UNIVERSITY OF WATERLOO

CS 856: Programmable Networks

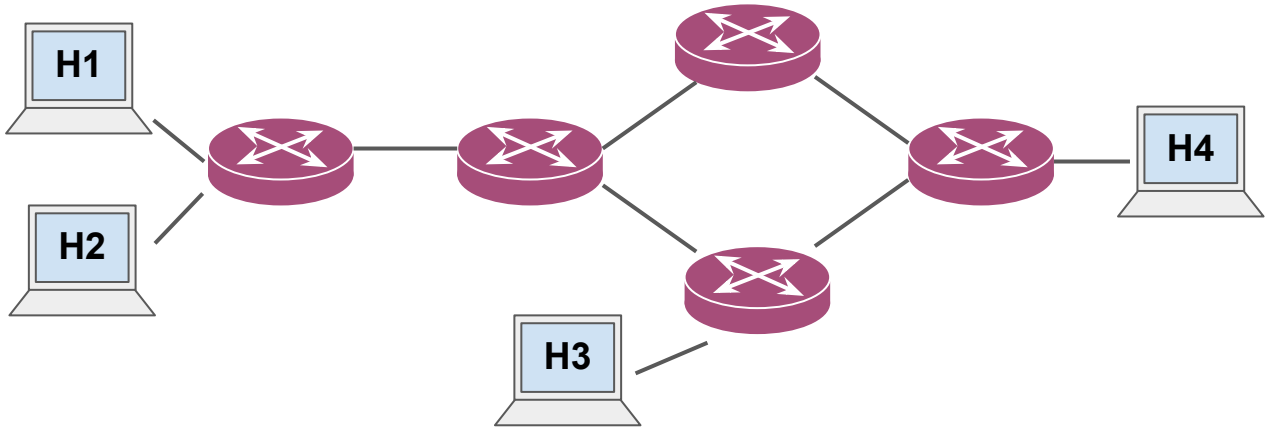
Lecture 9: Applications to Transport, Network QoS, and In-Network Computing

Mina Tahmasbi Arashloo

Winter 2024

Part 1: Transport and Network Quality of Servcie (QoS)

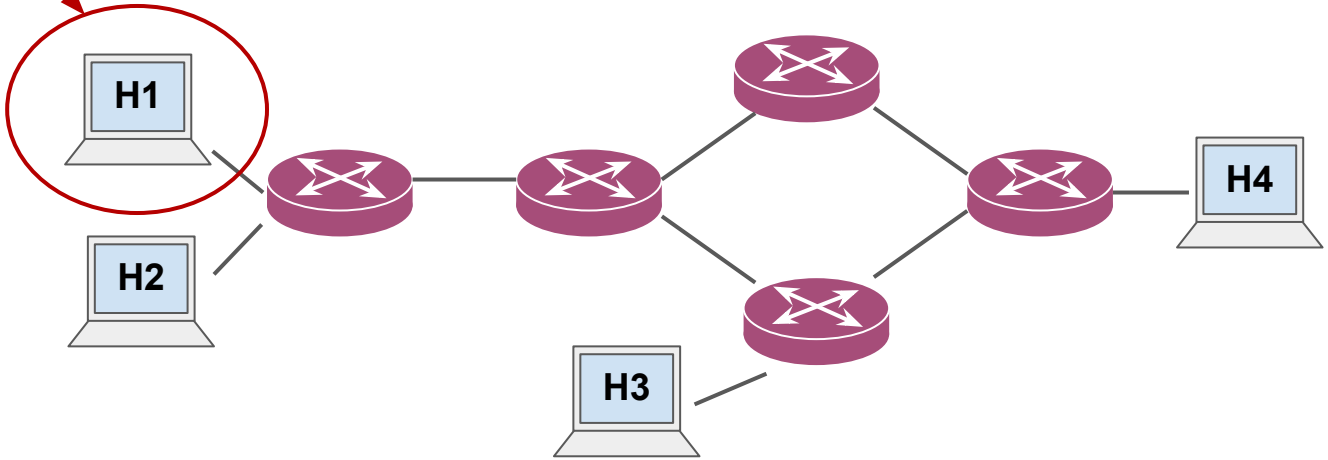
Networks are shared infrastructure



Networks are shared infrastructure

Traffic from multiple "flows" share

- The link to the network
- The memory and computational resources of the host



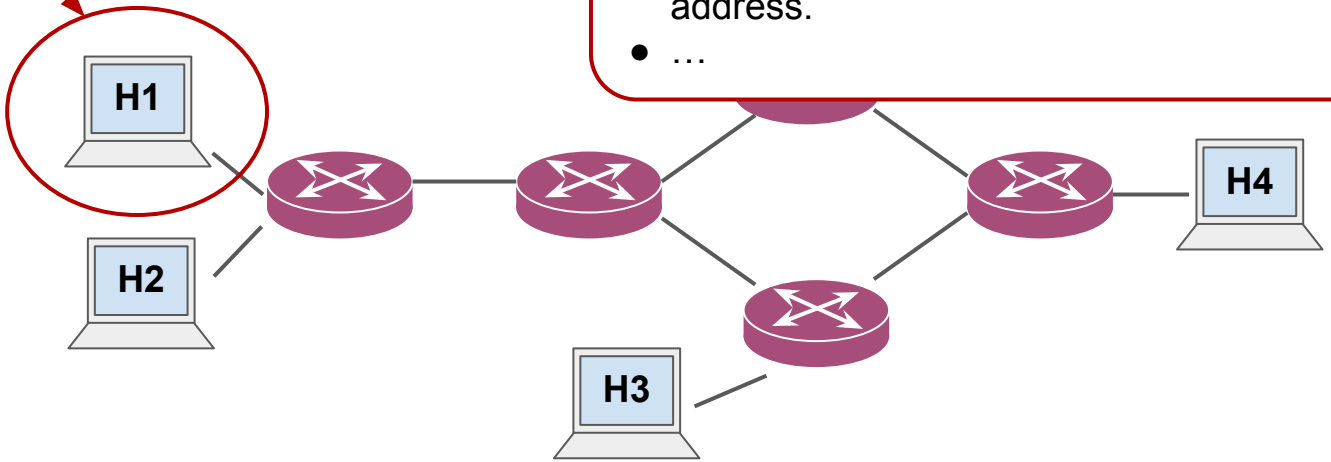
Networks are shared infrastructure

Traffic from multiple "flows" share

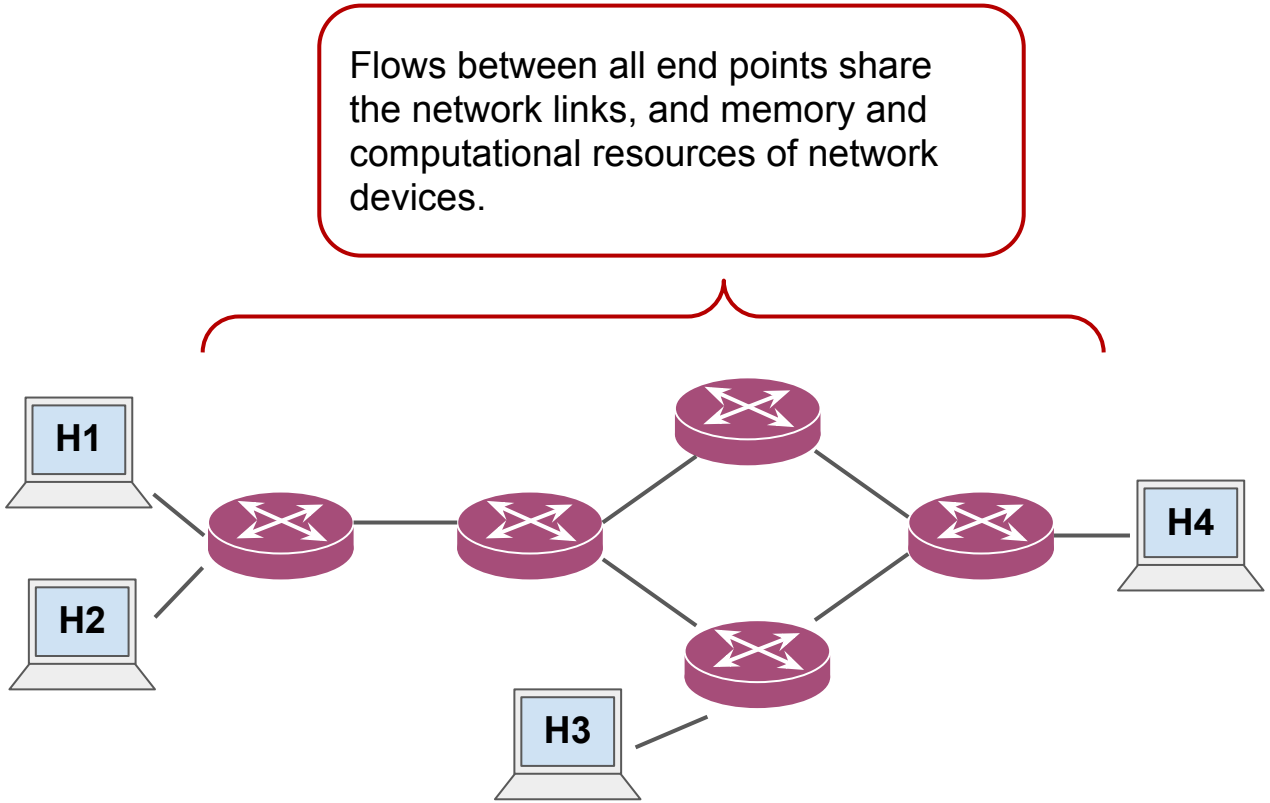
- The link to the network
- The memory and computational resources of the host

You can think of a flow in the broad sense of the term – a set of packets that are treated together as a group for network processing purposes:

- A TCP flow
- Packets originating from the same VM
- All the DNS packets from the same IP address.
- …

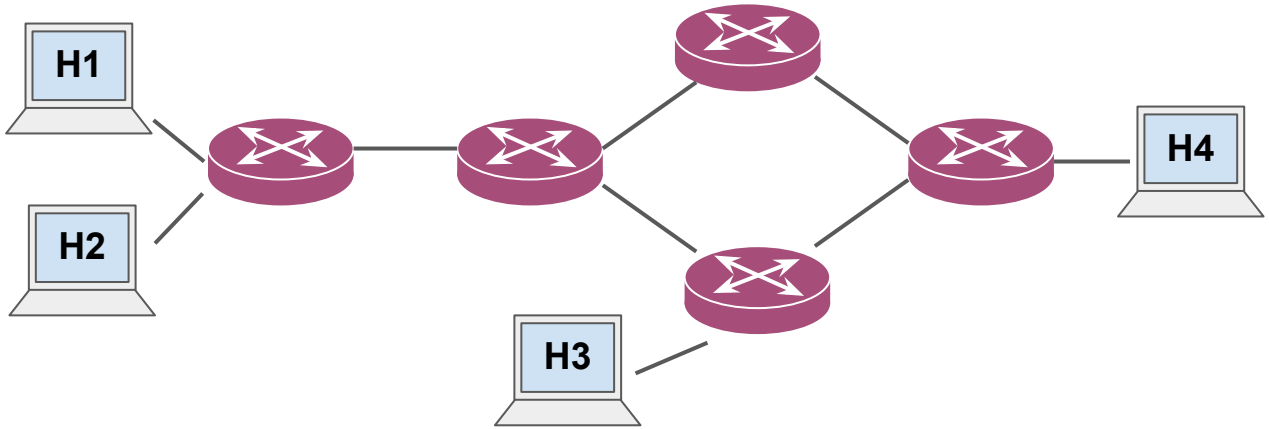


Networks are shared infrastructure



Networks are shared infrastructure

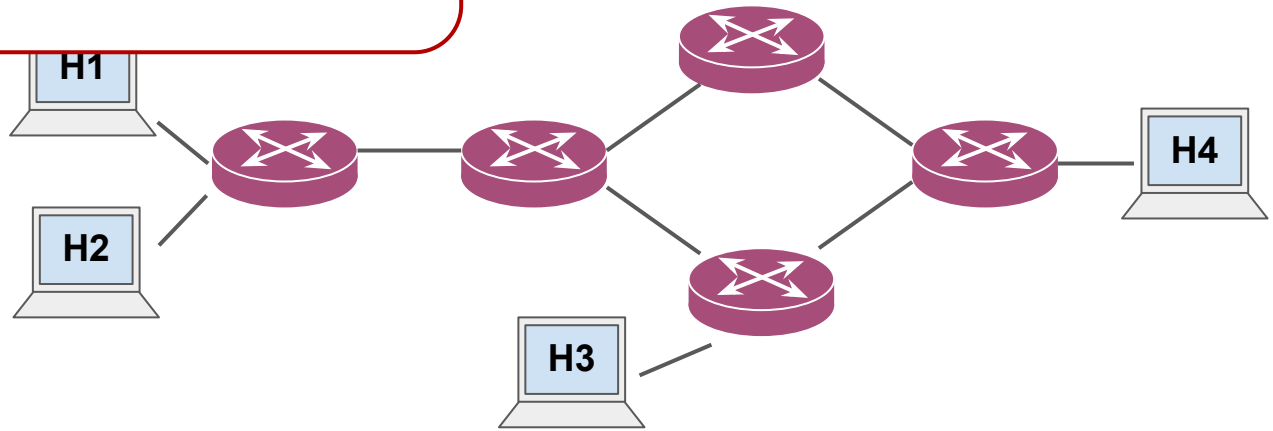
IP only provides *best-effort packet delivery*



Networks are shared infrastructure

IP only provides *best-effort packet delivery*

- Packets can get lost
- No performance (e.g., throughput, latency, jitter) bounds for a flow or classes of flows
- No notion of fairness
- …

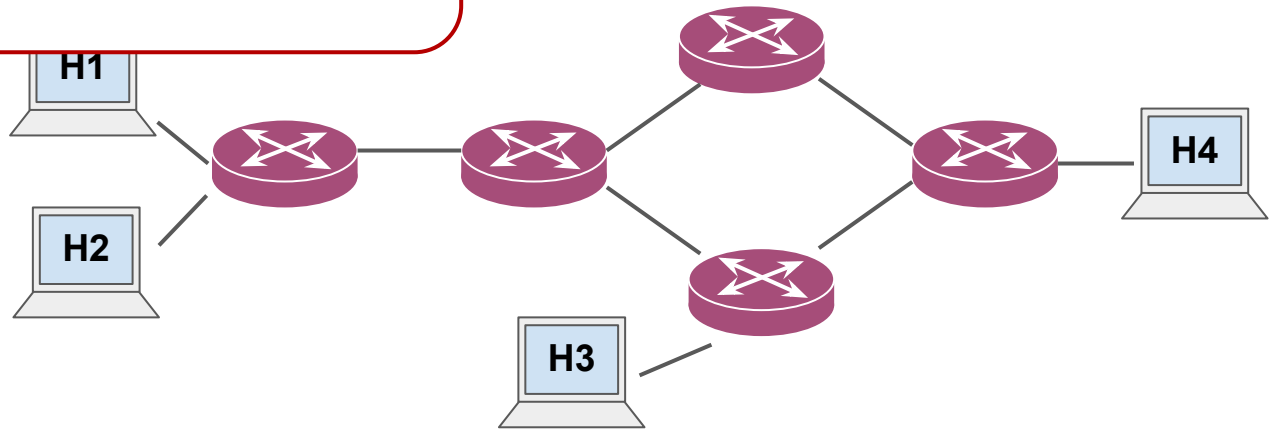


Networks are shared infrastructure

IP only provides *best-effort packet delivery*

- Packets can get lost
- No performance (e.g., throughput, latency, jitter) bounds for a flow or classes of flows
- No notion of fairness
- …

Doesn't have a notion of (or care about) flows

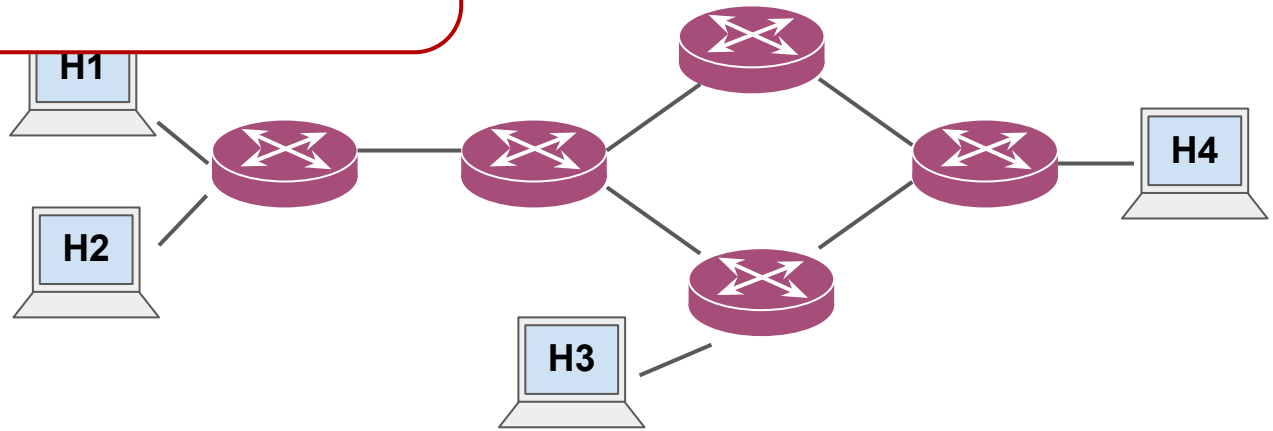


Networks are shared infrastructure

IP only provides *best-effort packet delivery*

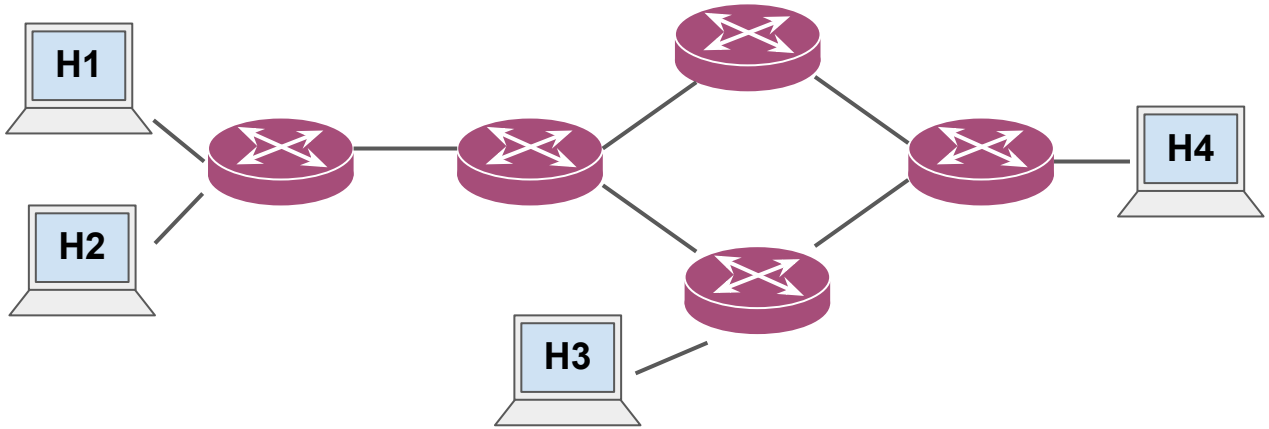
- Packets can get lost
- No performance (e.g., throughput, latency, jitter) bounds for a flow or classes of flows
- No notion of fairness
- …

Doesn't have a notion of (or care about) flows



Networks are shared infrastructure

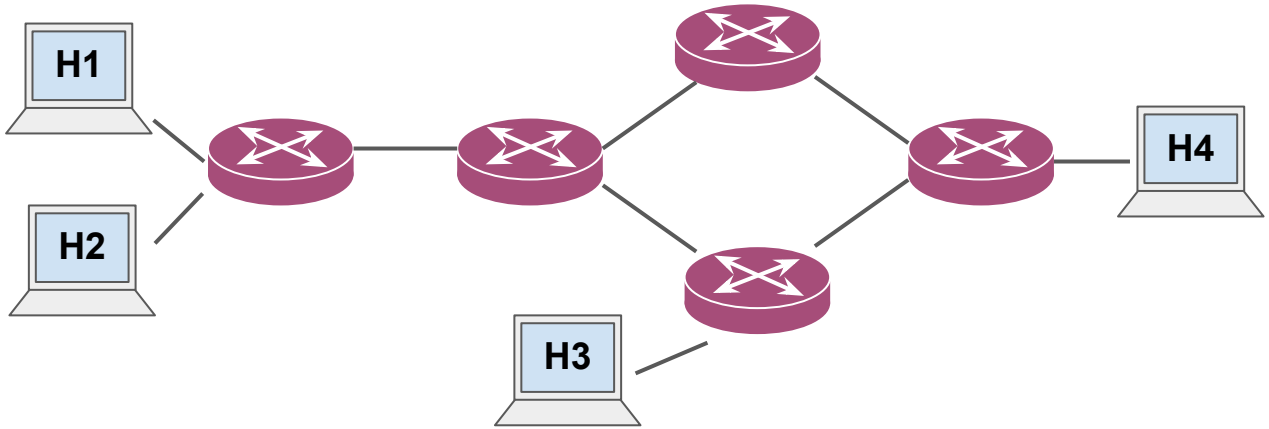
IP only provides *best-effort packet delivery*



Networks are shared infrastructure

IP only provides *best-effort packet delivery*

There are other mechanisms to control/customize how different flows share network resources.

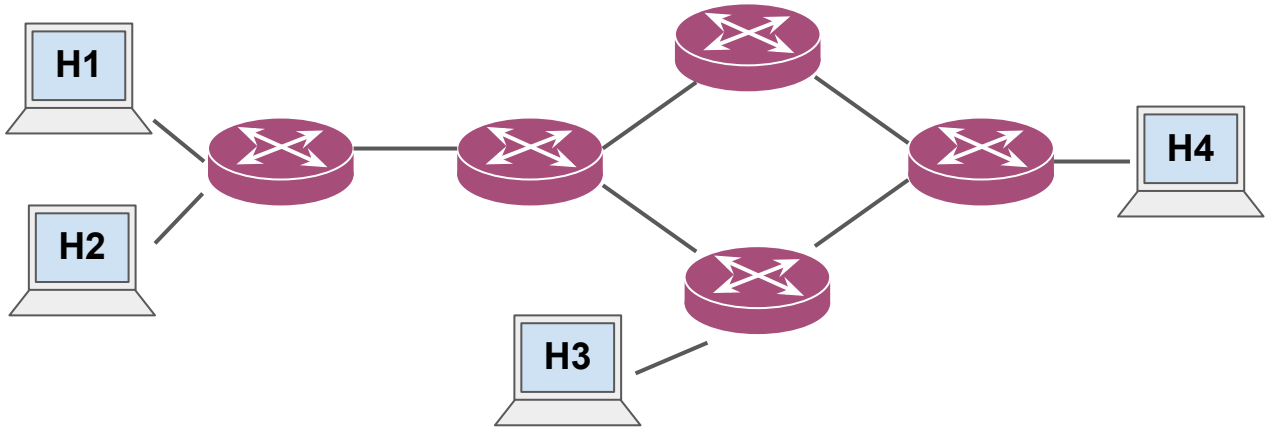


Networks are shared infrastructure

IP only provides *best-effort packet delivery*

There are other mechanisms to control/customize how different flows share network resources.

- end-to-end congestion control
- packet scheduling
- active queue management
- …

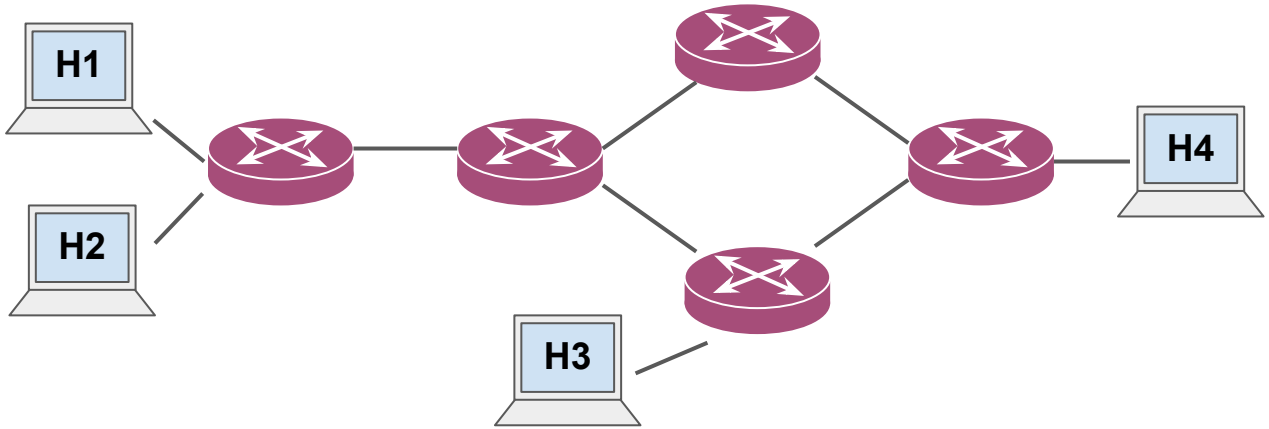


Networks are shared infrastructure

IP only provides *best-effort packet delivery*

There are other mechanisms to control/customize how different flows share network resources.

- end-to-end congestion control
- packet scheduling
- active queue management
- …

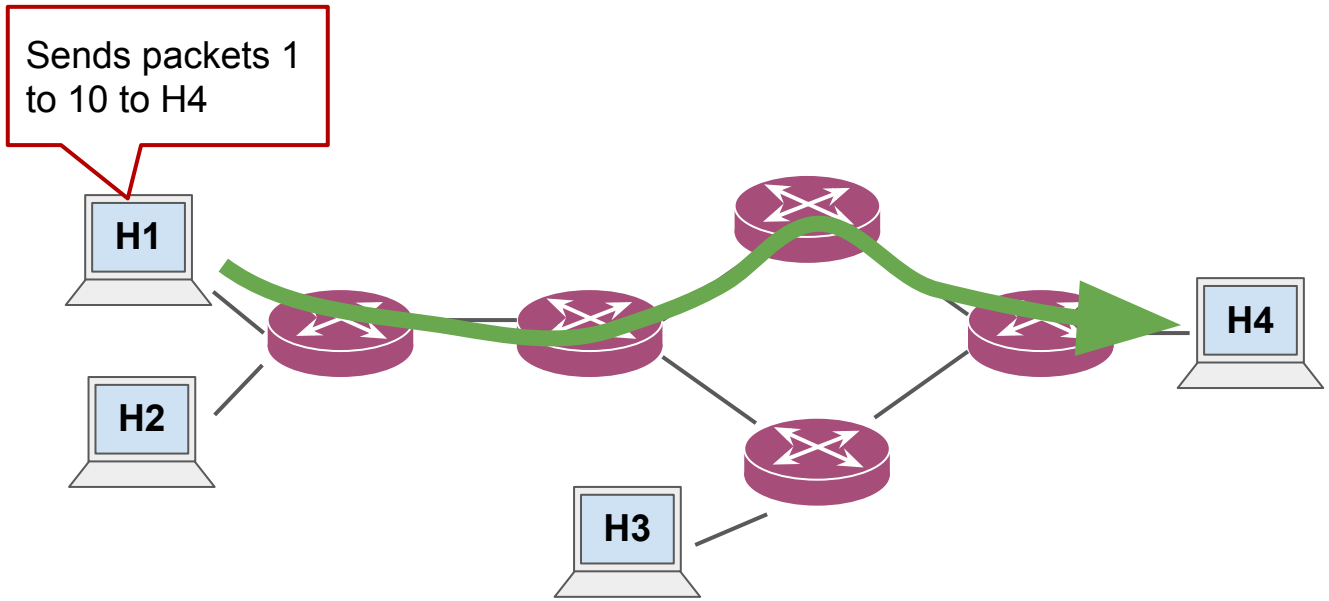


End-to-End Congestion Control

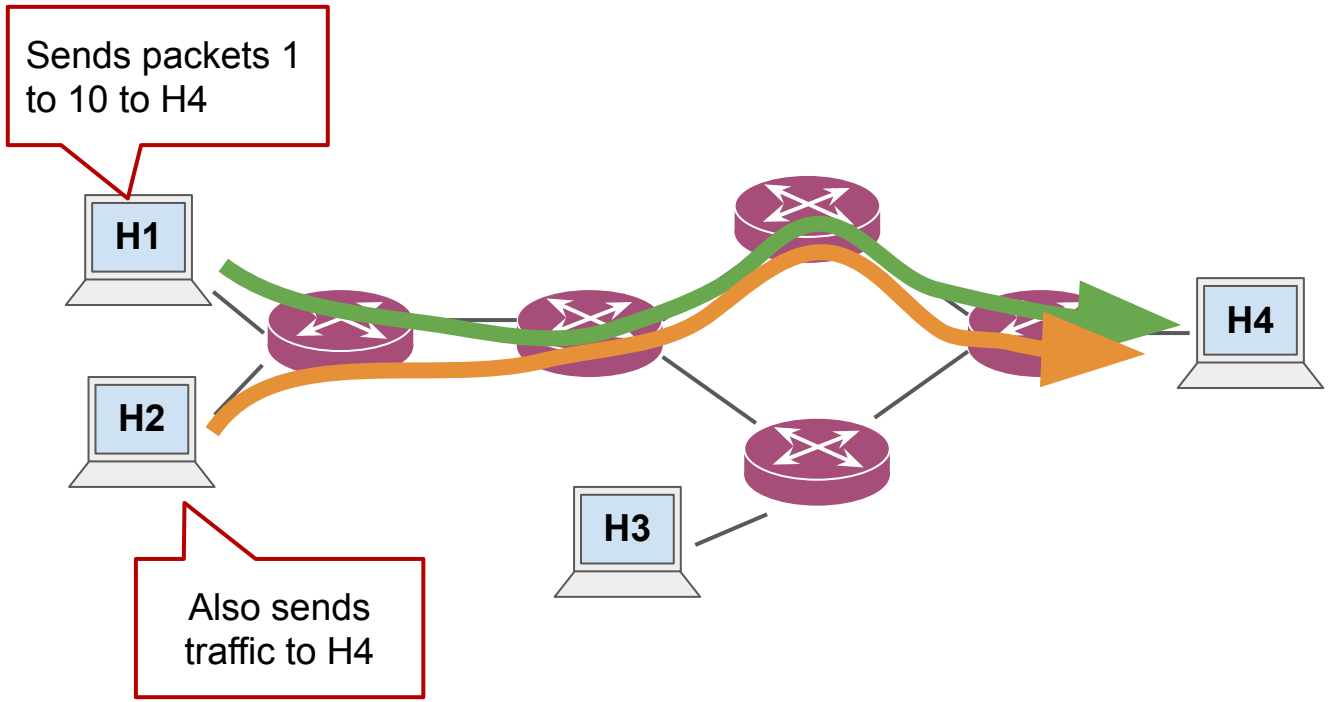
For every flow, at the sender

- Send some packets out.
- Use some signals to detect congestion in the network:
  - packets getting lost
  - packets taking longer to get to the receiver
  - network/receiver telling you it is congested
  - …
- Adjust sending rate accordingly.

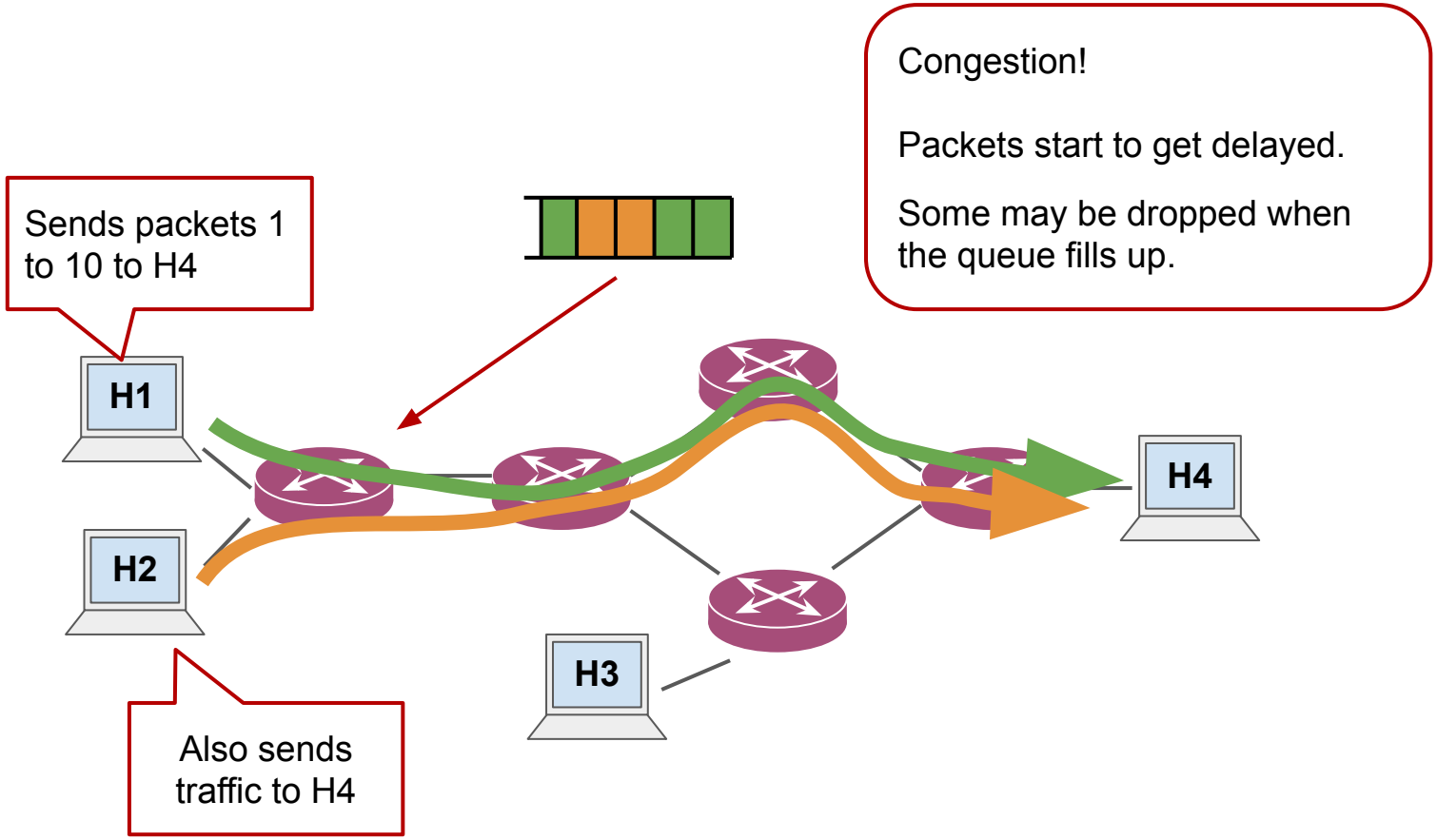
End-to-End Congestion Control



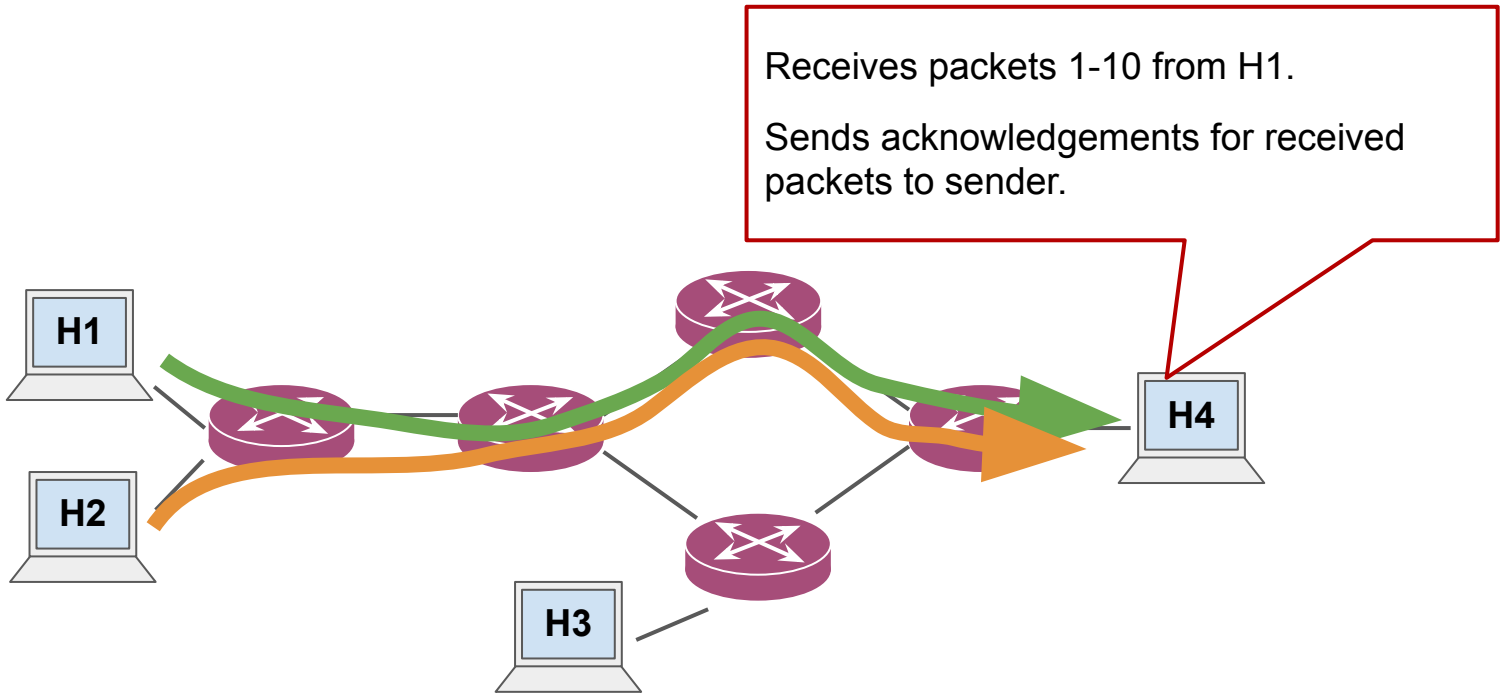
End-to-End Congestion Control



End-to-End Congestion Control



End-to-End Congestion Control



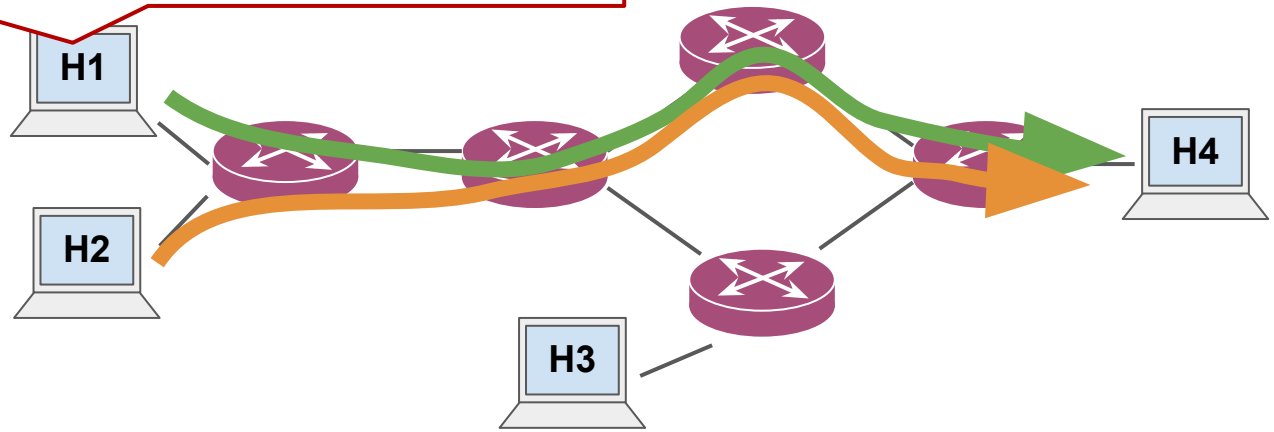
End-to-End Congestion Control

Receives acknowledgements for packets 1-10.

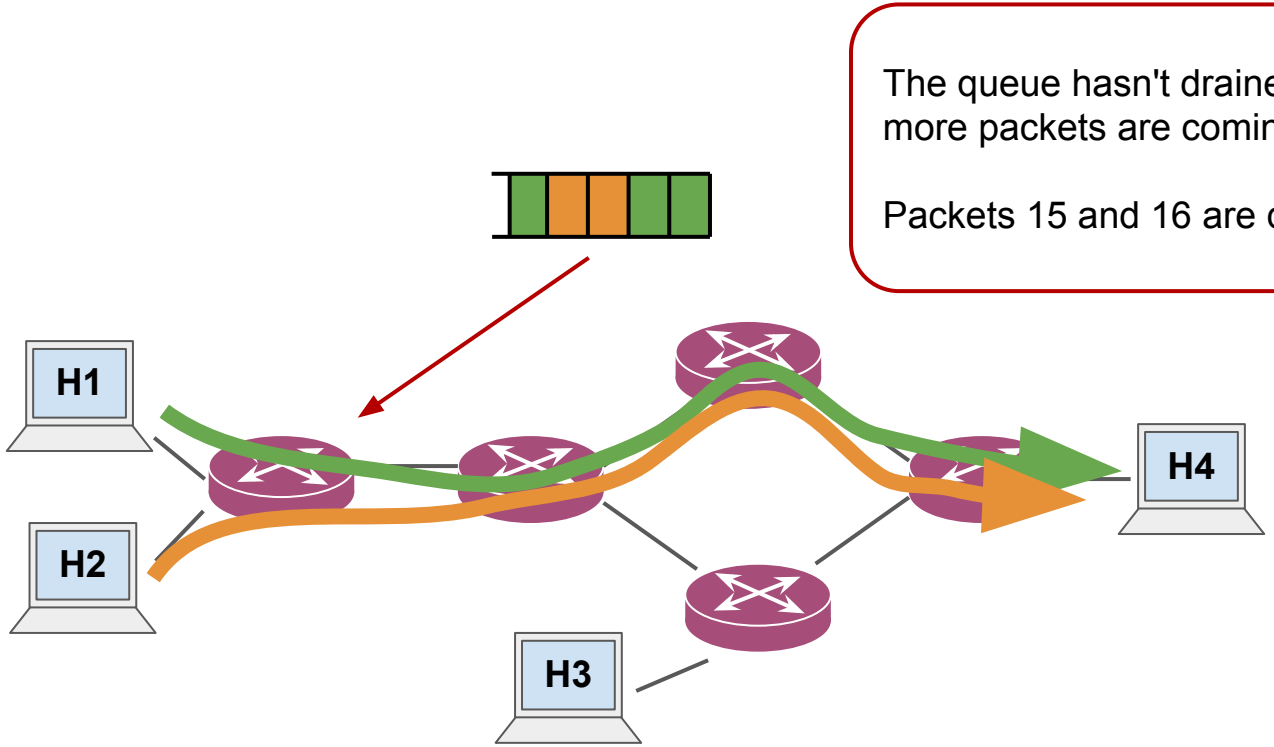
Notices packets 8-10 took longer to get acknowledged.

Maybe there is a minor congestion?

Next time, only sends 7 packets out.



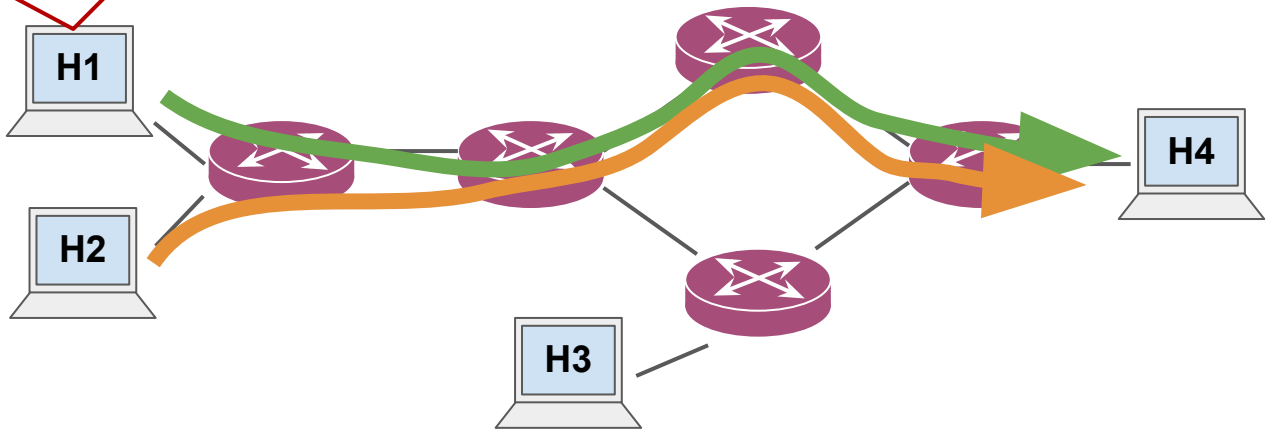
End-to-End Congestion Control



End-to-End Congestion Control

When it doesn't receive acks for packets 15 and 16, decides there is severe congestion going on.

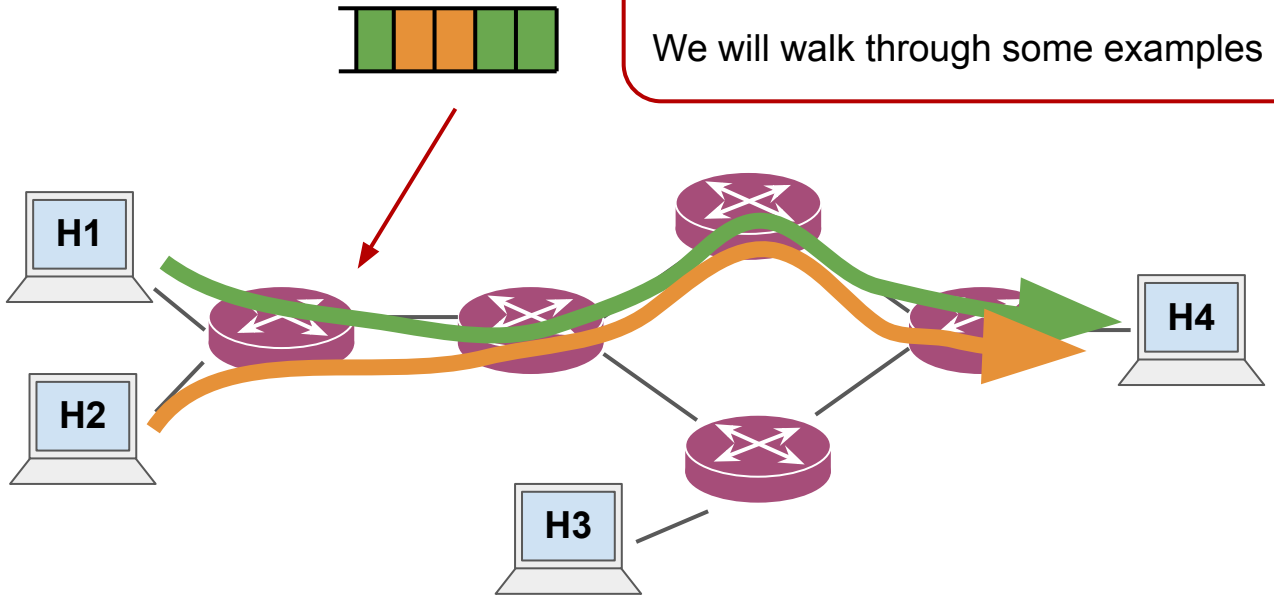
Only sends 2 packets out next time.



End-to-End Congestion Control

When the queue starts building up, the switch can signal the end-points to slow down.

We will walk through some examples later.



How does it affect resource sharing?

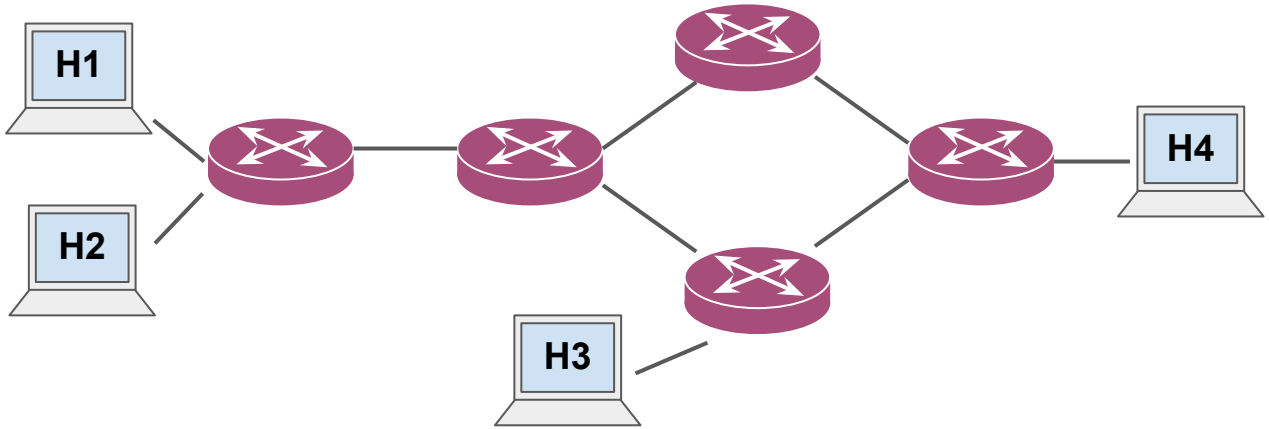
- The congestion-control algorithm decides when and how much to change the pace for each flow
- It affects how different flows in the network interact with each other at bottlenecks.
- E.g., the flows that back off quickly and to larger extents can end up with a lower share of the bottleneck bandwidth.

Networks are shared infrastructure

IP only provides *best-effort packet delivery*

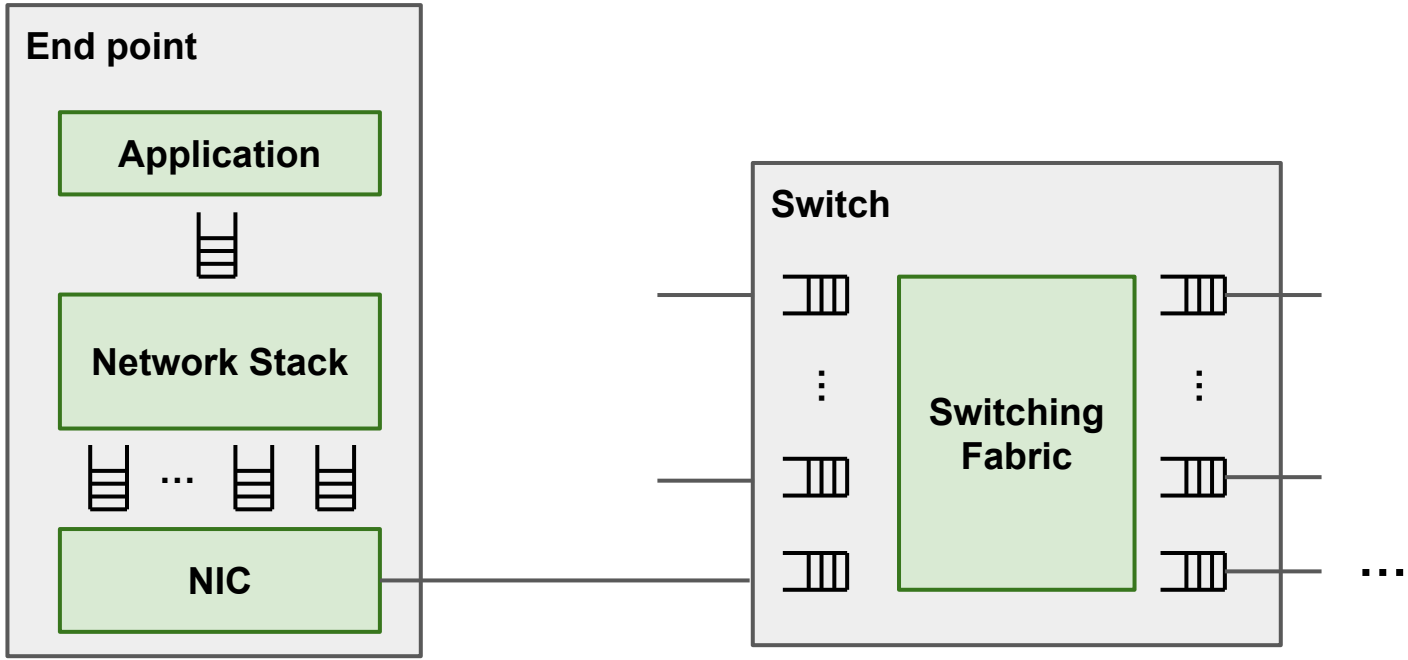
There are other mechanisms to control/customize how different flows share network resources.

- end-to-end congestion control
- packet scheduling
- active queue management
- …



Packet Scheduling

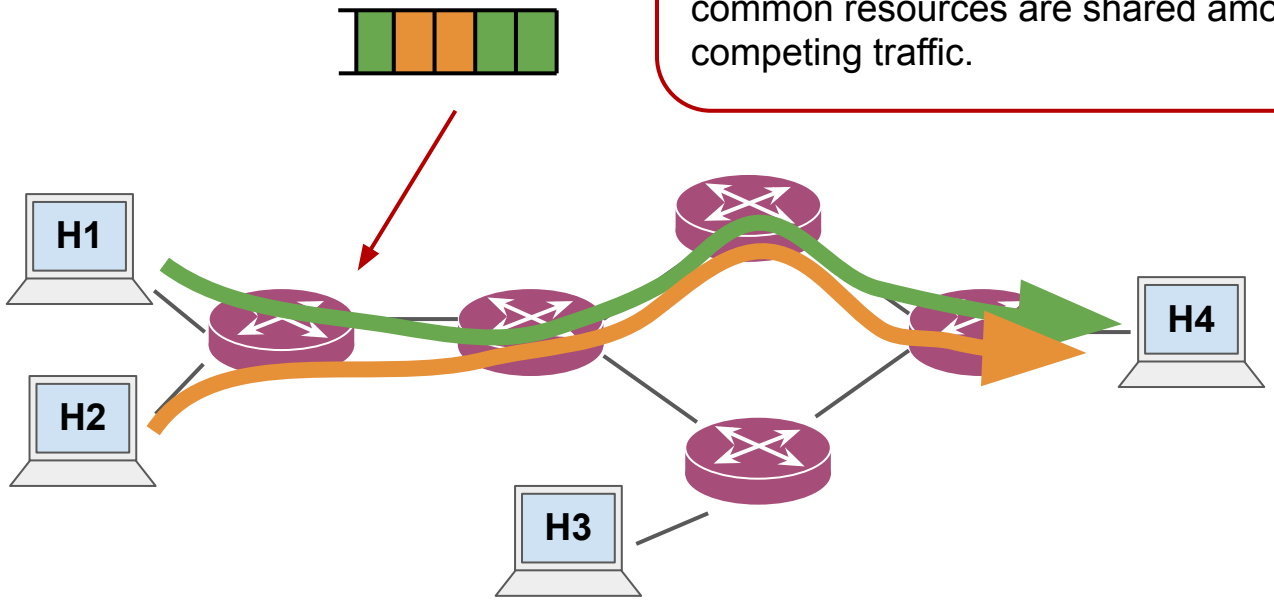
Queues are an integral part of networks



Packet Scheduling

Packet schedulers decide how to line up packets in queues and who gets to go next.

When there is contention, they can affect how common resources are shared among competing traffic.



Packet Scheduling

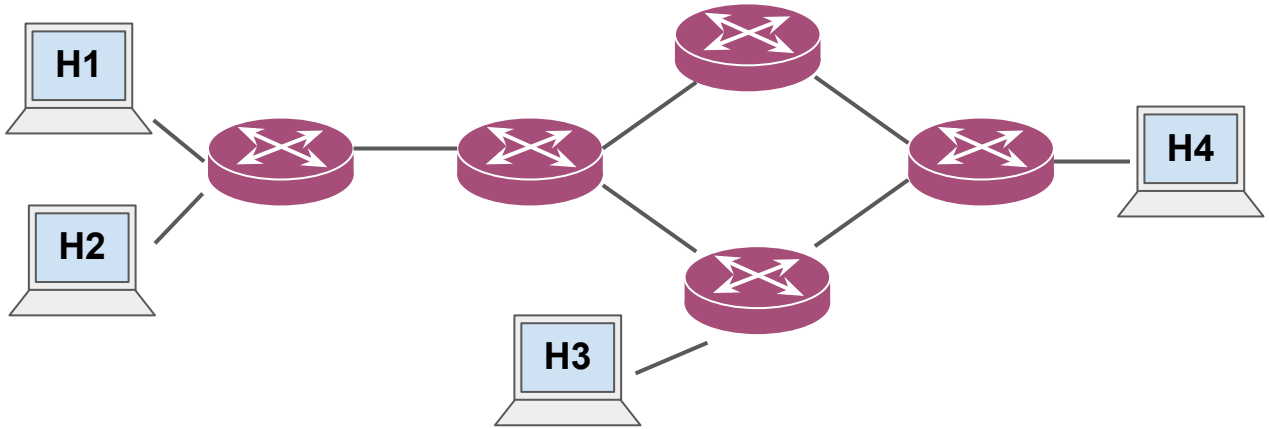
- The simplest "scheduler" is First In First Out (FIFO)
  - Packets are queued up in the order they arrive and exit in the same order
- There are many other, more complex, schedulers
  - A single priority queue (a packet's priority is decided on enqueue)
  - A set of FIFOs, each with its own priority
  - A set of FIFOs, serviced in round robin fashion
  - A FIFO, but packets can only be dequeued at a specific rate
  - A hierarchy of schedulers
  - …

Networks are shared infrastructure

IP only provides *best-effort packet delivery*

There are other mechanisms to control/customize how different flows share network resources.

- end-to-end congestion control
- packet scheduling
- active queue management
- …



Active Queue Management (AQM)

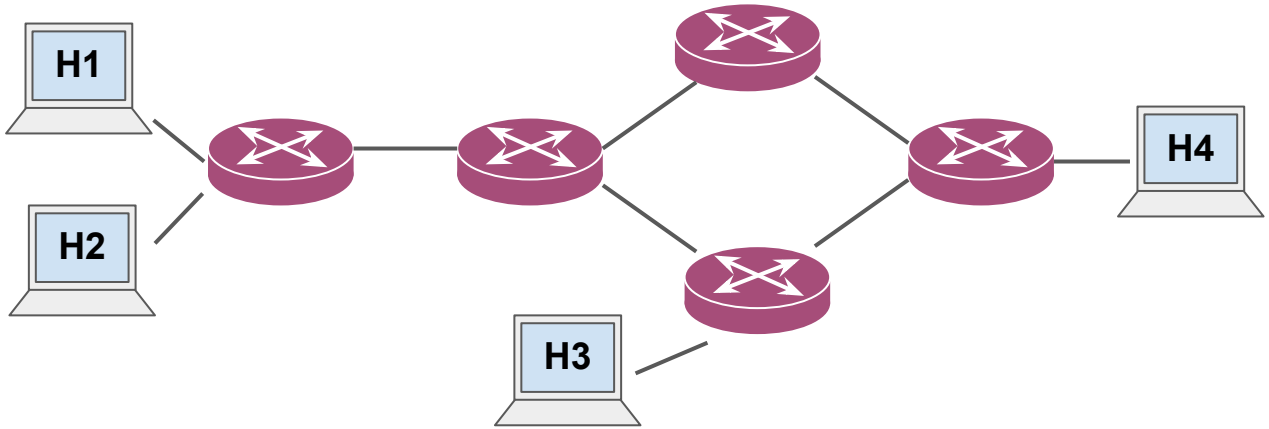
- AQM algorithms manage the occupancy of a single queue
- They try to drop/mark packets before the queue is full
    - To keep the queue occupancy, and therefore, latency, within desirable bounds.
- Different algorithms have different ways of deciding when to start dropping/marking, whether to drop or mark, and which packets to drop/mark.

Networks are shared infrastructure

IP only provides *best-effort packet delivery*

There are other mechanisms to control/customize how different flows share network resources.

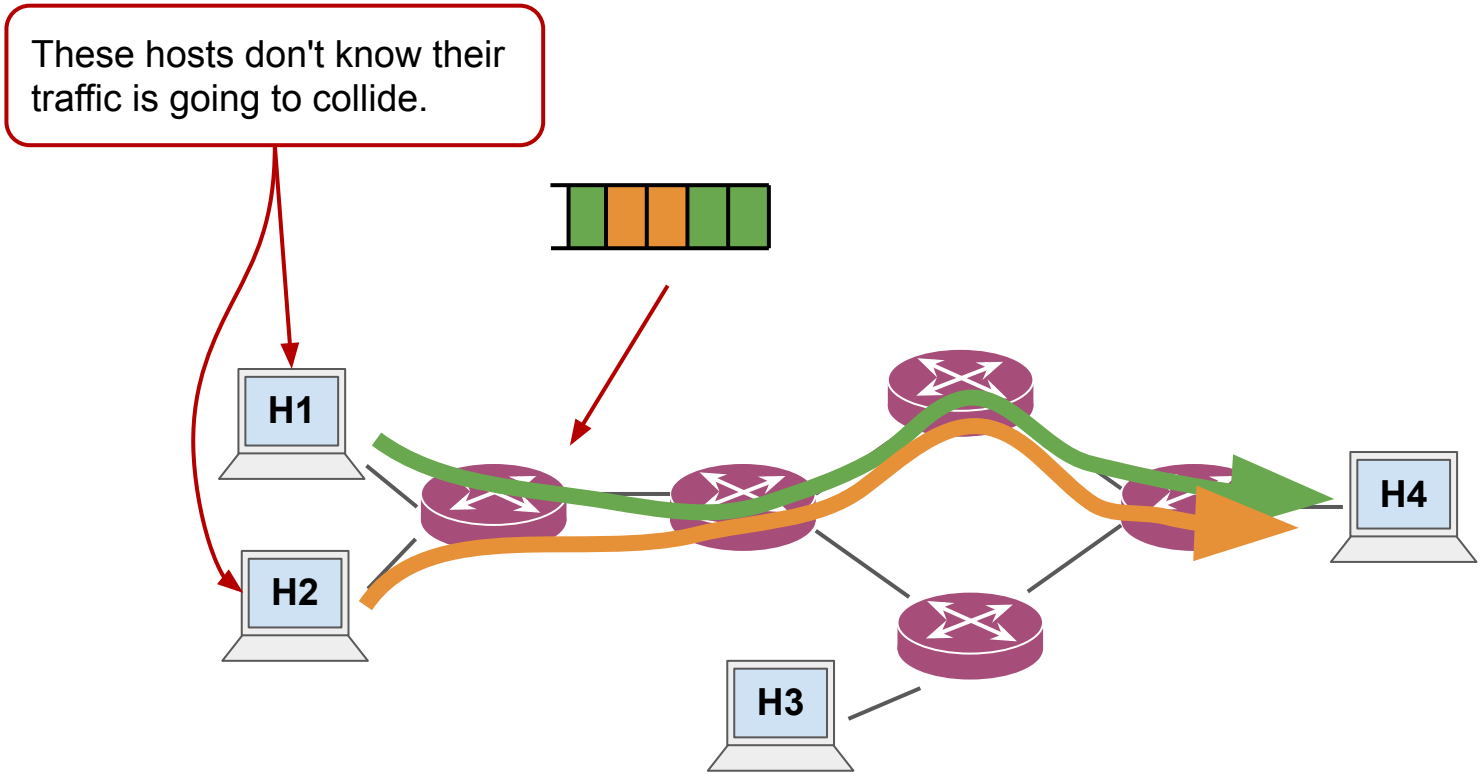
- end-to-end congestion control
- packet scheduling
- active queue management
- …



Traditional networks mostly rely on end-to-end congestion control

- keeping the functionality in the network quite simple
  - No explicit signals or a simple fixed set of signals for end-to-end congestion control algorithms
  - A few FIFO queues and a few schedulers (e.g., priorities, DRR, etc.)
  - No AQM or a simple fixed set of AQM algorithms
- Why?
  - end-to-end principle
  - Keeping network devices simple and fast

A little help from the network can go a long way

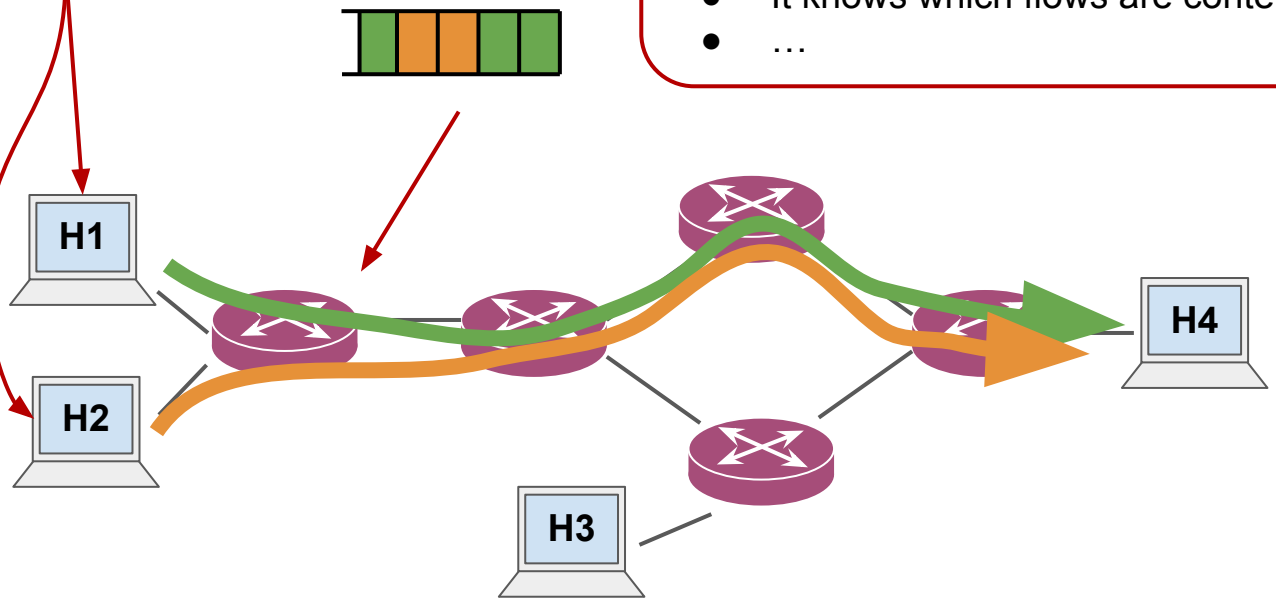


But a little help from the network can go a long way

These hosts don't know their traffic is going to collide.

The switch knows a lot more about the contention

- It is where the contention is happening
- It sees the queue building up
- It knows which flows are contending
- …



But a little help from the network can go a long way

- In traditional networks, the sender has to infer what is happening at the switch from indirect signals (delays, loss, marked packets).
- Why not have the the switch provide the information more directly and explicitly to senders?
- Why not have the switch play a more active role in handling contention with more sophisticated scheduling and AQM algorithms?

Nevertheless, resistance to changing network hardware to accommodate extra help

- Some researchers showed the benefits, but needed change to the switch hardware
- Conflicted with the goal of keeping the functionality in the network simple and general

And we may be forced to do more in the network anyway

- End-to-end congestion control relies on sending packets out, getting signals, and adjusting it sending strategy.
- It works best when flows can take a couple of round-trip-times (RTTs) to complete
  - They can send packets for a few rounds, and get an accurate picture of the available network capacity.
- However, this may not happen in certain networks

And we may be forced to do more in the network anyway

- For example, in data centers:
  - Link speeds are increasing
  - Flows are getting shorter
- Flows can take fewer RTTs (even just one or two) to complete.
- Not enough time for the end-to-end congestion control algorithms to "figure out" the right rate.

How has network programmability helped?

- Customizing the signals to e2e congestion control, scheduling, AQM, etc. to the each network and the requirements of its applications.
- Motivating new signaling, scheduling, AQM, etc. techniques
  - Implement it in a programmable switch
  - show it can run at line rate
  - show it provides significant benefits
  - so you can convince vendors to include it in their switches

How has network programmability helped?

- Better signals for congestion control algorithms
  - e.g., use INT to add information about queue lengths to the packets (HPCC, SIGCOMM 2019)
- More complex (and flexible) packet scheduling
  - e.g., fair queuing is hard to implement at line-rate but you can implement and approximation on programmable switches (AFQ, NSDI'18).
  - a programmable hardware architecture for packet scheduling, so we can configure the switch for different scheduling algorithms (PIFO, SIGCOMM'16)

How has network programmability helped?

- Targeted fine-grained measurements
  - can help provide better signals to congestion control algorithms
  - can help create more effective AQM schemes
  - e.g., if we could detect which flow(s) contribute most to the queue build up, we can mark/drop those packets in our AQM scheme (Conquest, CoNEXT'19)

Paper 1: HPCC: High Precision Congestion Control

- Uses INT to provide more accurate signals to congestion control algorithms
- Demonstrates the benefit in low-latency high-bandwidth RDMA networks.

Paper 2: Approximating Fair Queueing on Reconfigurable Switches

- Fair queuing ensure all flows sharing a link get a fair share of the bandwidth
    - providing the illusion that each flow has its own separate queue with a "round-robin" service across queues
- It has been deemed too complex to implement in switches.
- This paper shows that an approximate version can be implemented in programmable switches.

Additional Resources

- Back-pressure Flow Control (BFC) (NSDI'22)
- Programmable packet scheduling at line-rate (SIGCOMM'16)
- Loom: Flexible and Efficient NIC Packet Scheduling (NSDI'19)
- Fine-Grained Queue Measurement in the Data Plane (CoNEXT'19)
- ABM: Active Buffer Management in Datacenters (SIGCOMM'22)

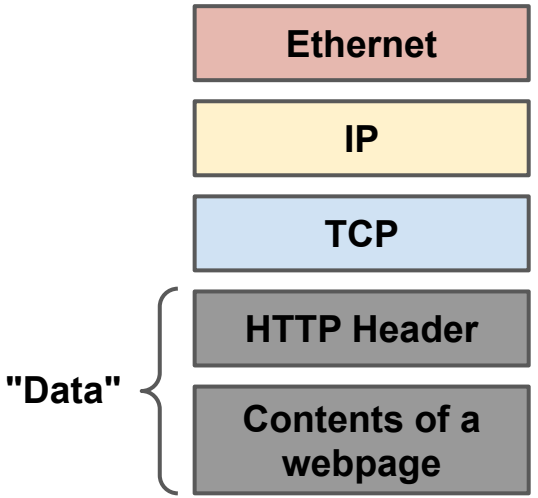
Part 2: In-Network Computing

So far: using network programmability to improve the network

- Trying out new algorithms/protocols
- Customizing packet processing to the specific needs of a network
- Helping with network verification
- Flexible and fine-grained monitoring
- In-network support for quality of service and transport-layer algorithms
- …

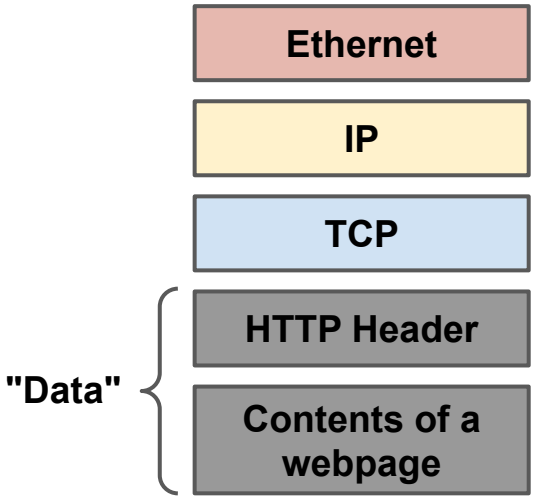
Using network programmability to accelerate applications?

- With programmable parsing, we can specify what we want to parse from the packet.
- Why stop after the transport-layer headers?
- We can look into the data that networked applications put into packets.



Using network programmability to accelerate applications?

- A programmable network device has limited computational resources and capabilities.
- But it can still do basic arithmetic operations
- and keep track of some information across packets.

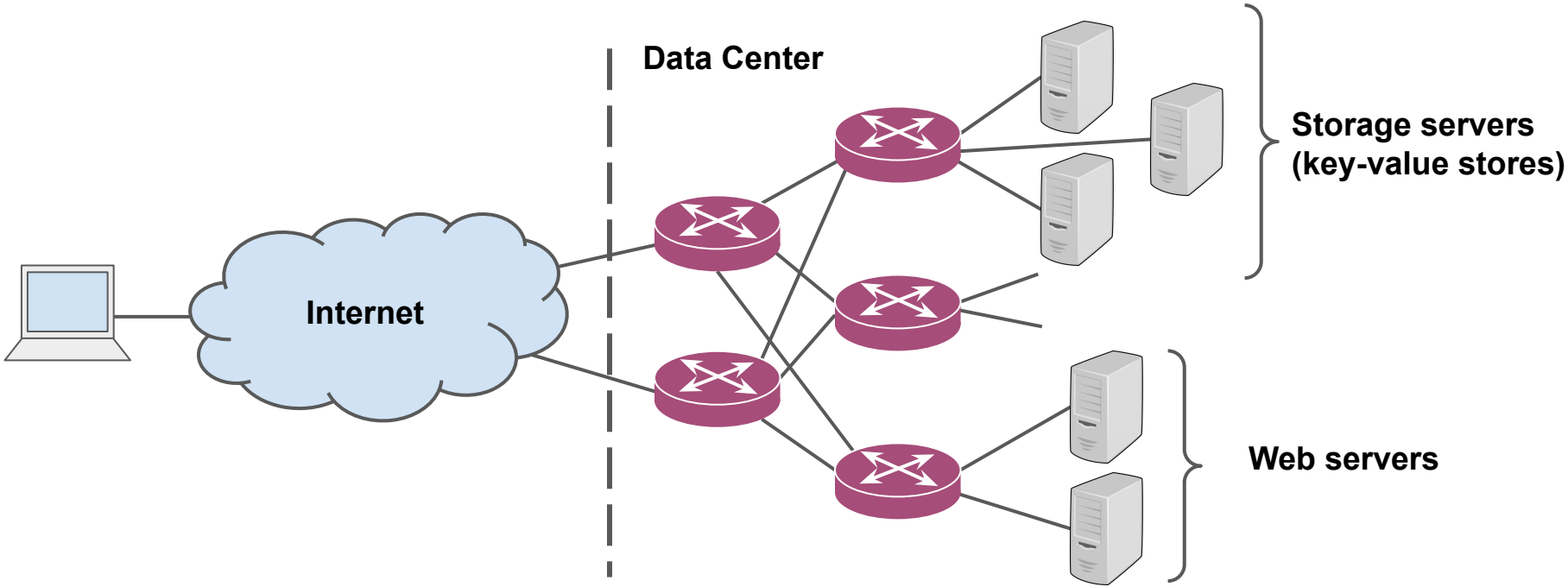


In-Network Computing

Offloading part of the application processing (i.e., compute) to the network

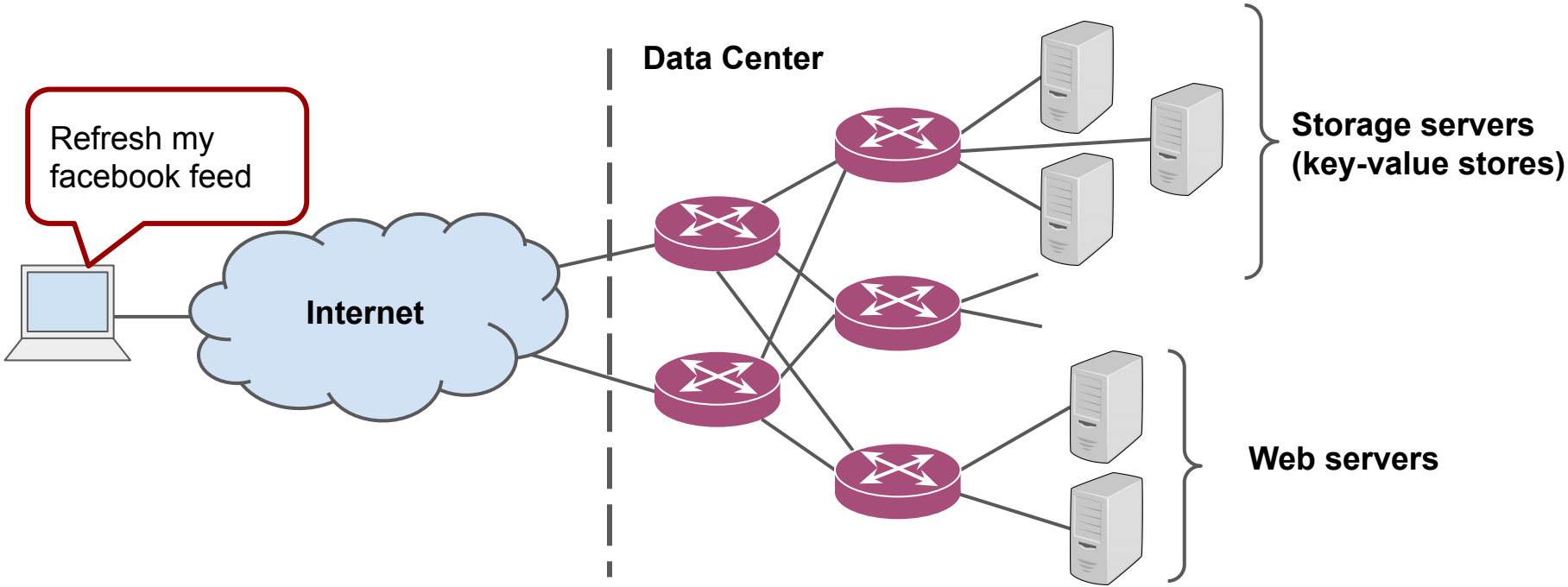
Example 1: In-network caching

- Online services rely quite heavily on distributed key-value stores.



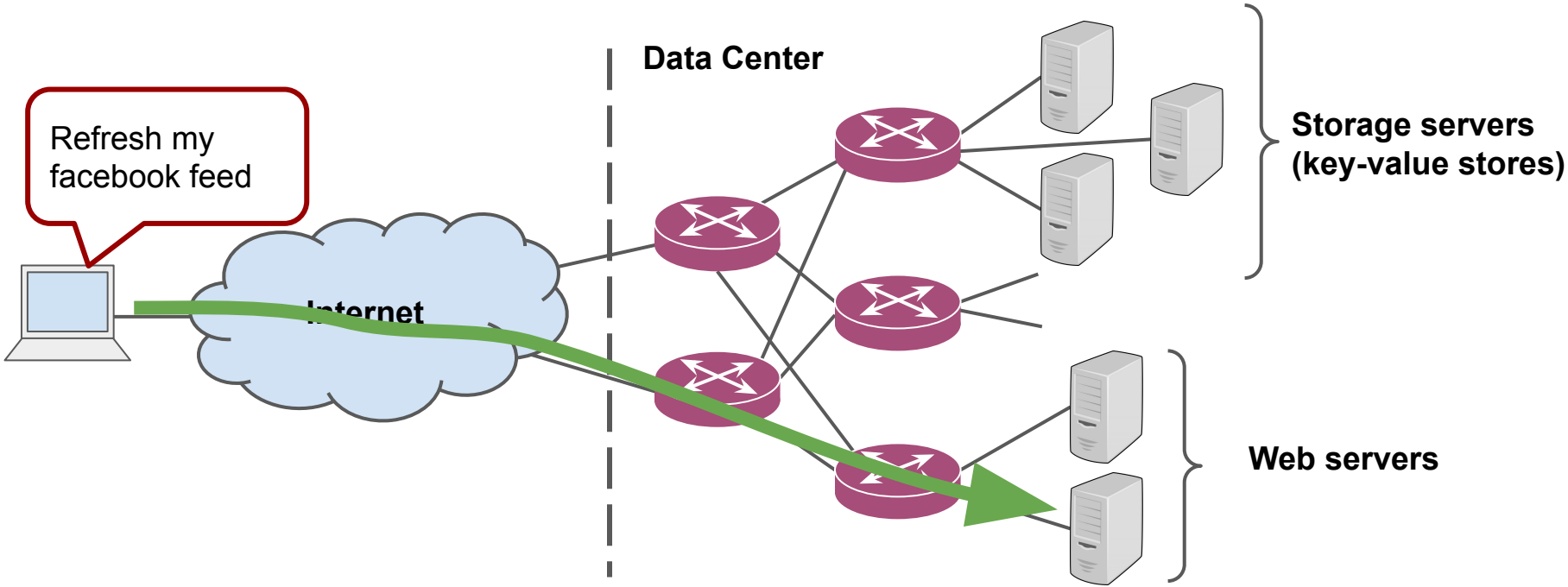
Example 1: In-network caching

- Online services rely quite heavily on distributed key-value stores.



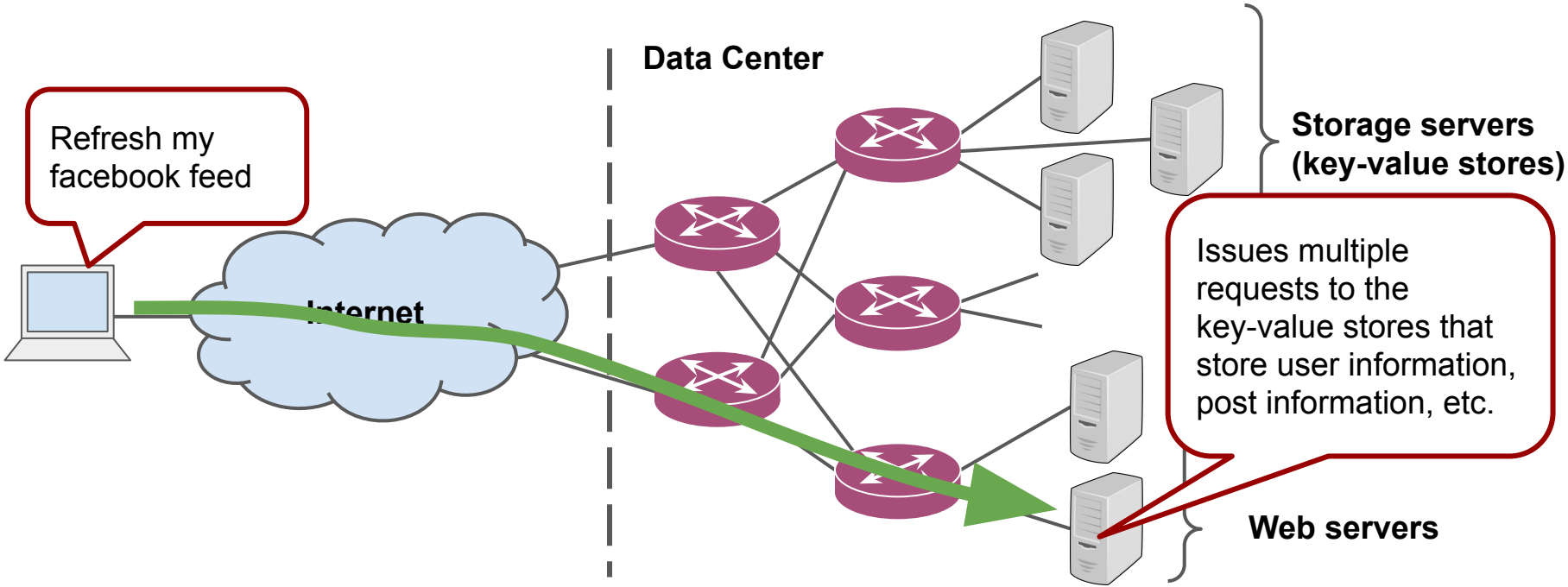
Example 1: In-network caching

- Online services rely quite heavily on distributed key-value stores.



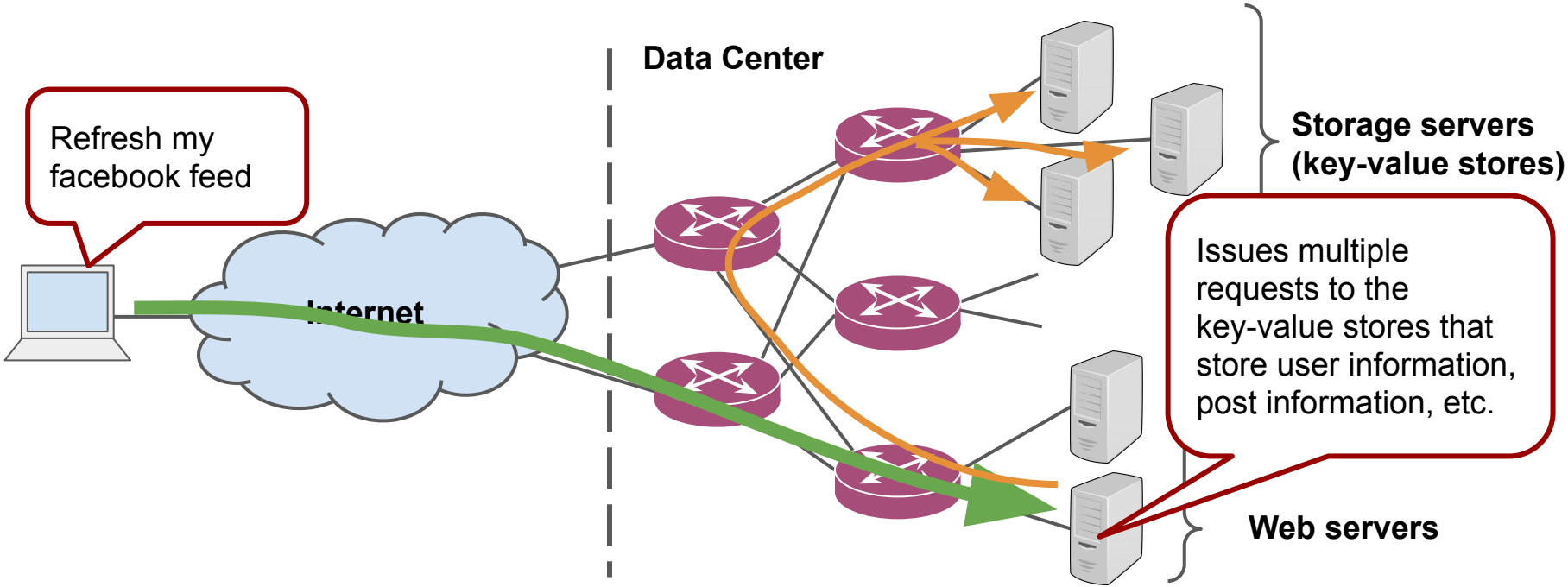
Example 1: In-network caching

- Online services rely quite heavily on distributed key-value stores.



Example 1: In-network caching

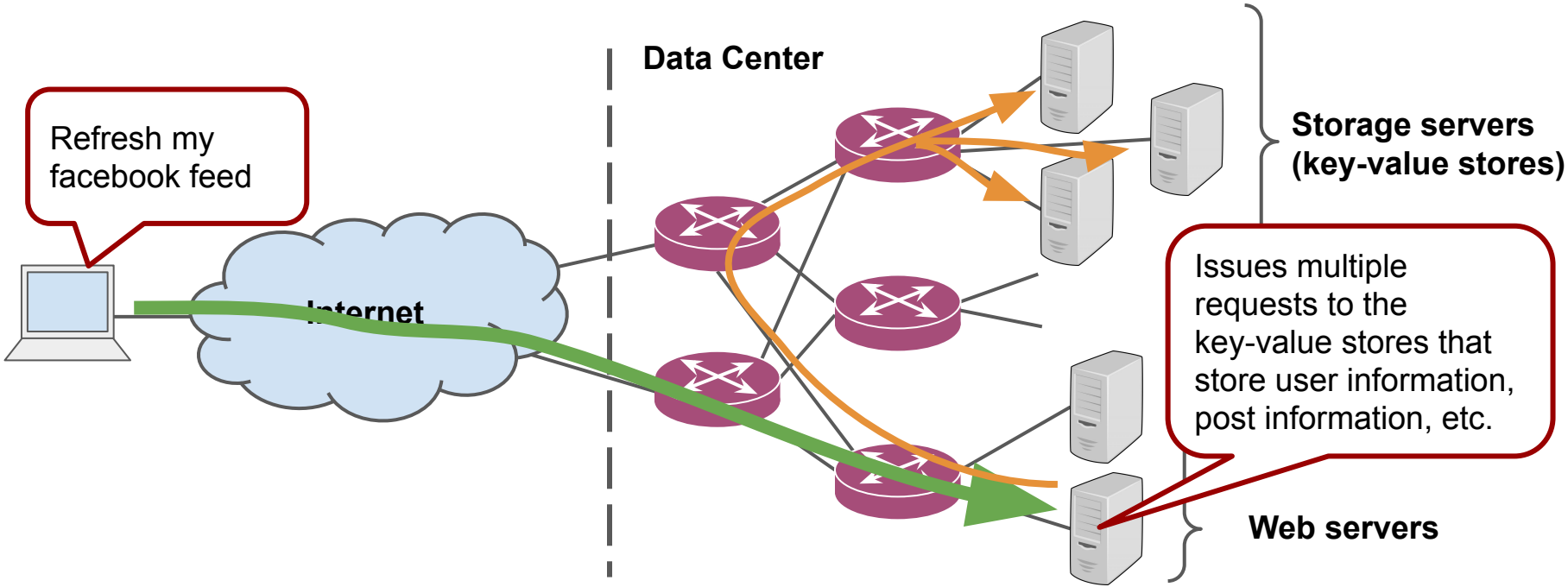
- Online services rely quite heavily on distributed key-value stores.



Example 1: In-network caching

- Key-value stores can get millions if not billions of requests every second.
- To handle such load, there are usually several storage servers, each taking care of part of the key-value store.
- Requests are load-balanced across storage servers.
- Problem?
    - Hot items change all the time
    - This can create load imbalance.
    - That is, one server (or a subset of them) can get overwhelmed and not be able to answer queries fast enough for good user quality of experience.

Example 1: In-network caching

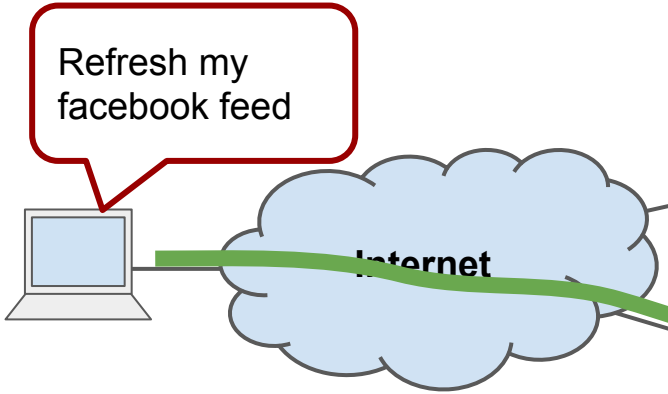


Example 1: In-network caching

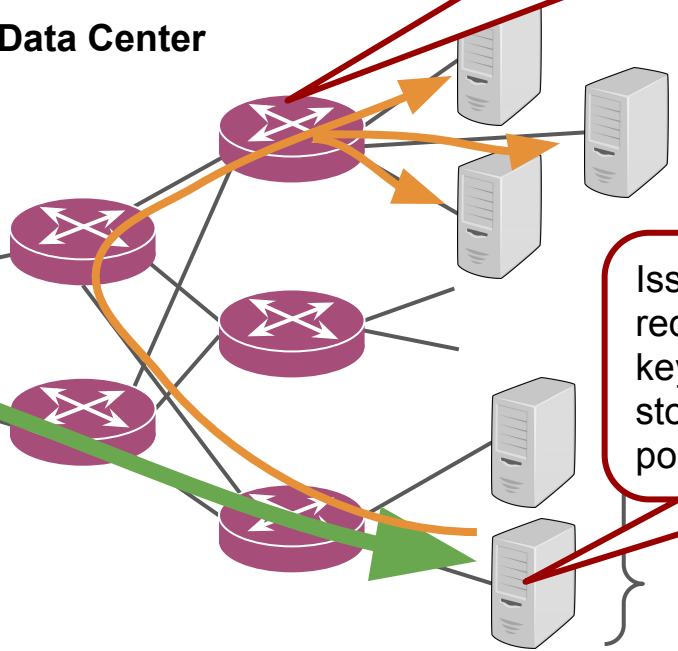
All the requests are going through the top of rack switch!

Can we store (i.e., cache) some of the "hot items" there?

Refresh my facebook feed



Data Center



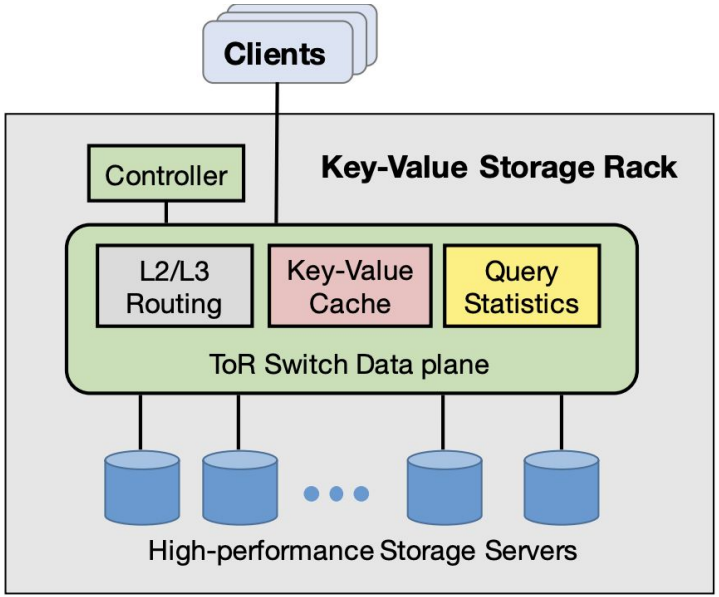
Storage servers (key-value stores)

Issues multiple requests to the key-value stores that store user information, post information, etc.

Web servers

Example 1: In-network caching

- NetCache (SOSP'17) proposes to do just that!



Example 1: In-network caching

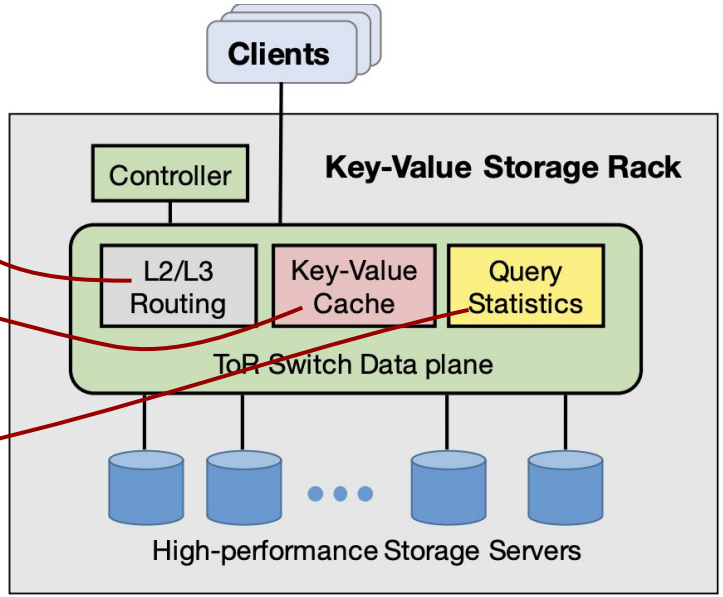
- NetCache (SOSP'17) proposes to do just that!

Regular switch functionality

Maintains "hot" items

Gather statistics about the queries.

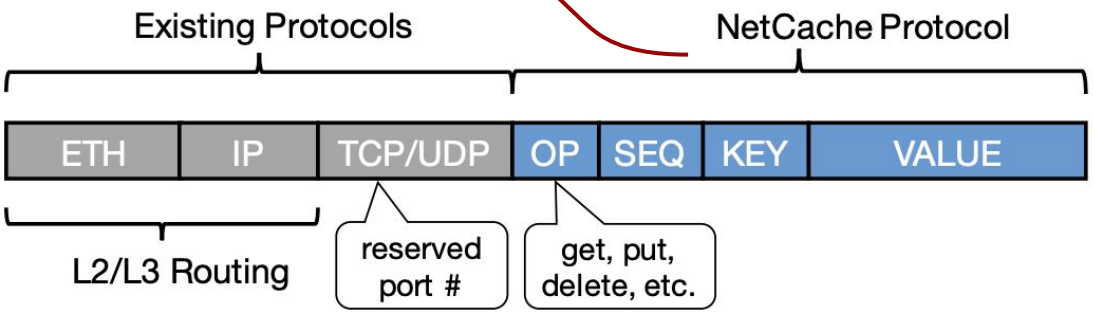
so the controller can update the cache as query patterns change.



Example 1: In-network caching

- NetCache (SOSP'17) proposes to do just that!

with a programmable parser, NetCache can define its own header.

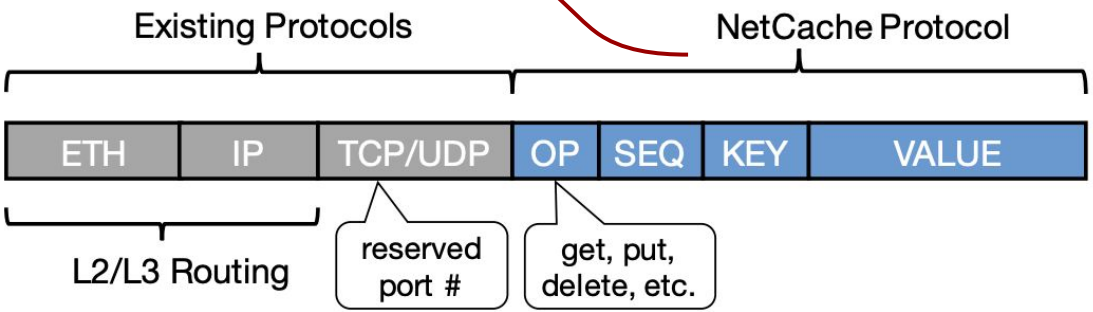


Example 1: In-network caching

- NetCache (SOSP'17) proposes to do just that!

with a programmable parser, NetCache can define its own header.

Applications are provided with a library that translates their requests to packets with NetCache headers.



Example 1: In-network caching

- NetCache (SOSP'17) proposes to do just that!

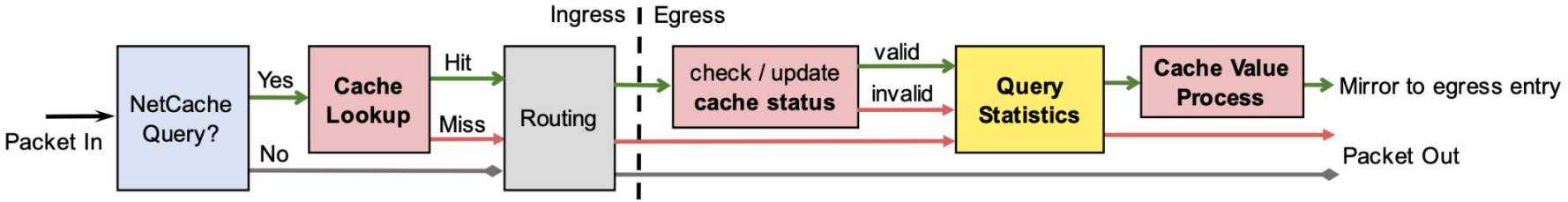


Figure 8: Logical view of NetCache switch data plane.

Example 2: In-network consensus

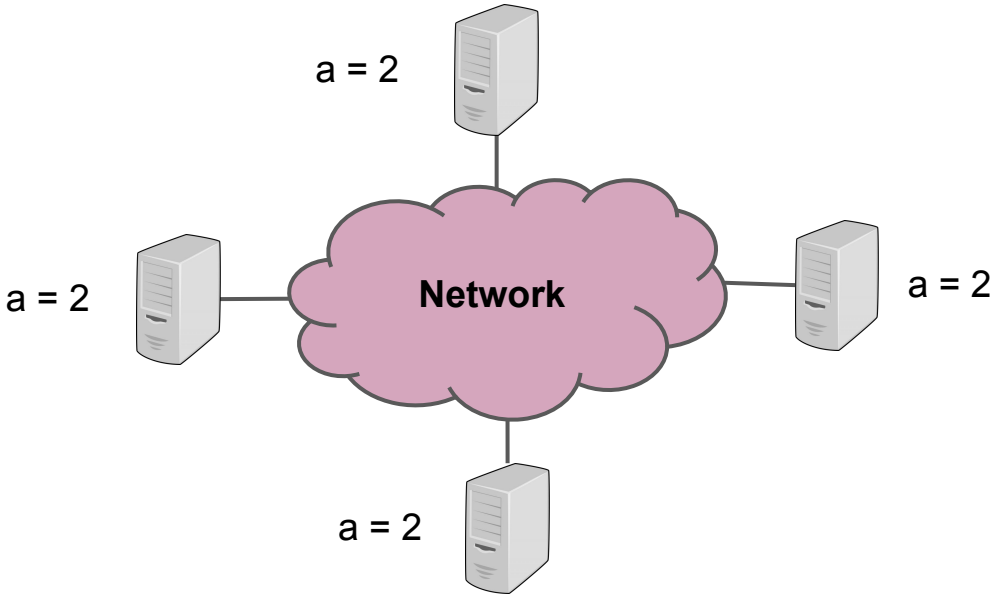
- What is consensus?
- You have a distributed set of participants .
  - e.g., servers keeping track of the store inventory
- You want all of them to agree on some values.
  - e.g., the total number of available trash cans to buy

Example 2: In-network consensus

- How is consensus/agreement usually implemented?

Each participant has its own view of the values of interest

Before any changes, participants communicate to make sure everyone is aware of the change.

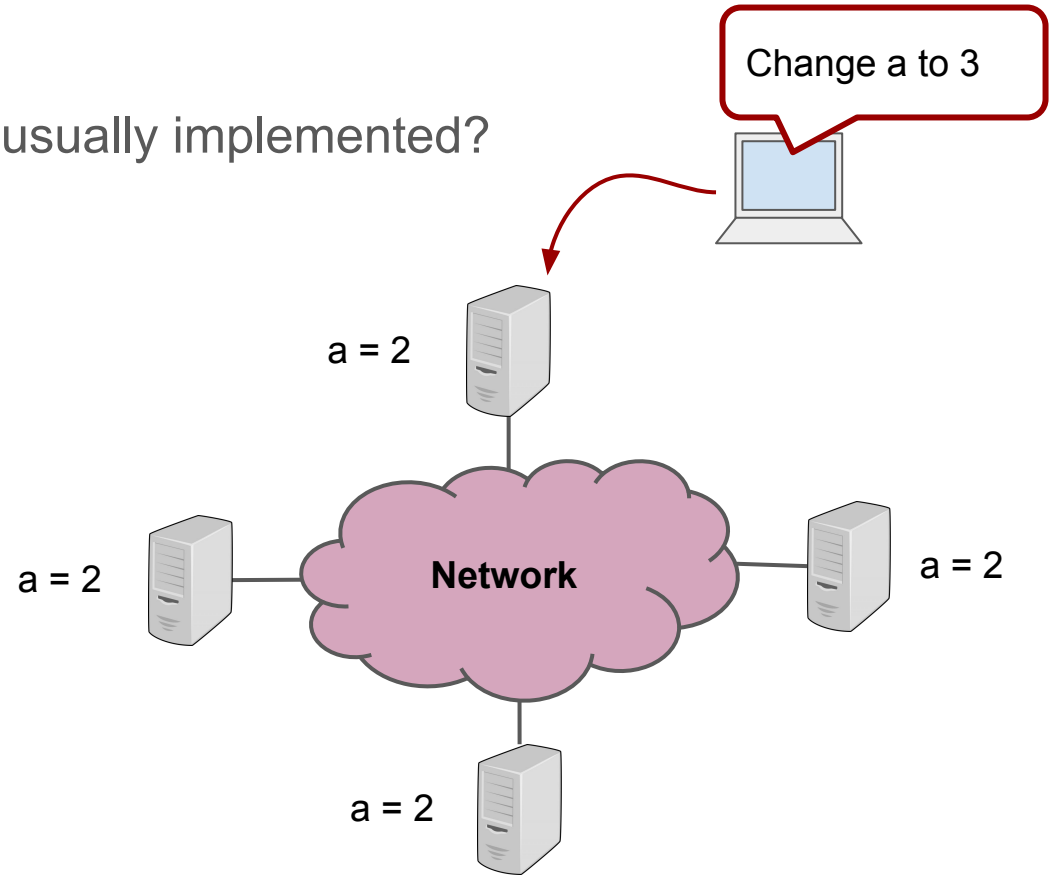


Example 2: In-network consensus

- How is consensus/agreement usually implemented?

Each participant has its own view of the values of interest

Before any changes, participants communicate to make sure everyone is aware of the change.

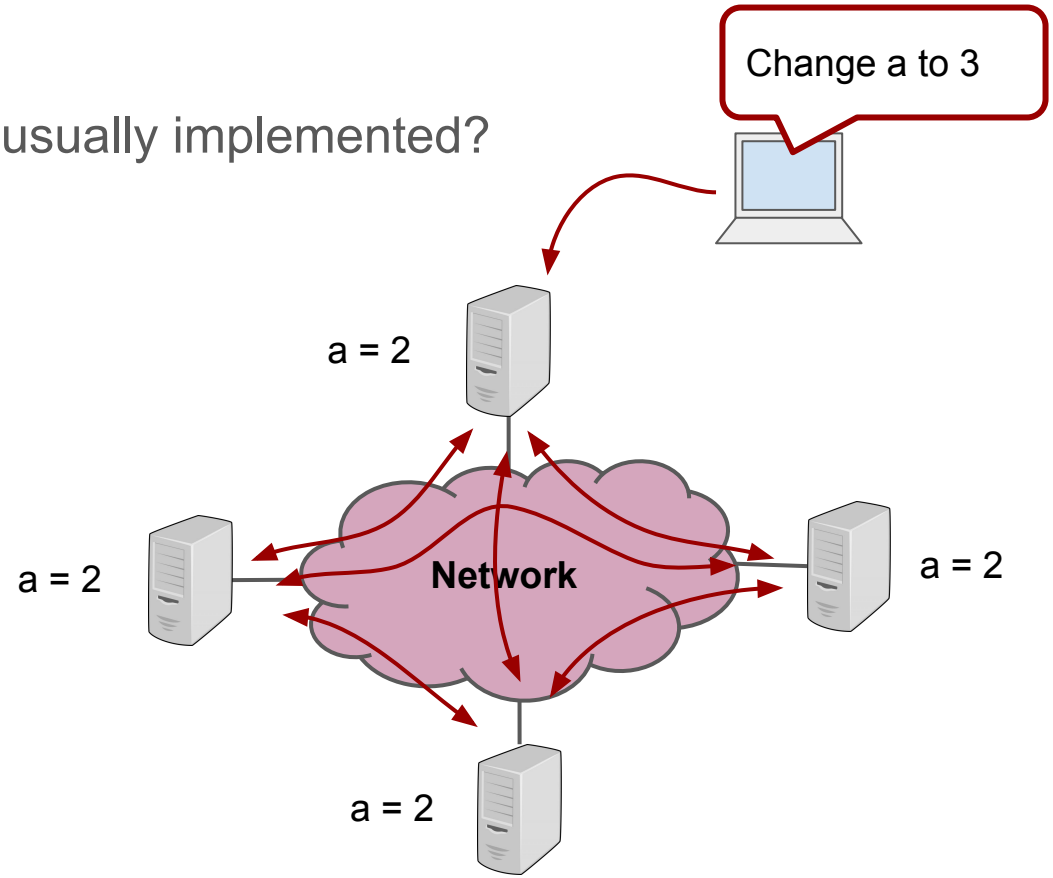


Example 2: In-network consensus

- How is consensus/agreement usually implemented?

Each participant has its own view of the values of interest

Before any changes, participants communicate to make sure everyone is aware of the change.

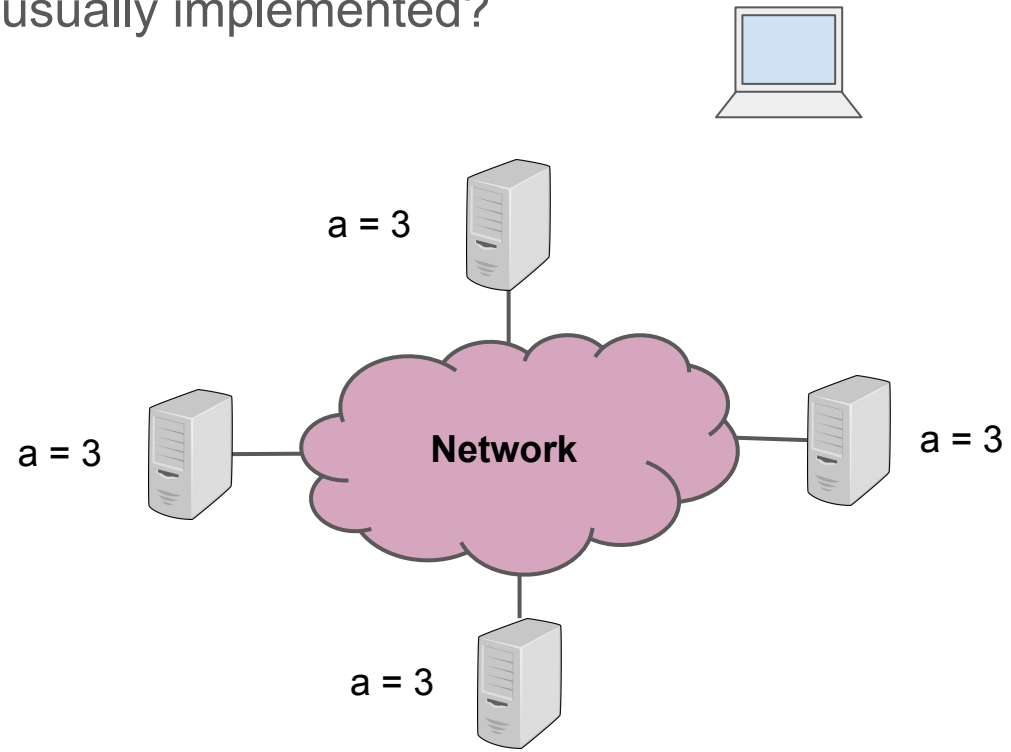


Example 2: In-network consensus

- How is consensus/agreement usually implemented?

Each participant has its own view of the values of interest

Before any changes, participants communicate to make sure everyone is aware of the change.

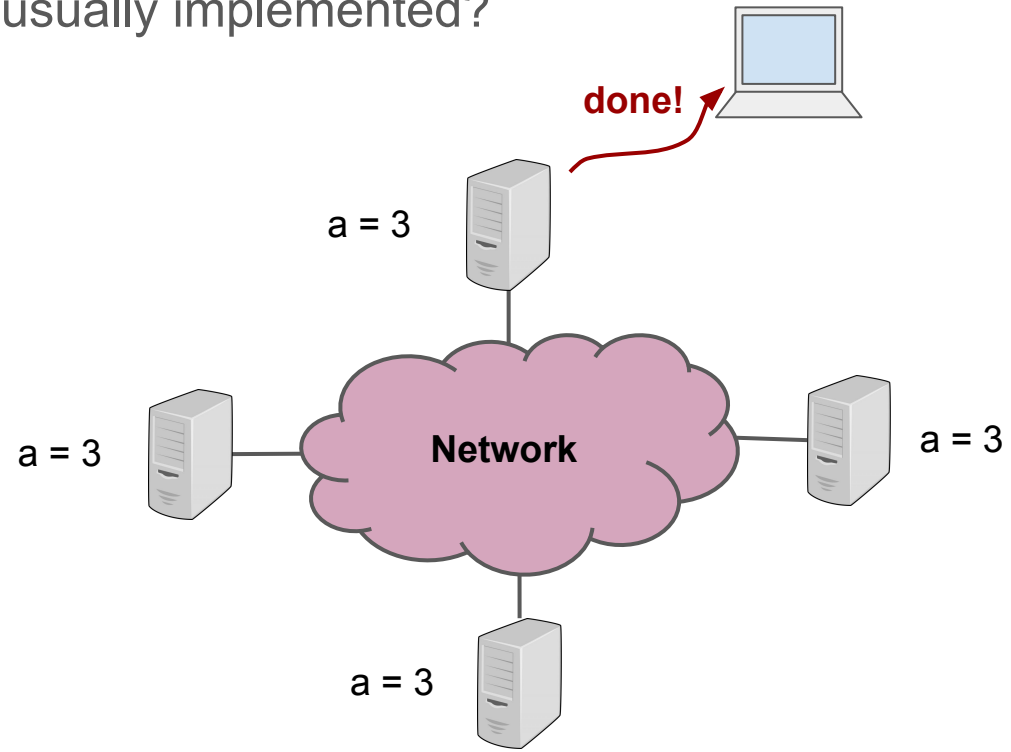


Example 2: In-network consensus

- How is consensus/agreement usually implemented?

Each participant has its own view of the values of interest

Before any changes, participants communicate to make sure everyone is aware of the change.



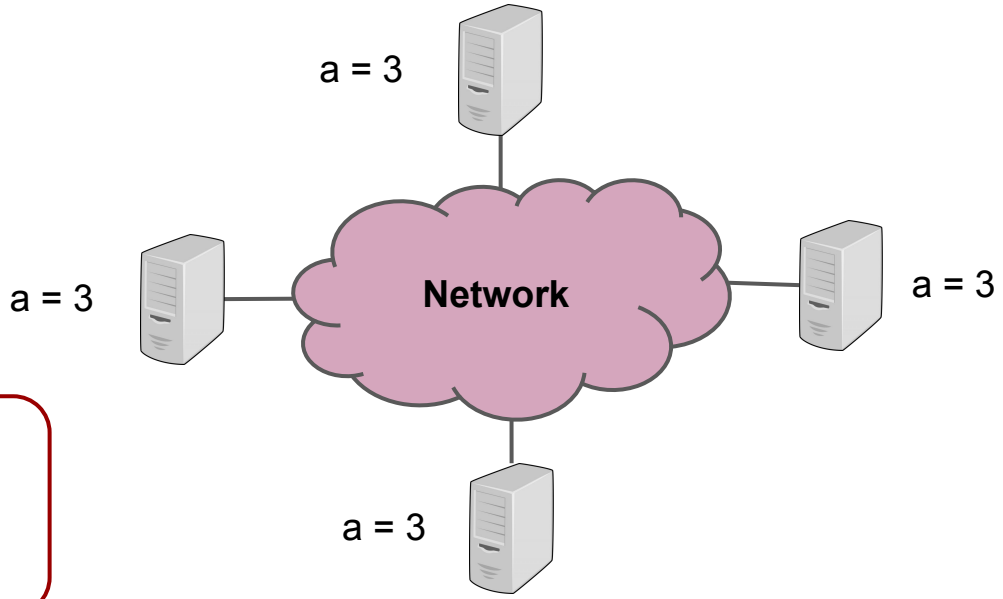
Example 2: In-network consensus

- How is consensus/agreement usually implemented?

Each participant has its own view of the values of interest

Before any changes, participants communicate to make sure everyone is aware of the change.

Paxos is a very famous and complex protocol that governs these communications to ensure consensus.



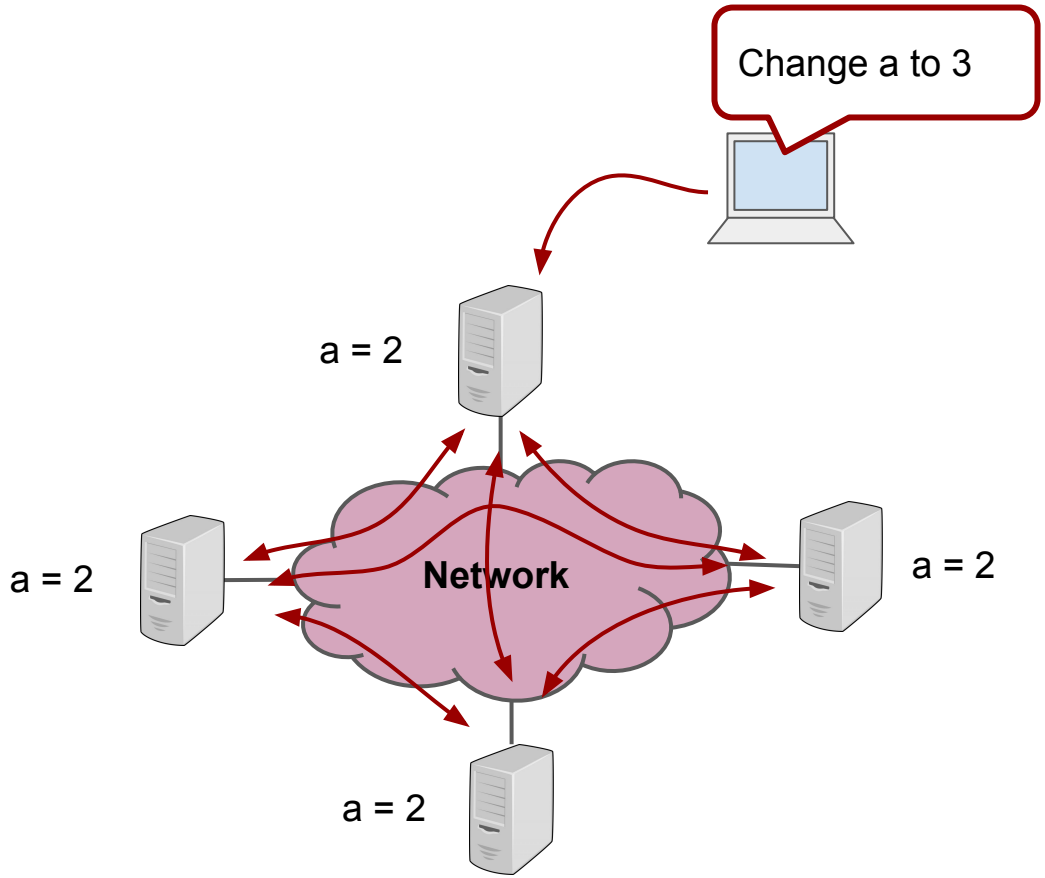
Example 2: In-network consensus

- Consensus is hard to implement efficiently.
  - Lots of communication to provide strong consistency.
- As such, it is typically only used for services that critically need such consistency.
- e.g., lock manager, configuration management, group membership
- Many distributed services depend on the above "coordination" services.
- And are bottlenecked by them…

Example 2: In-network consensus

Consensus is communication heavy

the actual computations done on each participant is quite simple.

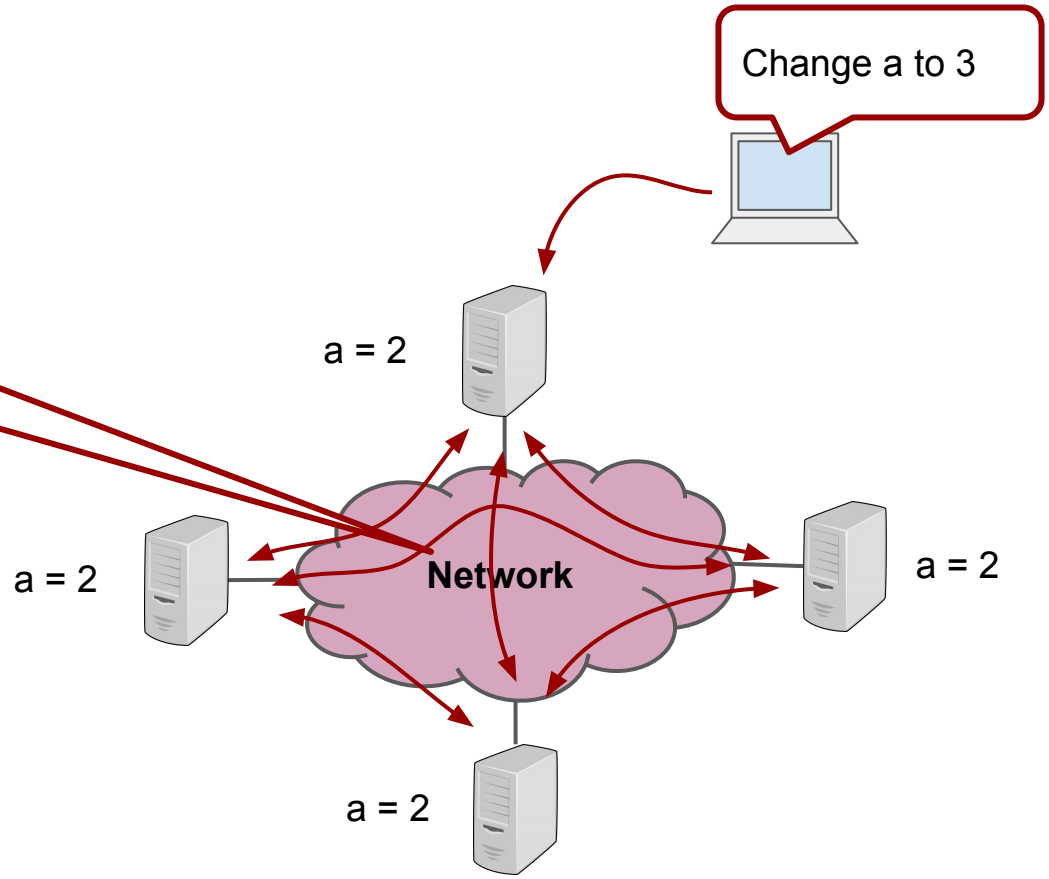


Example 2: In-network consensus

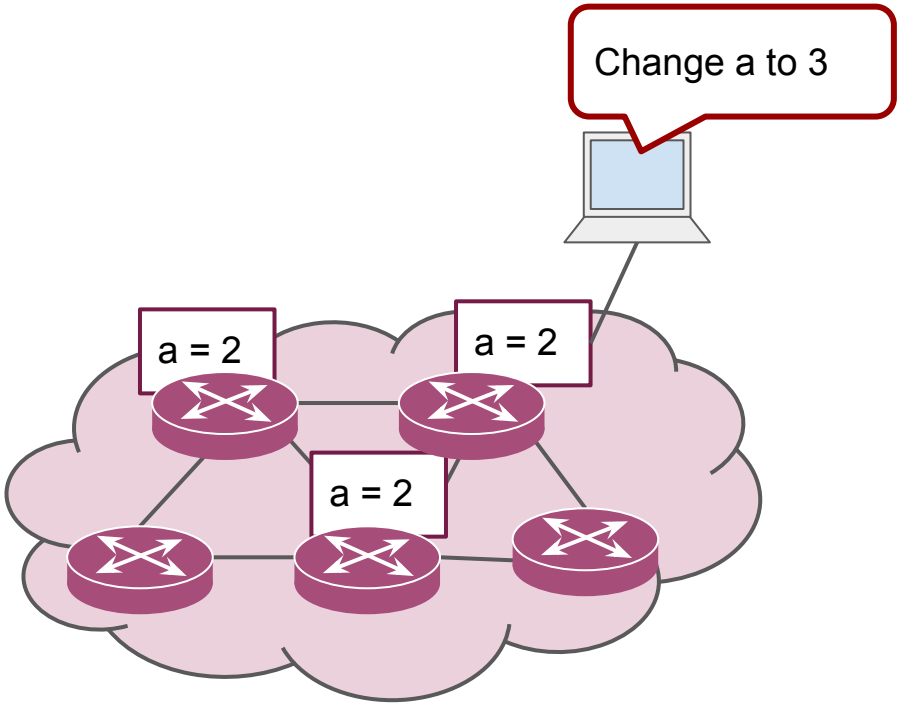
Can we implement it in the network?

Consensus is communication heavy

the actual computations done on each participant is quite simple.



Example 2: In-network consensus

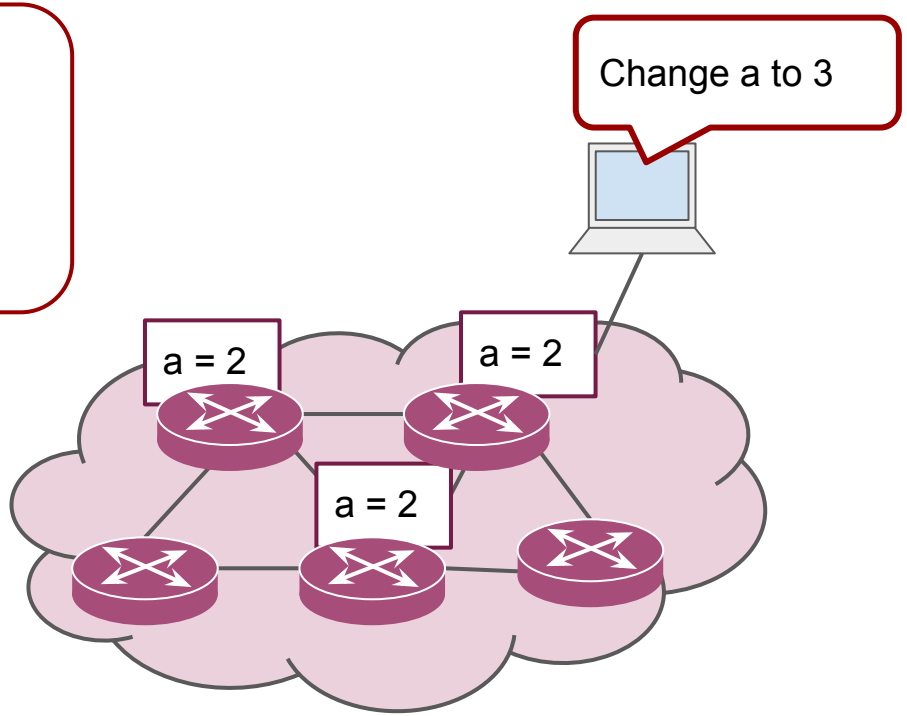


Example 2: In-network consensus

Switches keep all copies of the values.

Switches serve read and write requests.

Switches run the consensus (or coordination, or agreement) protocol.



Example 2: In-network consensus

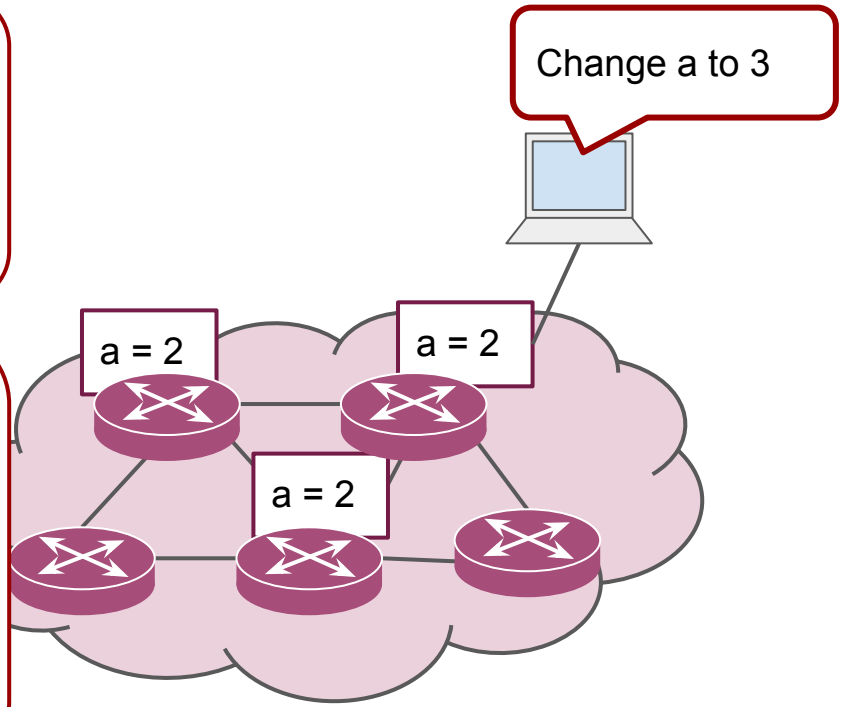
Switches keep all copies of the values.

Switches serve read and write requests.

Switches run the consensus (or coordination, or agreement) protocol.

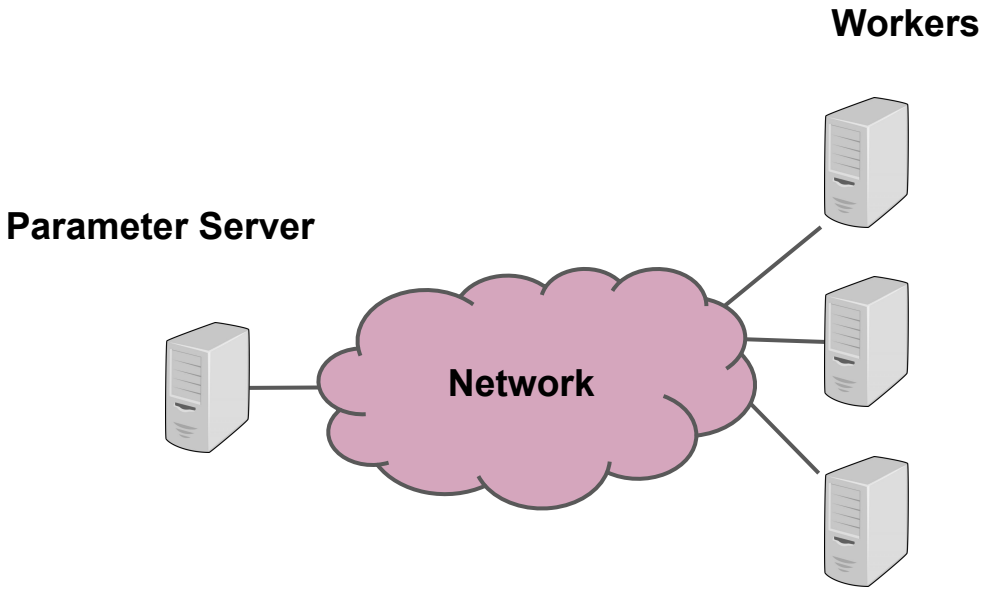
Benefits?

- Switches are faster than servers
- Communication between each pair of servers requires the traversal of multiple switches (multiple RTTs)
- Switches are "closer" to each other, so this can be done even in sub-RTT



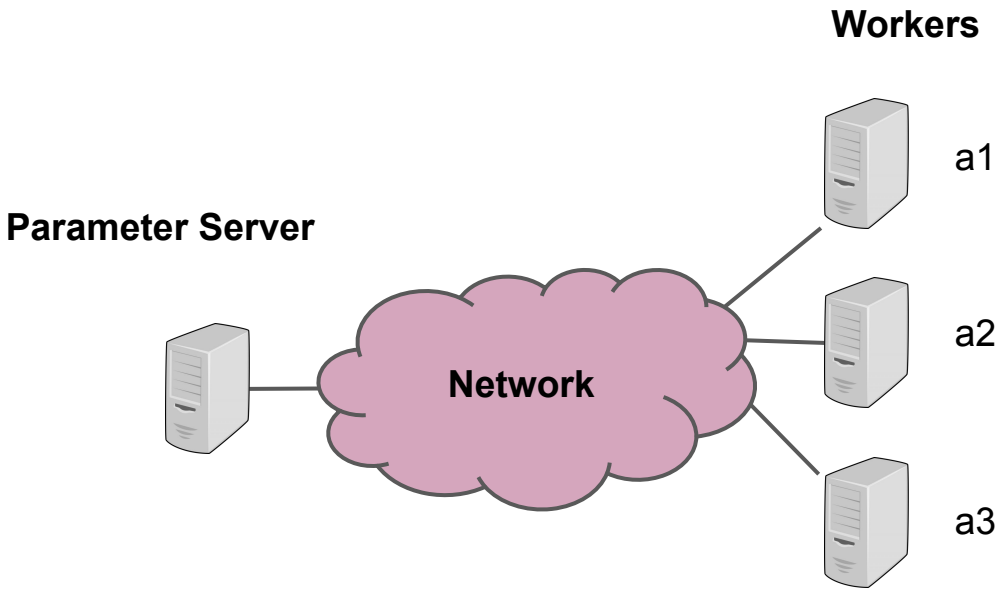
Example 3: Accelerating ML Training

- Distributed training of ML models can require a lot of network communication.



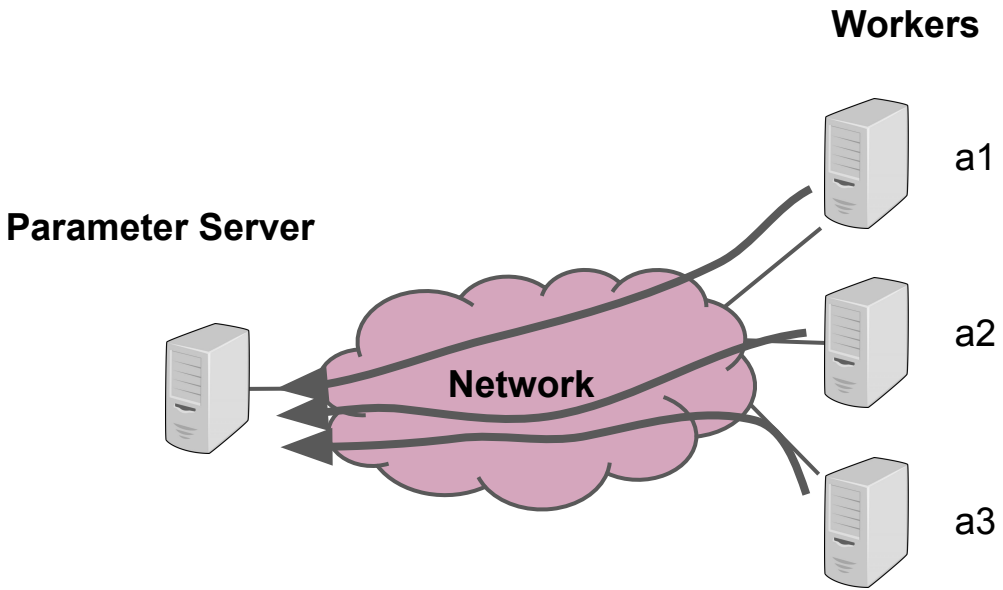
Example 3: Accelerating ML Training

- Distributed training of ML models can require a lot of network communication.



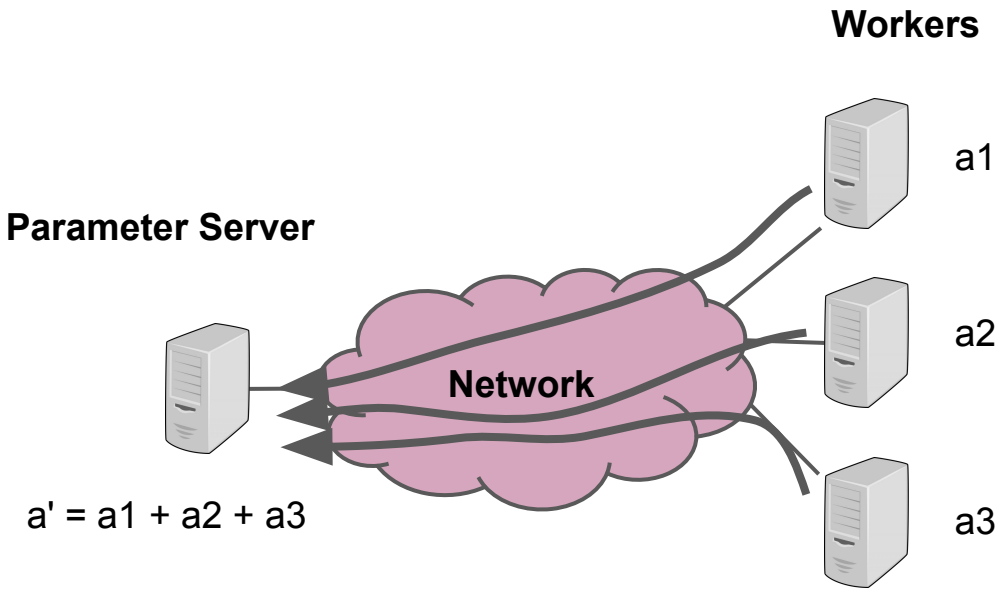
Example 3: Accelerating ML Training

- Distributed training of ML models can require a lot of network communication.



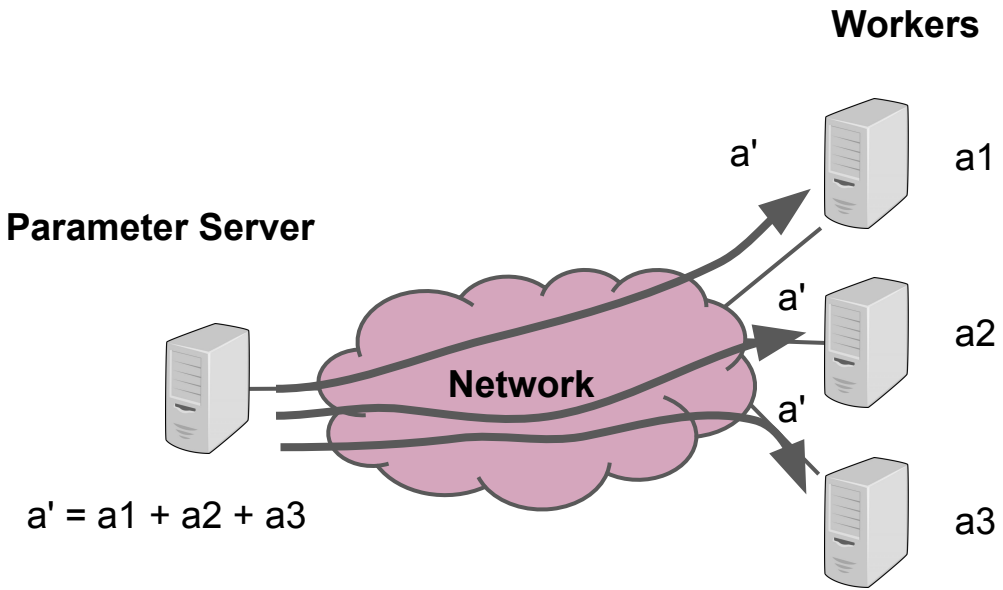
Example 3: Accelerating ML Training

- Distributed training of ML models can require a lot of network communication.



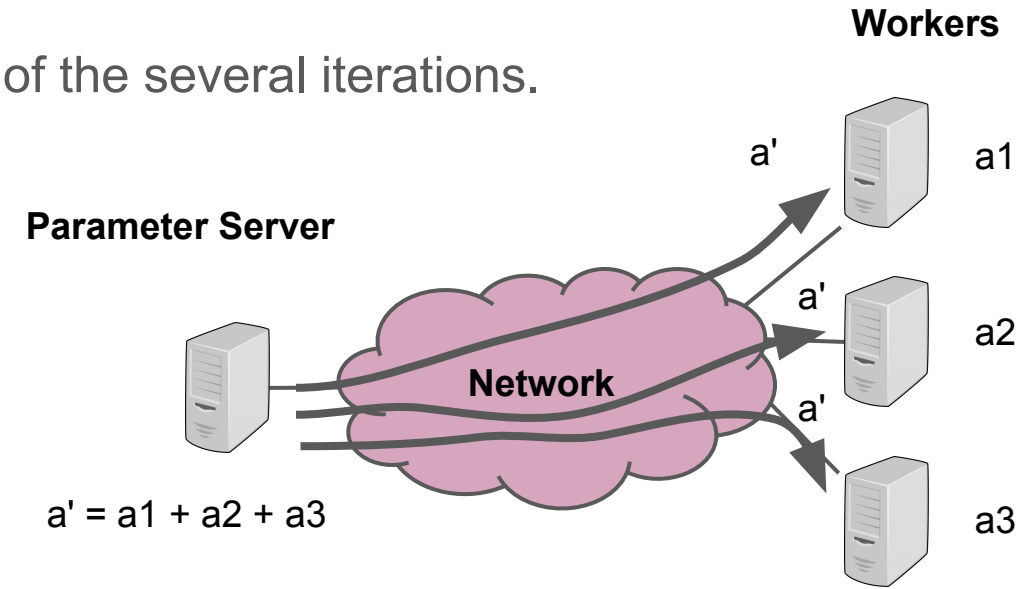
Example 3: Accelerating ML Training

- Distributed training of ML models can require a lot of network communication.

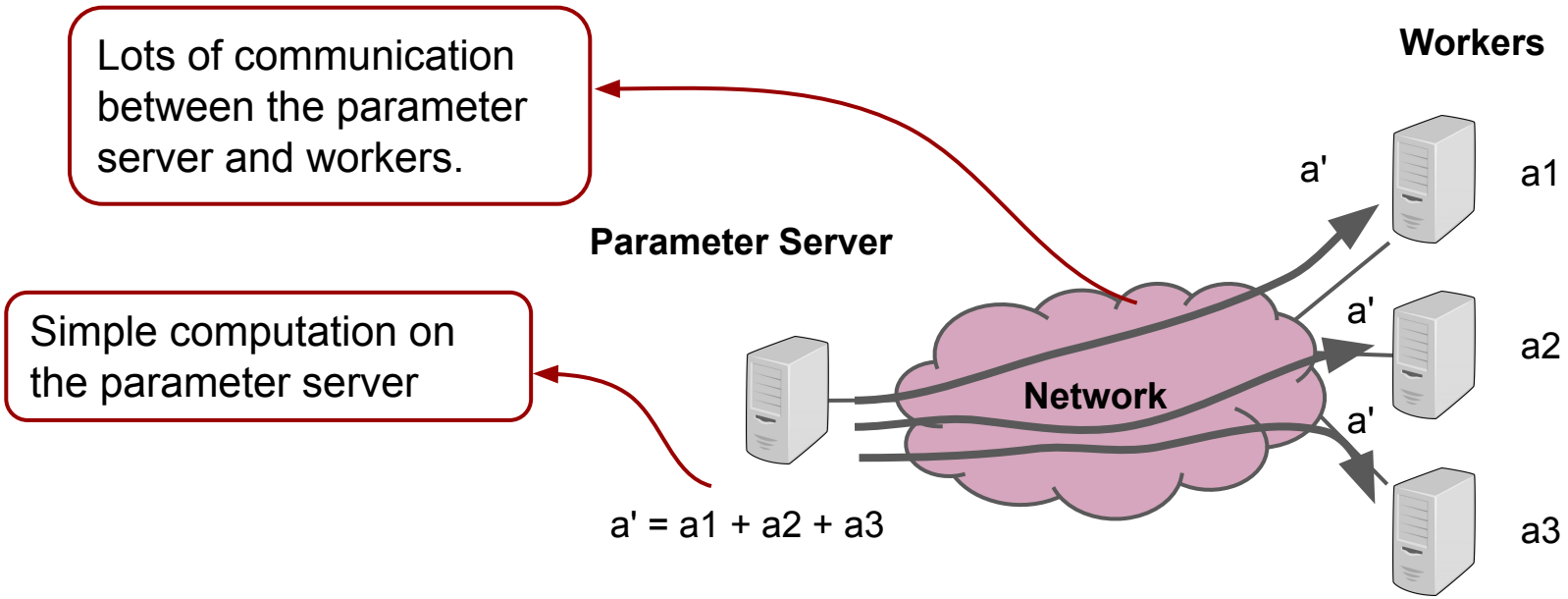


Example 3: Accelerating ML Training

- Distributed training of ML models can require a lot of network communication.
- This happens in every of the several iterations.



Example 3: Accelerating ML Training

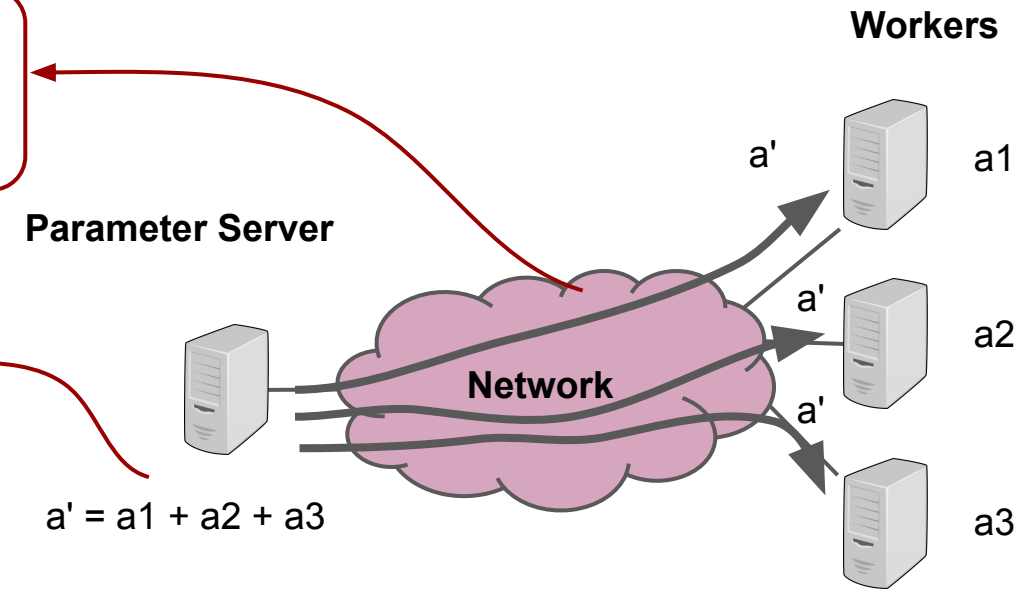


Example 3: Accelerating ML Training

Lots of communication between the parameter server and workers.

Simple computation on the parameter server

Familiar pattern?



Example 3: Accelerating ML Training

Implement the parameter server in network switches

- The switch can keep track of the sum (aggregate) in a register.
- As packets come from the workers, it can retrieve values from packets and update the sum.
- Once the switch receives values from all workers, it can send the sum back to the workers.
- Benefits? Same as before
    - Higher throughput and lower communication latency

Challenges of in-network computing

- What if the information we need from the applications spans multiple packets?
  - e.g., in Netcache, what if the value for a key-value pair doesn't fit into one packet?
- It is difficult to reconstruct a stream in the switch
  - reconstruct = put together packet contents from multiple packets

Challenges of in-network computing

- Application logic is typically stateful.
- Switches have limited memory, and only allow limited access to it
- Application logic can be more complex than network processing
- Switches have limited computational capabilities.

Challenges of in-network computing

- You can see these constraints play out in current applications of in-network computing
  - NetCache caches hot items with small-ish values.
  - Coordination services don't store a lot of data
  - same as ML training parameter aggregation
  - In all cases, computation is quite simple.
- There have been proposals for switches with computational resources and capabilities that are more suited for application acceleration
  - e.g., Trio, or Tofino + FPGA

Challenges of in-network computing

- What should the API be for the applications?
- Suppose you are writing a distributed/networked application.
- How should you specify which part should be "offloaded" and executed in the network?

Challenges of in-network computing

- There is a higher abstraction bar here for programming abstractions.
- If someone is implementing a new network protocol, you can assume they have networking knowledge.
- We don't want application developers to have to learn all the details about network processing (packets, headers, protocols, etc.) to be able to accelerate their application.
- There are recent proposals that try to extend familiar programming abstractions like connections and RPCs for this purpose.

Paper: ATP: In-network Aggregation for Multi-tenant Learning

- Provides a framework for accelerating ML training by performing the aggregation in the network.
- Address many challenges of doing so at large scale:
  - Multiple training jobs running simultaneously.
  - Aggregation across multiple racks, i.e., over multiple switches, when workers and parameter servers are scattered across multiple racks.
  - Handling packet loss and congestion control
  - …

Additional Resources

- When Should The Network Be The Computer? (HotOS'19)
- In-network caching: NetCache
- In-network consensus: NetChain, NetLock, P4xos.
- ML acceleration: ATP, Trio
- Programming interfaces/abstractions: NetRPC, NCL, Bertha