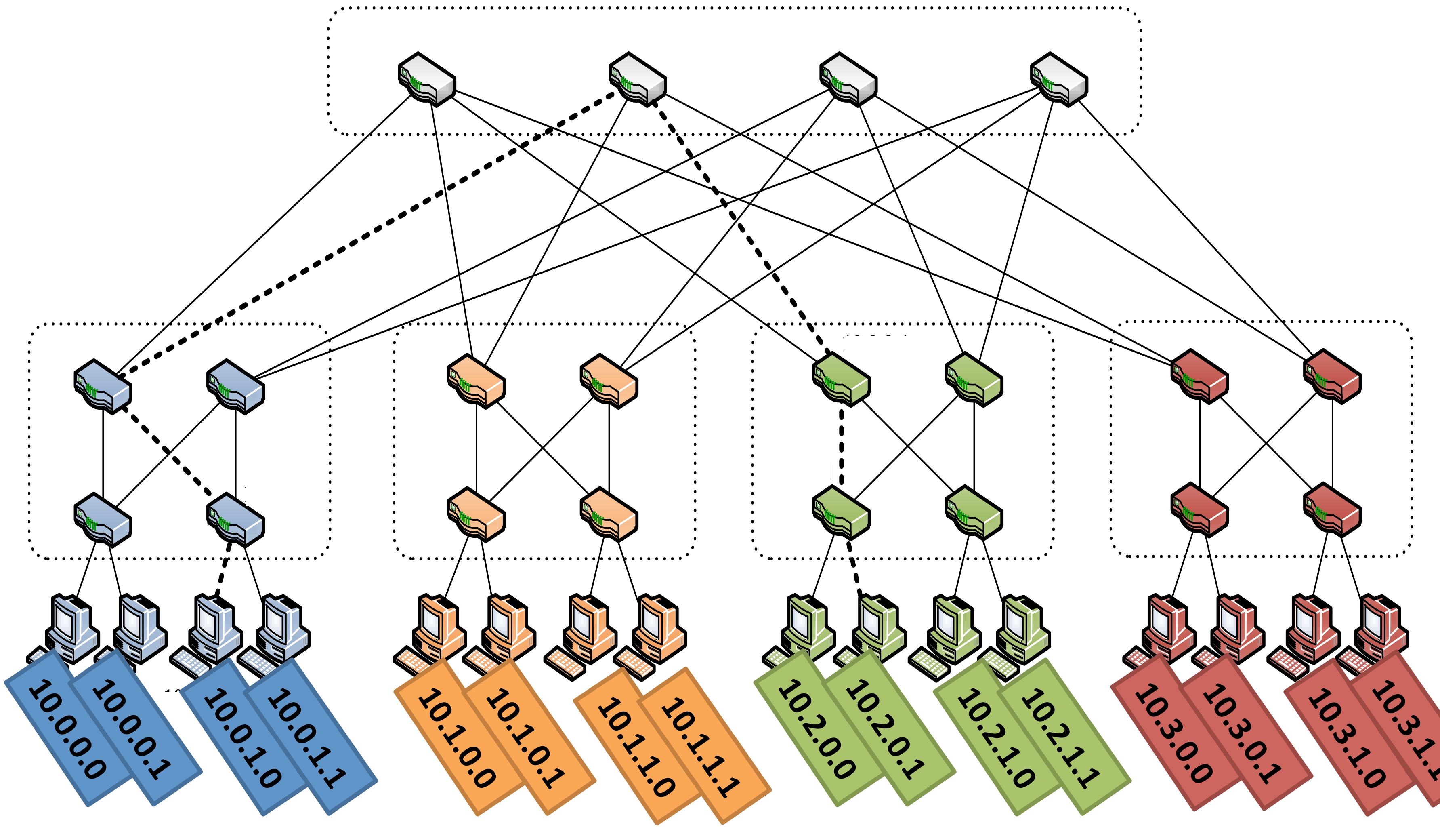
Solution 1: Topology-aware addressing



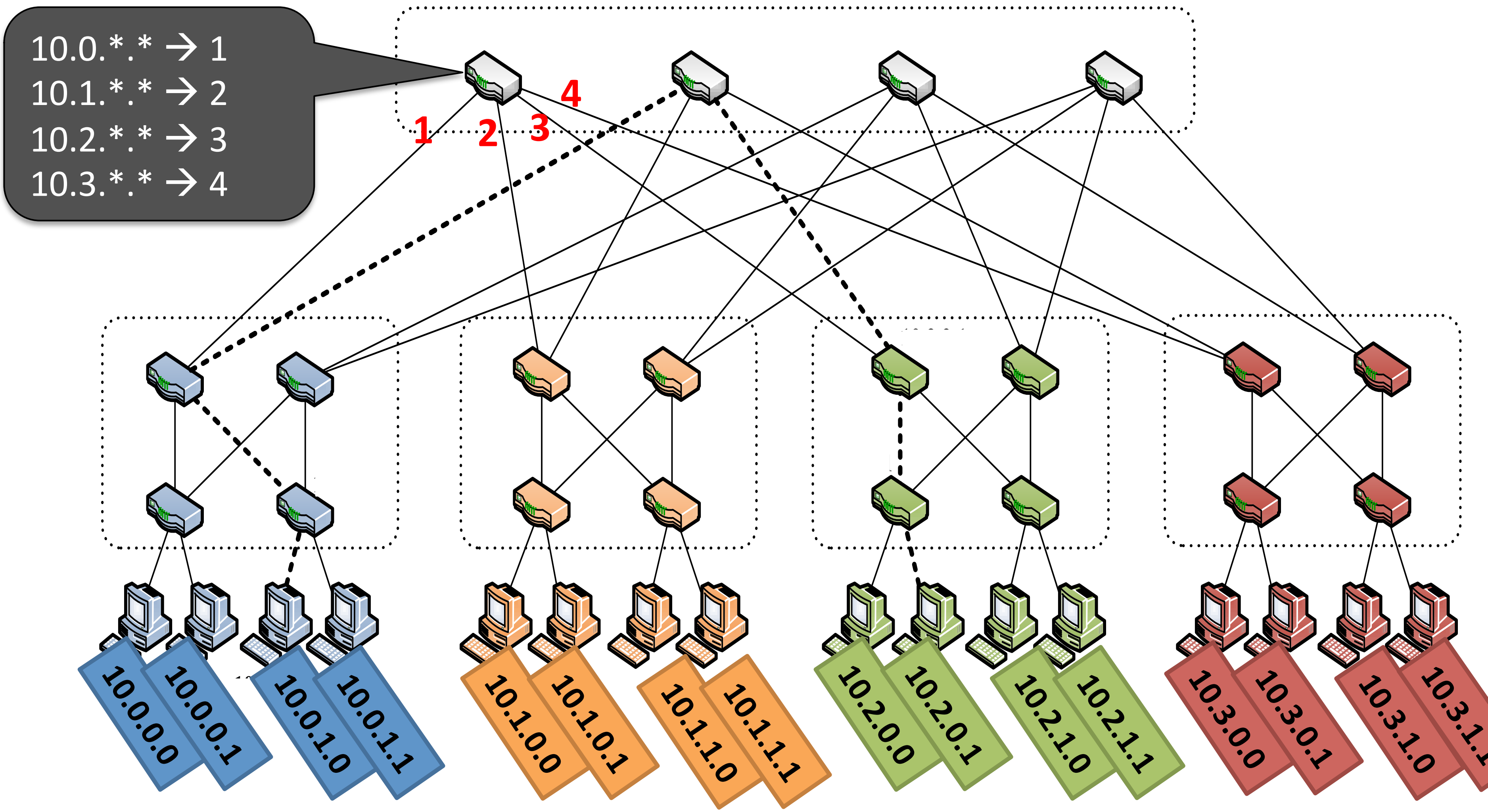
Solution 1: Topology-aware addressing



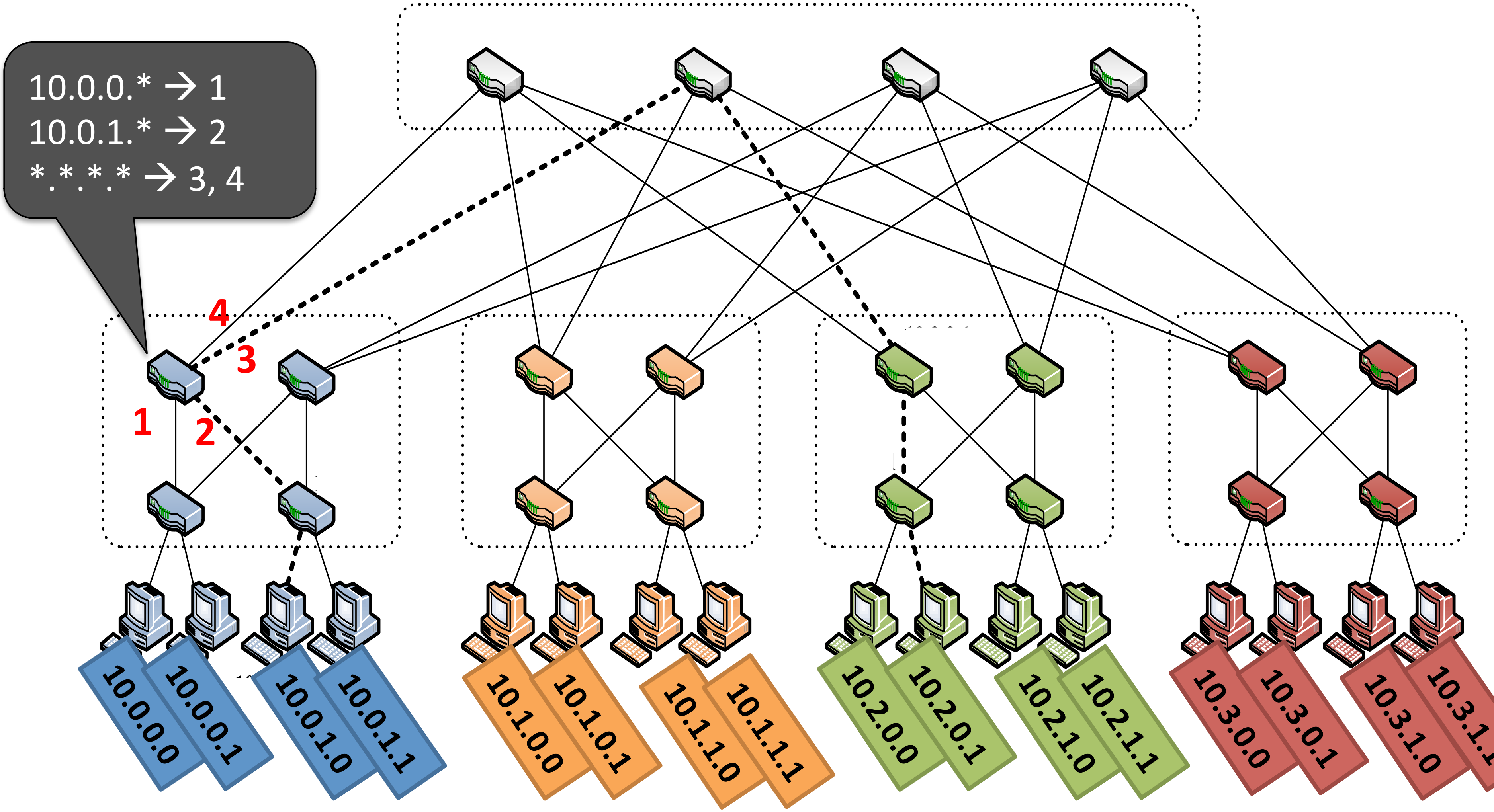
Solution 1: Topology-aware addressing



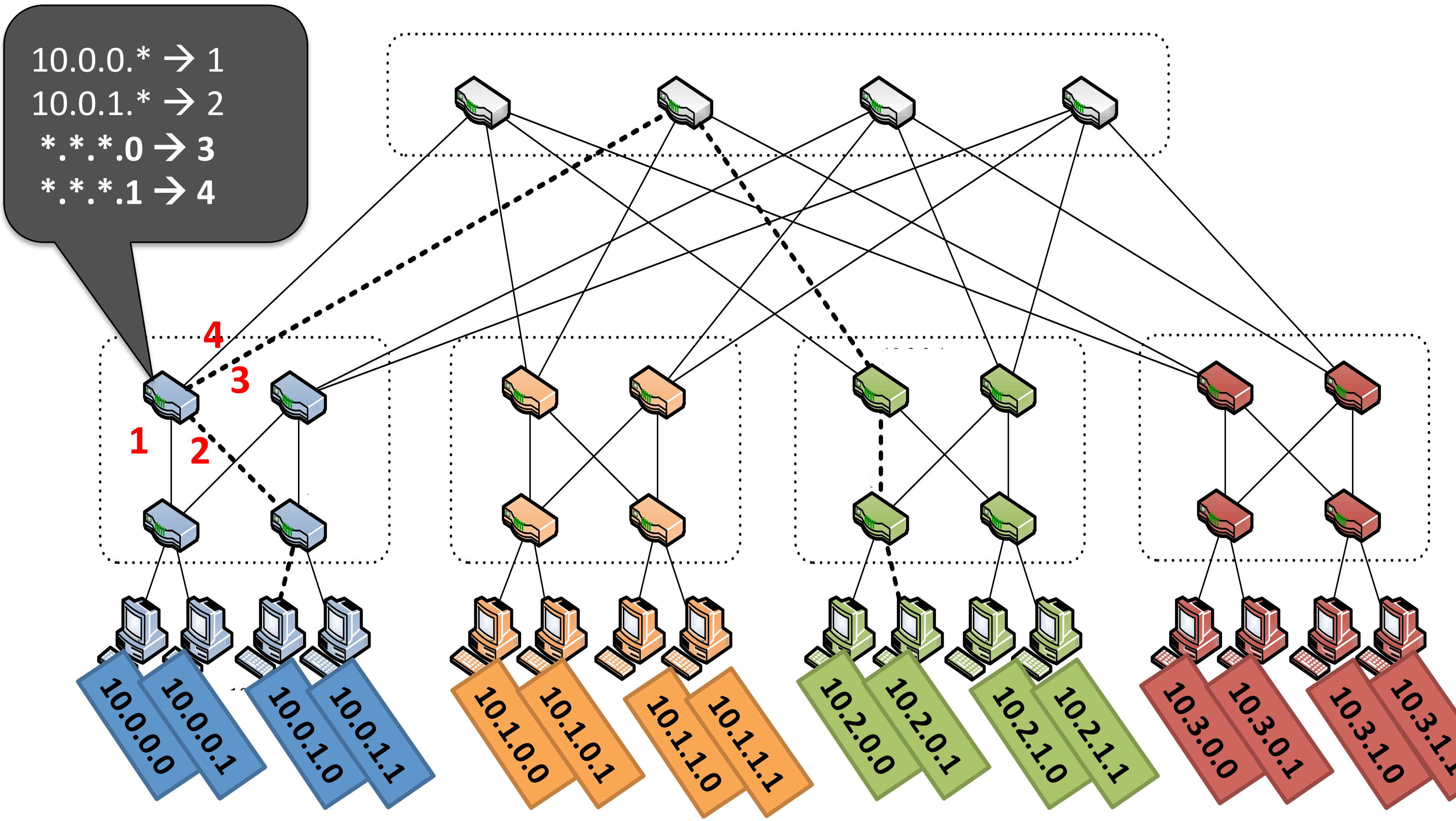
Solution 1: Topology-aware addressing



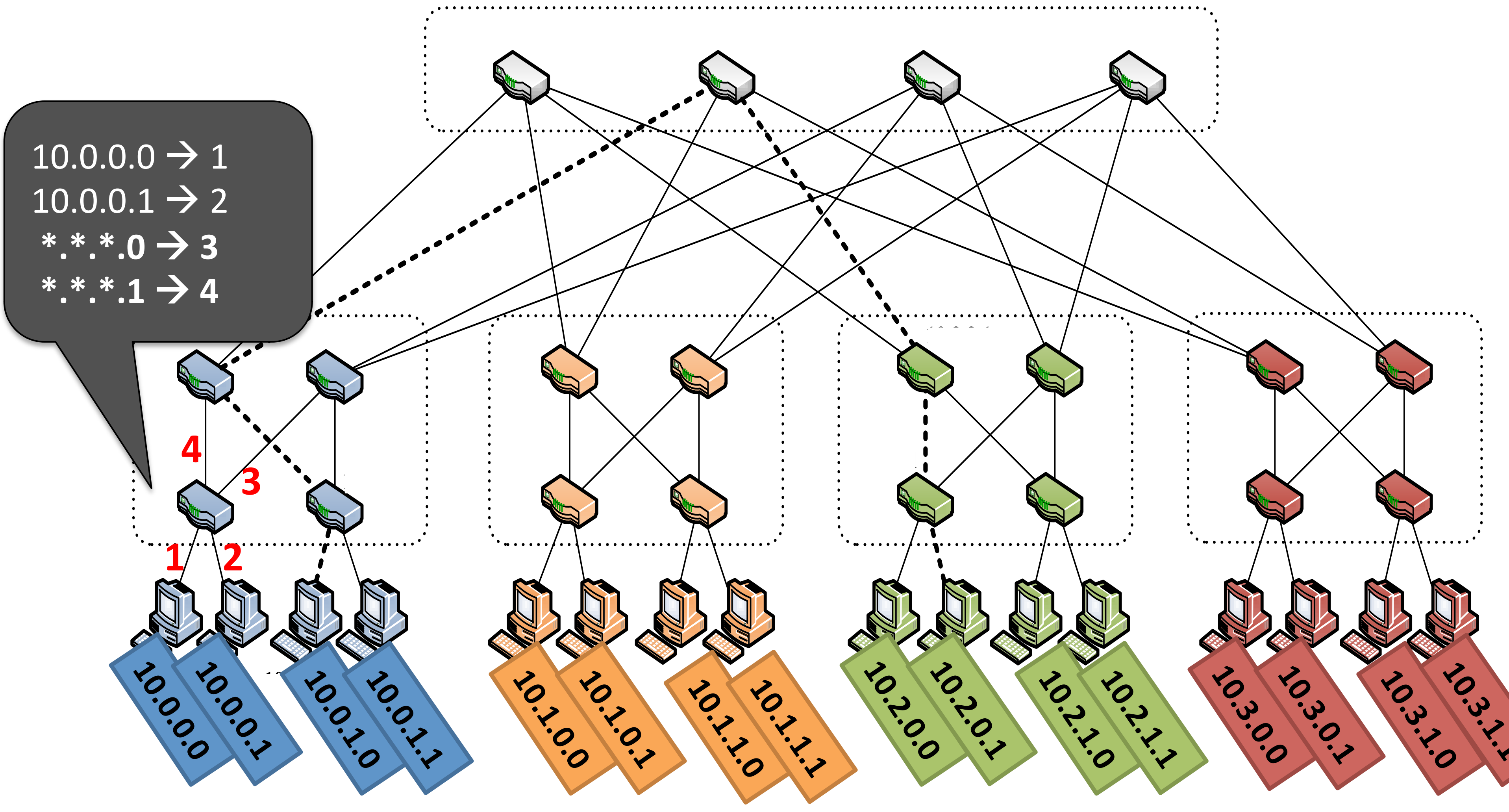
Solution 1: Topology-aware addressing



Solution 1: Topology-aware addressing



Solution 1: Topology-aware addressing



Solution 1: Topology-aware addressing

- Idea: addresses embed location in regular topology
- Maximum #entries/switch: *k (= 4 in example)*
  - Constant, independent of #destinations!
- No route computation / messages / protocols
  - Topology is "hard coded"
  - (but still need localized link failure detection)
- Problems?
  - VM migration: ideally, VM keeps its IP address when it moves
  - Vulnerable to misconfiguration (in topology or addresses)

Solution 2: Centralize + Source Routes

- Centralized "controller" server knows topology and computes routes
- Controller hands server all paths to each destination
  - O(#destinations) state per server
  - But server memory cheap (e.g., 1M routes x 100B/route=100MB)
- Server inserts entire path vector into packet header ("source routing")
  - E.g., header=[dst=D | index=0 | path={S5,S1,S2,S9}]
- Switch forwards based on packet header
  - index++; next-hop = path[index]

Solution 2: Centralize + Source Routes

- #entries per switch?
  - None!
- #routing messages?
  - akin to a broadcast from controller to all servers
- Pro:
  - switches very simple and scalable
  - flexibility: end-points (hence apps) control route selection
- Cons:
  - scalability / robustness of controller (SDN addresses this)
  - Clean-slate design of everything

Questions for today

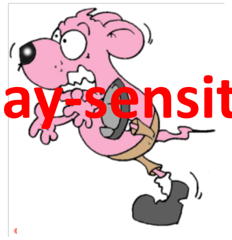
- L2/L3 design:
  - addressing / routing / forwarding in the Fat-Tree

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

Many slides courtesy of Mohammad Alizadeh, Stanford University

Workloads

- Partition/Aggregate
  (Query) → **Delay-sensitive**



- Short messages [50KB-1MB]
  (Coordination, Control state) → **Delay-sensitive**

- Large flows [1MB-50MB]
  (Data update) → **Throughput-sensitive**

Slide from Mohammad Alizadeh, Stanford University

Tension Between Requirements

High Throughput ***vs.*** **Low Latency**

High Throughput

Deep queues at switches:

- Queuing delays increase latency

Low Latency

Shallow queues at switches:

- Bad for bursts & throughput

Objective:
Low Queue Occupancy & High Throughput

Slide from Mohammad Alizadeh, Stanford University

Data Center TCP (DC-TCP)

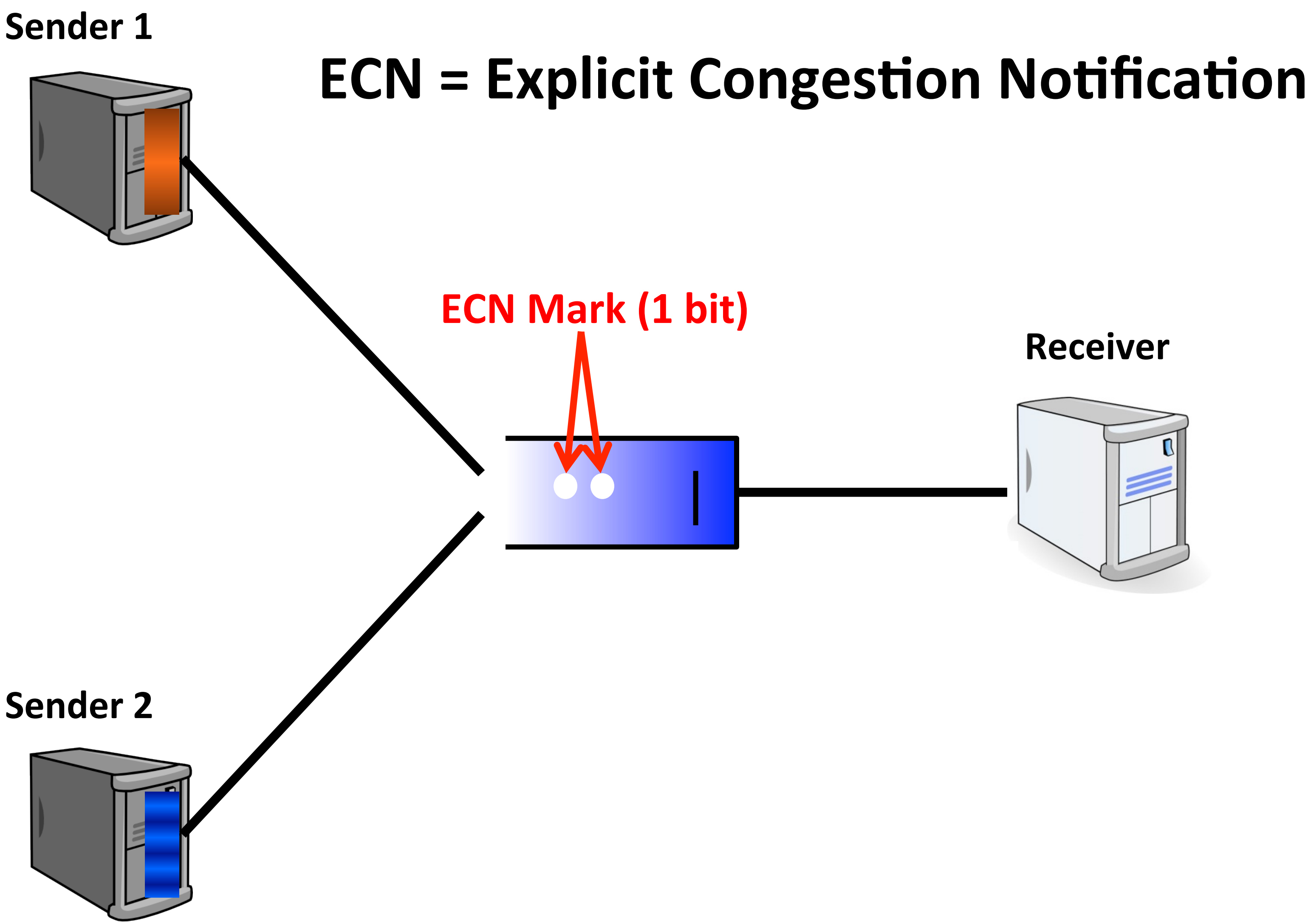
- Proposal from Microsoft Research, 2010
  - Incremental fixes to TCP for DC environments
  - Deployed in Microsoft datacenters (~rumor)

- Leverages Explicit Congestion Notification (ECN)

Lec#14: Explicit Congestion Notification (ECN)

- Defined in RFC 3168 using ToS/DSCP bits in the IP header
- Single bit in packet header; set by congested routers
  - If data packet has bit set, then ACK has ECN bit set
- Routers typically set ECN bit based on average queue length
- Congestion semantics exactly like that of drop
  - I.e., sender reacts as though it saw a drop

Review: The TCP/ECN Control Loop



Slide from Mohammad Alizadeh, Stanford University

DC-TCP: key ideas

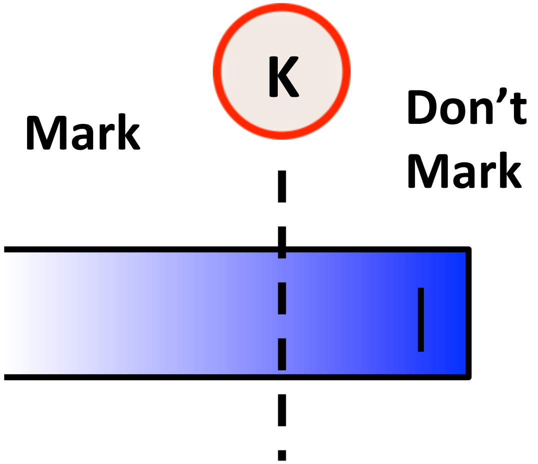
1. React early and quickly: use ECN
2. React in proportion to the **extent** of congestion, not its **presence**

ECN Marks	TCP	DCTCP
1 0 1 1 1 1 0 1 1 1	**Cut window by 50%**	**Cut window by 40%**
0 0 0 0 0 0 0 0 0 1	**Cut window by 50%**	**Cut window by 5%**

Slide from Mohammad Alizadeh, Stanford University

At the switch

– If **Queue Length > K**
(note: queue length is instantaneous, not average)
  - Set ECN bit in packet



At the sender

- Maintain running average of the *fraction* of packets marked **(*α*)**.

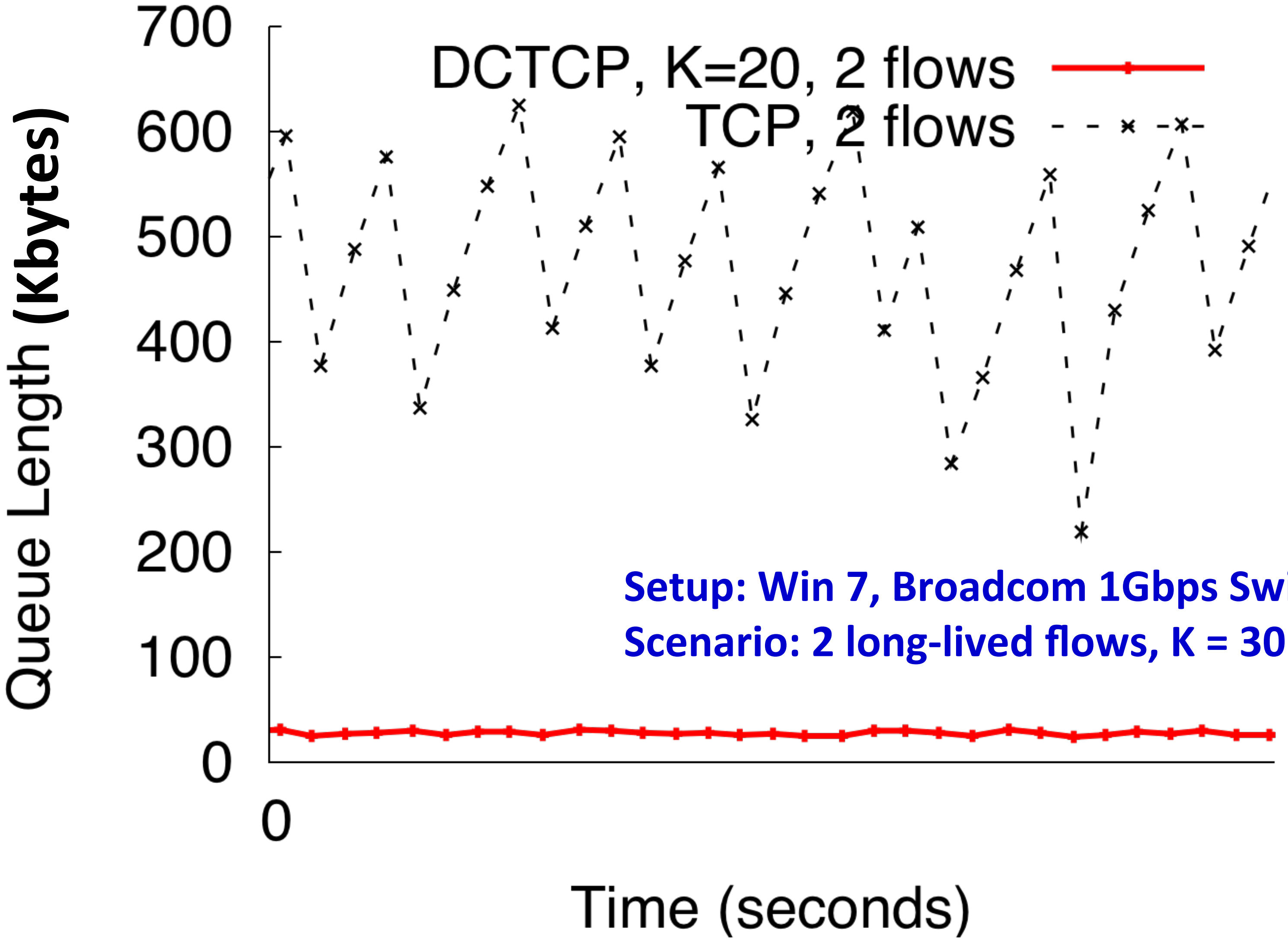
$$\text{each RTT}: \; F = \frac{\#\ \text{of marked ACKs}}{\text{Total}\ \#\ \text{of ACKs}} \;\Rightarrow\; \alpha \leftarrow (1-g)\alpha + gF$$

- Window adapts based on ***α*:** $W \leftarrow (1-\frac{\alpha}{2})W$

α equal to 1 (high congestion): $W \leftarrow \frac{W}{2}$ (same as TCP!)

Slide from Mohammad Alizadeh, Stanford University

DCTCP in Action



Setup: Win 7, Broadcom 1Gbps Switch
Scenario: 2 long-lived flows, K = 30KB

Slide from Mohammad Alizadeh, Stanford University

DC-TCP: why it works

1. React early and quickly: use ECN
   → Avoid large buildup in queues → lower latency
2. React in proportion to the **extent** of congestion, not its **presence**
   → Maintain high throughput by not over-reacting to congestion
   → Reduces variance in sending rates, lowering queue buildups

Slide from Mohammad Alizadeh, Stanford University

Data Center TCP (DC-TCP)

- Proposal from Microsoft Research, 2010
  - Incremental fixes to TCP for DC environments
  - Deployed in Microsoft datacenters (~rumor)

- Leverages Explicit Congestion Notification (ECN)

- An improvement; but still far from "ideal"

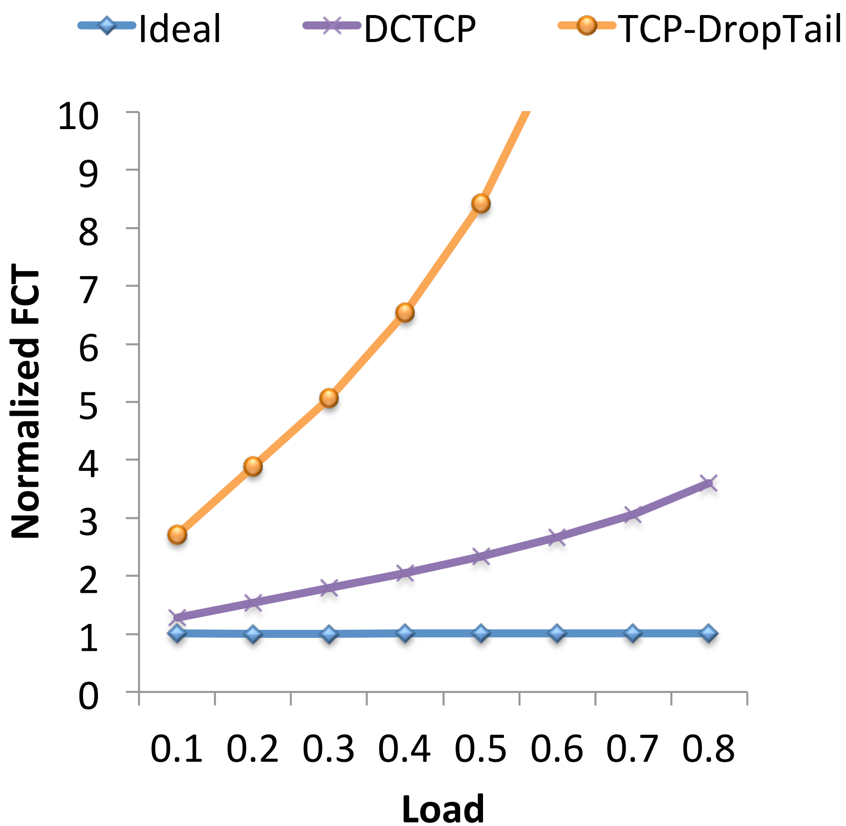
What’s “ideal” ?

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

Flow Completion Time (FCT)

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

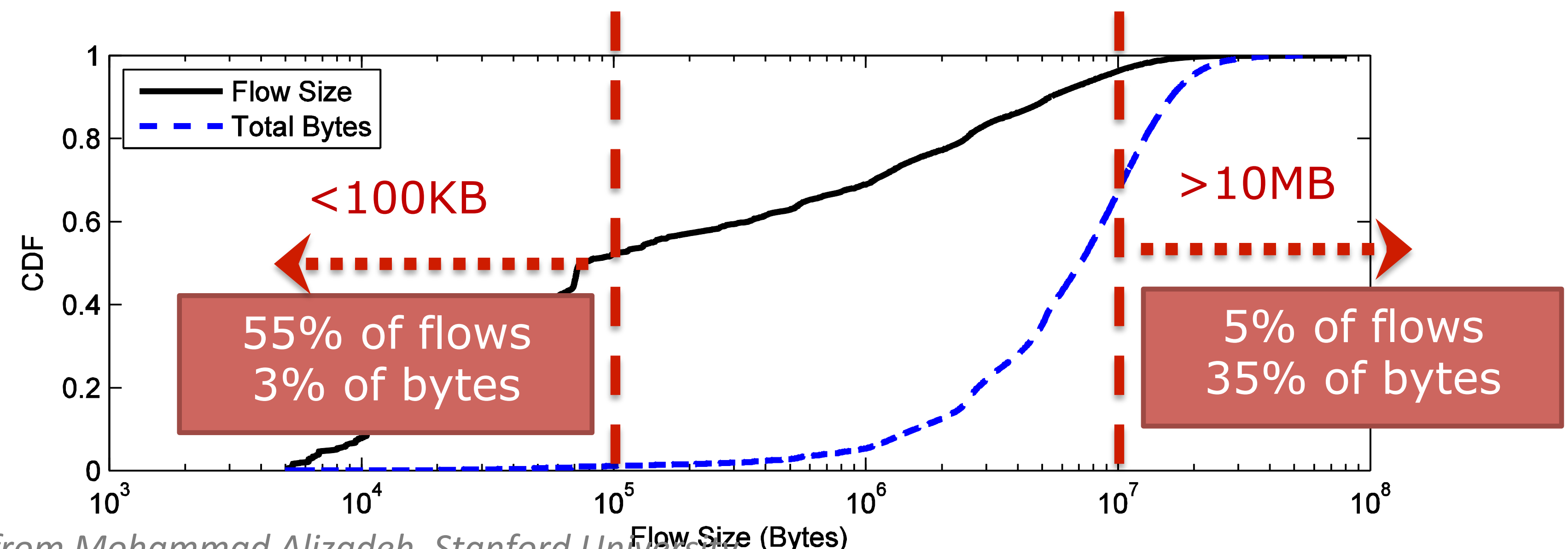
FCT with DCTCP



Slide from Mohammad Alizadeh, Stanford University

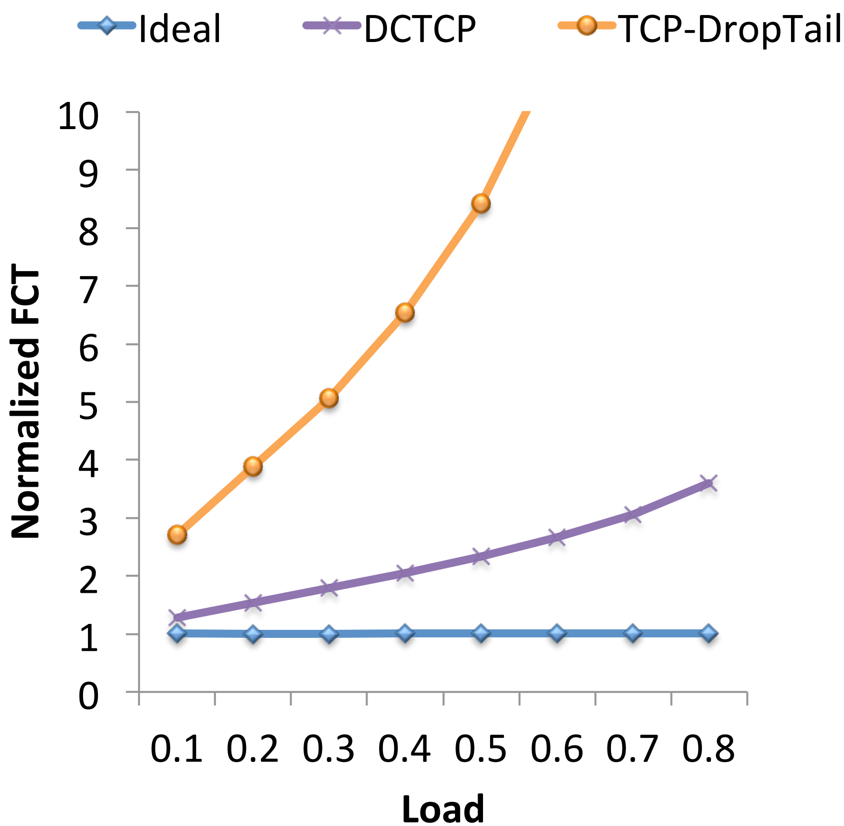
Recall: “Elephants” and “Mice”

- Web search, data mining (Microsoft) [Alizadeh 2010]



Slide from Mohammad Alizadeh, Stanford University

FCT with DCTCP



Problem: the mice are delayed by the elephants

Slide from Mohammad Alizadeh, Stanford University

Solution: use priorities!

[pFabric, Sigcomm 2013]

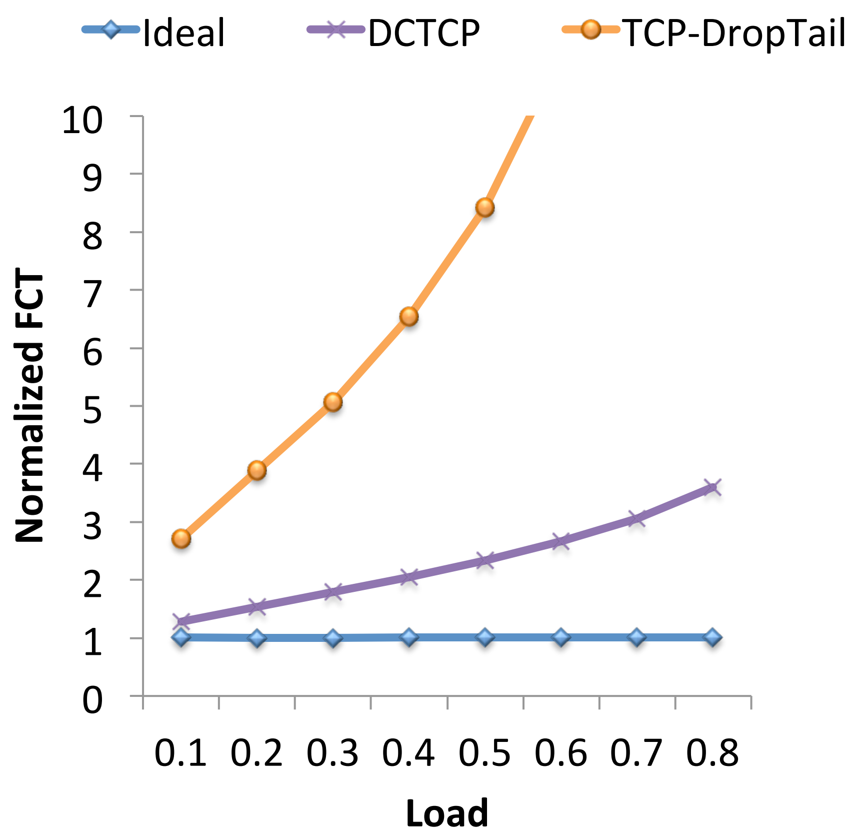
- **Packets carry a single priority number**
  - priority = remaining flow size (e.g., #bytes un-acked)

- **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)

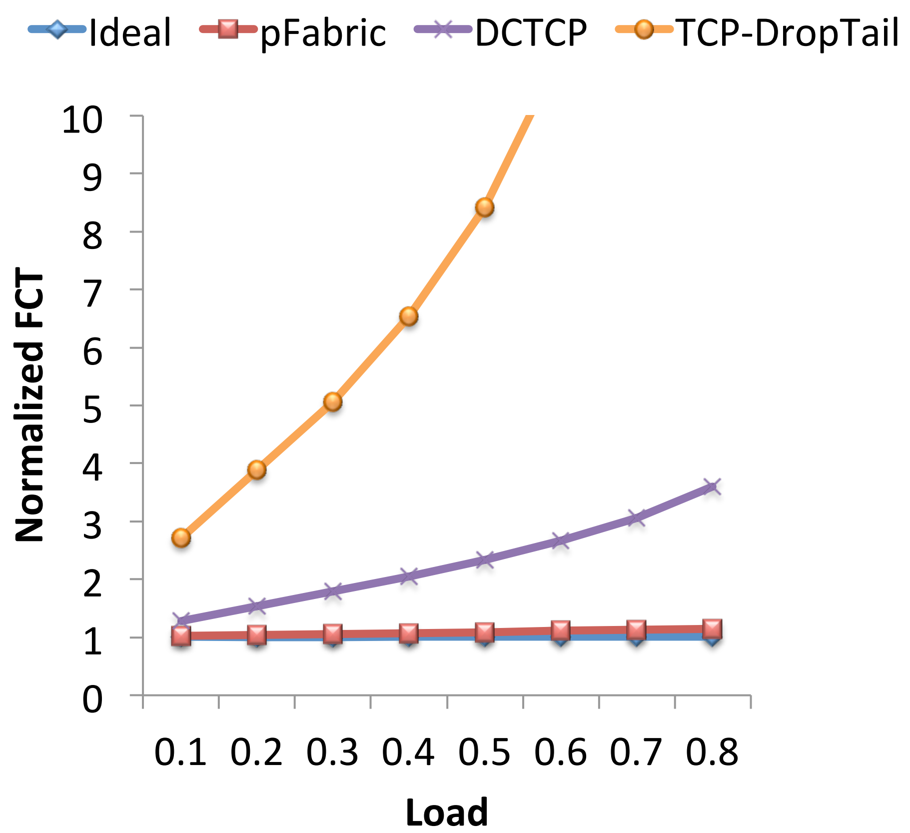
Slide from Mohammad Alizadeh, Stanford University

FCT



Slide from Mohammad Alizadeh, Stanford University

FCT



Slide from Mohammad Alizadeh, Stanford University

Why does pFabric work?

- Consider problem of scheduling N jobs at a single queue/processor
  - $J_{1,}$ J_2, … , J_n with duration T_1, T_2, …T_n respectively
- "Shortest Job First" (SJF) scheduling minimizes average Job Completion Time
  - 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

DIY exercise: why does SJF work?

- Consider 4 jobs with duration 1,2,4,8
- Compute finish times w/ Round-Robin vs. SJF
- Assume scheduler:
  - Picks best job (as per scheduling policy)
  - Runs job for one time unit
  - Update job durations
  - repeat
- Job completion times:
  - With RR: 1, 5, 10, 15
  - With SJF: 1, 3, 7, 15

Summary

- Learn more:
  - https://code.facebook.com/posts/360346274145943/introducing-data-center-fabric-the-next-generation-facebook-data-center-network/

- Next time: Guest lecture by Stephen Strowes, IPv6!