NetStore

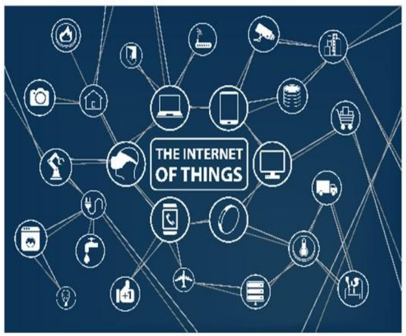
Leveraging Network Optimizations to Improve Distributed Transaction Processing Performance

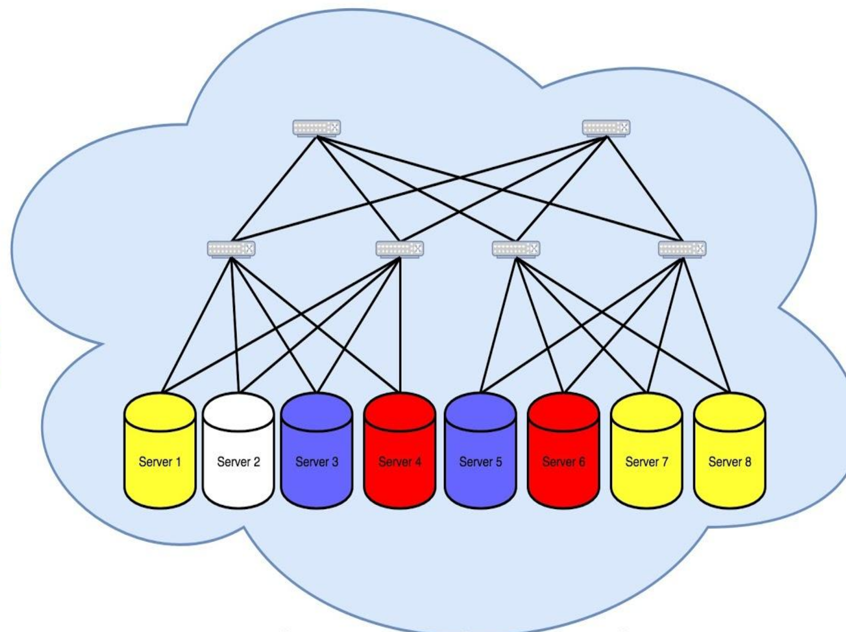
Xu Cui, Michael Mior, Bernard Wong,
Khuzaima Daudjee, Sajjad Rizvi

UNIVERSITY OF WATERLOO

ACTIVE Workshop @ Middleware 2017
Las Vegas, NV, USA
December 12, 2017

A tremendous amount of data is generated and stored in the cloud



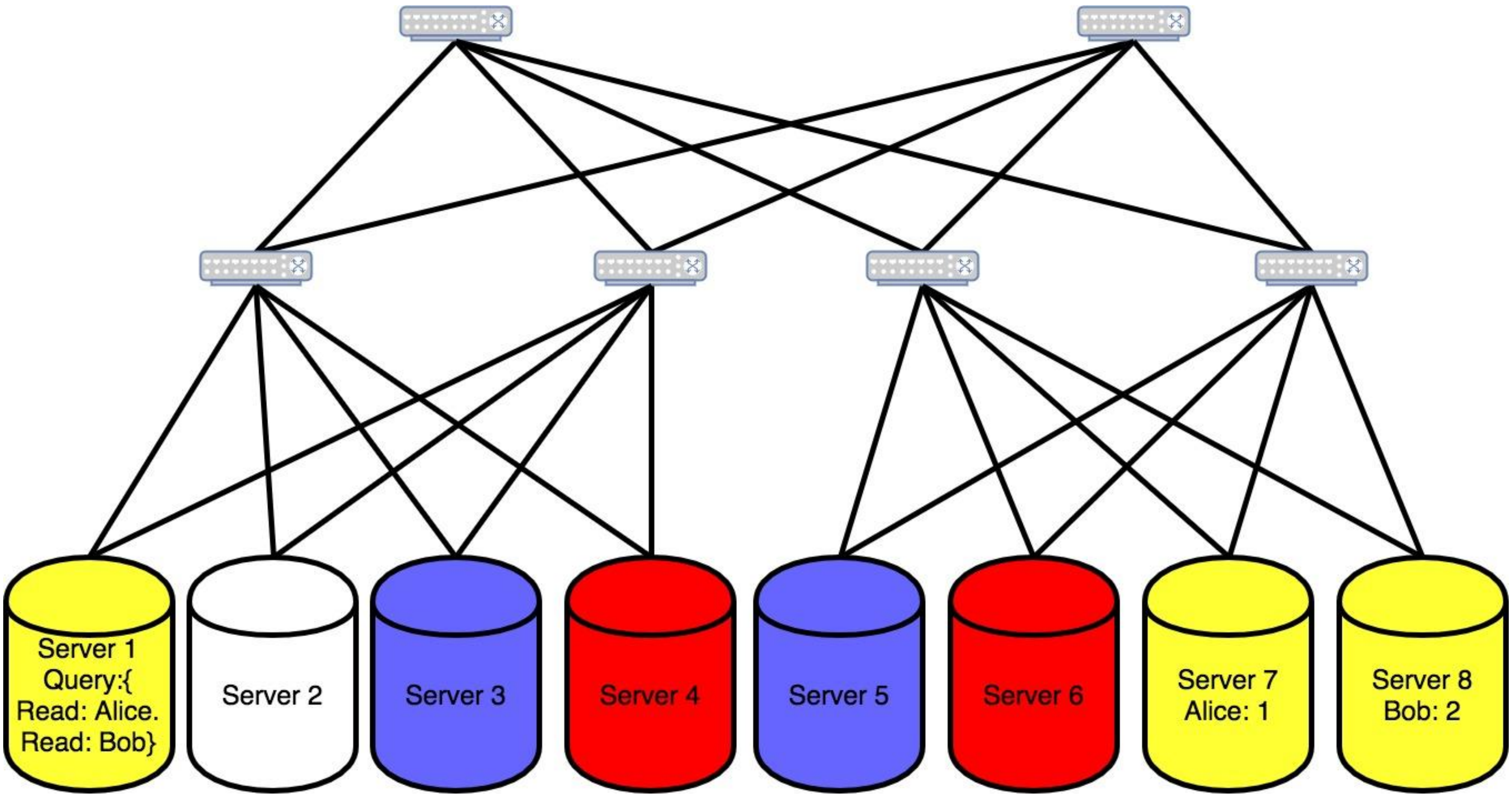


the cloud

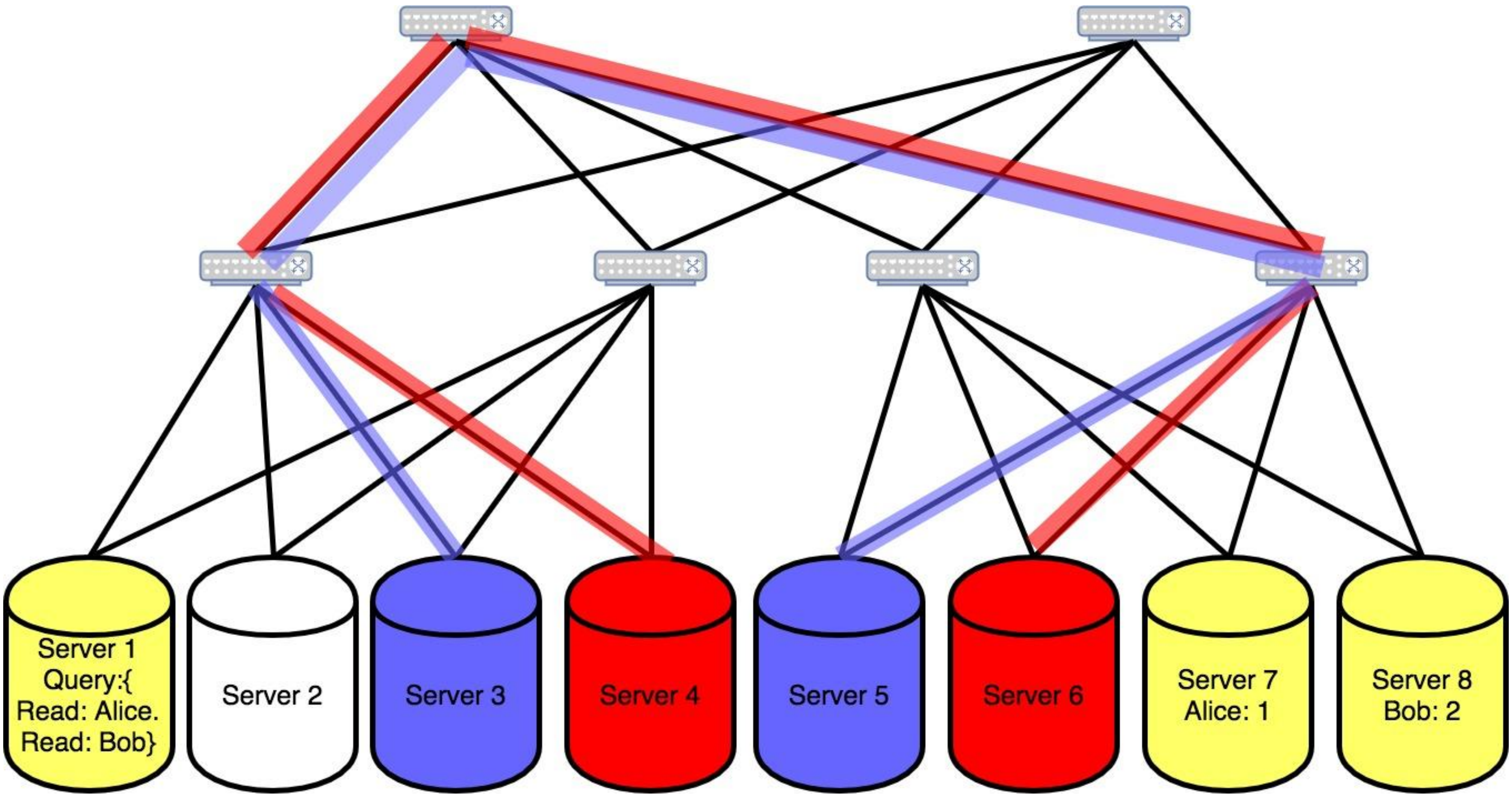
image sources:
smartbear, appian, imarticus

UNIVERSITY OF WATERLOO

A distributed query

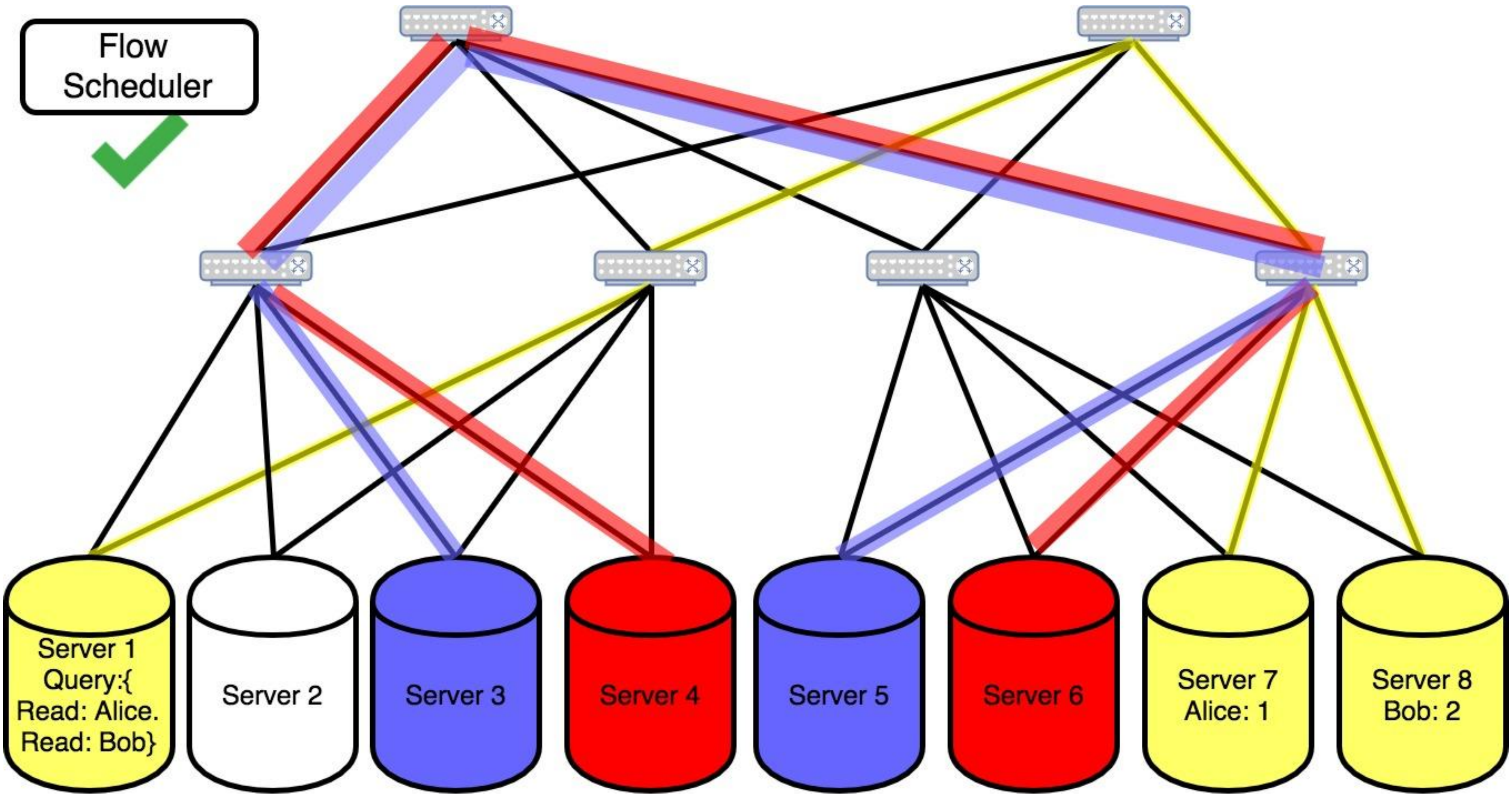


Part of the network is congested

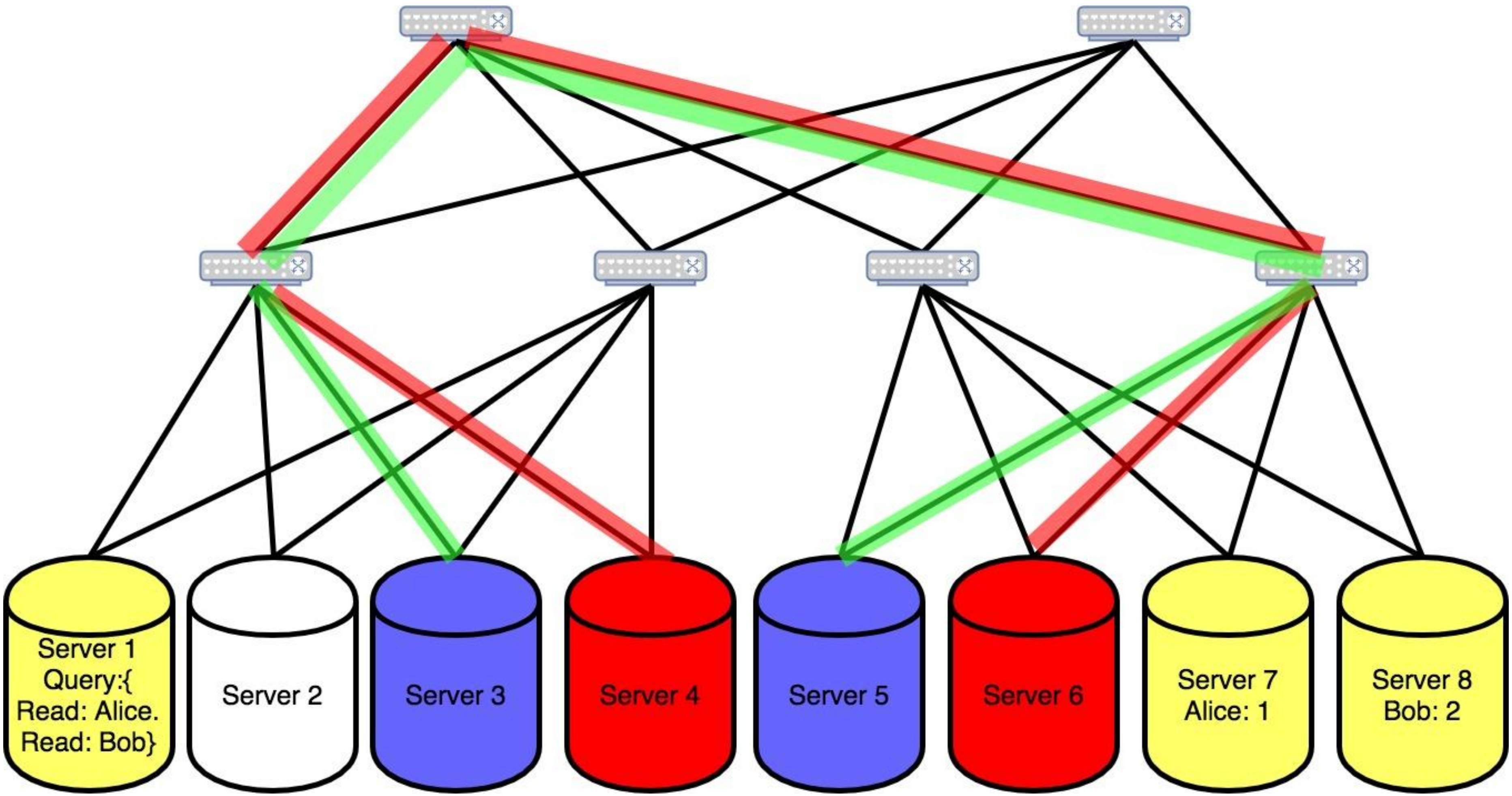


UNIVERSITY OF WATERLOO

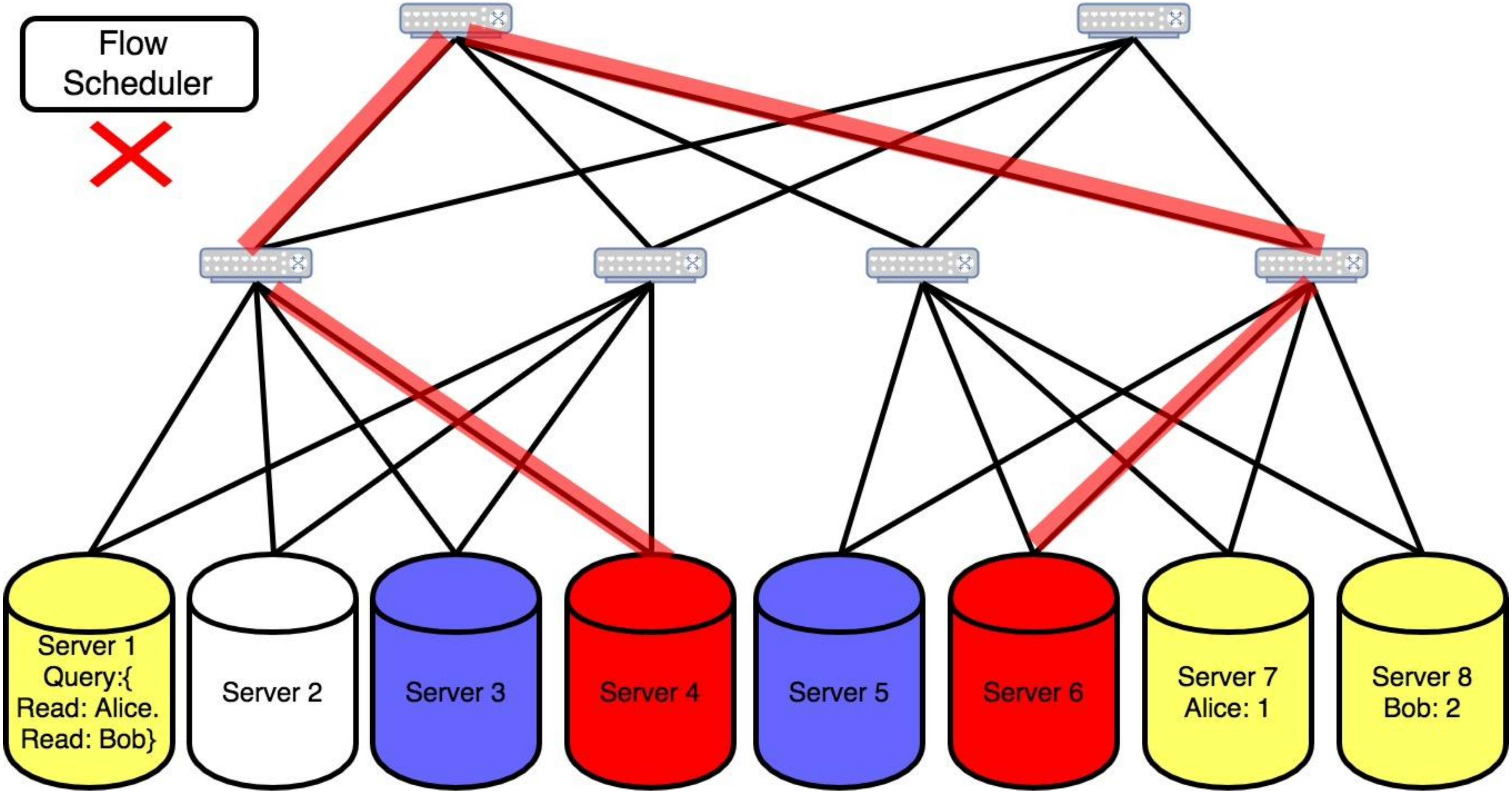
A flow scheduler can solve this



If the green flows are short-lived flows

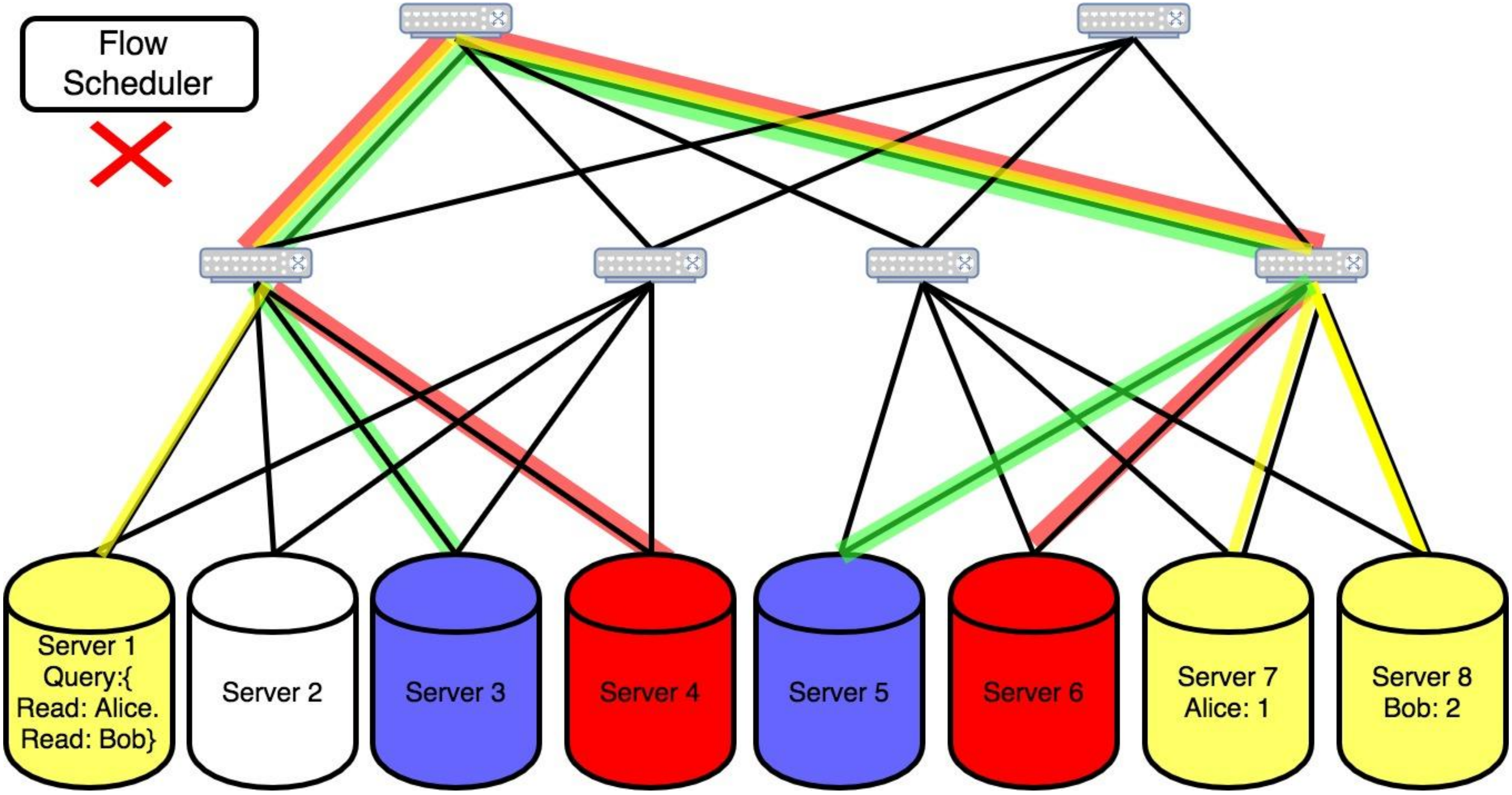


A flow scheduler cannot detect the transient congestion



UNIVERSITY OF WATERLOO

The transaction query may be routed on the congested paths



UNIVERSITY OF WATERLOO

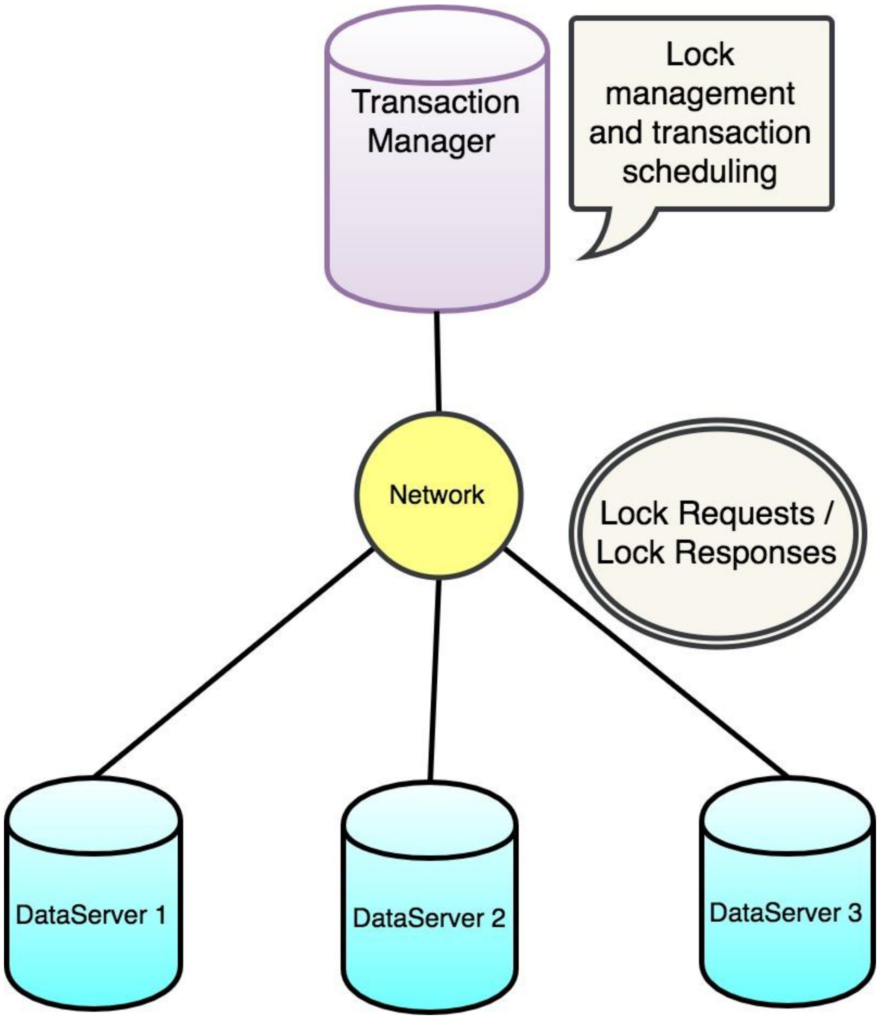
NetStore

- A transaction processing system co-designed with the network enables two network-aware optimizations

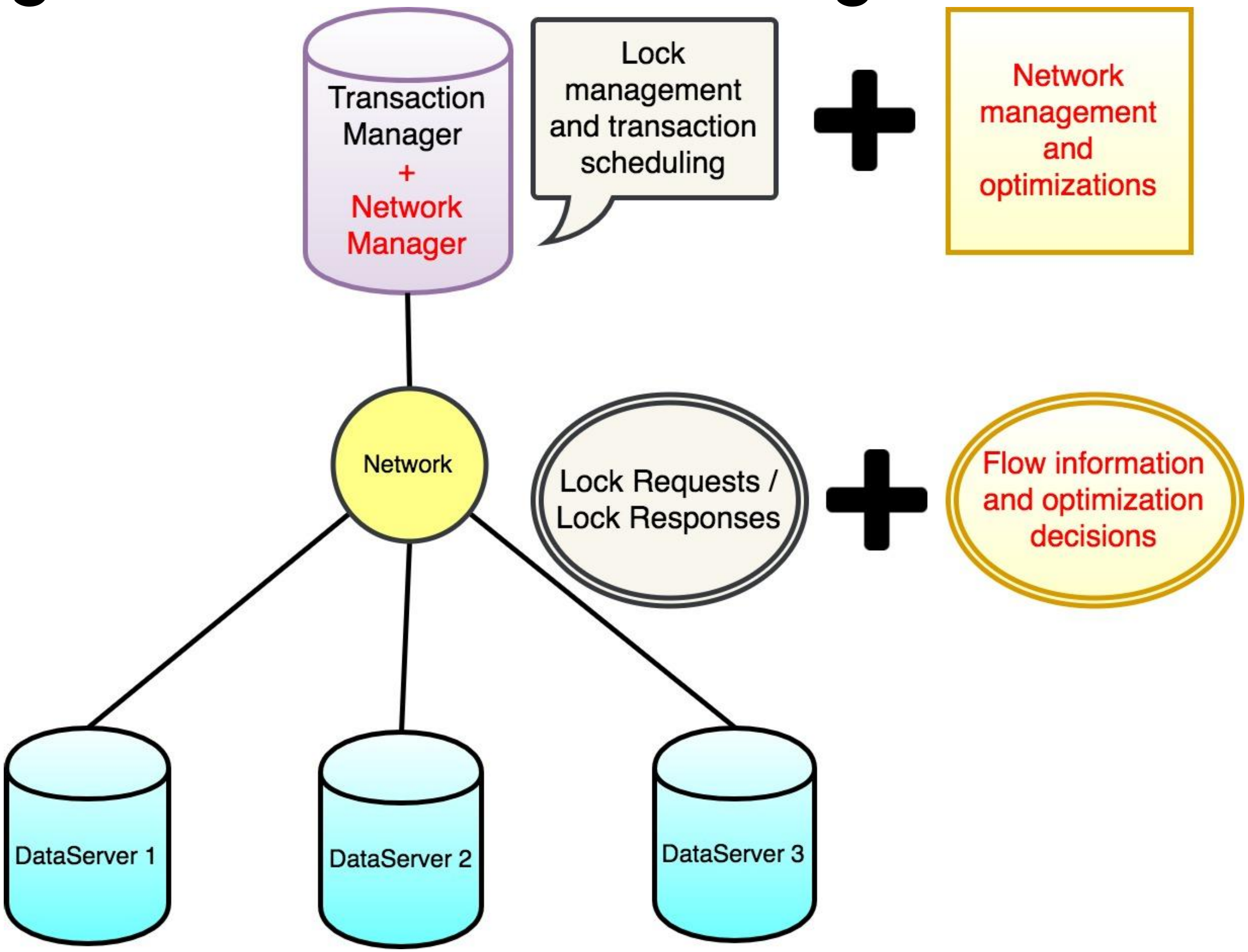
  ➢ **Least bottlenecked path (LBP)**: a dynamic flow scheduler that leverages information gathered from a transaction manager

  ➢ **Network-aware caching (NAC)**: a database caching optimization that makes caching decisions based on the network topology

Standard database architecture



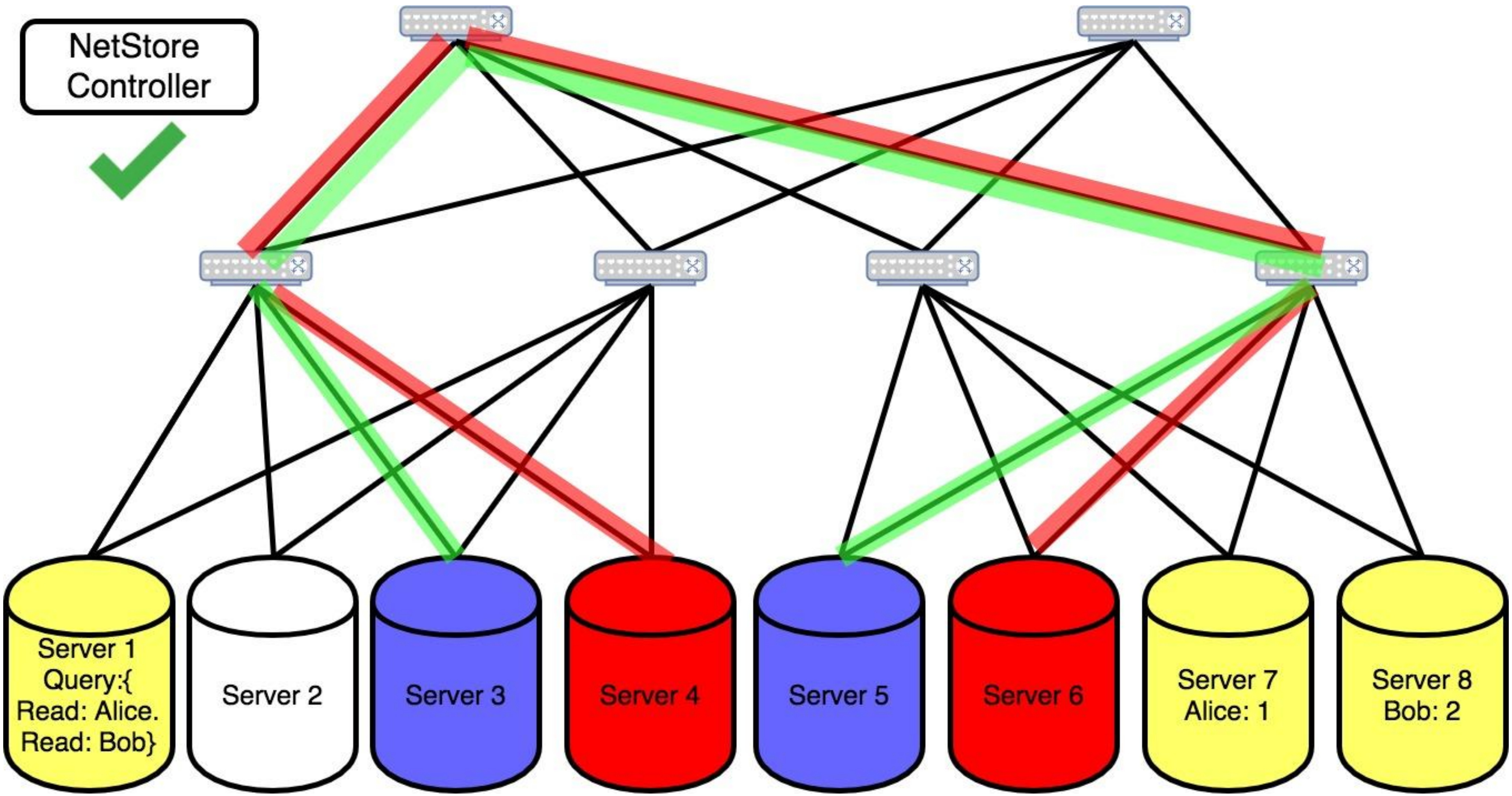
The NetStore controller extends the transaction manager with a network manager



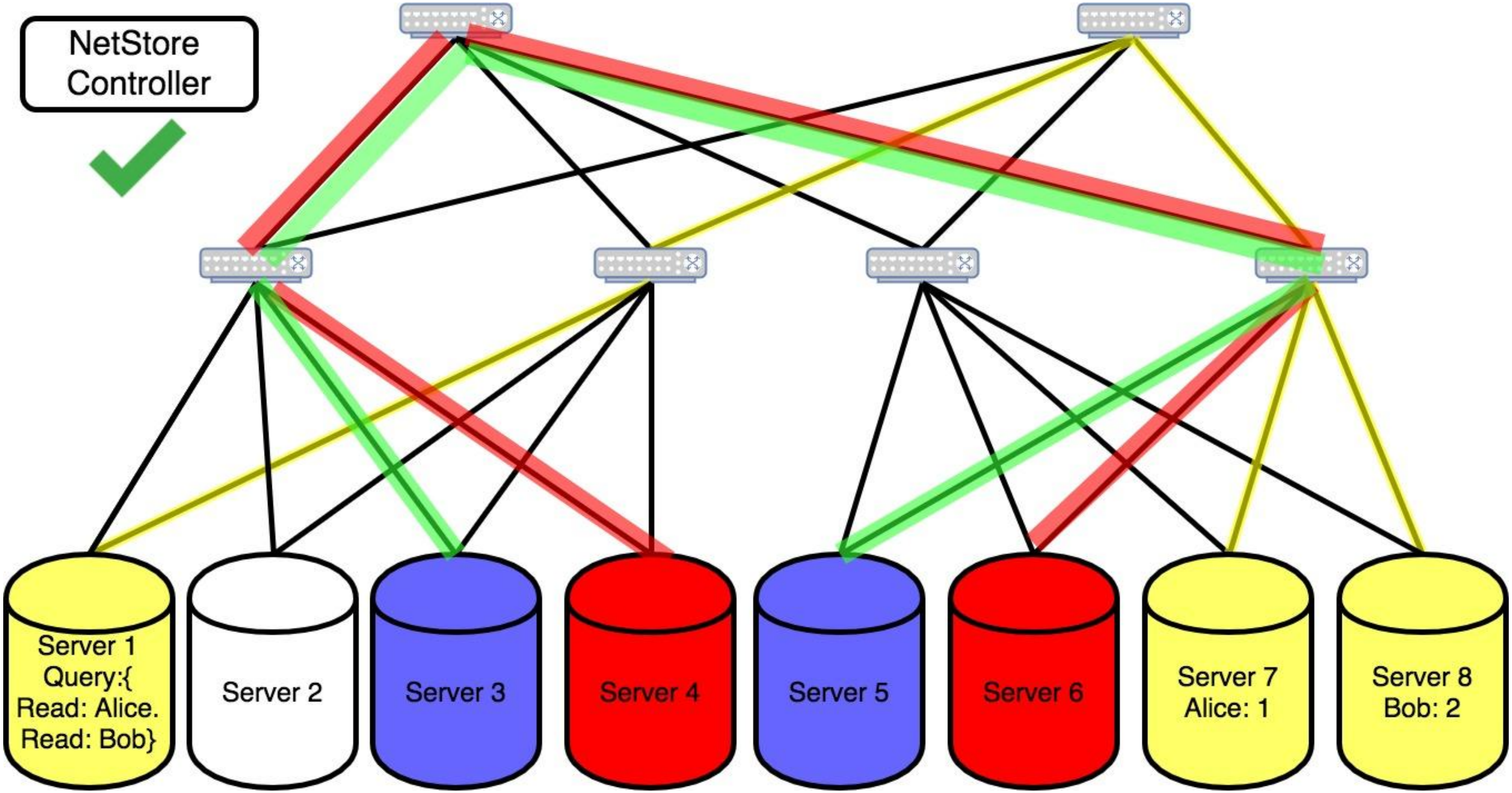
Least bottlenecked path (LBP)

- The database and network co-design enables NetStore to maintain a global view of the network
- LBP uses this dynamic flow information to approximate the bandwidth allocation for each new flow
- LBP routes the new flow through the best path

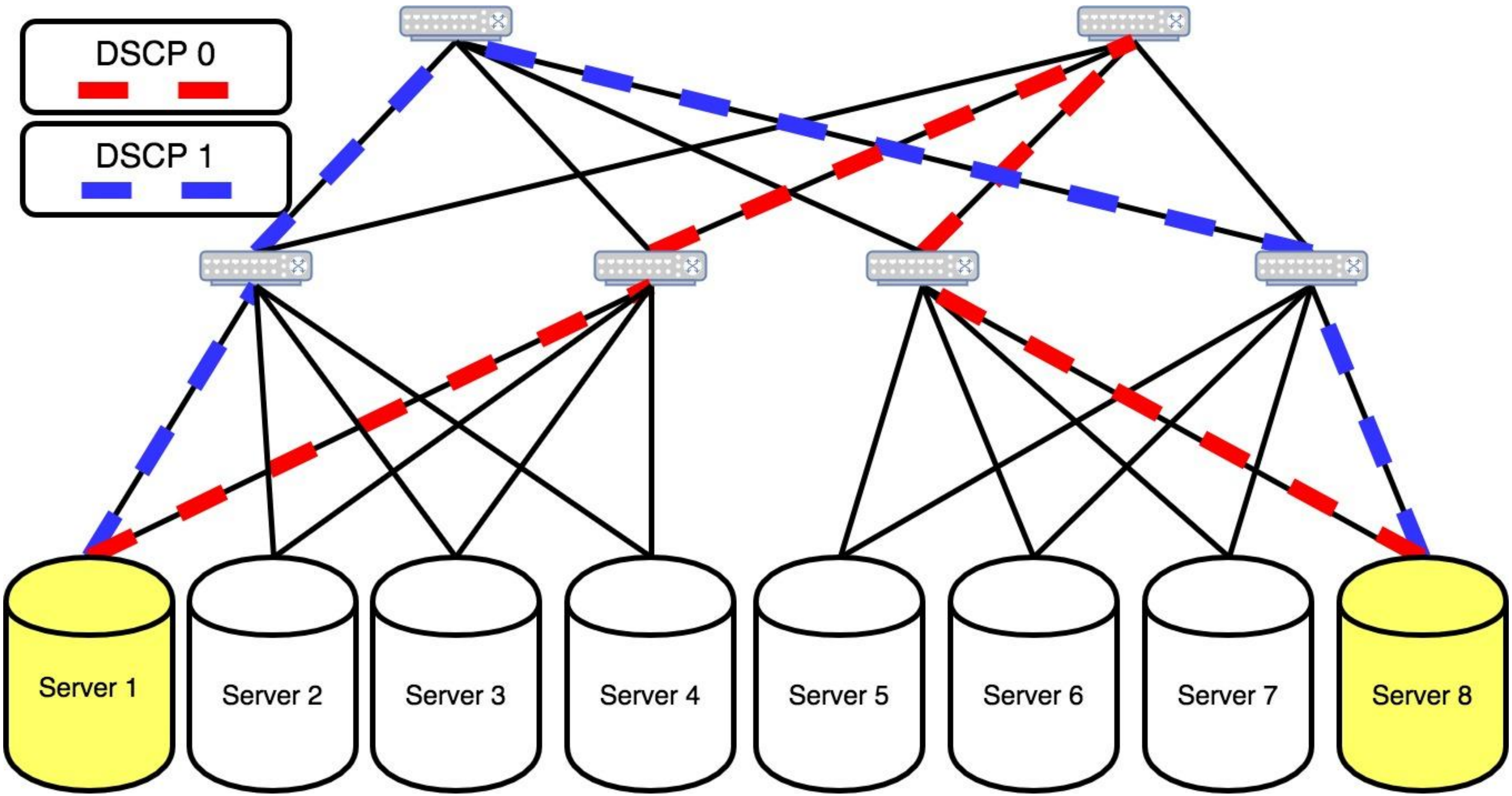
LBP can detect the transient network congestion caused by short-lived flows



LBP selects the best path for each transaction flow



NetStore configures network paths when the system bootstraps



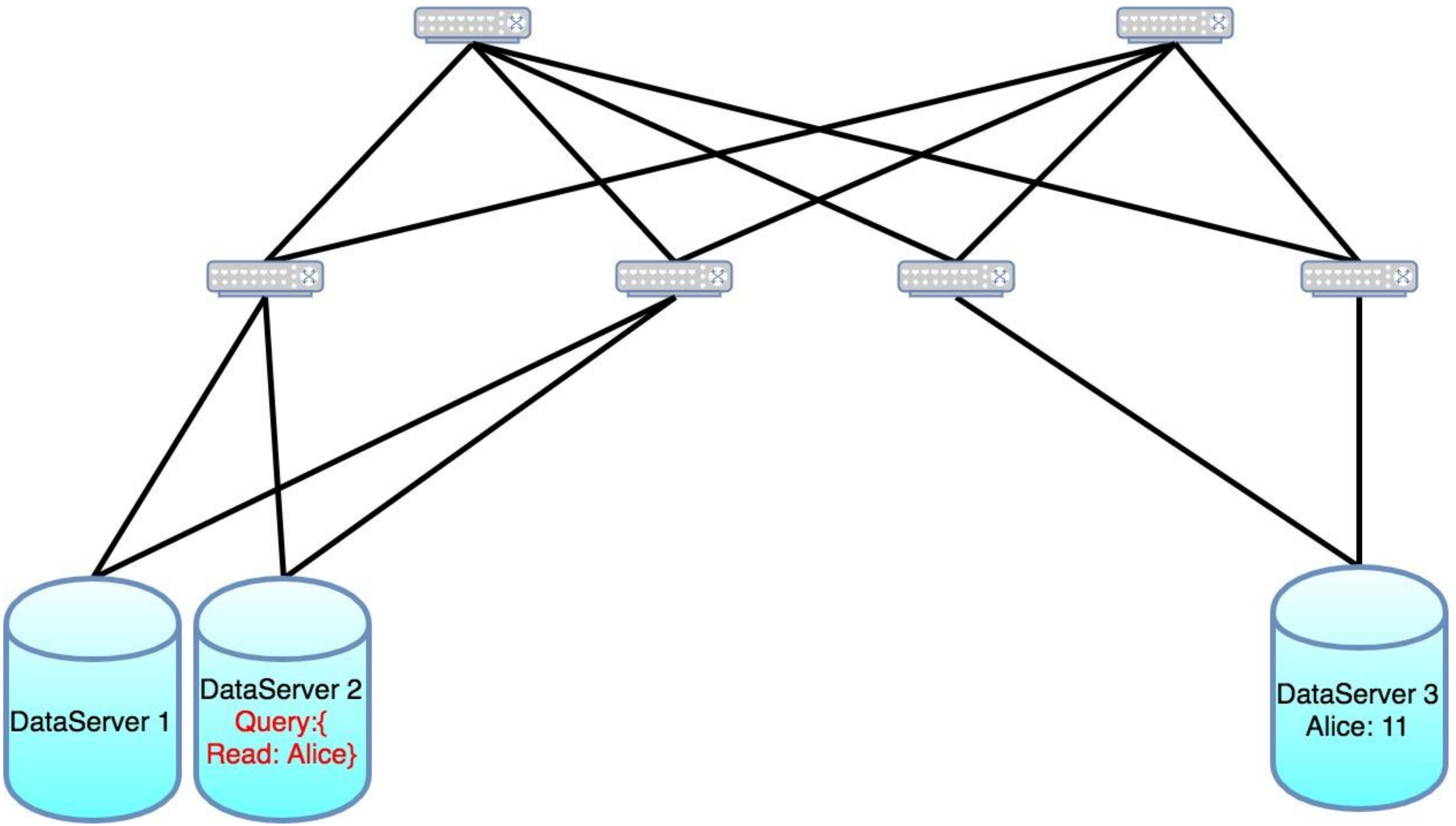
Benefits of least bottlenecked path

- Makes informed routing decisions based on the dynamic flow information gathered from the transaction manager
- Balances the network load for short-lived transactional flows when transient network congestion is present

Network-aware caching (NAC)

- The co-design enables network-aware caching
- NAC leverages cache replicas to reduce the load on the network
- NAC avoids cache invalidations which can increase the network load

DataServer 2 performs a read query on Alice



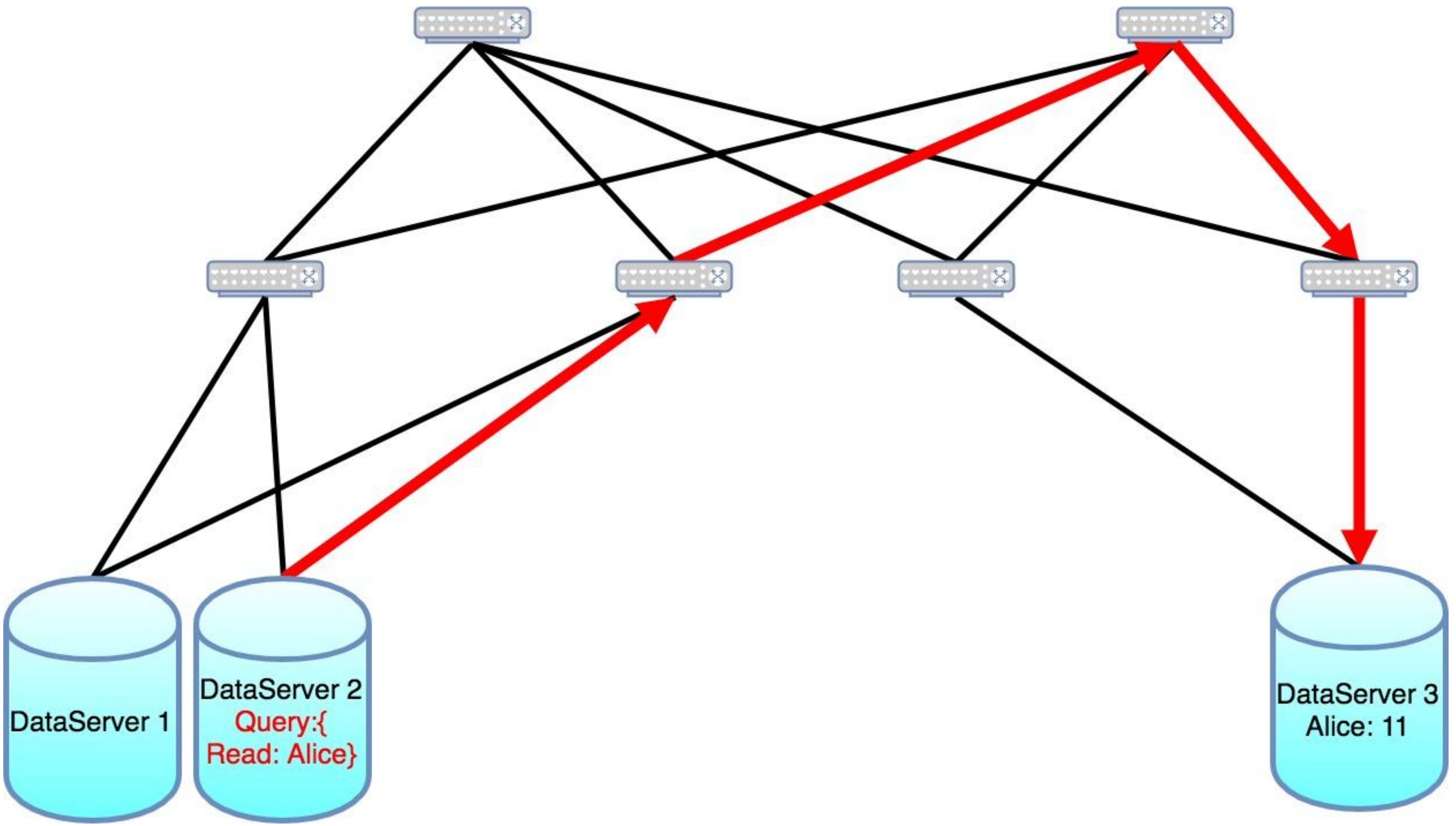
The NetStore controller maintains a cache index of the cache entries

Key	DataServer IDs	Cache Version #

The NetStore controller creates a version number for each cache entry

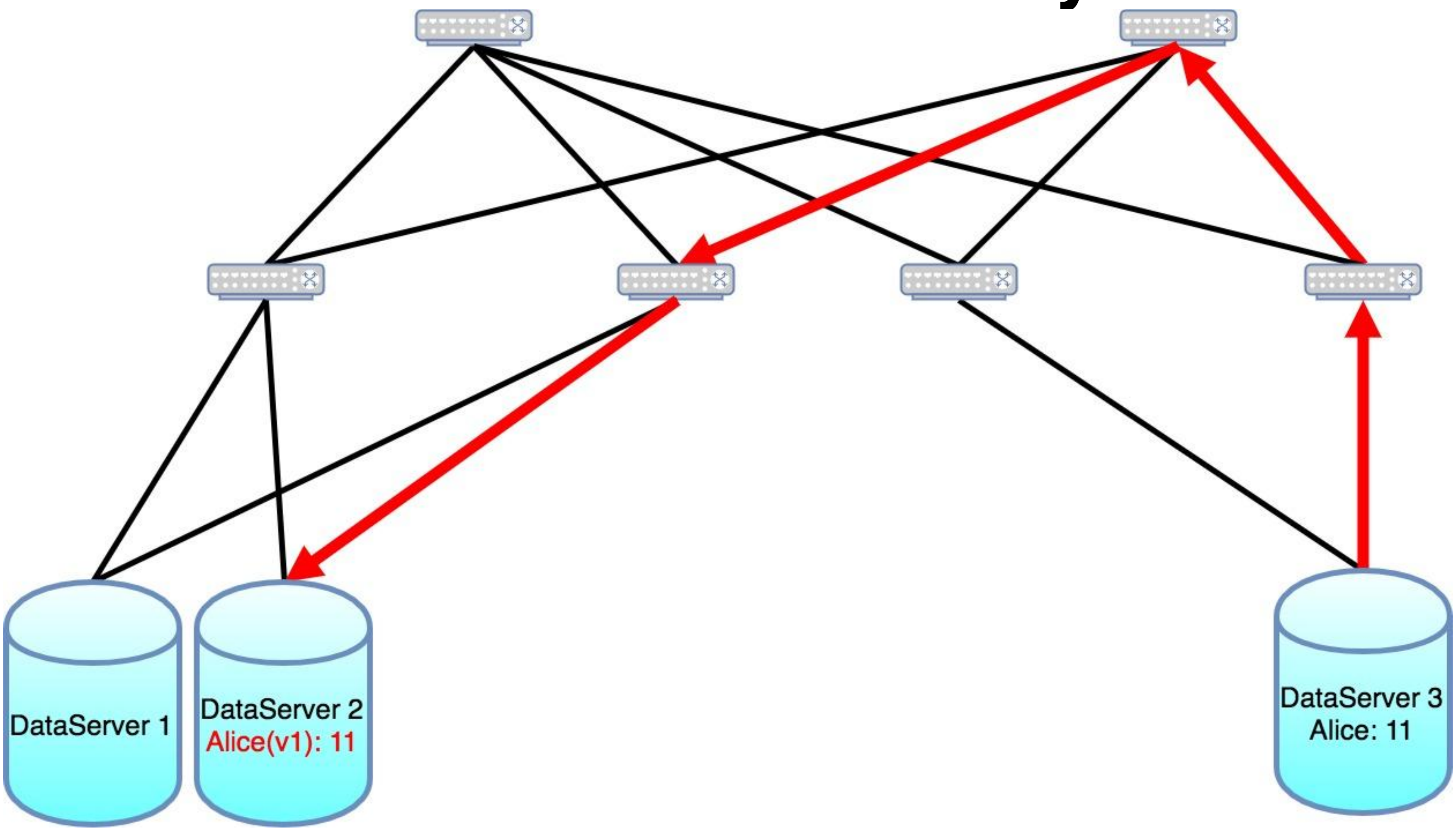
Key	DataServer IDs	Cache Version #
Alice	2	1

DataServer 2 fetches Alice from DataServer 3

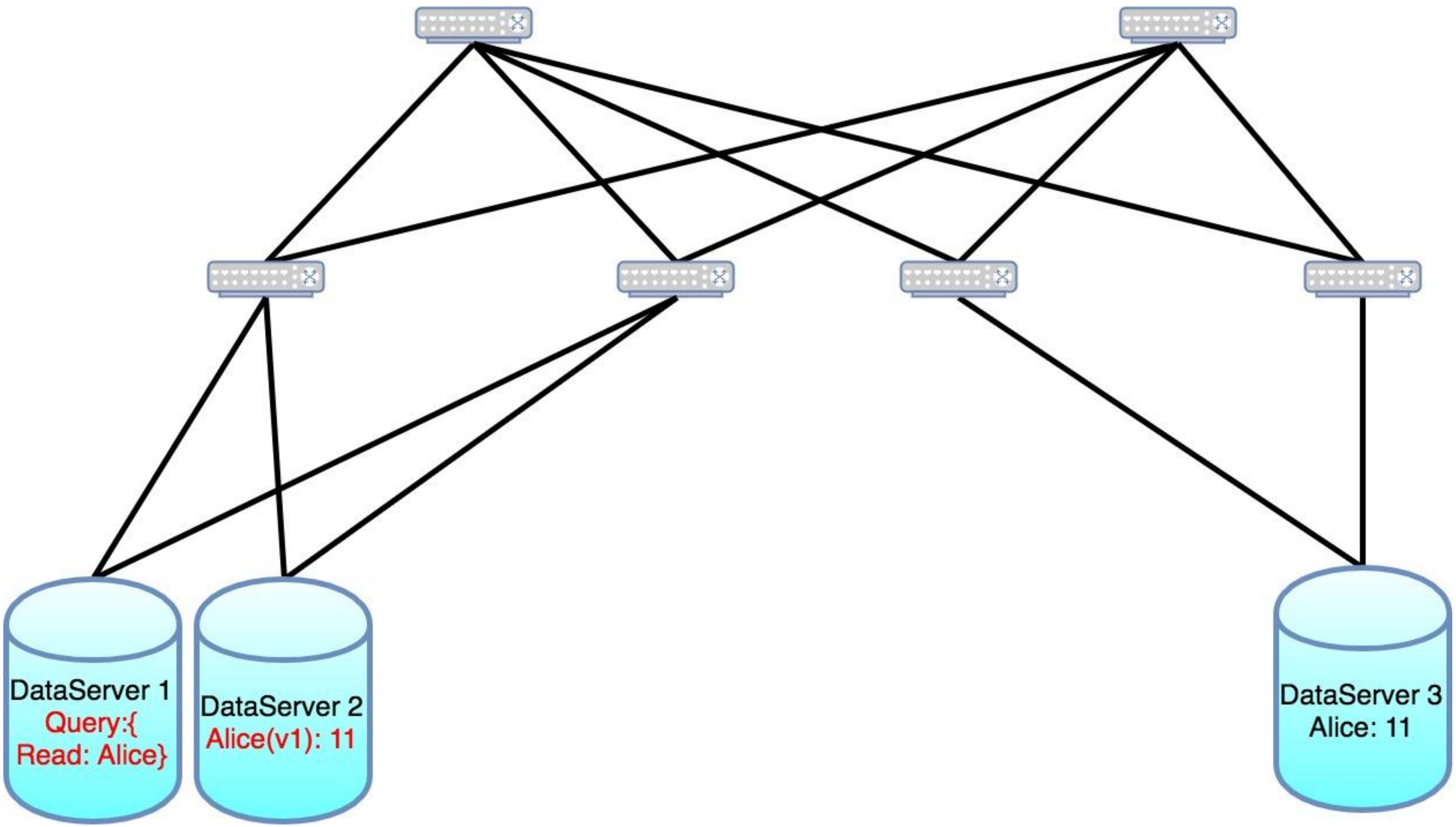


UNIVERSITY OF WATERLOO

DataServer 2 stores the cache replica and the version number locally



DataServer 1 performs a read operation on Alice



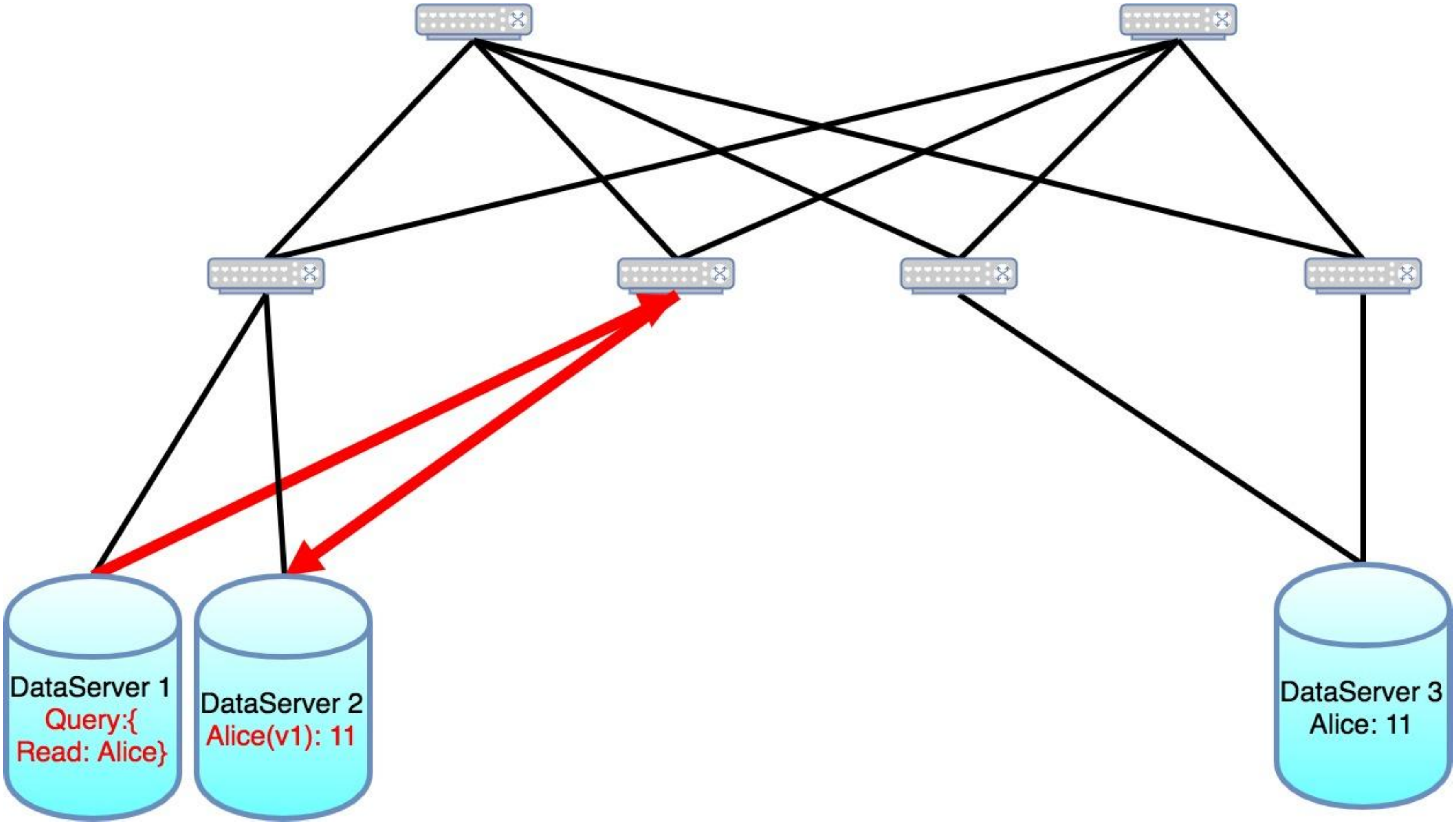
The NetStore controller determines the best cache replica location for this op

Key	DataServer IDs	Cache Version #
Alice	2	1

The NetStore controller adds server id 1 to Alice's cache index

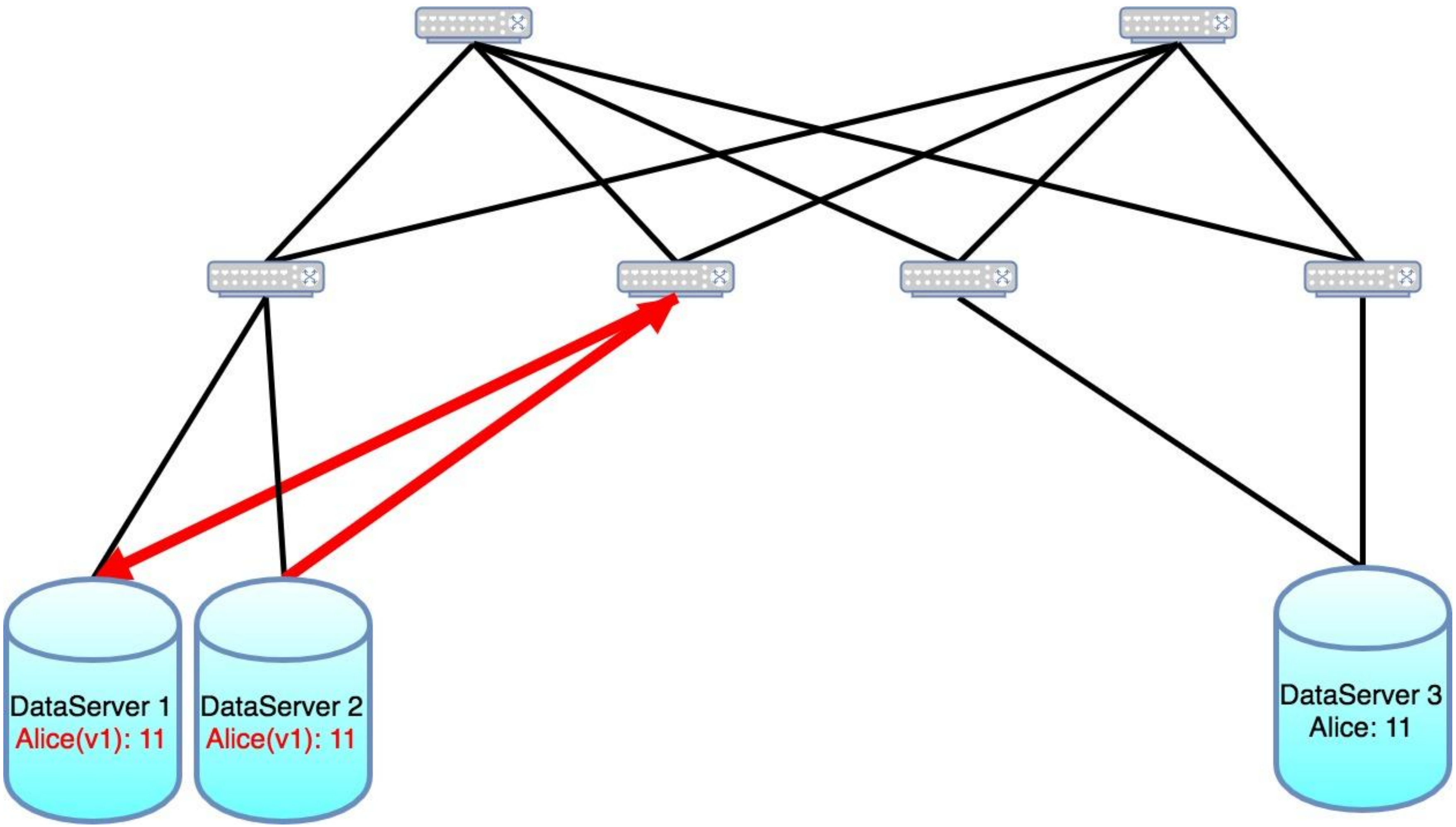
Key	DataServer IDs	Cache Version #
Alice	2 => 2, 1	1

DataServer 1 fetches the data from DataServer 2

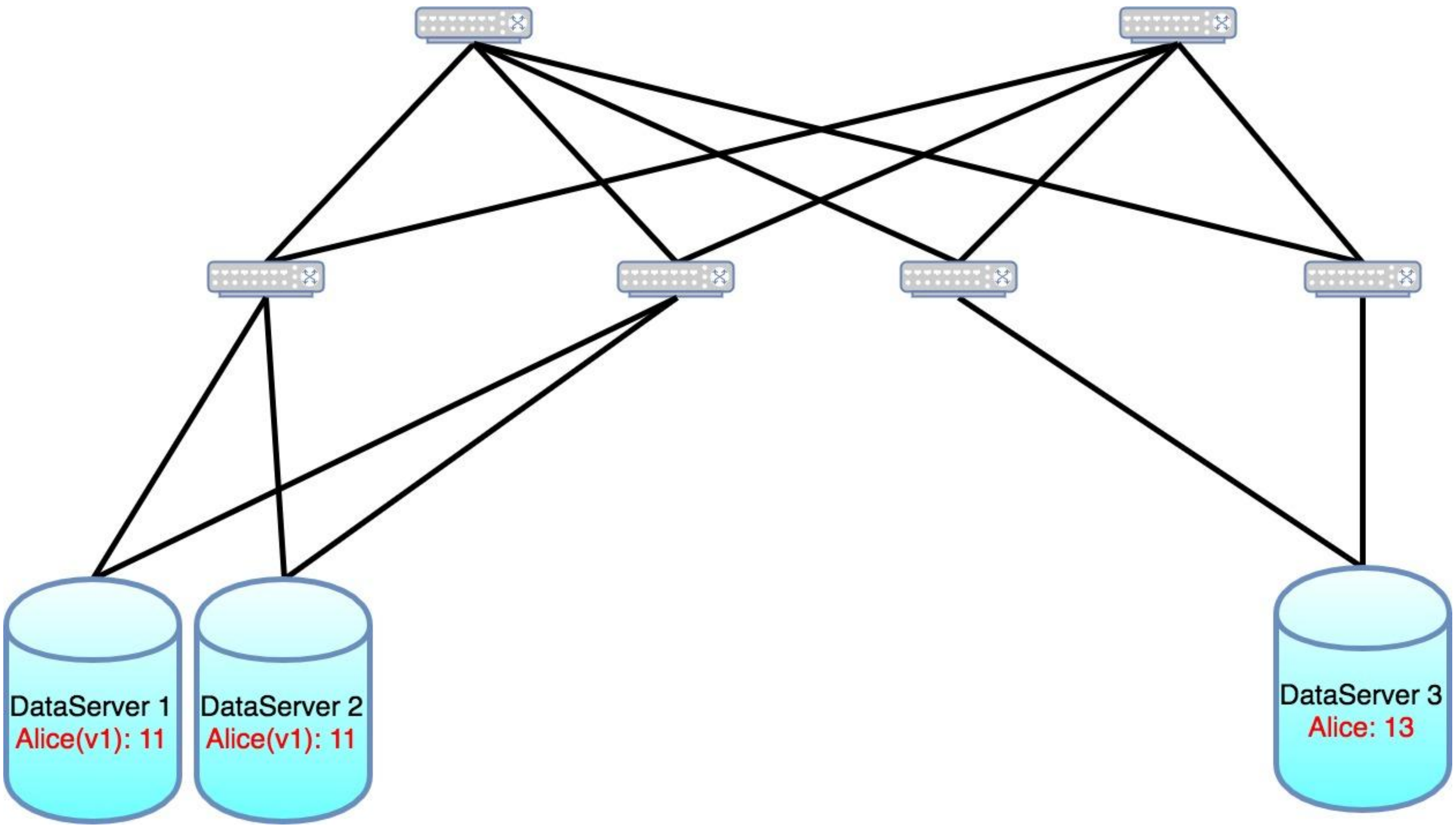


UNIVERSITY OF WATERLOO

DataServer 1 stores the result in its local cache



A write operation is performed on Alice



The NetStore controller erases Alice

Key	DataServer IDs	Cache Version #
~~Alice~~	~~2, 1~~	~~1~~

A new version number is generated when another read operation happens

Key	DataServer IDs	Cache Version #
Alice	1	2

UNIVERSITY OF WATERLOO

Benefits of network-aware caching

- Augments a database optimization with network-awareness
- Reduces the load on the network
- Avoids cache invalidations
- Performs batch-processing to further improve performance

Experimental setup

- We use Mininet to build a distributed virtual multi-rooted tree network
  - 64 virtual servers
  - Each virtual server runs a transaction client, a transaction server, a background client and a background server
  - 1 Gbps capacity on each link

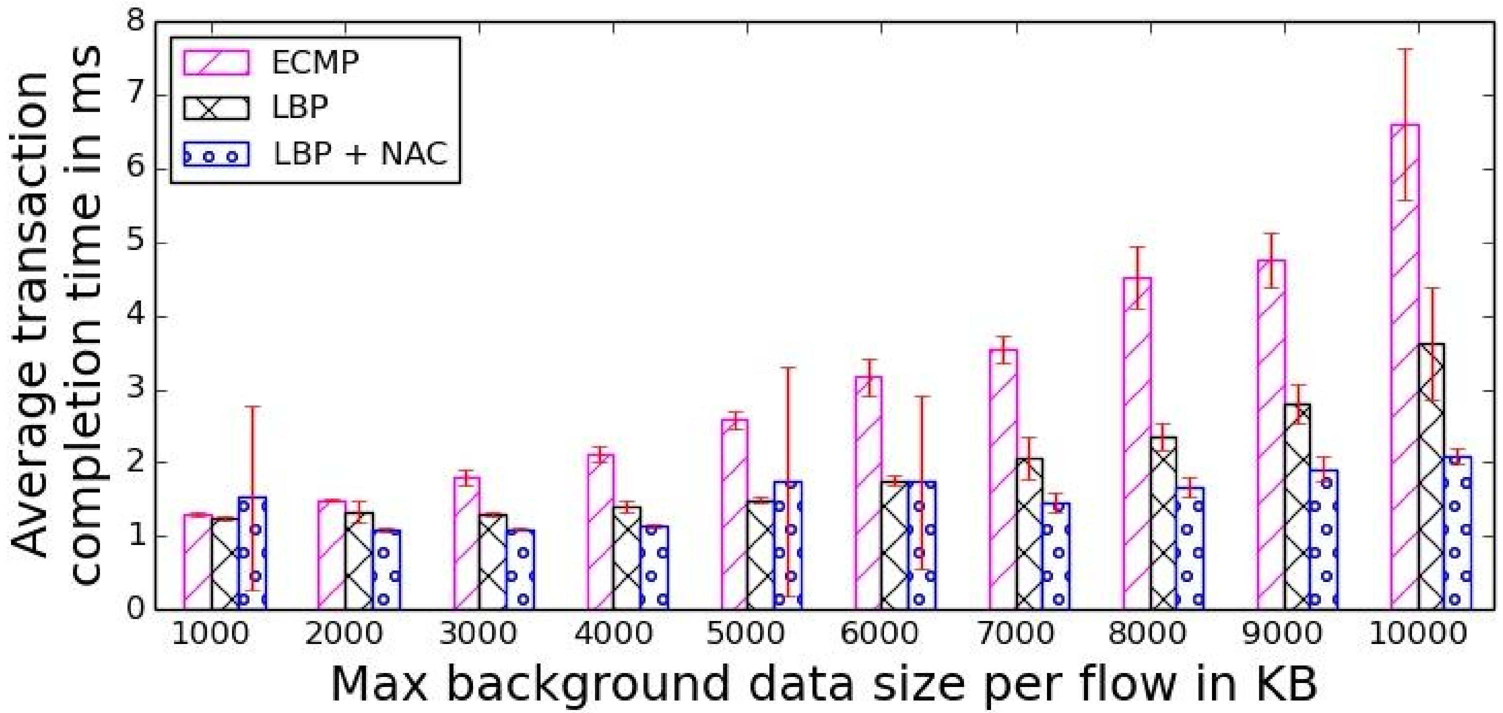
Experimental setup

- The controller runs on a dedicated machine
- We use a synthetic workload that performs read and write operations
- The key selection process follows a Zipfian distribution with a distribution constant of 0.99
- We use ECMP as a baseline for comparison

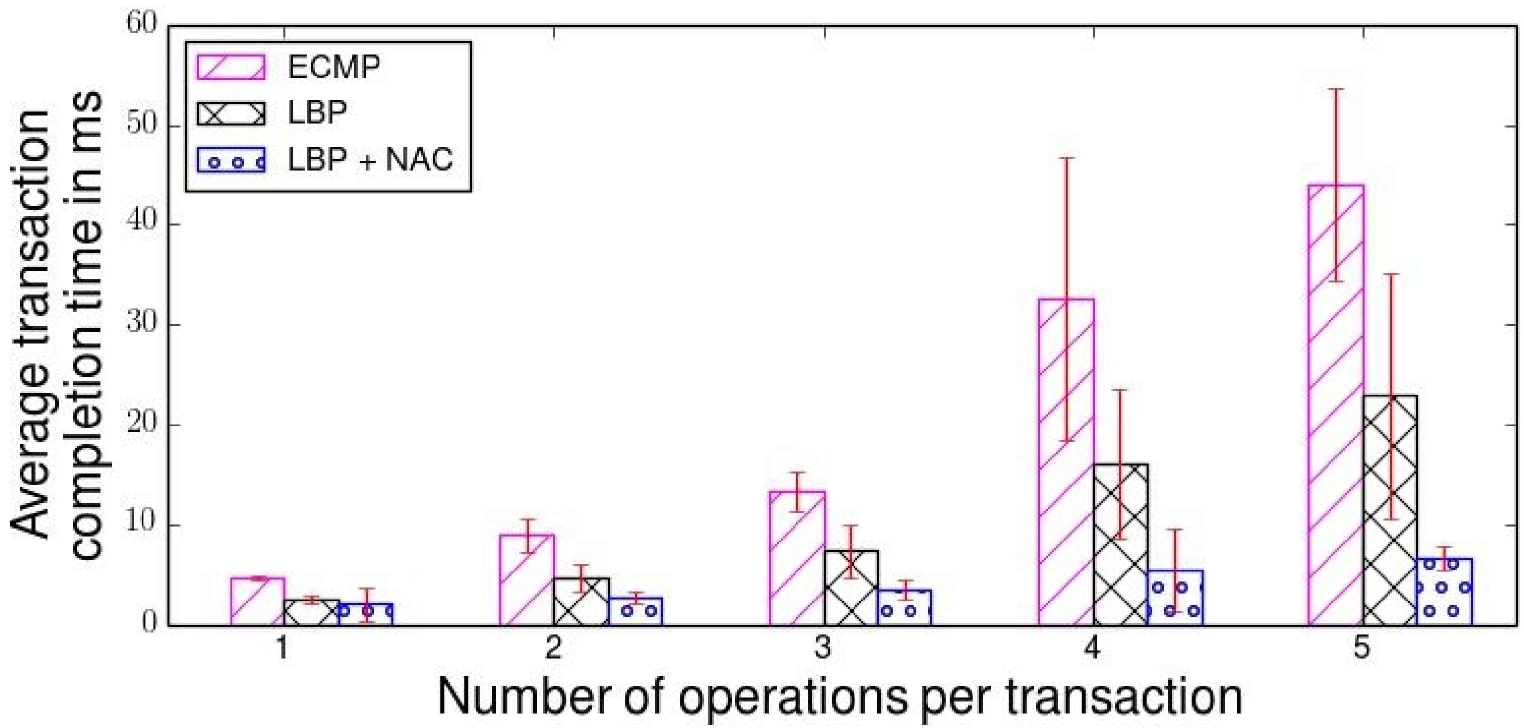
Experimental setup: default system parameters

NetStore parameter	default values
write transaction percentage	1%
foreground op data size	6KB
max background flow data size	8000KB
average fg interarrival time	50ms
average bg interarrival time	1000ms
# of key-value pairs in database	2,000,000
# of metadata cache entries in controller	20,000
# of cache entries in each data server	20,000

ECMP vs NetStore: varying the size of background flows



ECMP vs NetStore: varying number of operations in transactions



Conclusion

- We made the case for co-designing cloud applications with network optimizations to improve performance
- NetStore is distributed transaction processing system that offers network-aware optimizations
- NetStore significantly reduces average transaction completion time when parts of the network are saturated

Thank you.

Contact: xcui@uwaterloo.ca