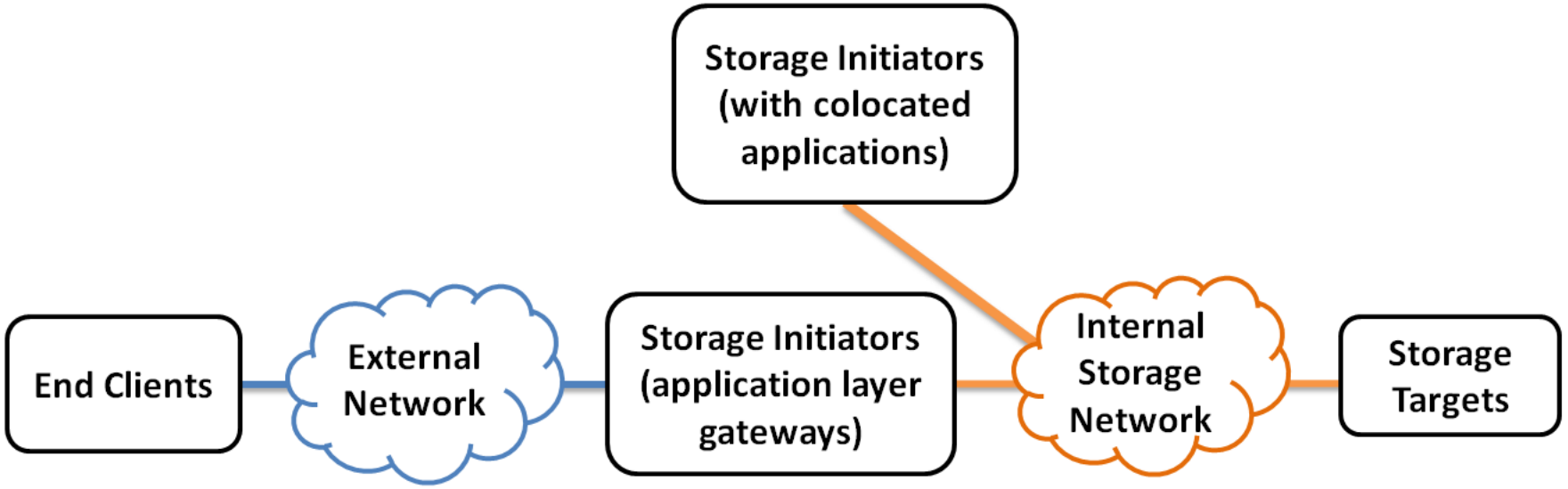
Scalable Object Storage

with Resource Reservations and Dynamic Load Balancing

Alex Aizman
Nexenta Systems

The Setup

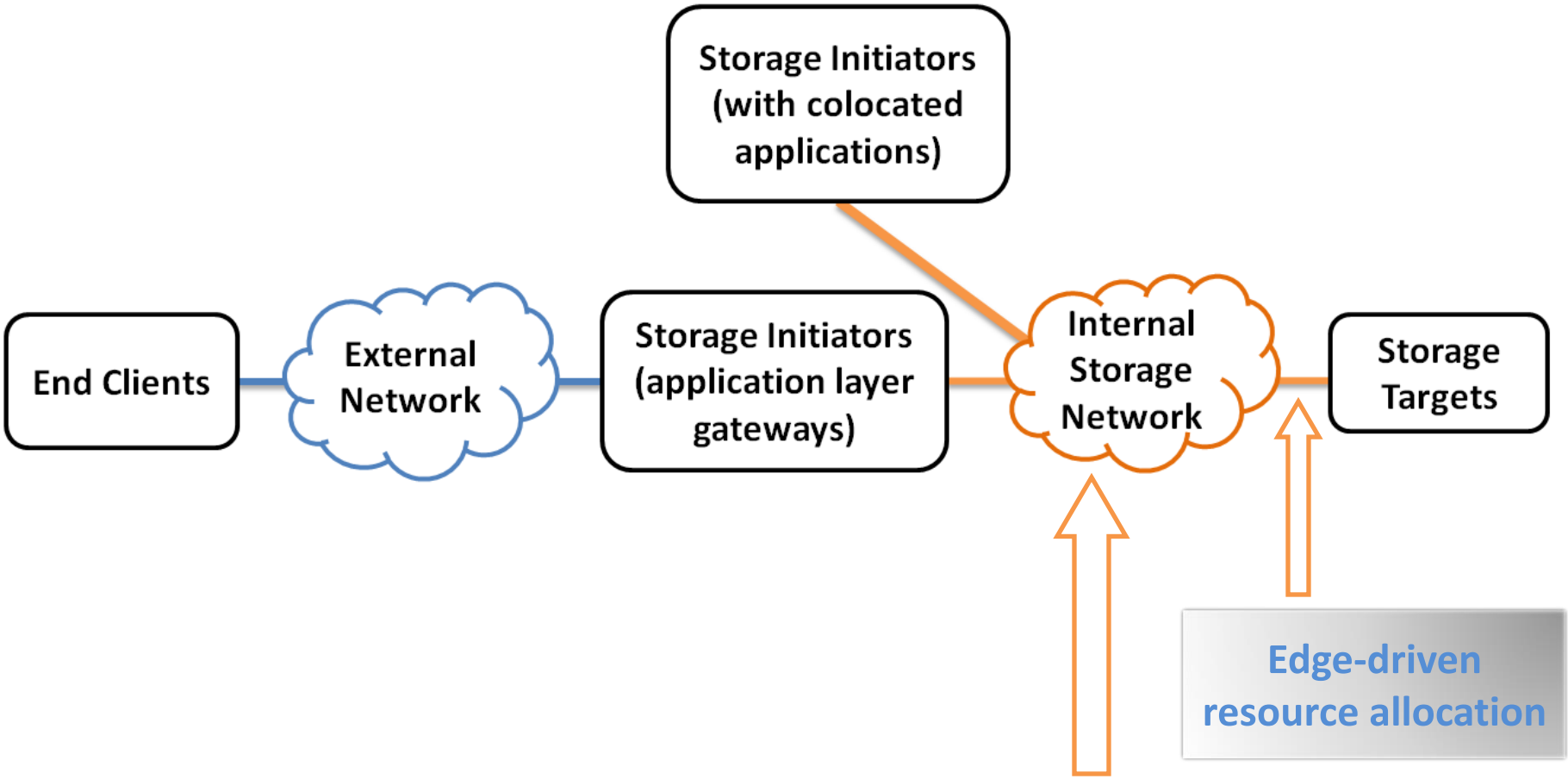


- Within Data Center
- Scale: 100+ nodes to unlimited
- Optimized for latency; no spikes at high utilization
  - No “fat tails”
- Layer 1 of storage stack is object
  - Storing and transporting immutable crypto-checksummed KVT

More Requirements

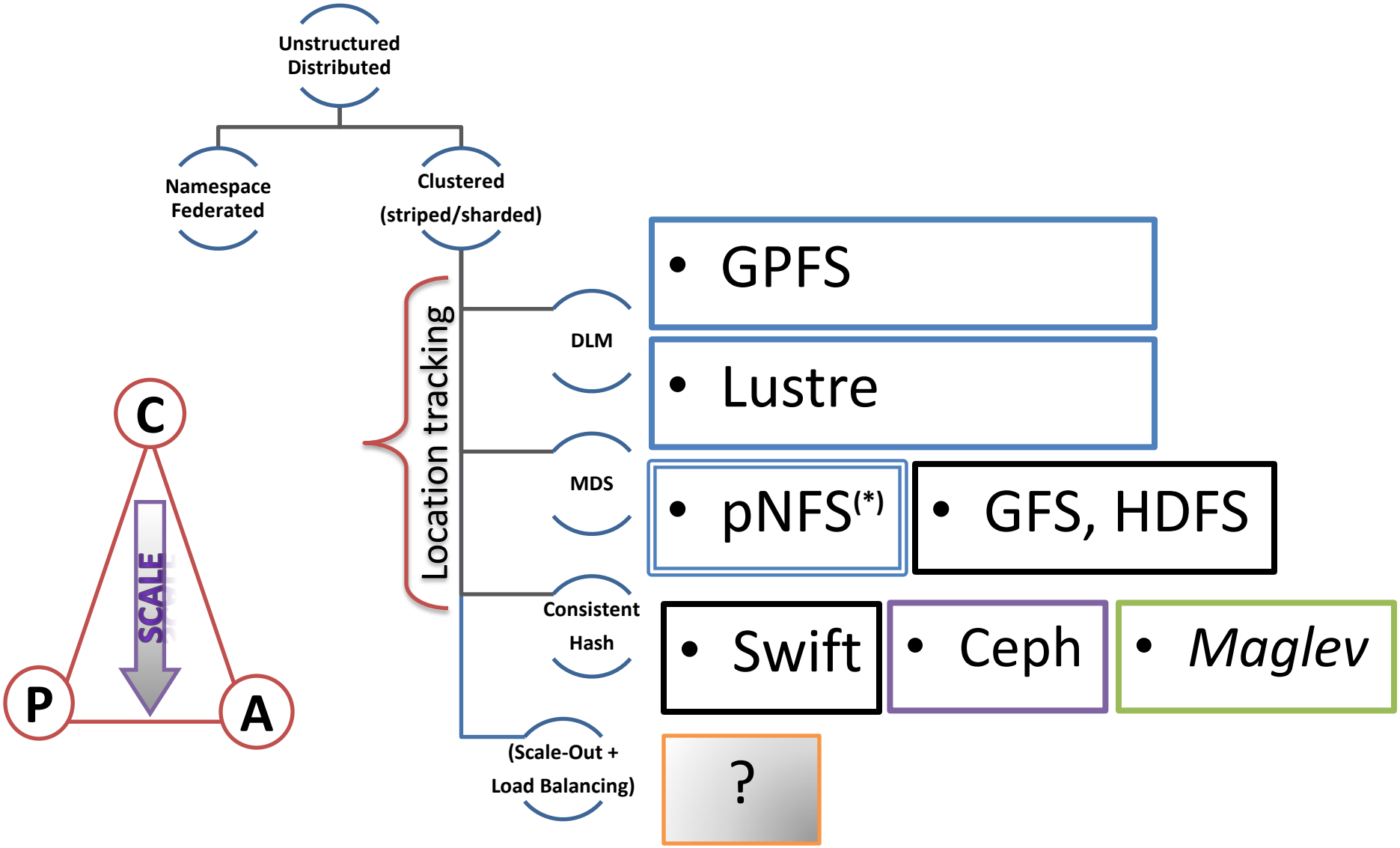
- Copy-on-write, eventually consistent
  - Put creates a new version
- Multiple replicas
  - Multiple replicas on the wire?
- “Rampant Layering Violation”
- No Incast
  - Mostly known as TCP Incast
- No/Minimized Convergence
  - Multiple link-sharing flows “converge” to fair share
- Linearly scalable and load balanced at all times
  - Uniform distribution != balanced distribution

The Claim

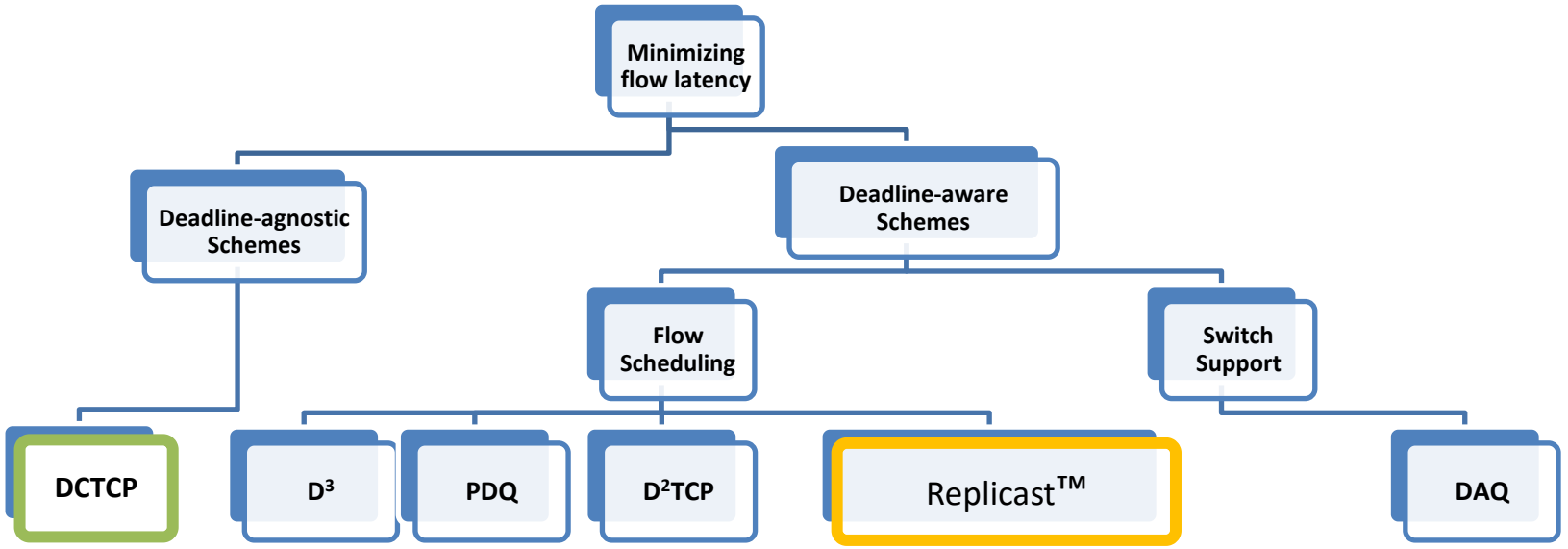


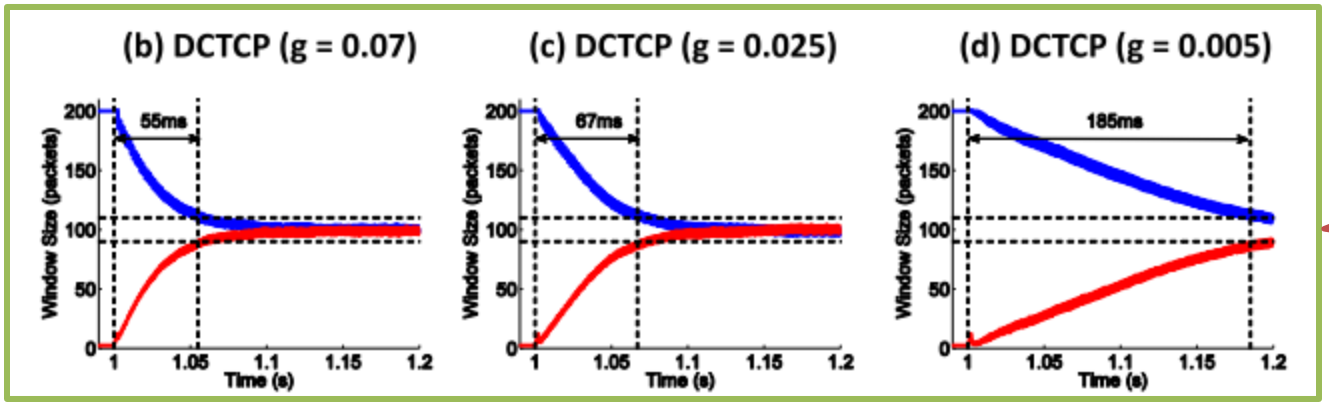
New Storage Protocol Required

Distributed clusters



DCN: transmission of short-lived flows



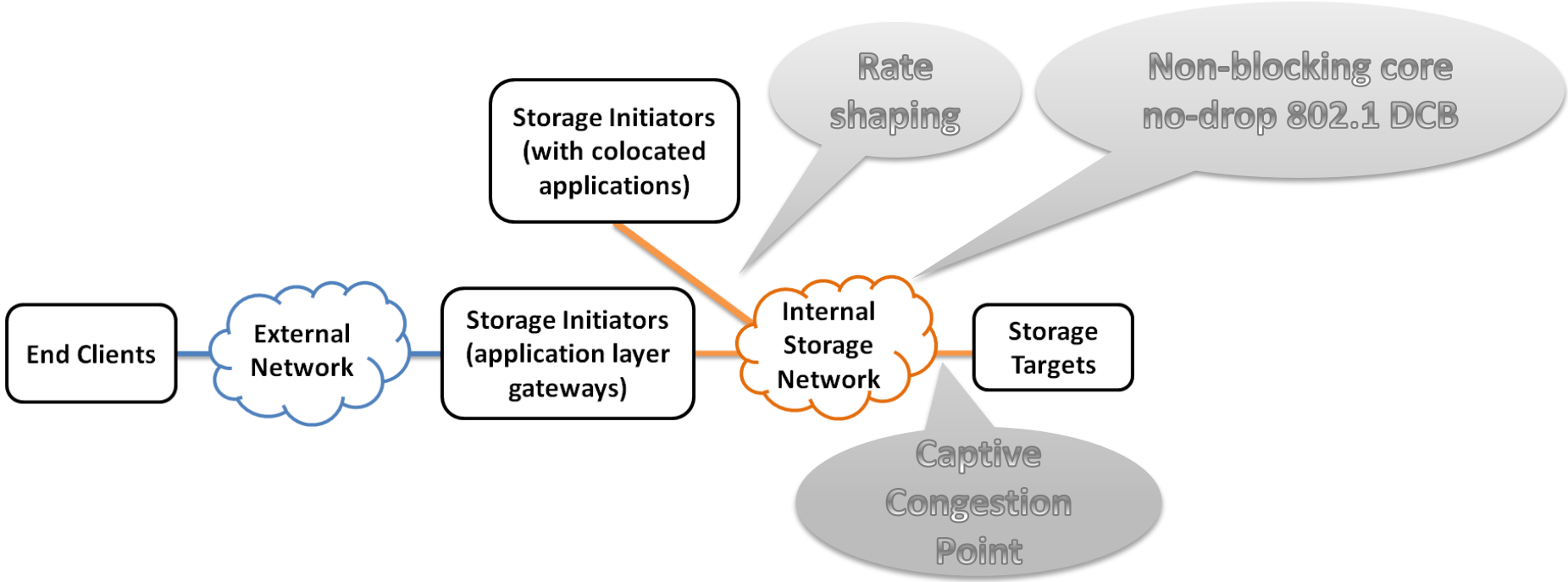


Connection setup and termination, both

(*) Schemes for Fast Transmission of Flows in Data Center Networks

(**) Analysis of DCTCP: Stability, Convergence, and Fairness

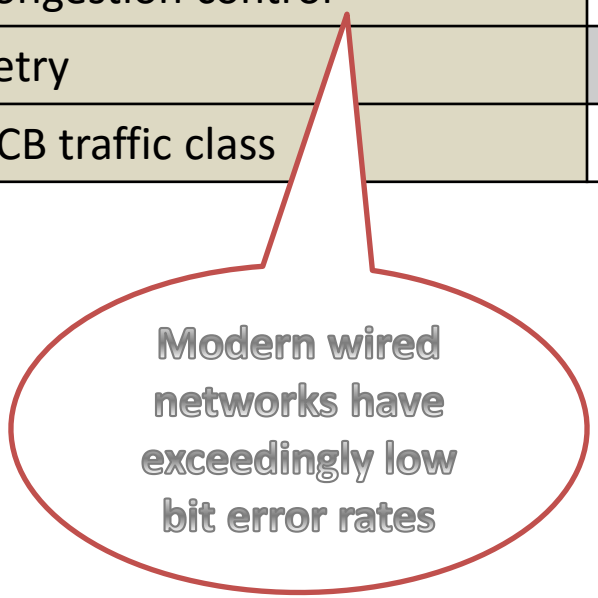
Congestion: give control to the target!



- Reserved bandwidth 100% utilized
  - Impact of one connection terminating?
  - Zero (or minimal) competition between flows
- Compare with SJF/EDF/PDQ..

Motivations: Transport

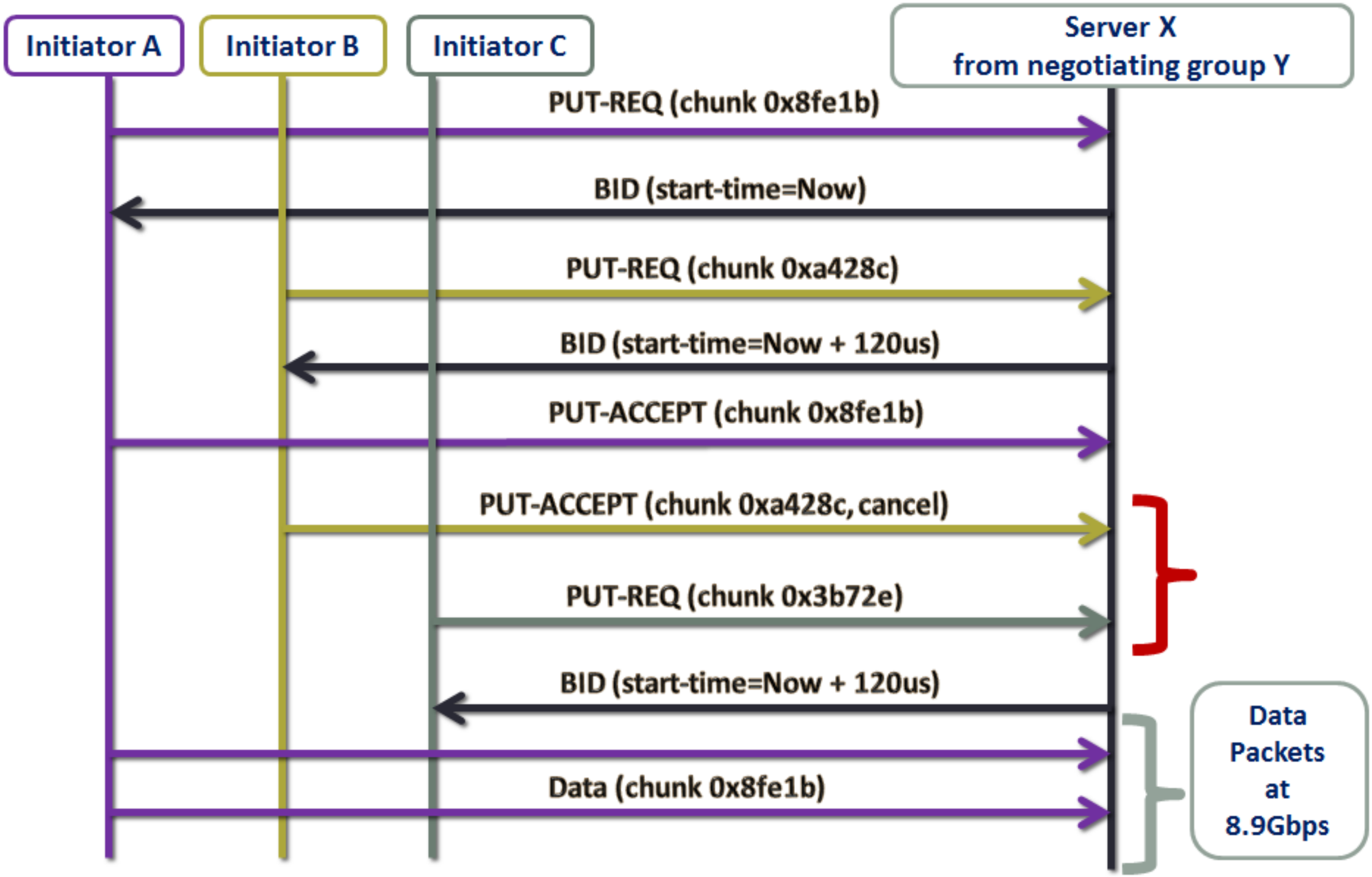
	L5 over TCP	Replicast
Performance	Throughput + fare-share	Completion time
General purpose	Yes	No
Multiple replicas on the wire	Yes	No
Mature and stable L4	Yes	No
(TCP) Incast	Yes	No
Congestion control	(L2) + L4	L2 + Replicast
Retry	L4	Replicast
DCB traffic class	Depending on the app	Yes



Motivations: Storage

	Replicast
Built-in deduplication	Yes
Consistent hashing + Inline load balancing	Yes
Target Resource reservation (Network, Disk)	Yes (Yes, Yes)

Replicast: edge-based load balancer

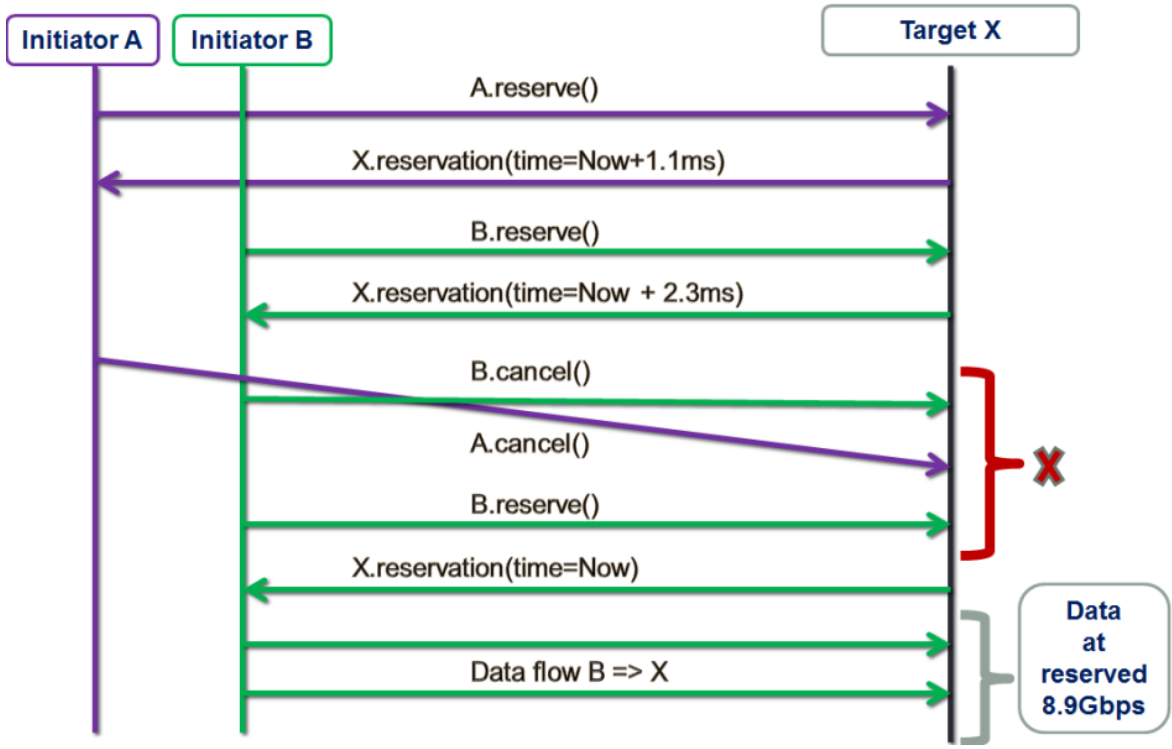


Tradeoffs – Protocol Variations

- There is always a cost and associated tradeoffs
- Replicast: all designated targets must share the timeslot

- Variations(*):
  1) Multicast control plane + unicast delivery
  2) Choosy Initiator
  3) The Better Protocol

- and more



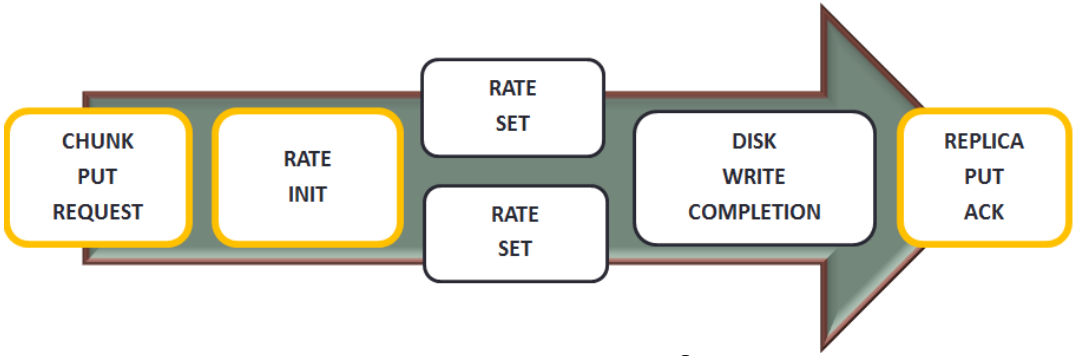
(*) https://storagetarget.com

Protocol Simulation

- Replicast is designed for 1000s of nodes
- SURGE framework @https://github.com/hqr/surge
- Each node is a goroutine; fully owns its configured resources
- Any-to-any connect via Go channels
- Time modeling

- Same-size payload chunks indexed by a cryptohash of their content
- And consistently hashed to: a) groups (Replicast), b) targets (unicast)
- Non-blocking no-drop network core that connects all 10GbE ports
- Flow isolation: protected VLAN
- Transmission errors are sufficiently rare and therefore not modeled
- *Reads are modeled but remain out of scope (and space)*

The “fair comparison” dilemma



- Unicast Consistent Hash, Captive Congestion Point
  - Consistent hashing for target selection
  - Unicast UDP for both control and data
  - Idealized bandwidth reservations: RATE INIT and RATE SET
  - Immediate start (as opposed to TCP slow start)
  - 3x lower connection-setup overhead vs. Replicast

Results

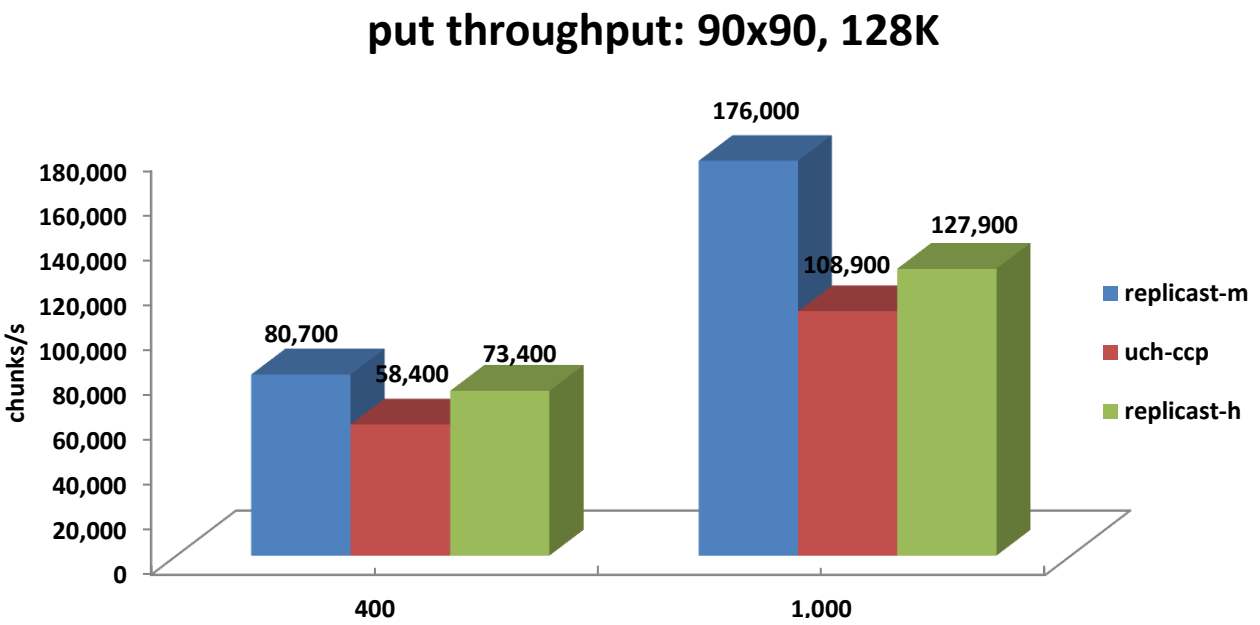
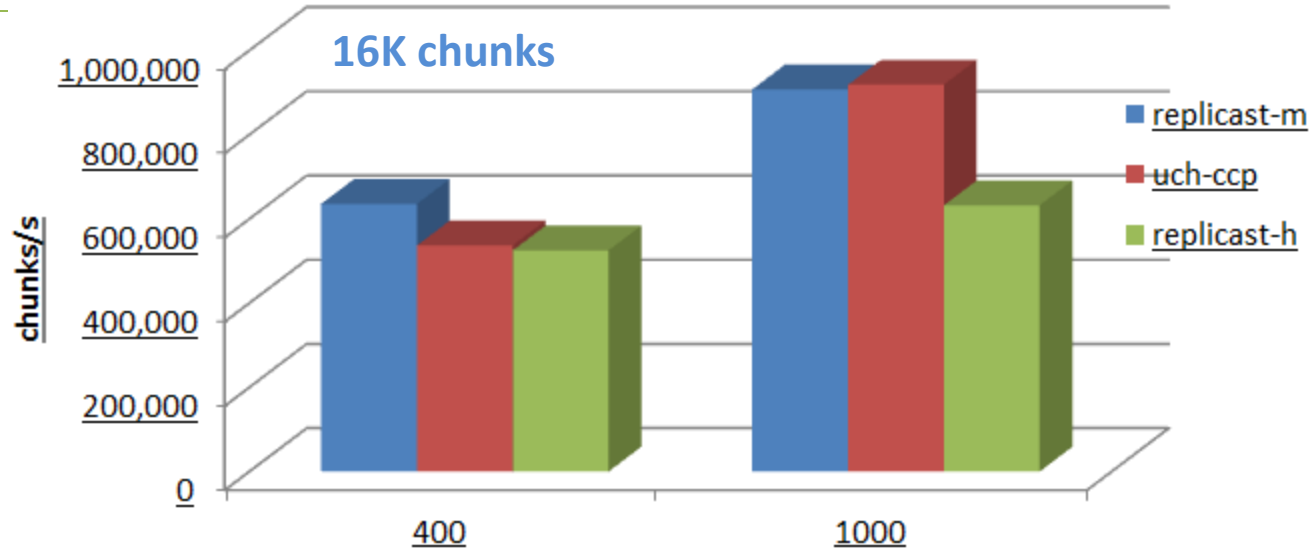


TABLE 2. 90 INITIATORS, 90 TARGETS, 400MB/S DRIVES

Chunk	uch-ccp	replicast-m	replicast-h
16K	536,950 c/s 70.6% 164.7us	635,750 c/s 83.6% 139.0us	525,100 c/s 69.0% 168.4us
128K	58,400 c/s 66.4% 1401.7us	80,700 c/s 90.6% 1039.8us	73,400 c/s 83.6% 1132.6us
1M	6,733 c/s 69.6% 10260.2us	8,067 c/s 85.6% 9210.8us	8,000 c/s 84.9% 8945.9us

Replicast: reservation conflicts

Chunk	Put interarrival time	λ	Poisson probability
16K	11us	0.09	46.7%
128K	50us	0.02	13%
1MB	500us	0.002	1.39%



Next Steps

- Optimizations for small chunks
- Optimizations for concurrent gets and puts
- Optimal ratios of initiators to targets
- Optimal sizing of the load-balancing groups
- Load balancing proxies
- Kernel bypass (DPDK)
- Bit Index Explicit Replication (BIER)
  - Stateless multi-point replication

Instead of conclusions: Guiding Principles

- Location independence: both chunks and MD
- No SPOF (no single-MDS, at least on this level)
- Inline load balancing | Inline global dedup
- Storage-level end-to-end resource reservation
- 100% bandwidth utilization
  - During the reserved timeslot
- Single copy on the wire
  - If possible
- Close-to-open, ACID/transactional and other types of consistency – by upper layers
- and more

Credits: Caitlin Bestler

Thank You