UNIVERSITY OF WATERLOO

CS 856: Programmable Networks

Lecture 10: In-Network Computing

Mina Tahmasbi Arashloo

Winter 2023

Logistics

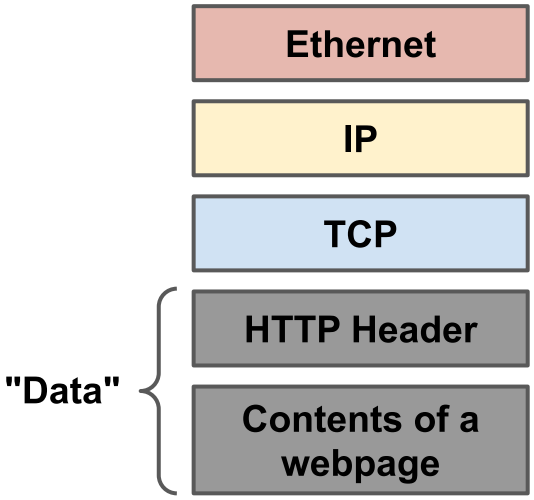
- Project presentations, April 4 and April 6
  - 20-minute presentation + 5 min Q&A
- Final project report, April 10
  - Will send template on Slack
- Final review (😊) due **Monday, March 27th, at 5pm**

Using network programmability to improve the network

- So far, we have mostly discussed how network programmability can help improve networks themselves.
    - 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
    - …

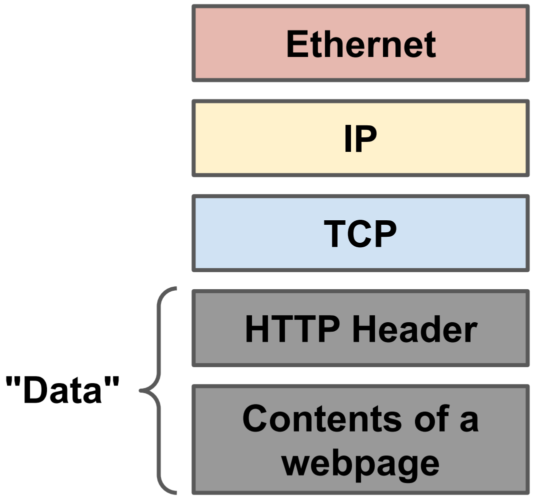
Using network programmability to accelerate applications?

- 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.

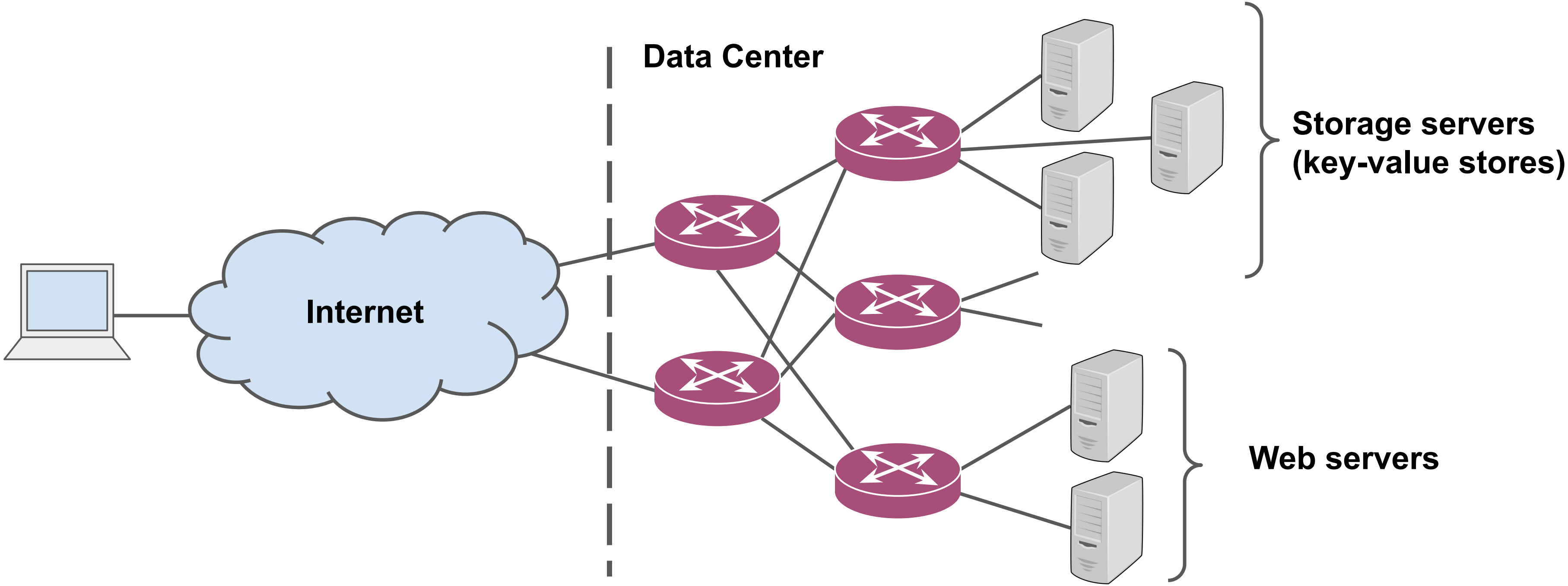


In-Network Computing

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

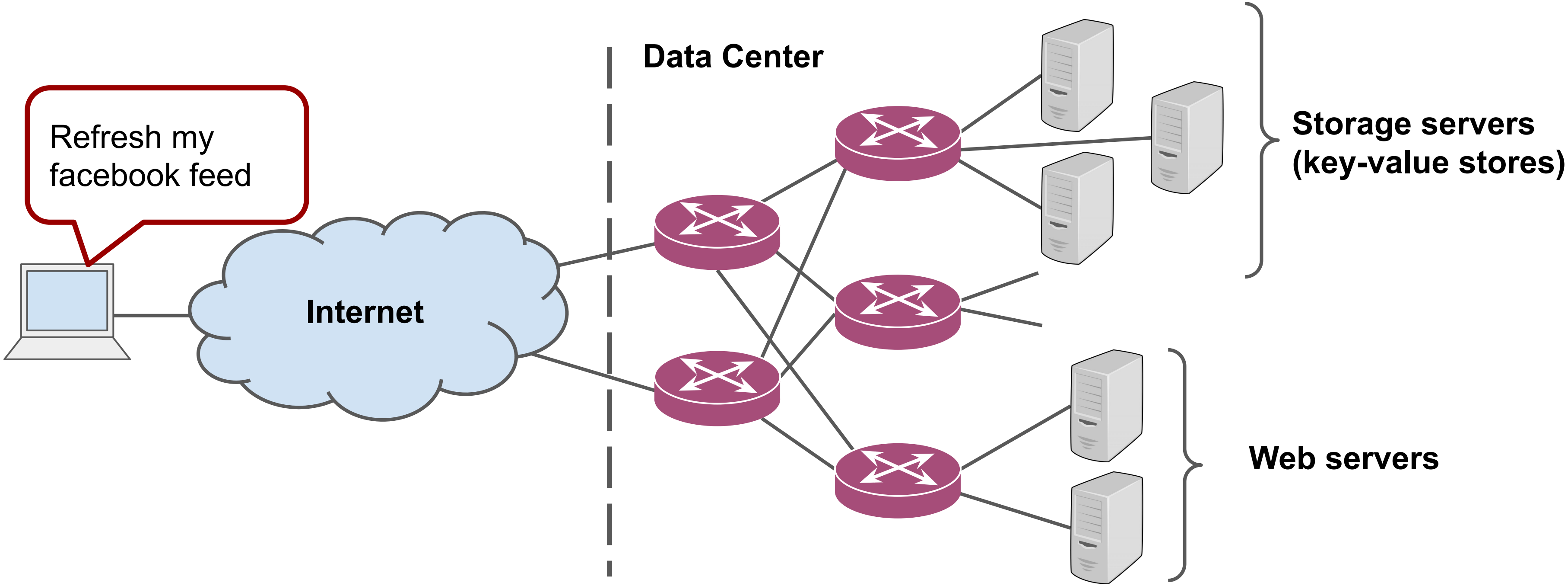
Example 1: In-network caching

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



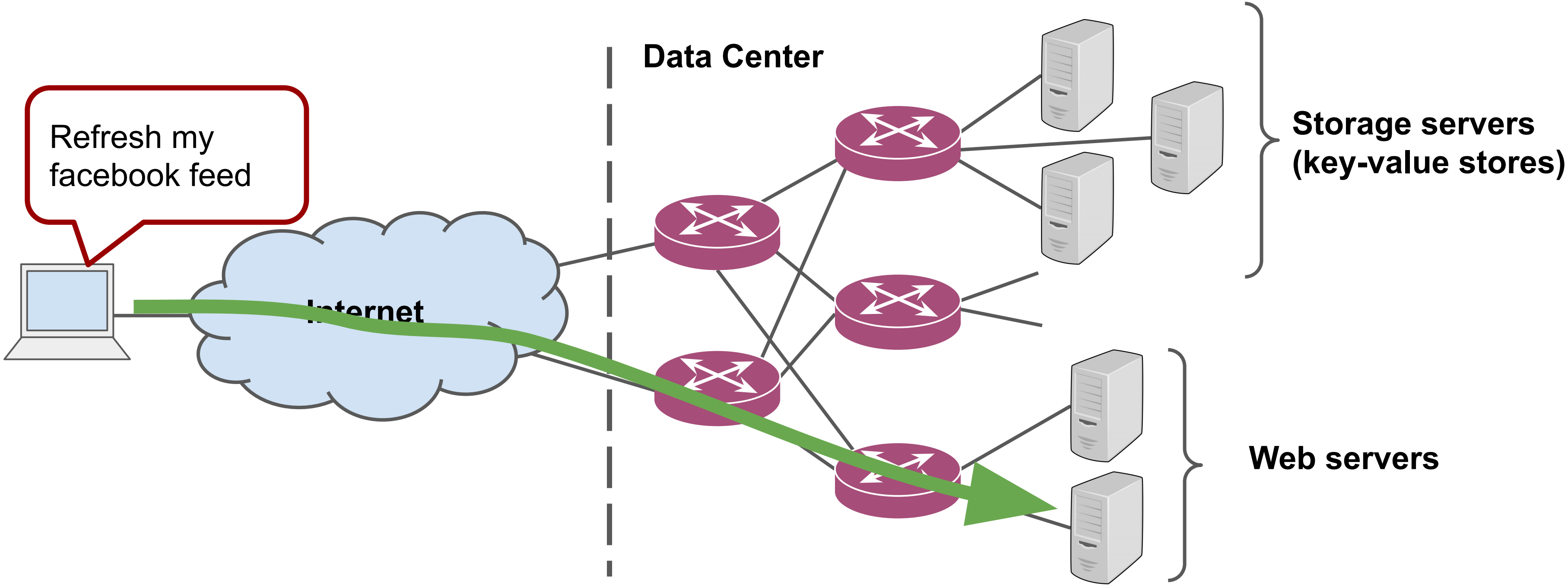
Example 1: In-network caching

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



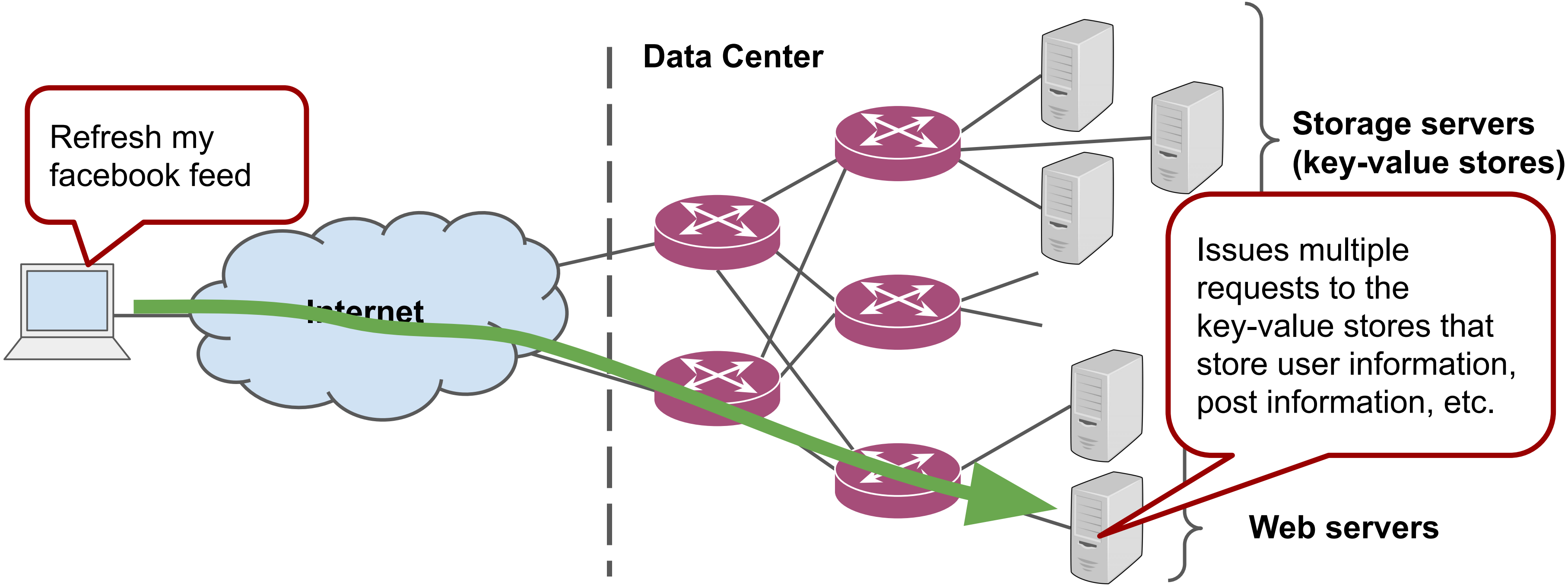
Example 1: In-network caching

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



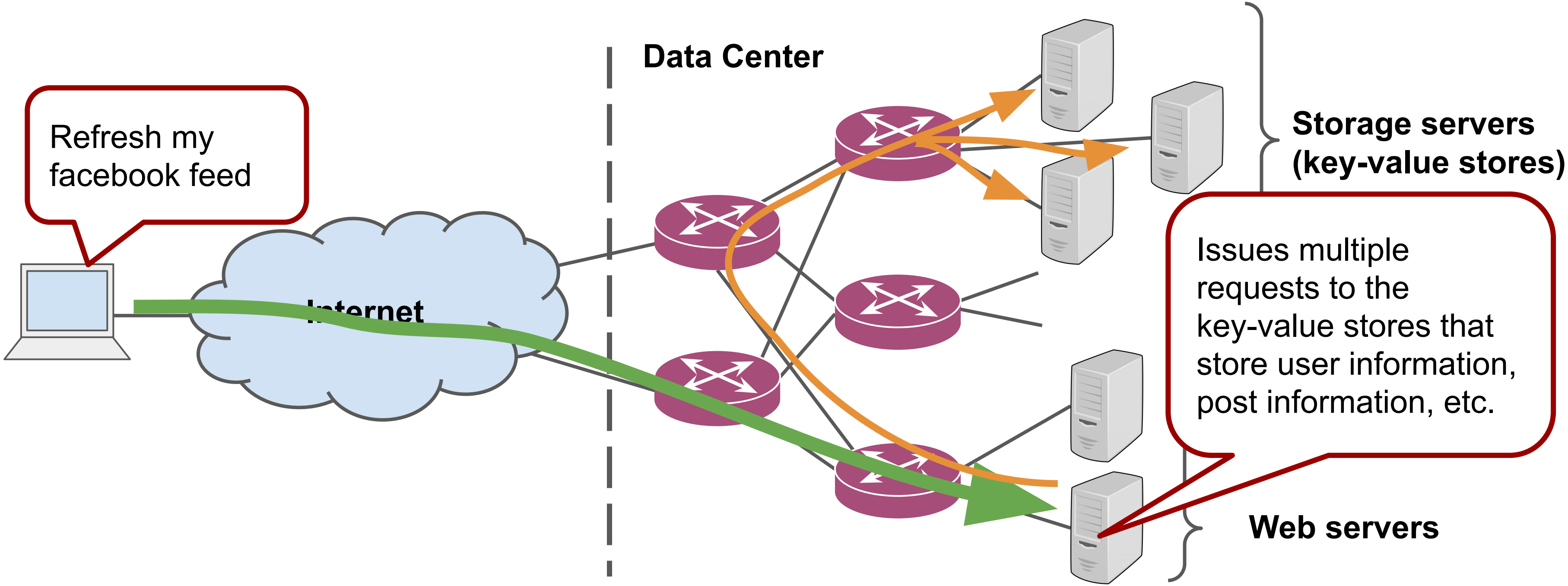
Example 1: In-network caching

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



Example 1: In-network caching

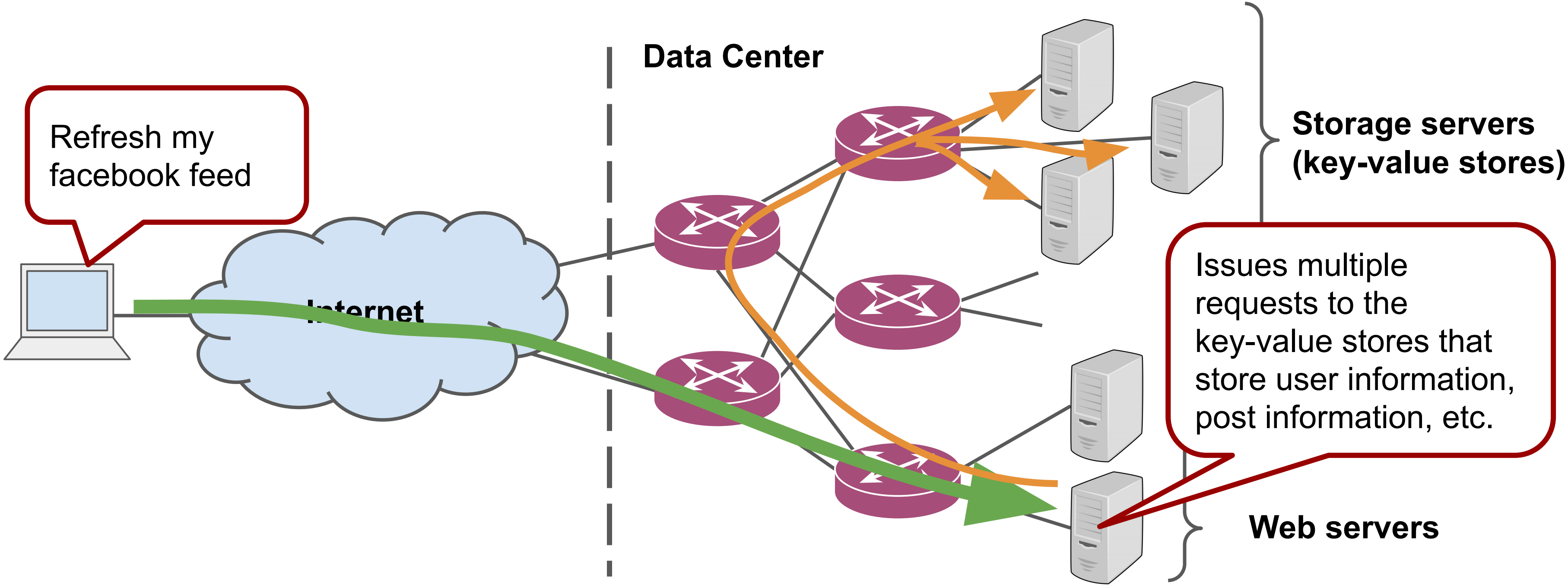
- Online services rely quite heavily on distributed key-value stores.



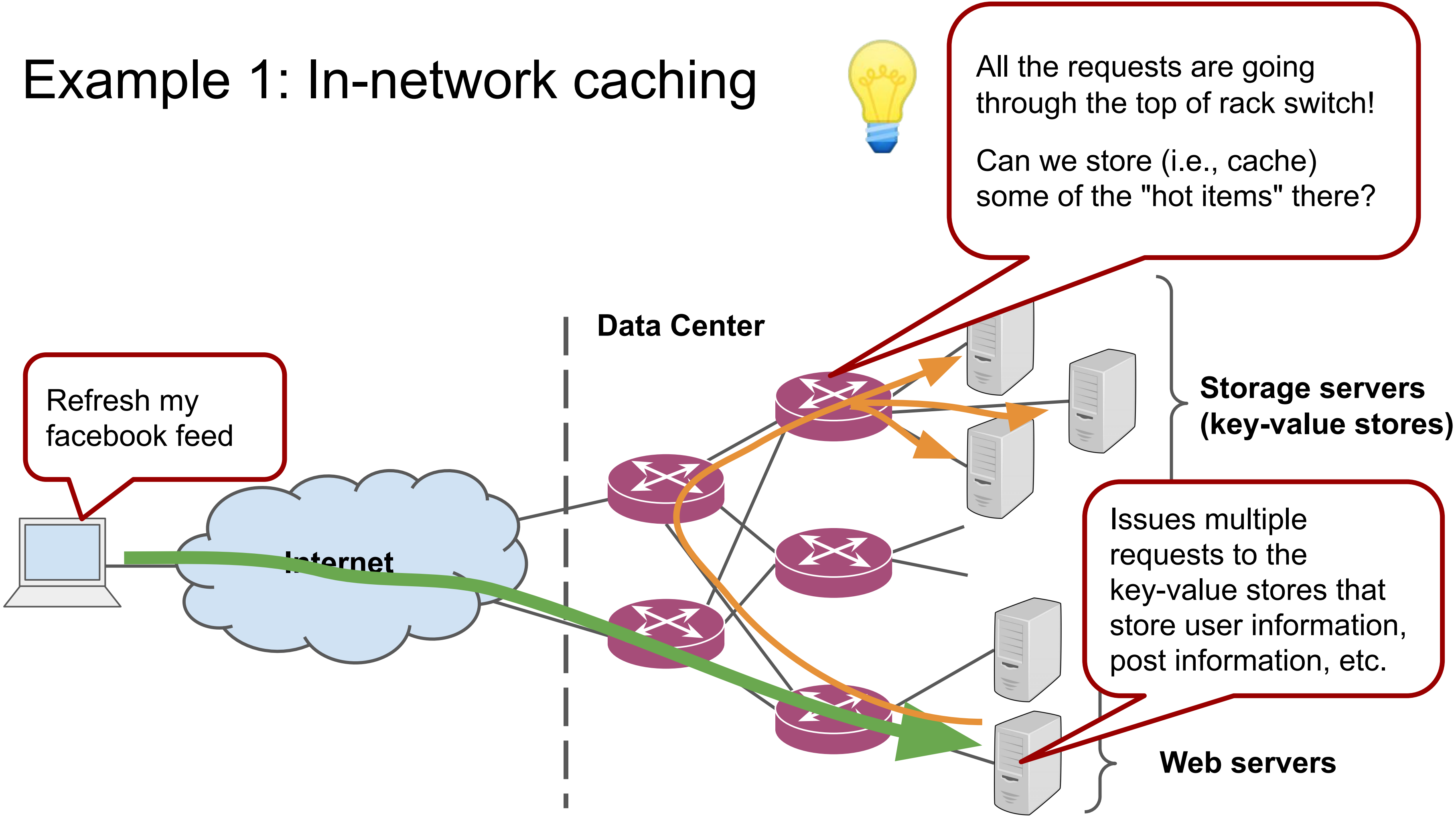
Example 1: In-network caching

- 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

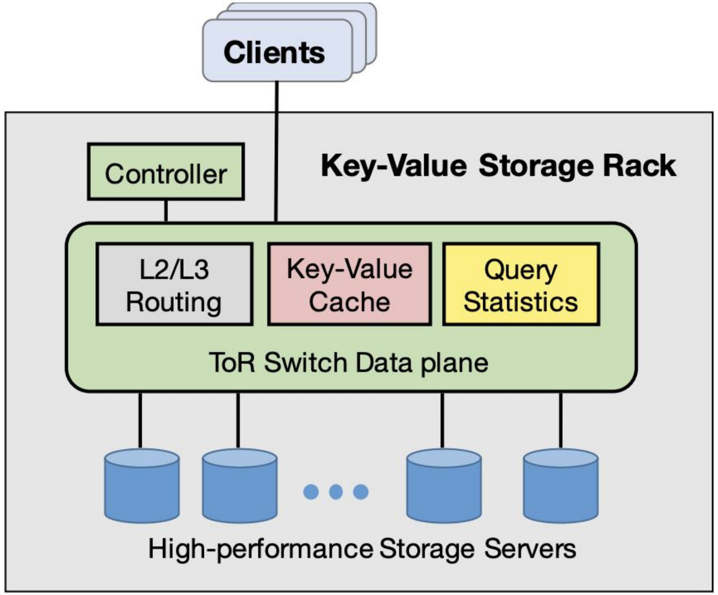


Example 1: In-network caching



Example 1: In-network caching

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



Example 1: In-network caching

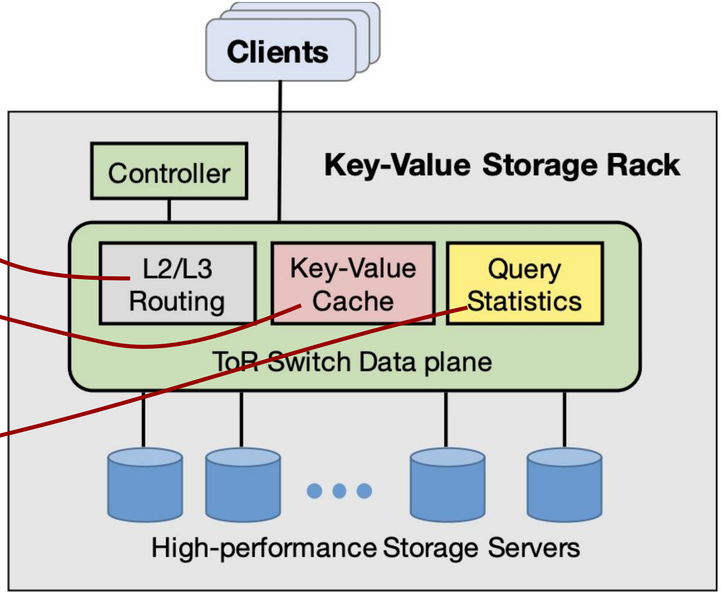
- NetCache (SOSP'17) proposes to do just that!

Regular switch functionality

Maintains "hot" items

Gather statistics about the queries.

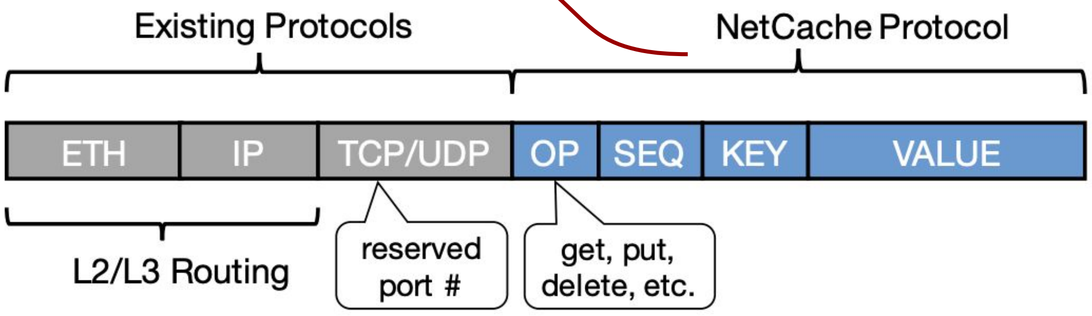
so the controller can update the cache as query patterns change.



Example 1: In-network caching

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

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

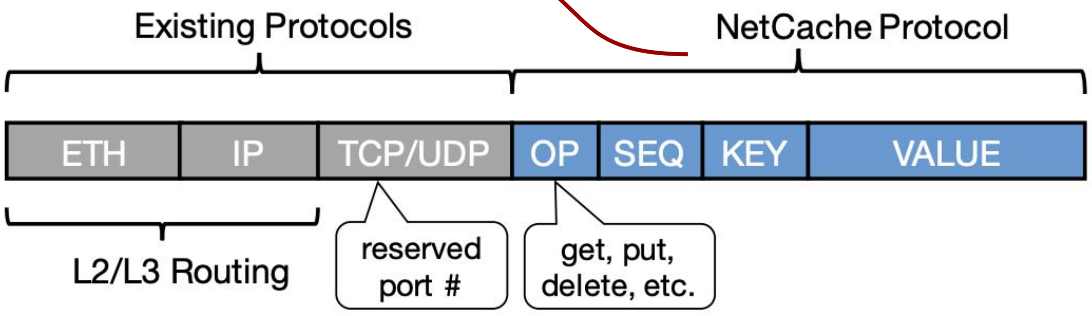


Example 1: In-network caching

- 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!

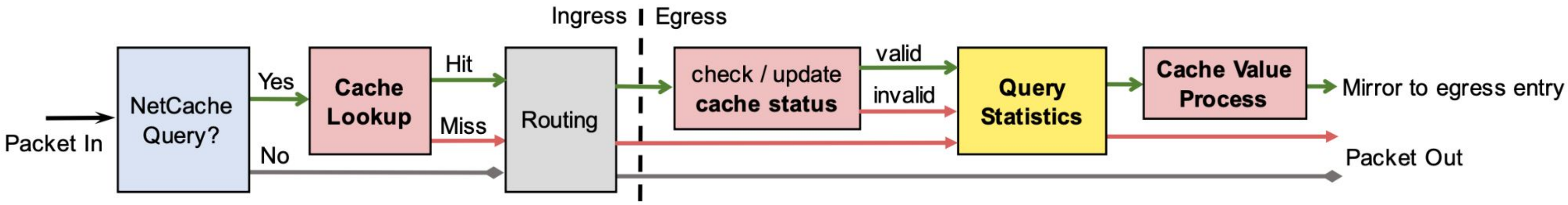


Figure 8: Logical view of NetCache switch data plane.

Example 2: In-network consensus

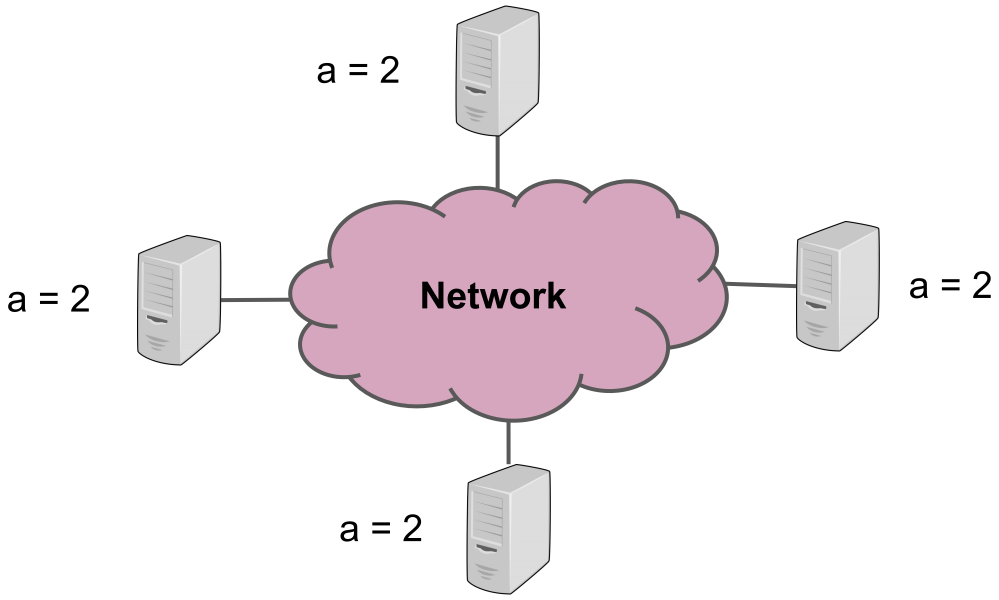
- What is consensus?
- You have a distributed set of participants .
  - e.g., servers keeping track of the store inventory
- You want all of them to agree on some values.
  - e.g., the total number of available trash cans to buy

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.

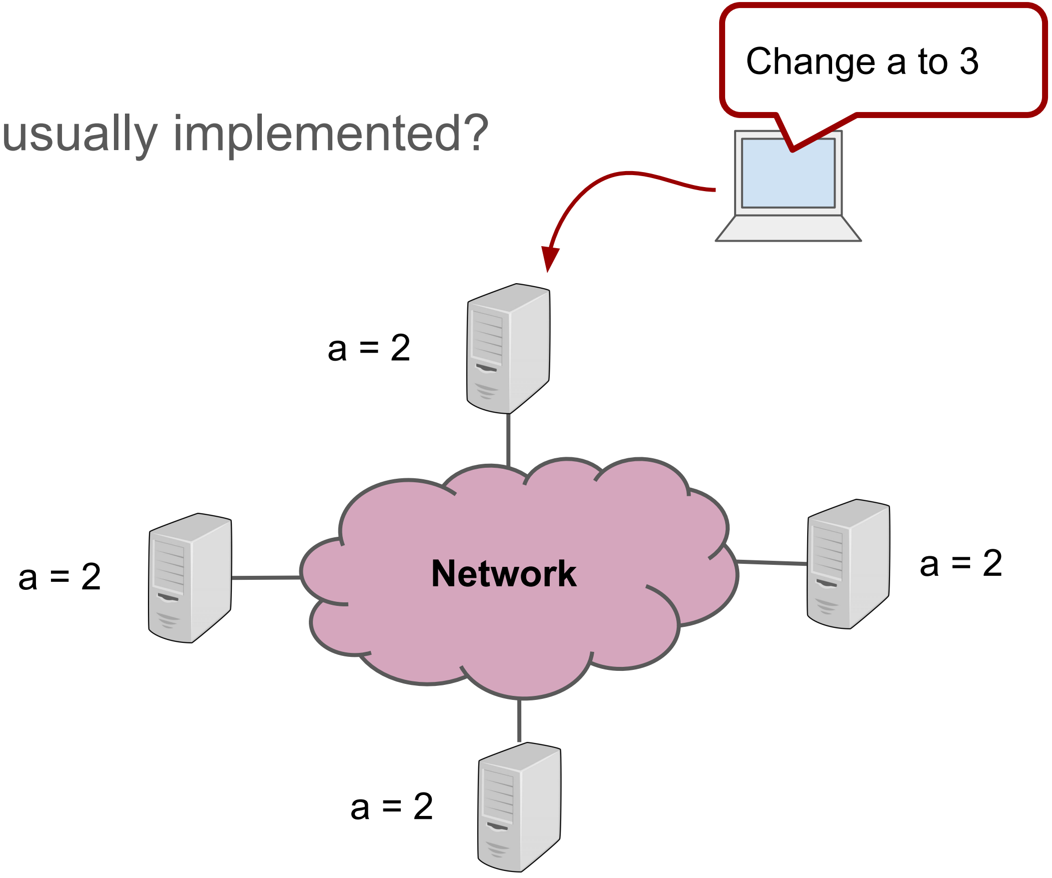


Example 2: In-network consensus

- 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.

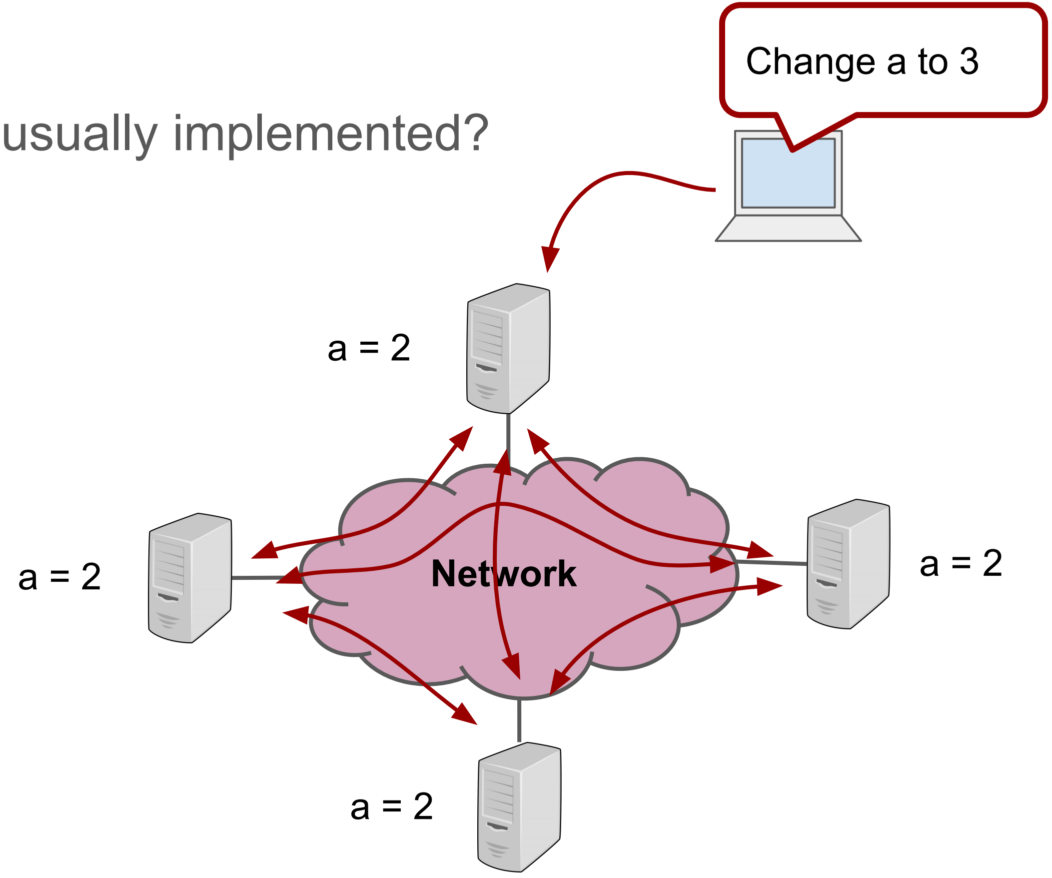


Example 2: In-network consensus

- 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.

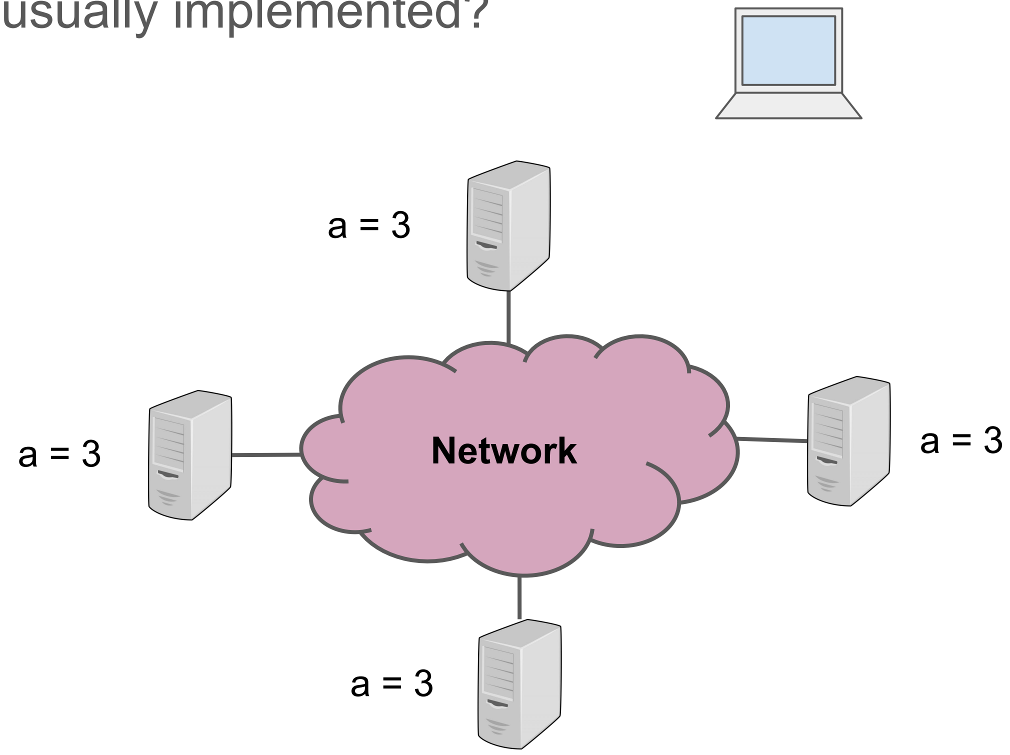


Example 2: In-network consensus

- 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.

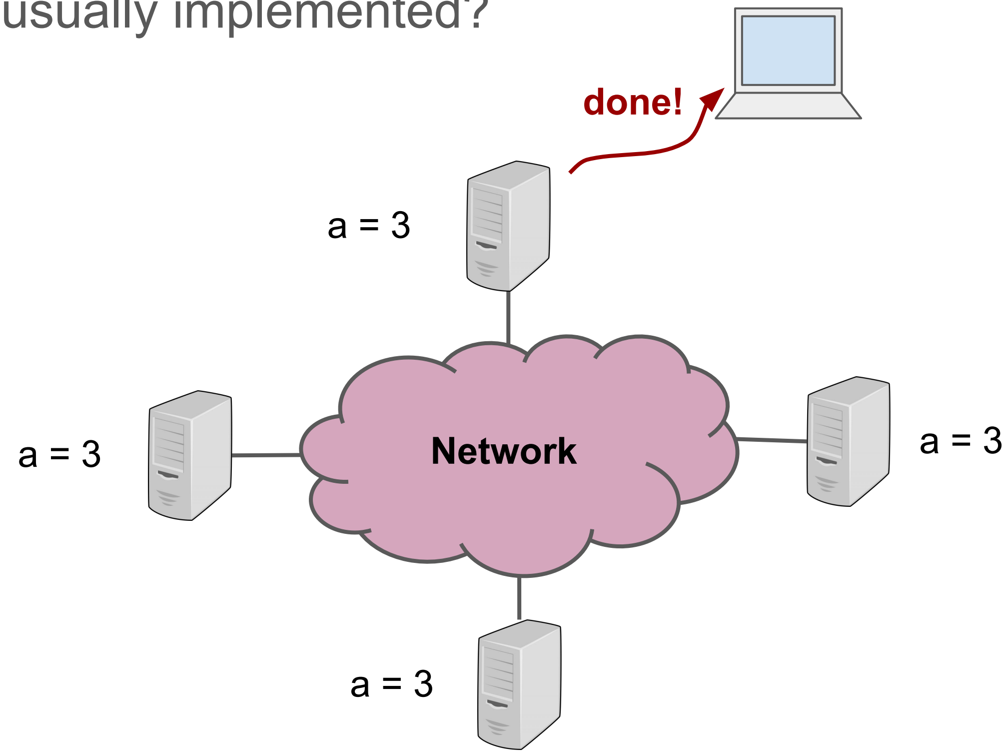


Example 2: In-network consensus

- 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.



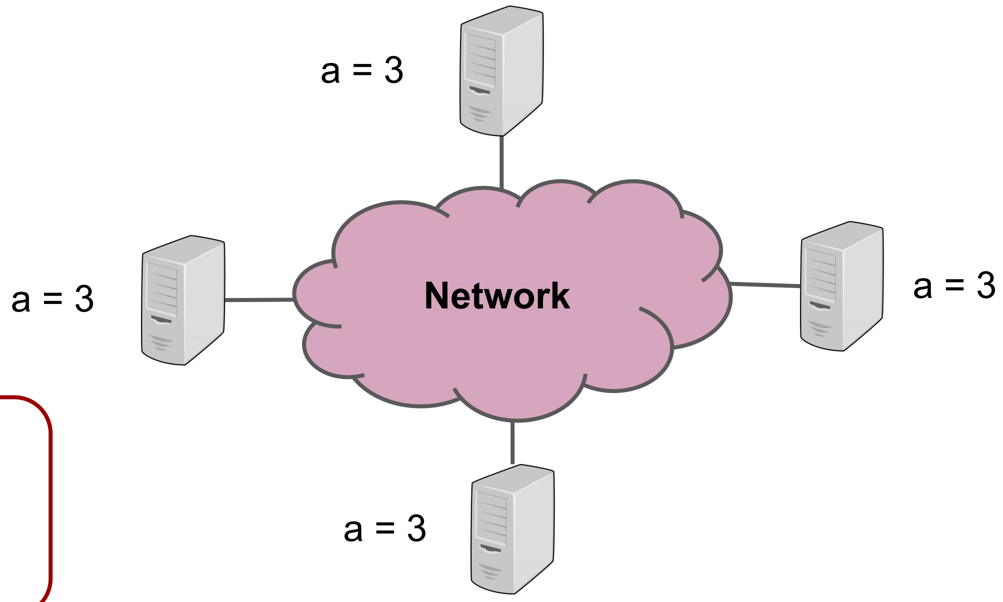
Example 2: In-network consensus

- 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.

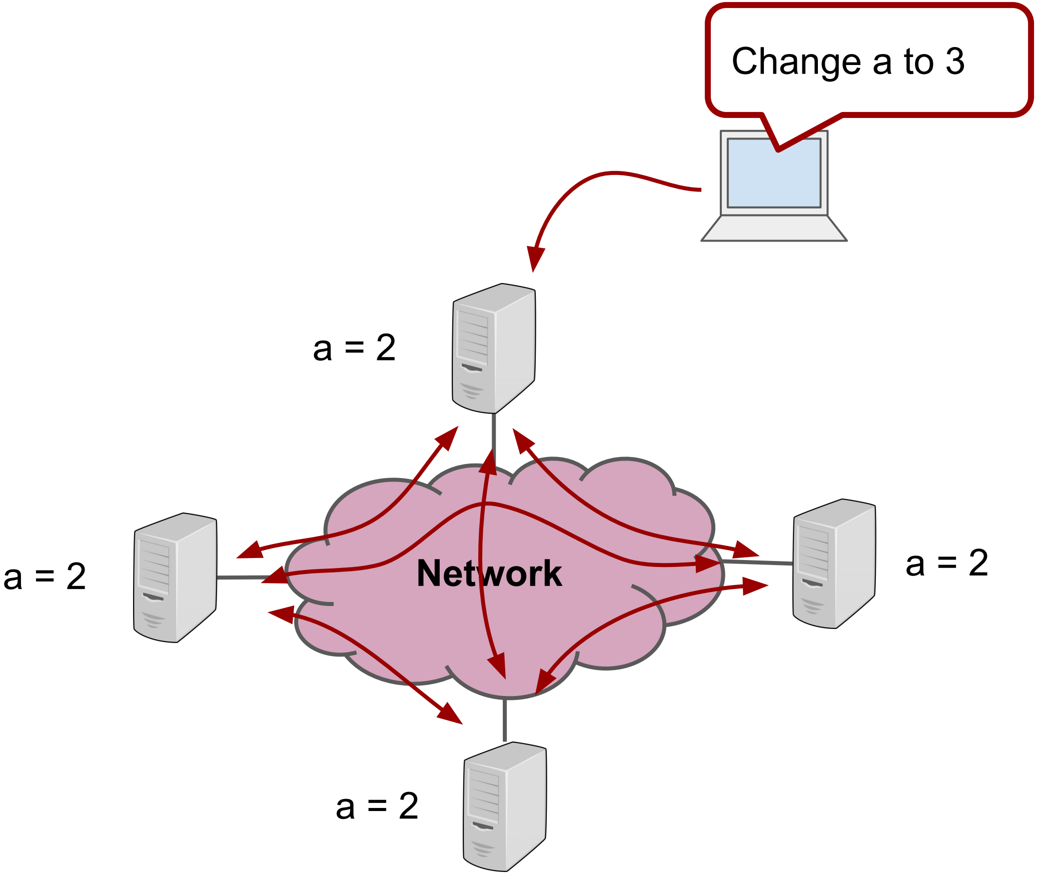
Paxos is a very famous and complex protocol that governs these communications to ensure consensus.



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

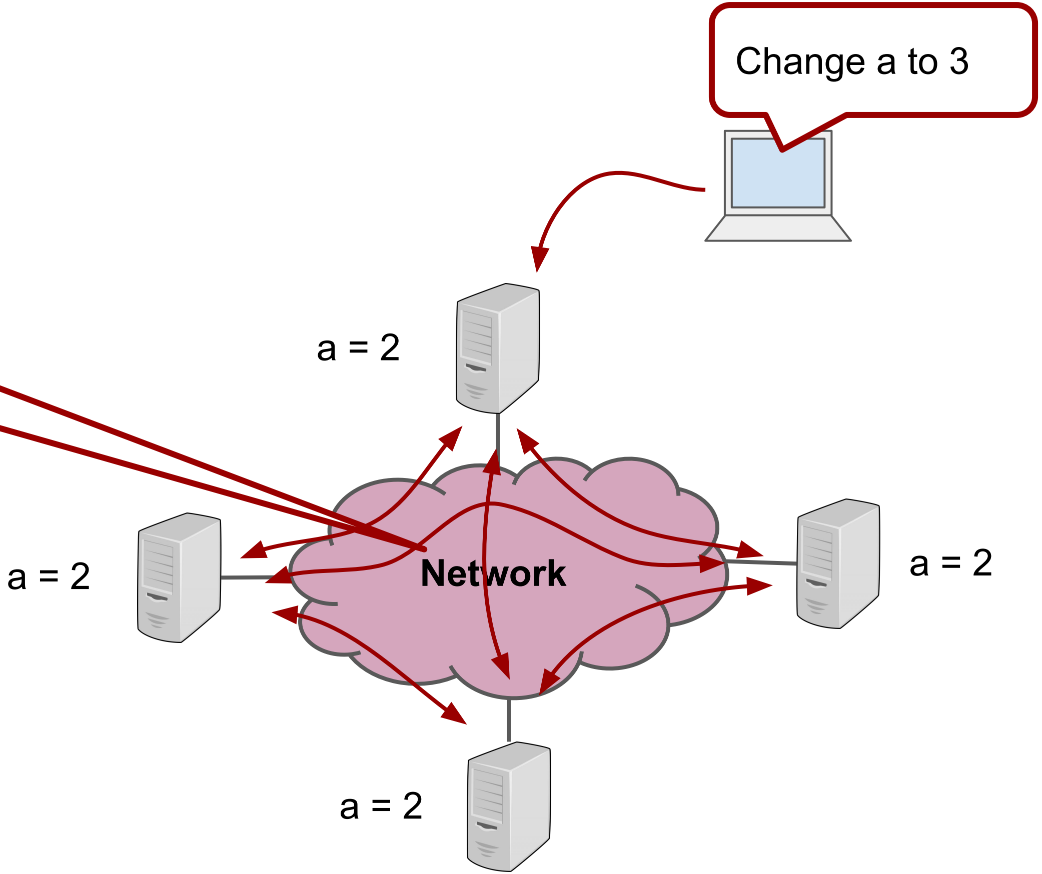
the actual computations done on each participant is quite simple.

Example 2: In-network consensus

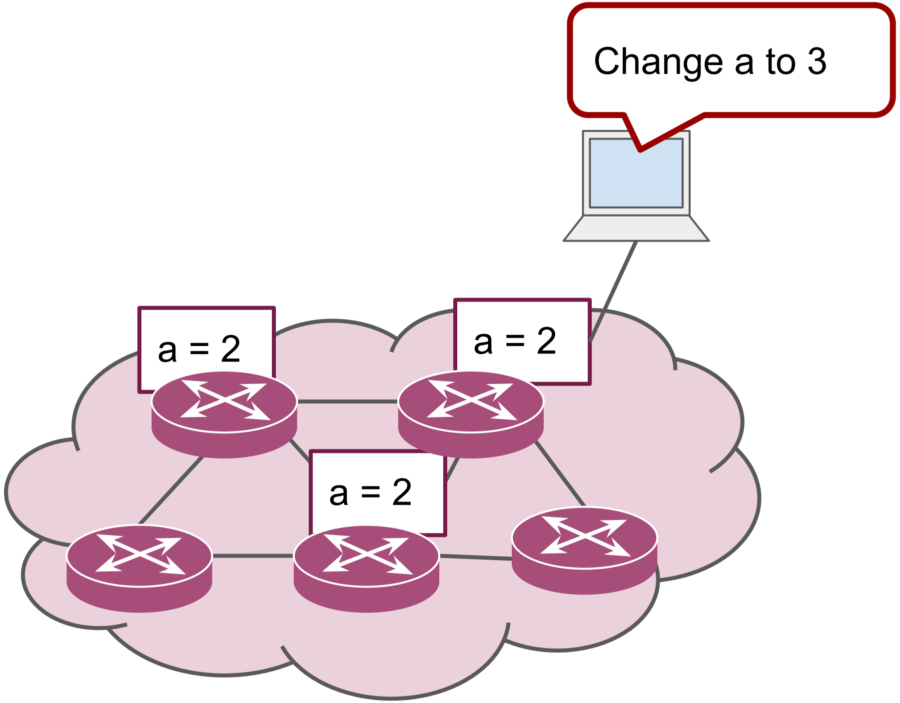
Can we implement it in the network?

Consensus is communication heavy

the actual computations done on each participant is quite simple.



Example 2: In-network consensus

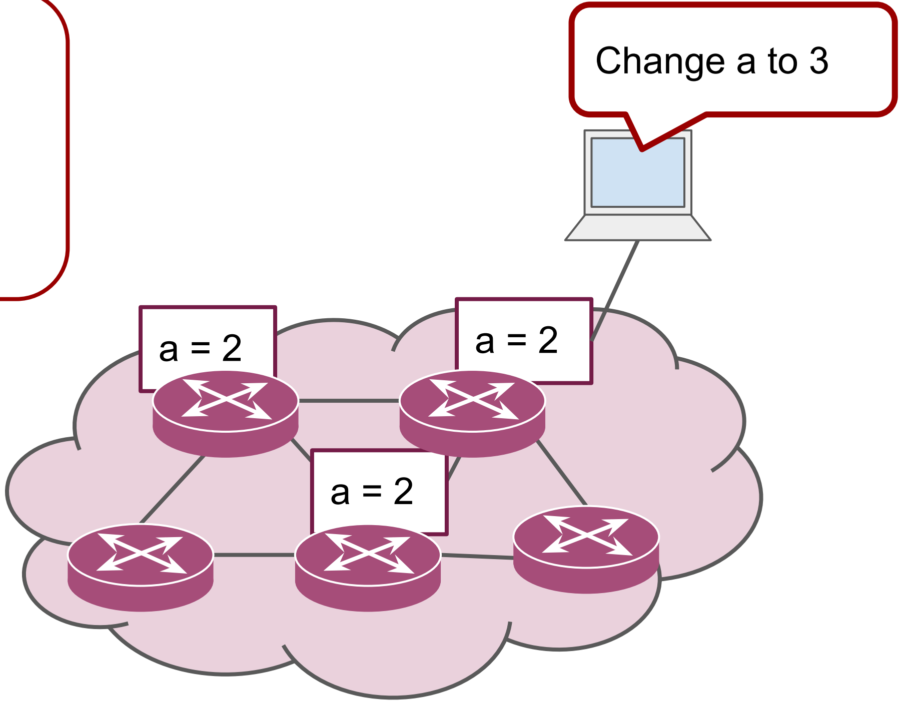


Example 2: In-network consensus

Switches keep all copies of the values.

Switches server read and write requests.

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



Example 2: In-network consensus

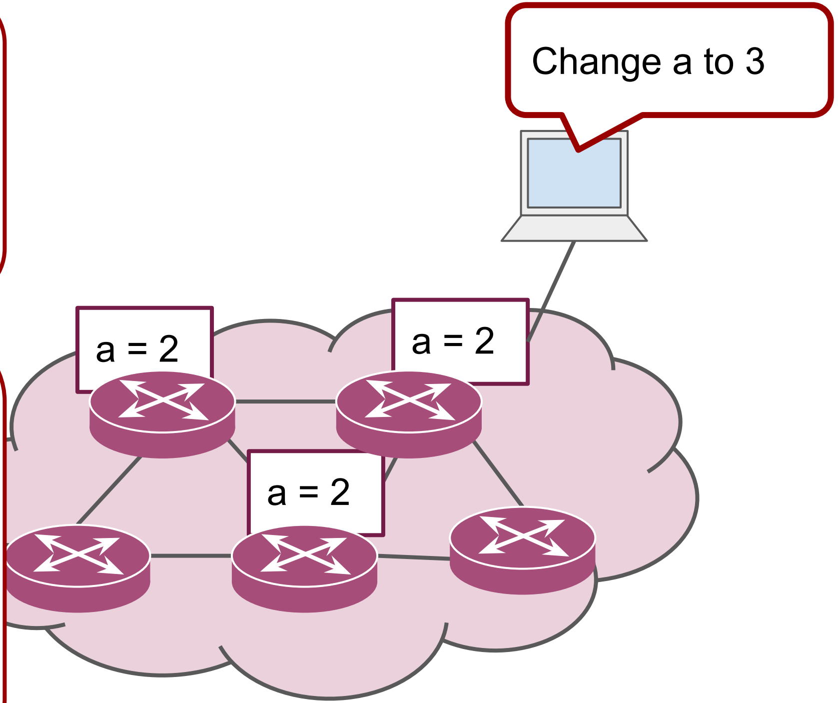
Switches keep all copies of the values.

Switches serve read and write requests.

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

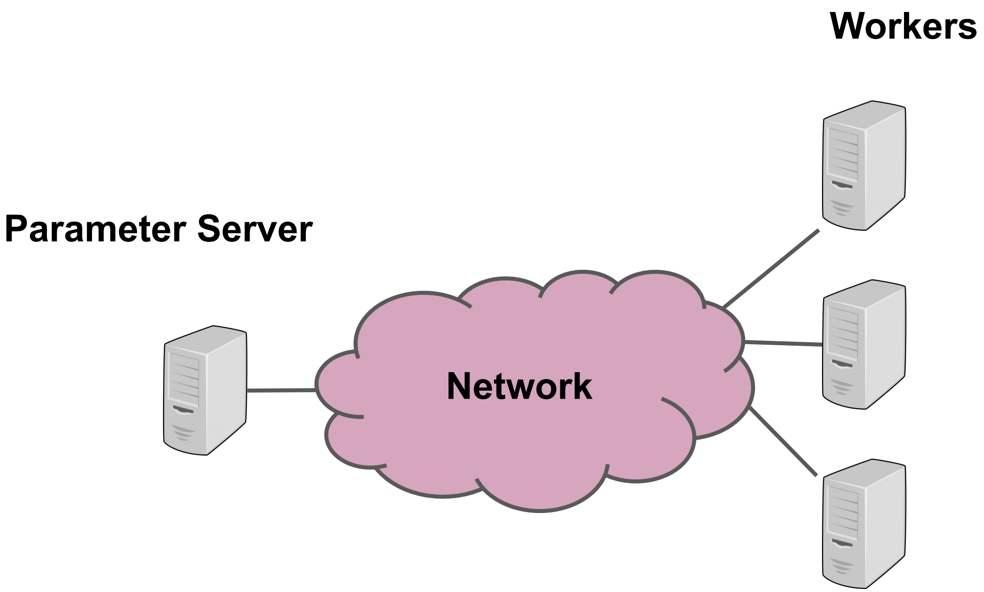
Benefits?

- 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



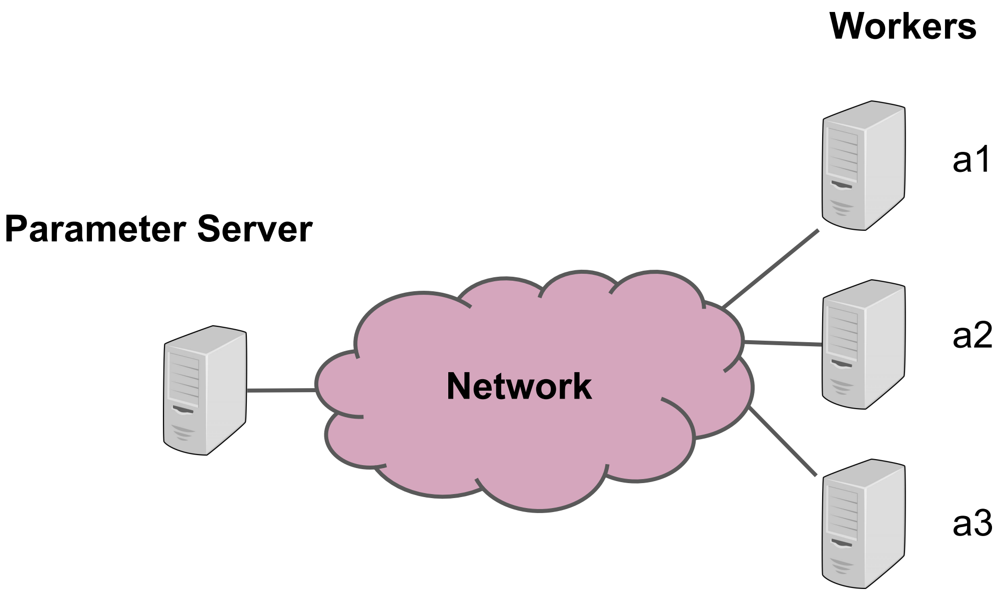
Example 3: Accelerating ML Training

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



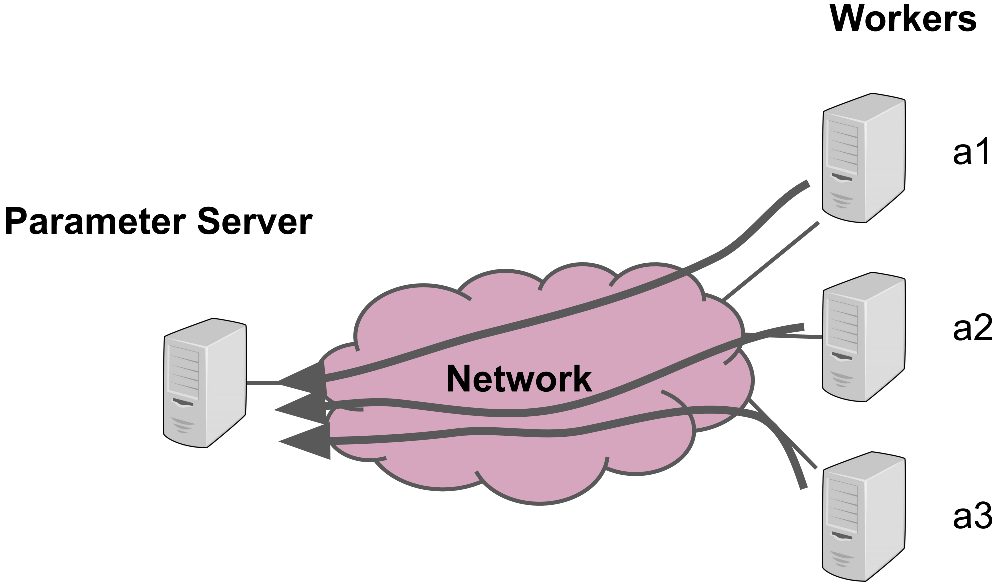
Example 3: Accelerating ML Training

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



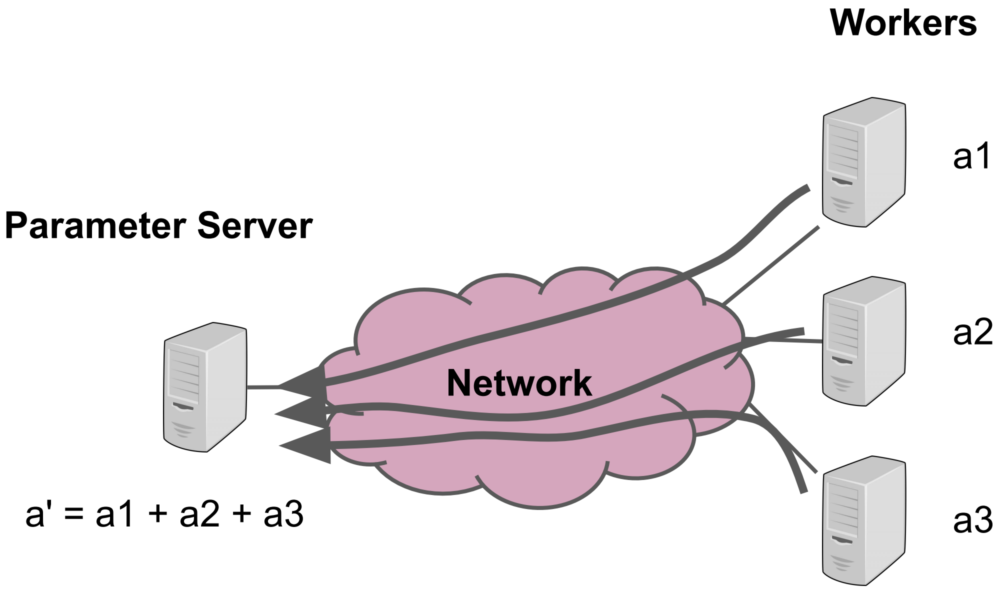
Example 3: Accelerating ML Training

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



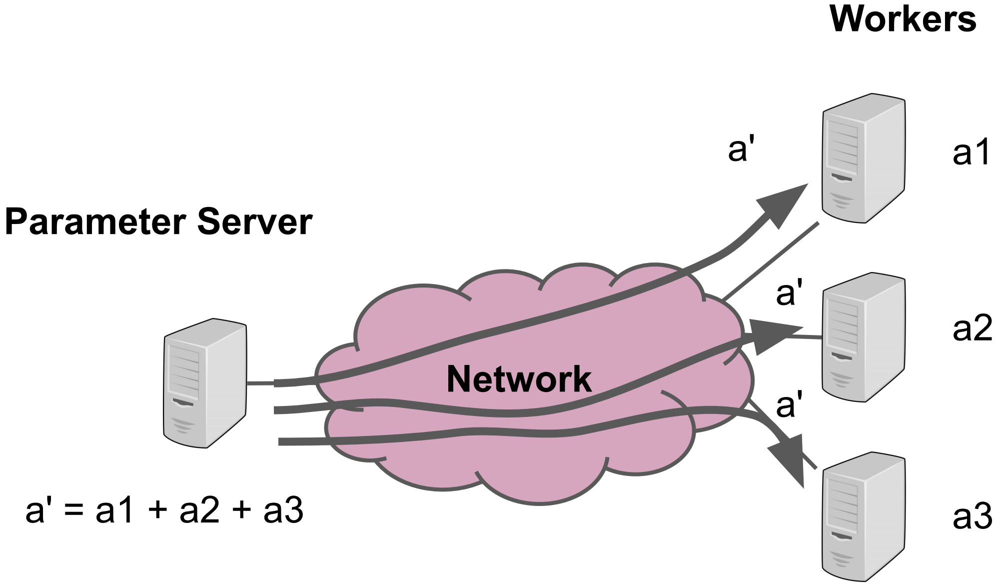
Example 3: Accelerating ML Training

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



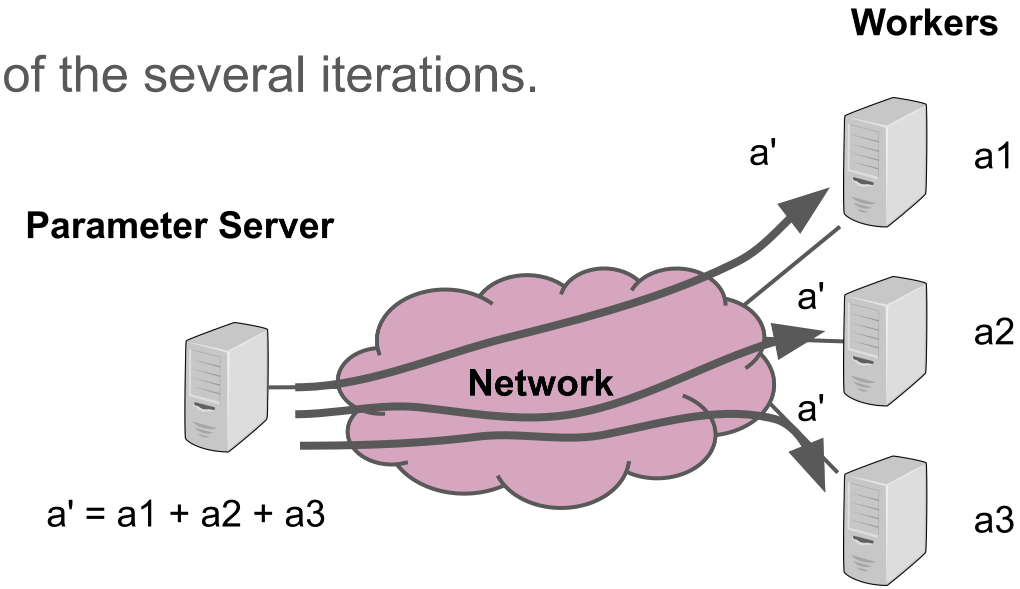
Example 3: Accelerating ML Training

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



Example 3: Accelerating ML Training

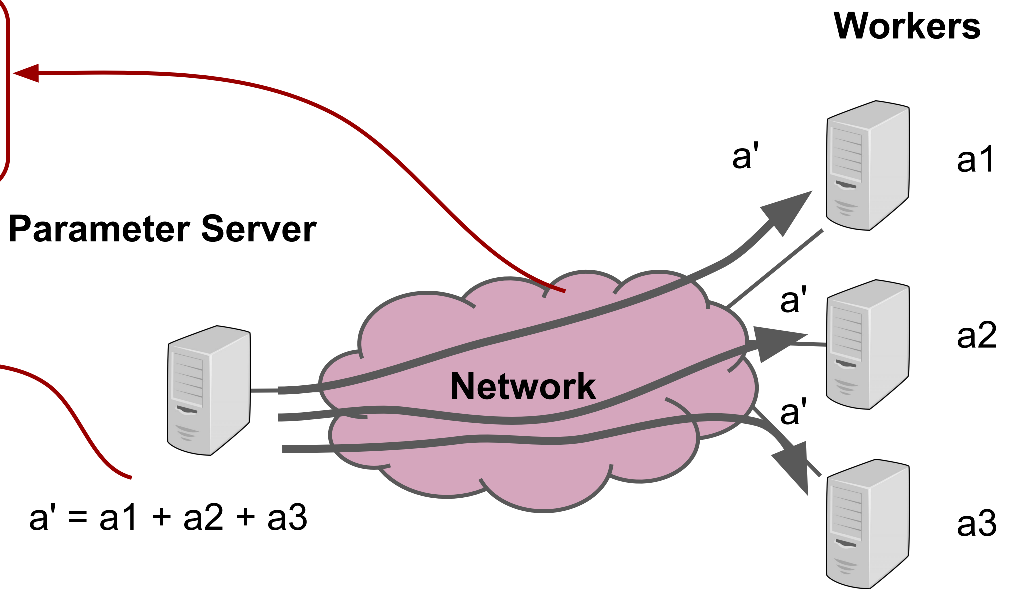
- Distributed training of ML models can require a lot of network communication.
- This happens in every of the several iterations.



Example 3: Accelerating ML Training

Lots of communication between the parameter server and workers.

Simple computation on the parameter server

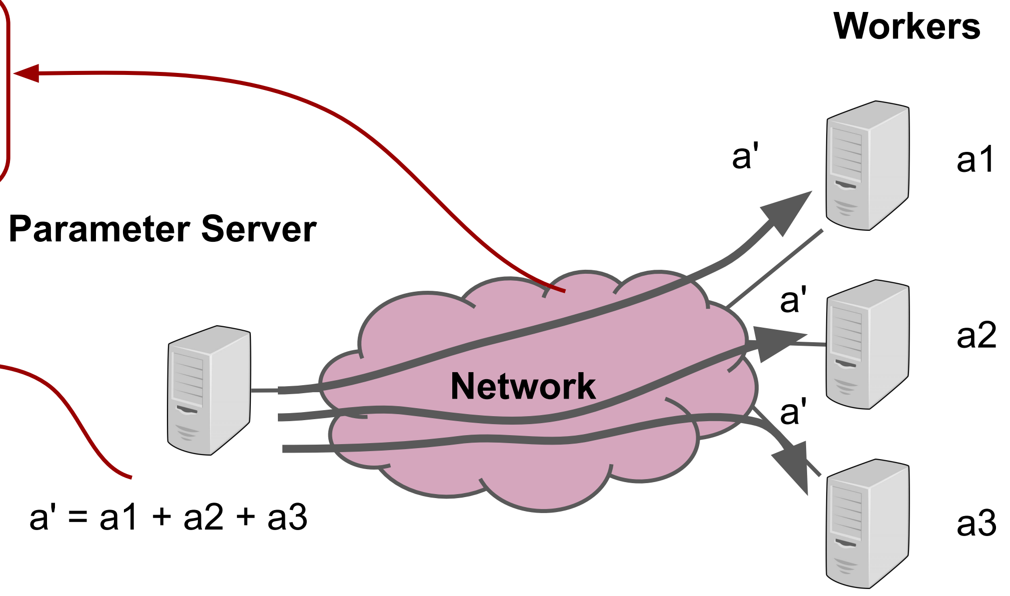


Example 3: Accelerating ML Training

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 smallish 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