Hydra: Leveraging Functional Slicing for Efficient Distributed SDN Controllers

Yiyang Chang, Ashkan Rezaei, **Balajee Vamanan**,
Jahangir Hasan, Sanjay Rao and T. N. Vijaykumar

UIC

PURDUE UNIVERSITY

Google

SDN is becoming prevalent

- Software defined Networking (SDN) becoming prevalent in datacenter and enterprise networks
  - ✓ Centralized state → Fine-grained management
    - high network utilization
  - ✓ Wide adoption in industry WANs
    - Google (B4, *SIGCOMM* '13)
    - Microsoft (SWAN, SIGCOMM'13)
- Consolidate state at a central controller
  - Single physical controller for small networks
  - Distributed implementation for large networks

Heterogeneity in SDN applications

- SDN applications place varying demands on underlying machine
  1. *Real-time*: periodically refresh state
     - e.g., heart-beats, link manager
     - deadline driven, light load
  2. *Latency-sensitive*: invoked during flow setup
     - e.g., path lookup, bandwidth reservation, QoS
     - latency sensitive, medium load
  3. *Computationally-intensive*: triggered during failures
     - e.g., shortest path calculation
     - affect convergence, heavy load

Distributed controller must handle both network size and application heterogeneity

Previous work: topological slicing

- Conventional approach: *topological slicing*
  - partition network topology: one physical controller for each network partition
    - all network functions run in each partition
- ✘ *one-size-fits-all*: all apps use same partition size
- Topological slicing co-locates all applications
  - Computationally-intensive apps susceptible to load spikes
  - ✘ affects co-located real-time/latency-sensitive apps
- ✘ Administrative constraints on partition sizing

topological slicing is agnostic of application heterogeneity and does not scale well

Hydra's contributions

1. *Functional slicing: split control plane applications across servers*
   - ✓ adds new dimension to partitioning, more freedom for placement
   - ✘ Increase latency if critical paths span multiple servers
2. Communication-aware placement
   - cast as optimization problem
     - ✓ multi-constraint graph partitioning
3. Shields real-time apps from other apps
   - uses thread prioritization

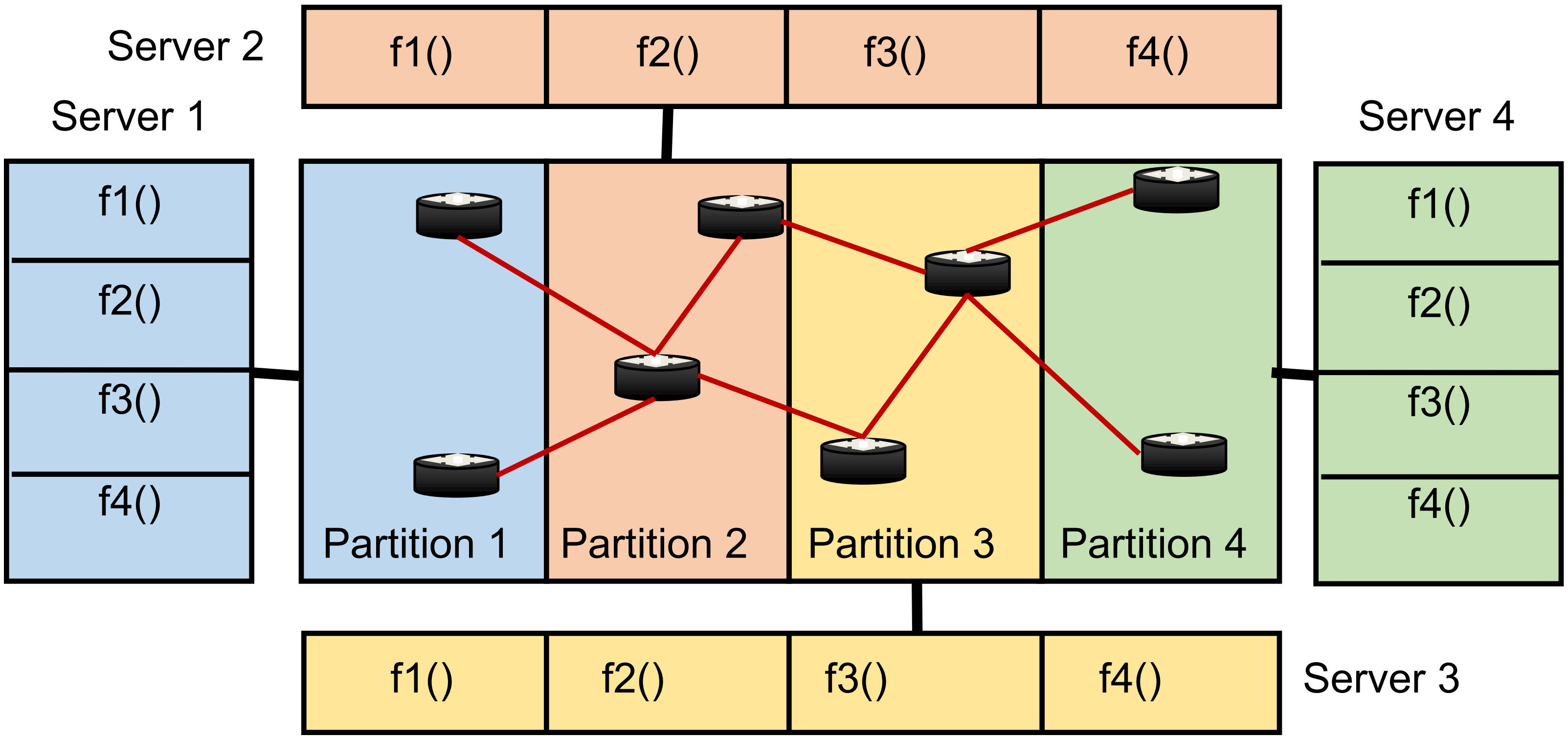
Hydra shows better performance for each app category and scales better than topological slicing

Outline

- Introduction
- **Background**
- Hydra
  - Goals
  - Functional slicing
  - Hybrid of functional and topological slicing
  - Communication-aware placement
- Key results
- Conclusion

Topological slicing

- Network is partitioned into multiple controller domains
- Each controller instance hosts all control-plane apps
  - but handles events *only* from switches in its partition



Shortcomings of topological slicing

- Sustainable throughput limited
  - compute/memory capacities of server must be sufficient to handle *all* apps
- Possible solution: increase # of partitions
- Increasing # of partitions causes other problems
  - ✘ Worsens convergence
    - time to recalculate paths during link/switch failure
  - ✘ Slow updates → Write-heavy apps suffer

Topological slicing co-locates all apps → requires higher # of partitions → affects scalability

Outline

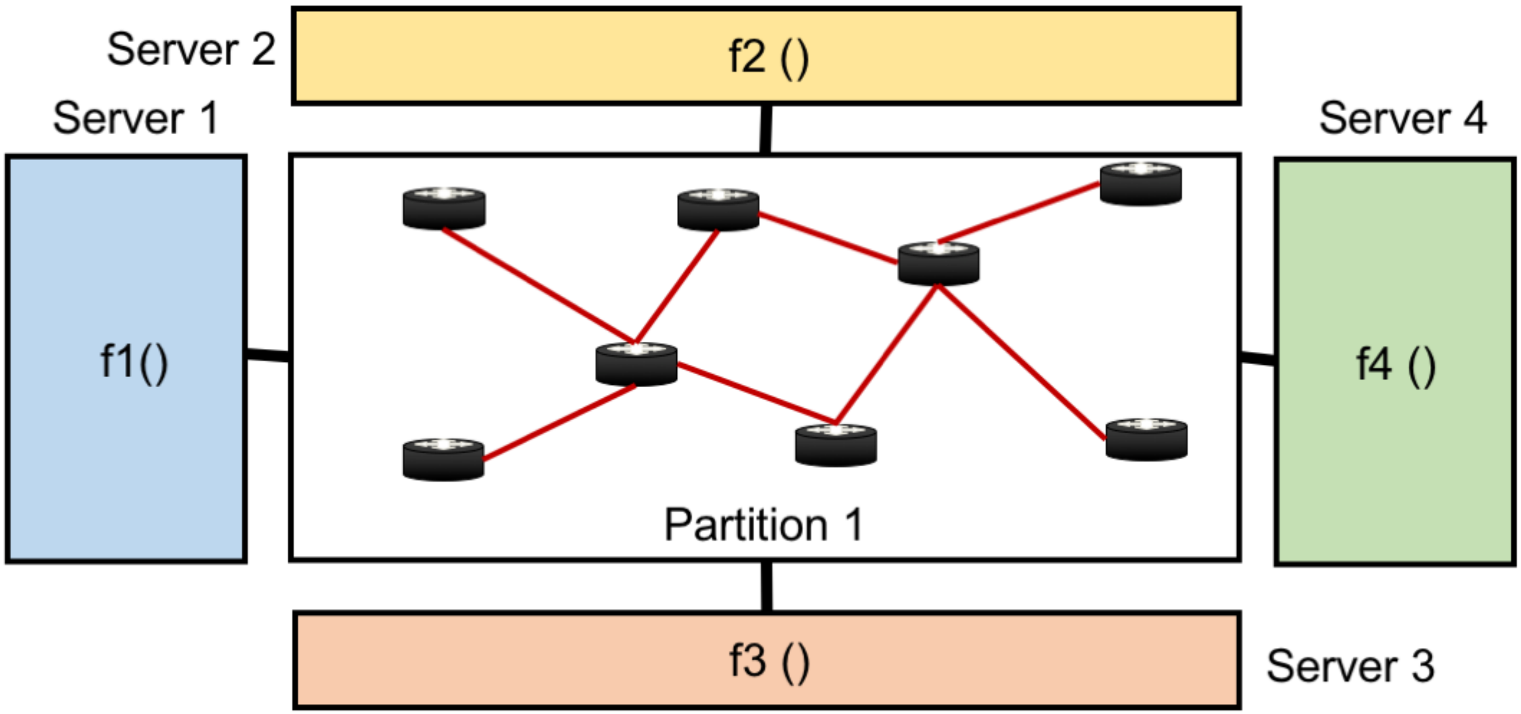
- Introduction
- Background
- **Hydra**
  - Goals
  - Functional slicing
  - Hybrid of functional and topological slicing
  - Communication-aware placement
- Key results
- Conclusion

Hydra: goals and techniques

1. Partitioning must help scalability without worsening state convergence
   - **Functional slicing**
2. Place applications without worsening latency
   - **Communication-aware placement**
3. Isolate real-time applications from load spikes
   - **prioritize** real-time apps over other apps

Functional slicing

- Split control-plane apps among multiple servers



✓ Better convergence (apps have complete network state)

✗ Increases latency for events spanning >1 app

- e.g., handling a **`packet-in`** might could involve inter-controller communication

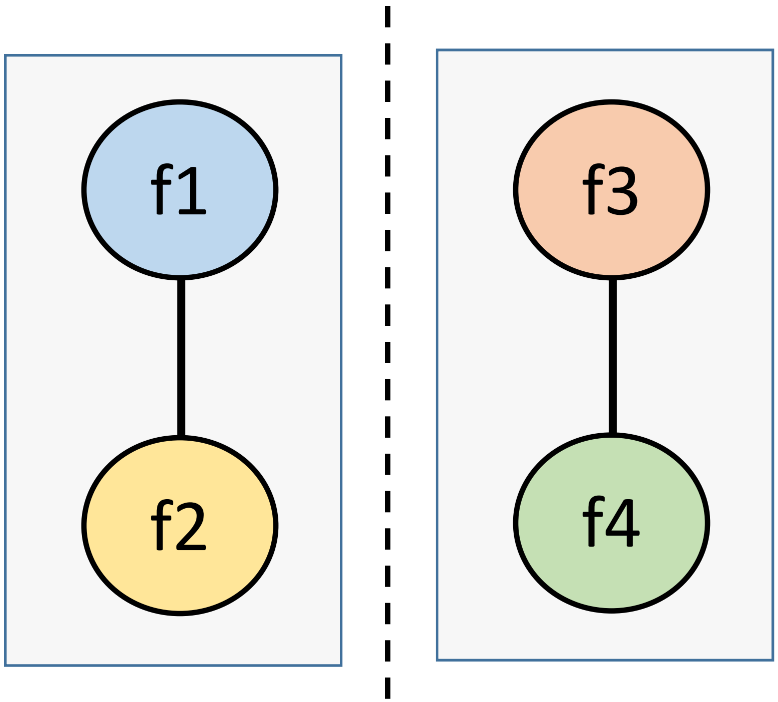
Communication-aware placement

- **Step-0**: find partition size based on critical app(s)
  - Typically, topology application is critical
- Determine app slice <> server mapping
  - Objective: minimize latency
  - subject to capacity and communication constraints
- Inputs
  - applications' CPU, memory demand
  - aggregate server CPU and memory capacity
  - communication graph
- compute-intensive apps (e.g., topology app) placed in separate machines to avoid interference

Mathematical formulation in the paper

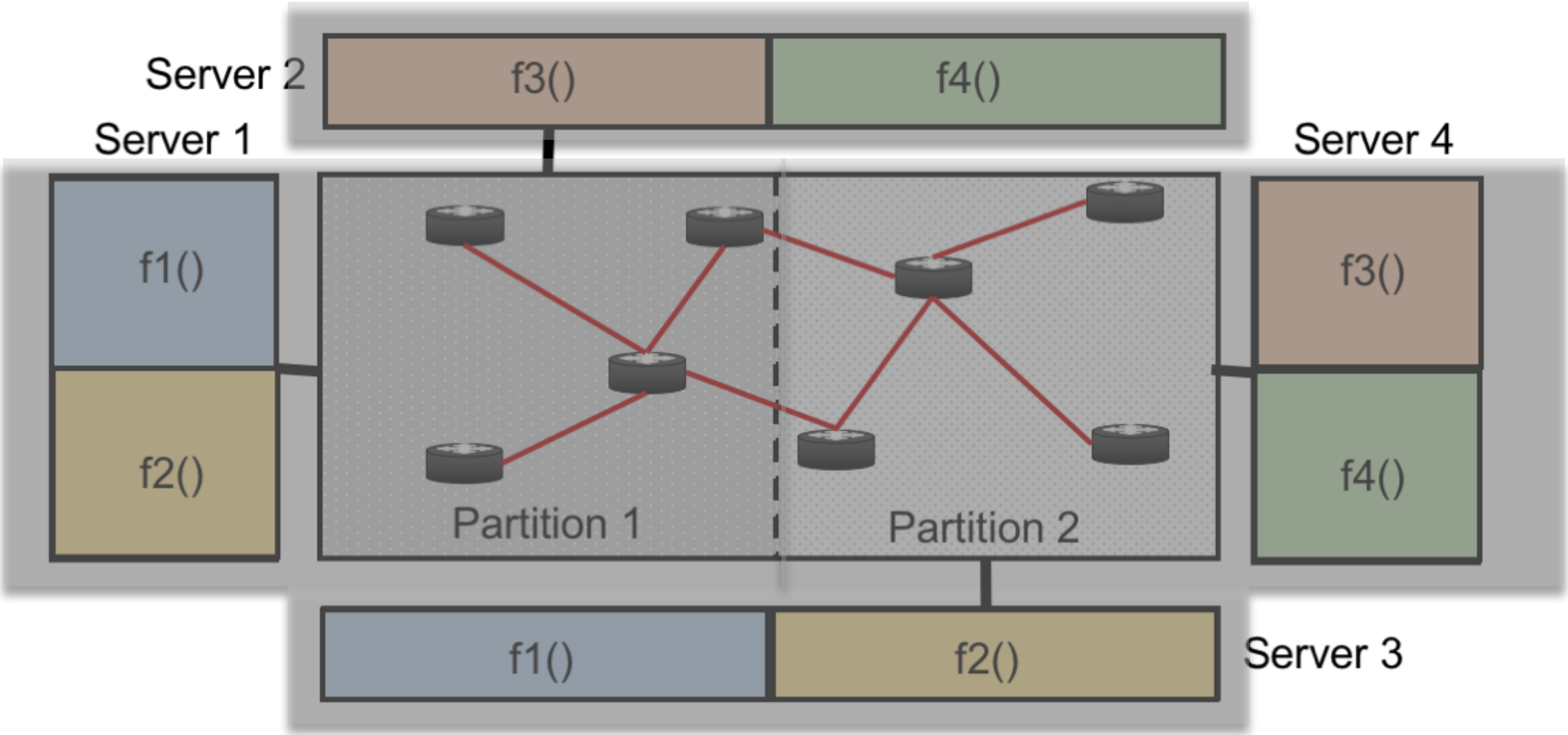
Communication-aware placement (cont.)

- An instance of *multi-constraint graph partitioning*
  - a known NP hard problem
  - use existing heuristics to solve in reasonable time



Hydra's approach: hybrid of functional and topological slicing

- hybrid of topological and functional slicing



Communication-aware placement achieves better convergence *without* increasing latency

Odds and ends

- Our model can be extended to accommodate
  - dynamic load changes
  - replicated controllers (fault tolerance)

Details in the paper

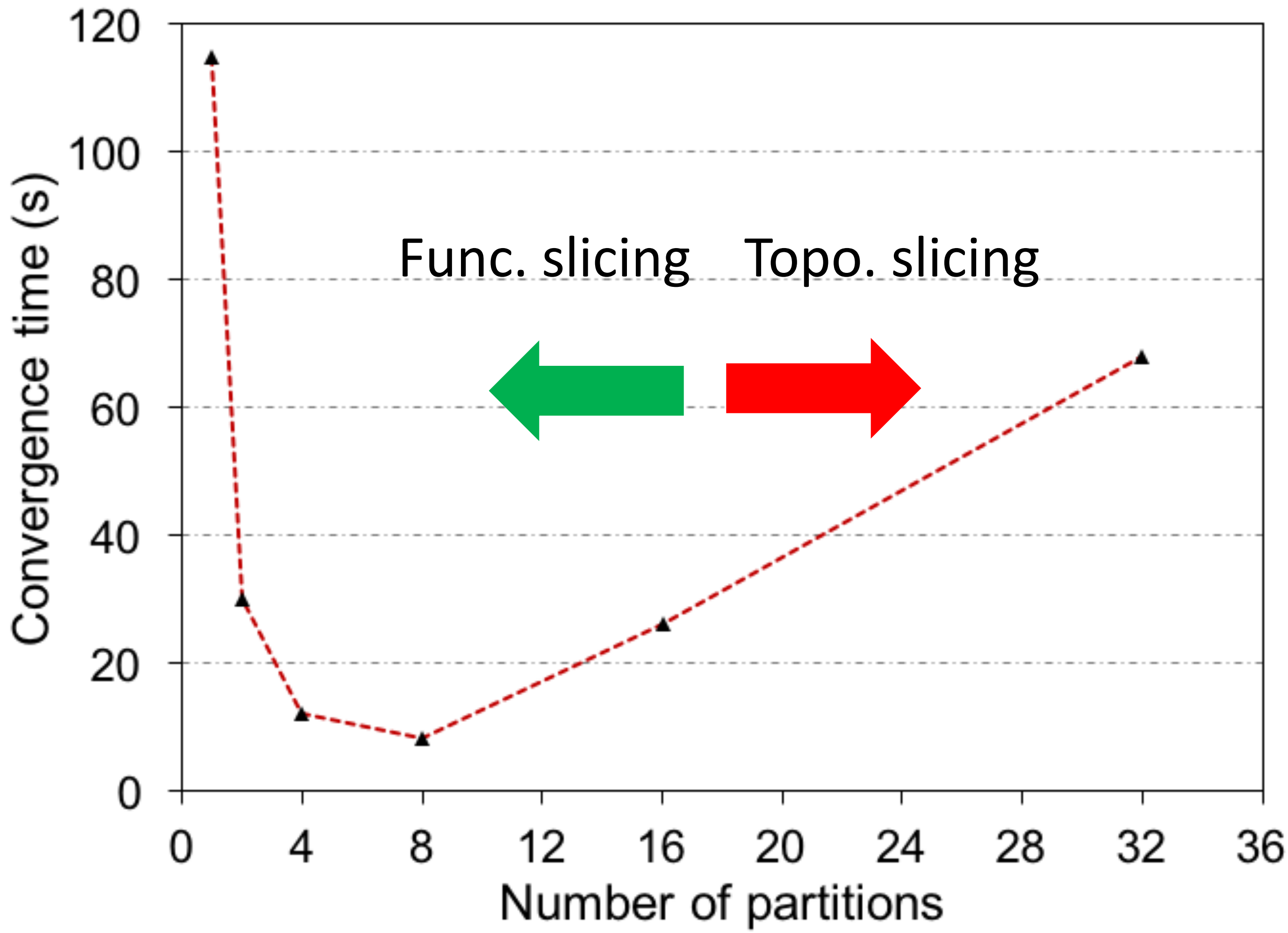
Outline

- Introduction
- Background
- Hydra
  - Goals
  - Functional slicing
  - Hybrid of functional and topological slicing
  - Communication-aware placement
- **Key results**
- Conclusion

Methodology

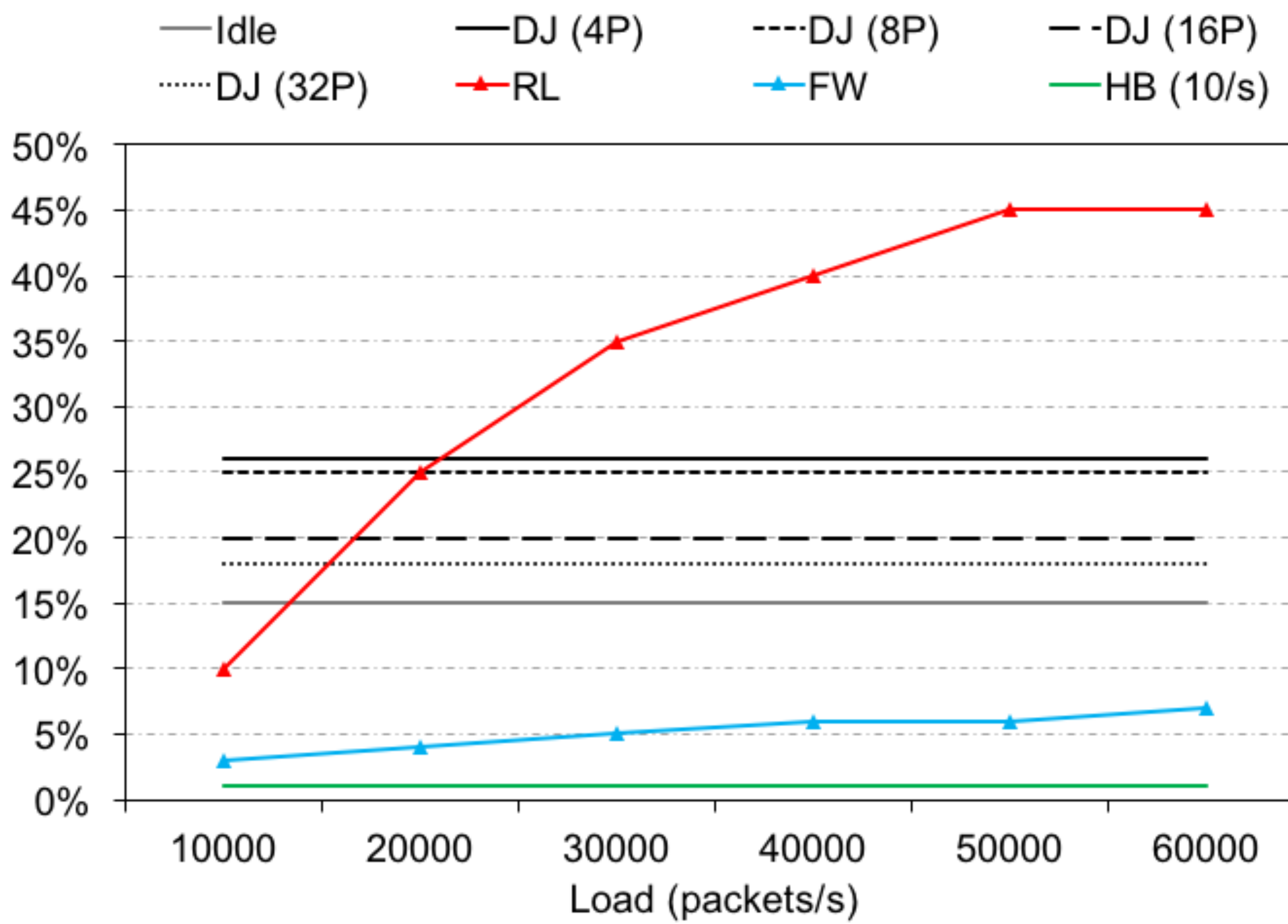
- *Floodlight* controller
- Apps
  - Dijkstra's shortest path computation (DJ)
  - Firewall (FW)
  - Route lookup (RL)
  - Heartbeat handler (HB)
- Modified *CBench* → load generation
  - *Mininet* models control plane → doesn't scale
- Topology
  - Datacenter network → fat-tree
    - ~2500 switches

Convergence time



Topological slicing requires higher # partitions → worsens convergence

CPU demand

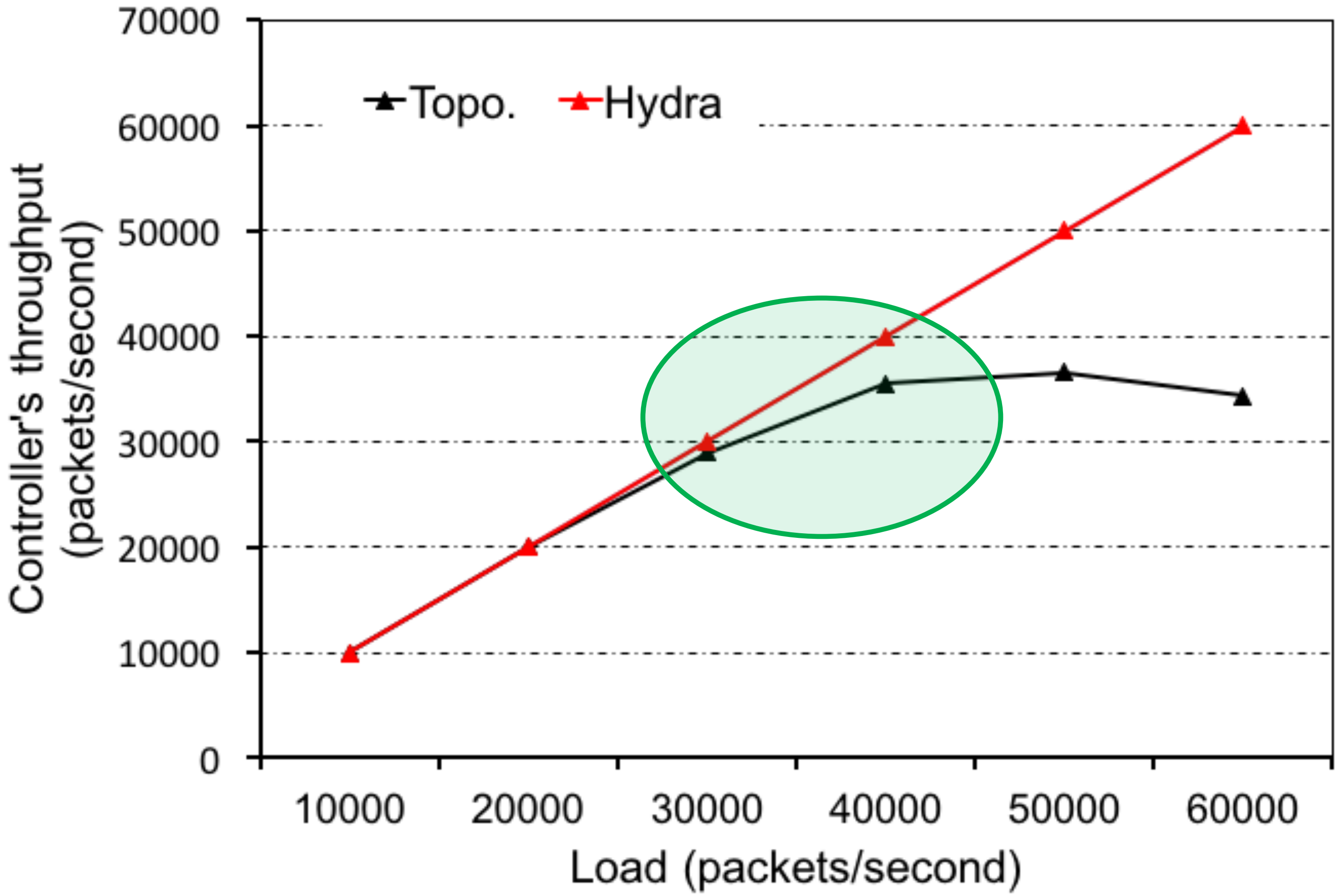


DJ's demand **depends on # partitions;**
other apps sensitive to network load (`packet-in`)

Placement results

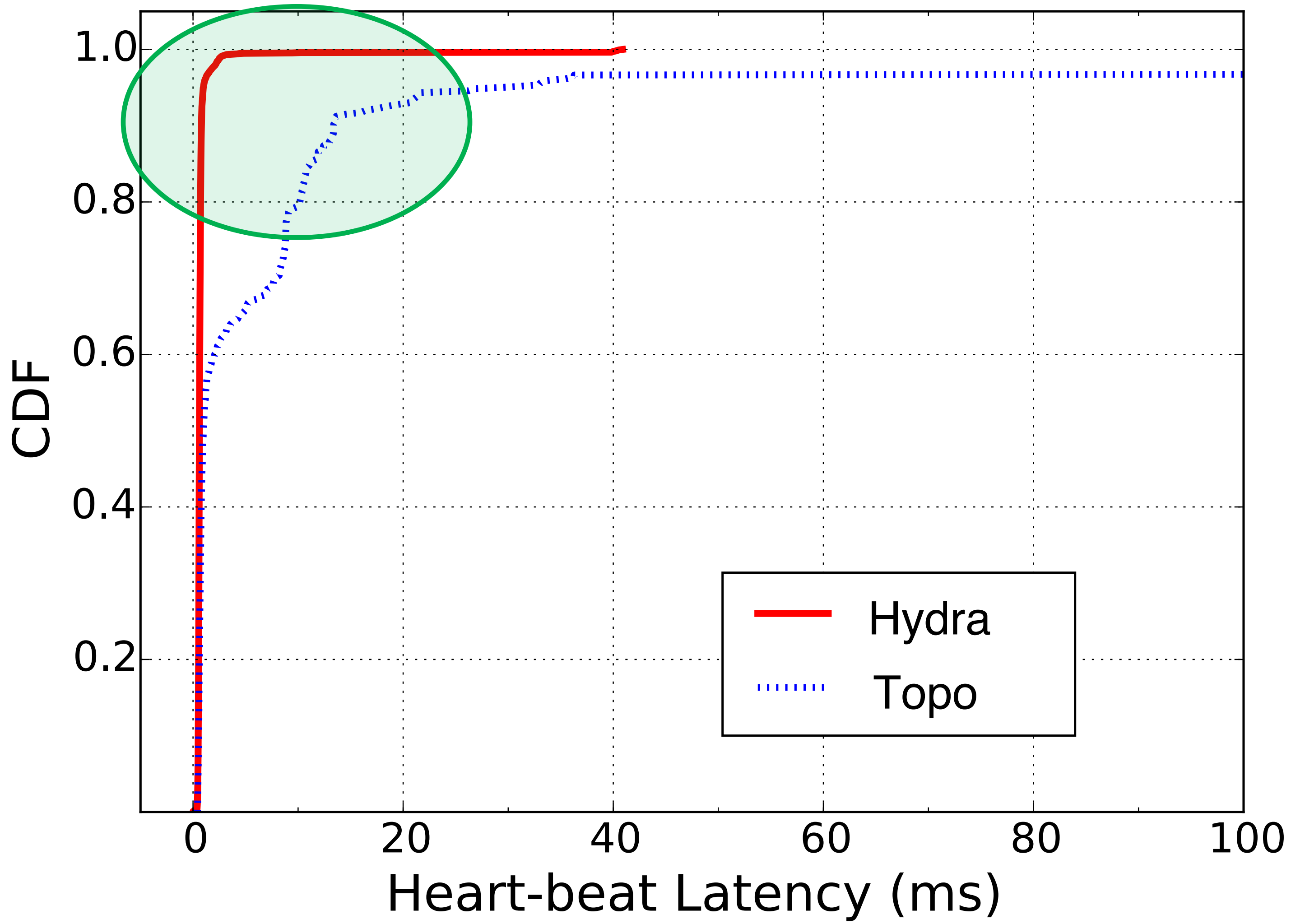
- Available capacity: 4 servers, 4 cores per server
- Topological partitioning co-locates all apps → requires >= **16** partitions
  - 16 partitions = 16 controller instances = one per core
- Functional slicing reduces demand → *Hydra* requires fewer than 16 partitions (i.e., **8**)
  - each partition hosts two controller instances
    - *Packet-in* pipeline: communication between RL and FW → one for DJ, one for {RL, FW, HB}
    - HB is prioritized over other apps

Hydra's scalability



***Hydra* scales better than topological slicing by isolating compute-intensive apps from other apps**

Real-time performance



***Hydra's thread prioritization* isolates heart-beats from other apps**

Conclusion

Hydra is framework for distributing SDN functions

- Incorporates *functional* slicing
- Communication-aware placement
- Thread prioritization
- Results show importance of Hydra's key ideas

- Future work
  - Infer communication graph using program analysis
  - Incorporate apps' consistency requirements into model

Hydra's gains potentially higher in large scale deployments