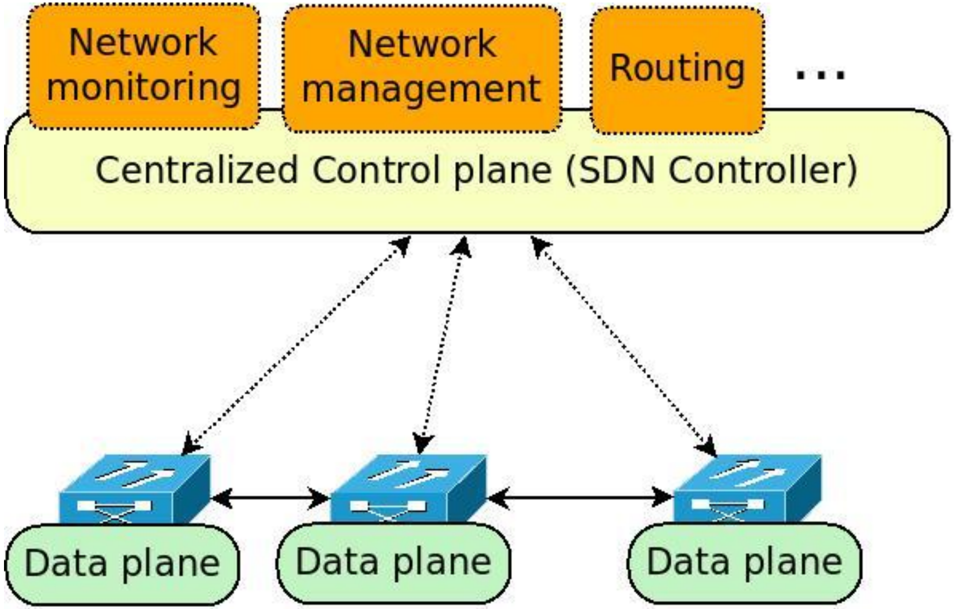
Cuttlefish: Hierarchical SDN Controllers with Adaptive Offload

Rinku Shah
Mythili Vutukuru
Purushottam Kulkarni

Department of Computer Science & Engineering
Indian Institute of Technology Bombay

ICNP 2018
September 26, 2018

What is Software-defined networking?



- **Software-Defined Network**
  - **Decouple** Control plane and Data plane

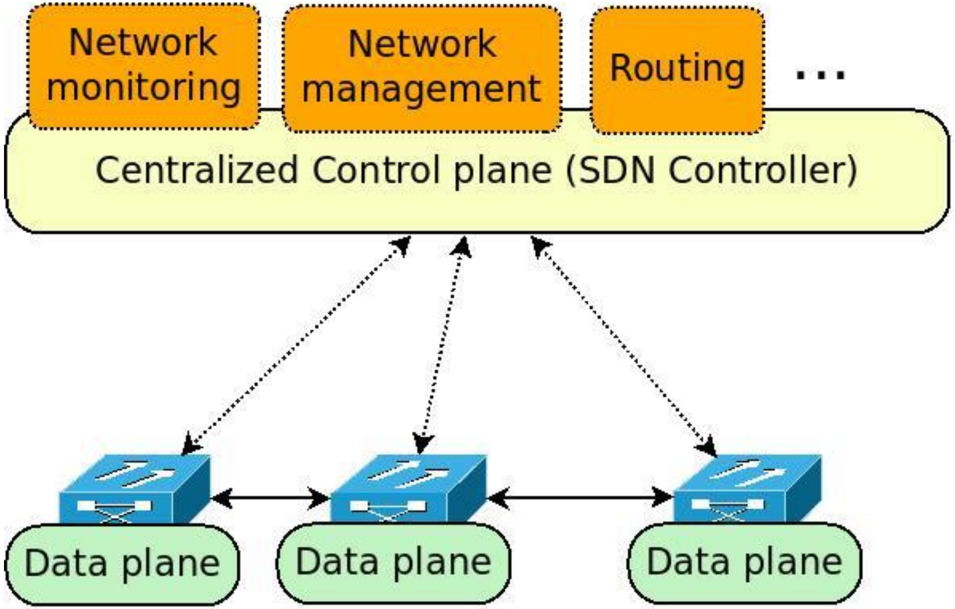
SDN BENEFITS

- Network state is **logically centralized**
  - Central network configuration and management possible
- **Network programmability**
  - Custom protocols on hardware switches

SDN CHALLENGES

- Scalability
- Security
- ...

What is Software-defined networking?



- **Software-Defined Network**
  - **Decouple** Control plane and Data plane

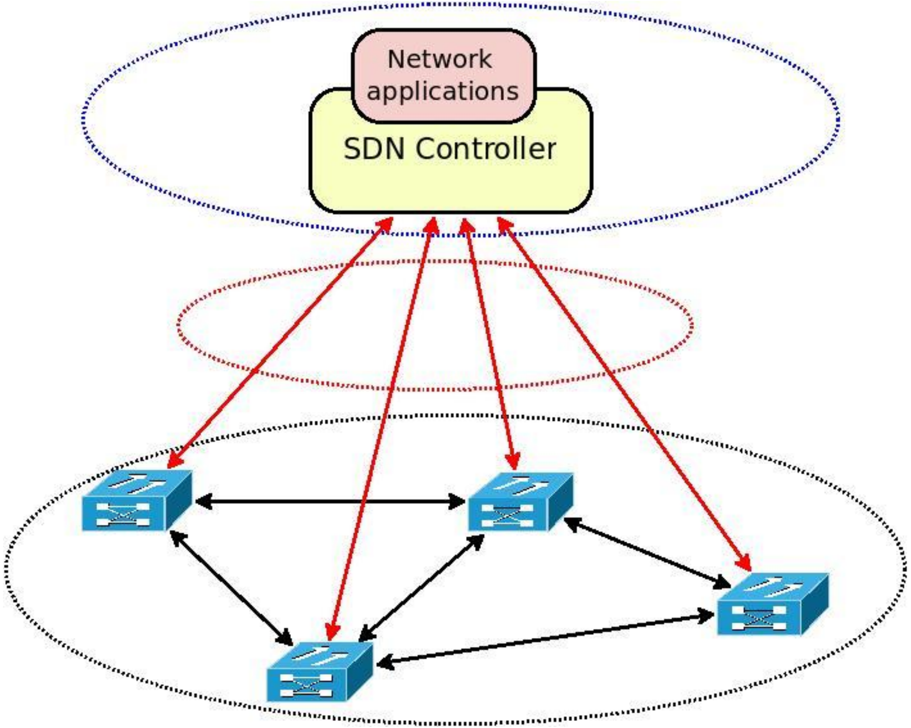
SDN BENEFITS

- Network state is **logically centralized**
  - Central network configuration and management possible
- **Network programmability**
  - Custom protocols on hardware switches

SDN CHALLENGES

- **Scalability**
- Security
- ...

SDN Scalability Problem: Bottleneck Domains



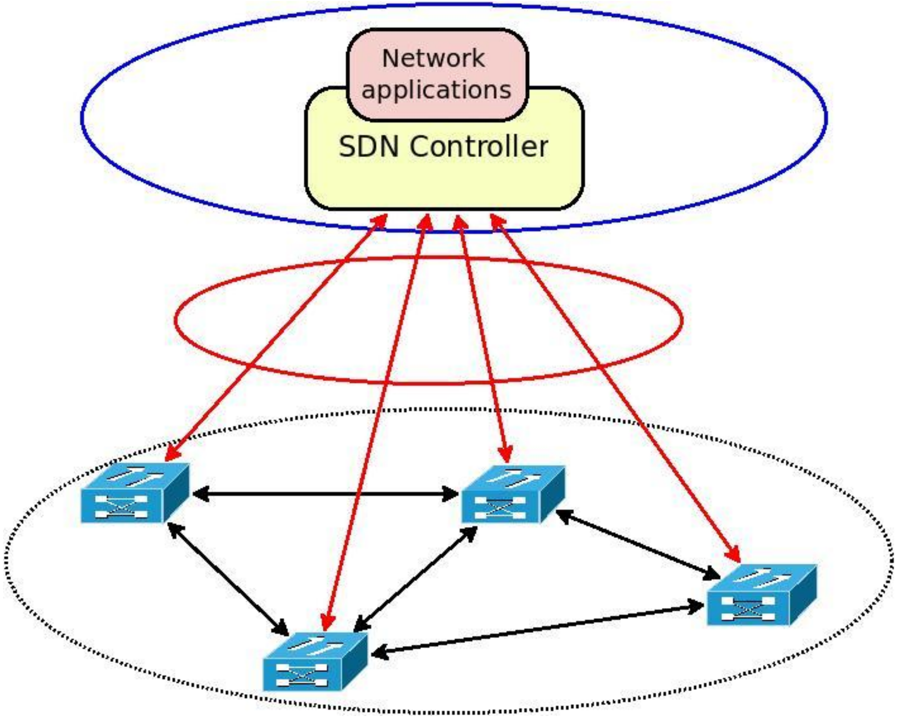
Controller Bottleneck

Switch-Controller Communication Bottleneck

Data-plane Bottleneck

SDN controller **Scalability** is a **vital** requirement to reap **SDN benefits**

SDN Scalability Problem: OUR FOCUS



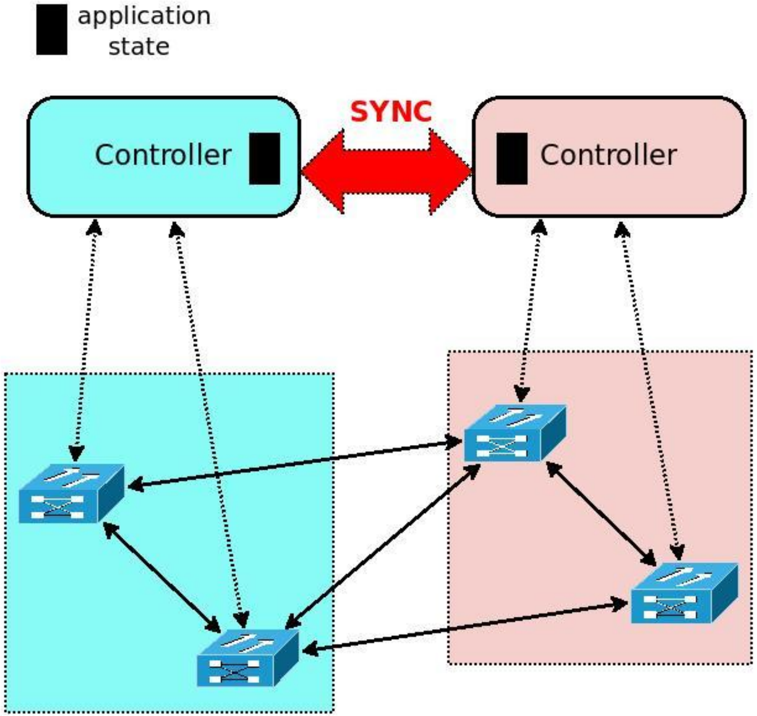
Controller Bottleneck

Switch-Controller Communication Bottleneck

Data-plane Bottleneck

SDN controller **Scalability** is a **vital** requirement to reap **SDN benefits**

Horizontal Scaling

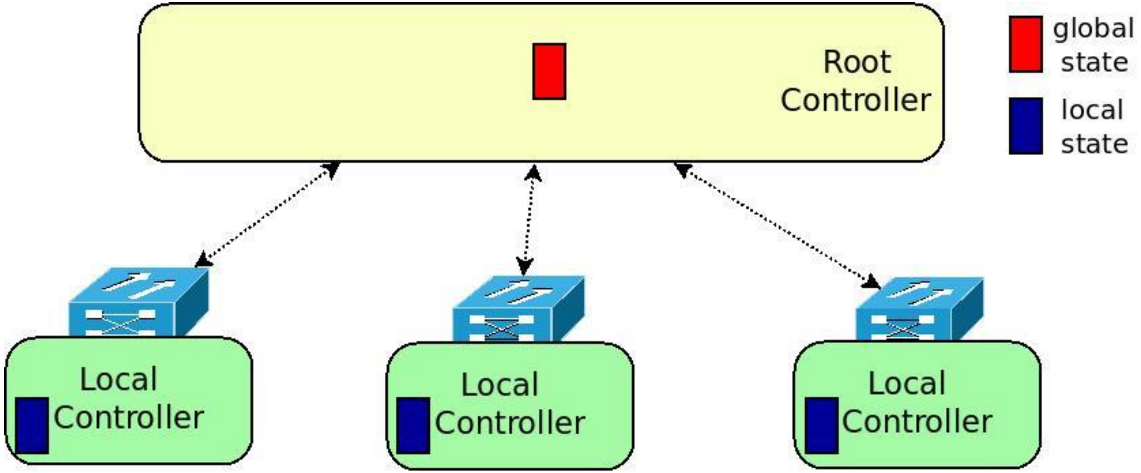


- ❑ **Single** physical controller to **multiple** controllers
- ❑ Each controller manages **subset** of the network topology
- ❑ Need for **synchronization** between controllers
- ❑ **Application state examples**
  - ❑ Topology information
  - ❑ Flow statistics at each switch

Onix[1]
Hyperflow[2]
Beehive[3]

[1] Teemu Koponen and others. Onix: A Distributed Control Platform for Large-scale Production Networks. In Proc of the Conference on OSDI, 2010.
[2] Amin Tootoonchian and Yashar Ganjali. HyperFlow: A Distributed Control Plane for OpenFlow. In Proc of the Internet Network Management Conference on Research on Enterprise Networking, 2010.
[3] S. H. Yeganeh and Y. Ganjali, “Beehive: Simple distributed programming in software-defined networks,” in Proc. of the Conference on SoSR 2016.

Hierarchical Scaling



Limited Applicability

- **Split computations** amongst **root** and **local** controller
- Application state classified as
  - **GLOBAL**
  - **LOCAL**
- GLOBAL state example:
  - Network topology
- LOCAL state example :
  - Flow statistics

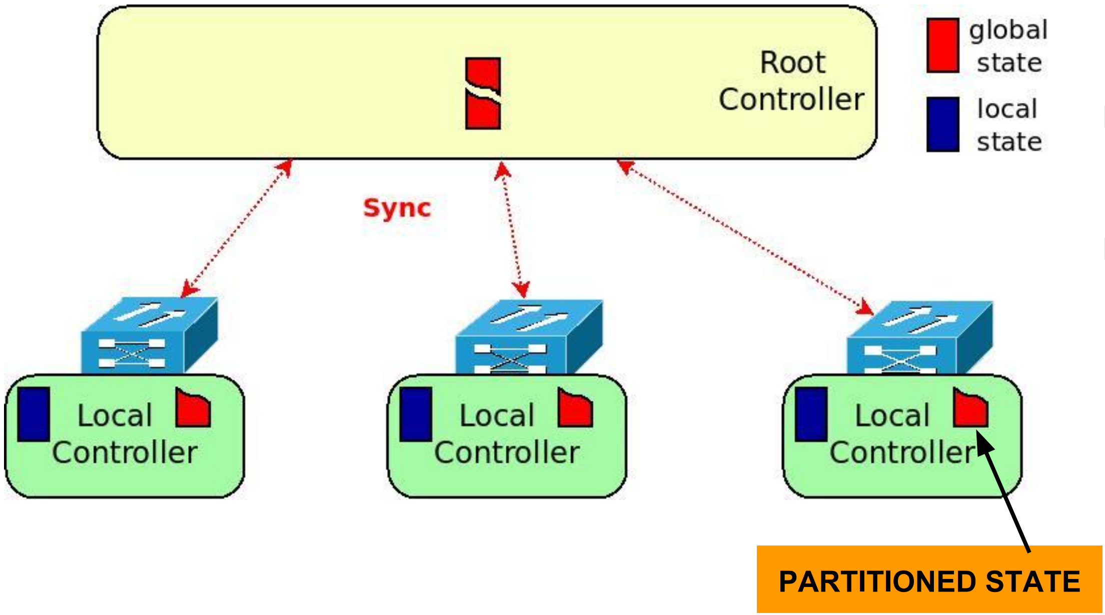
Devoflow[1]
Kandoo[2]
FOCUS[3]

[1] Andrew R. Curtis and others. DevoFlow: Scaling Flow Management for High-performance Networks. In Proc of the SIGCOMM, 2011.

[2] Soheil Hassas Yeganeh and Yashar Ganjali. Kandoo: A Framework for Efficient and Scalable Offloading of Control Applications. In Proc of the Workshop on HoTSDN, 2012.

[3] Ji Yang and others. FOCUS: Function Offloading from a Controller to Utilize Switch Power. In Proc of IEEE Conference on NFV-SDN, 2016.

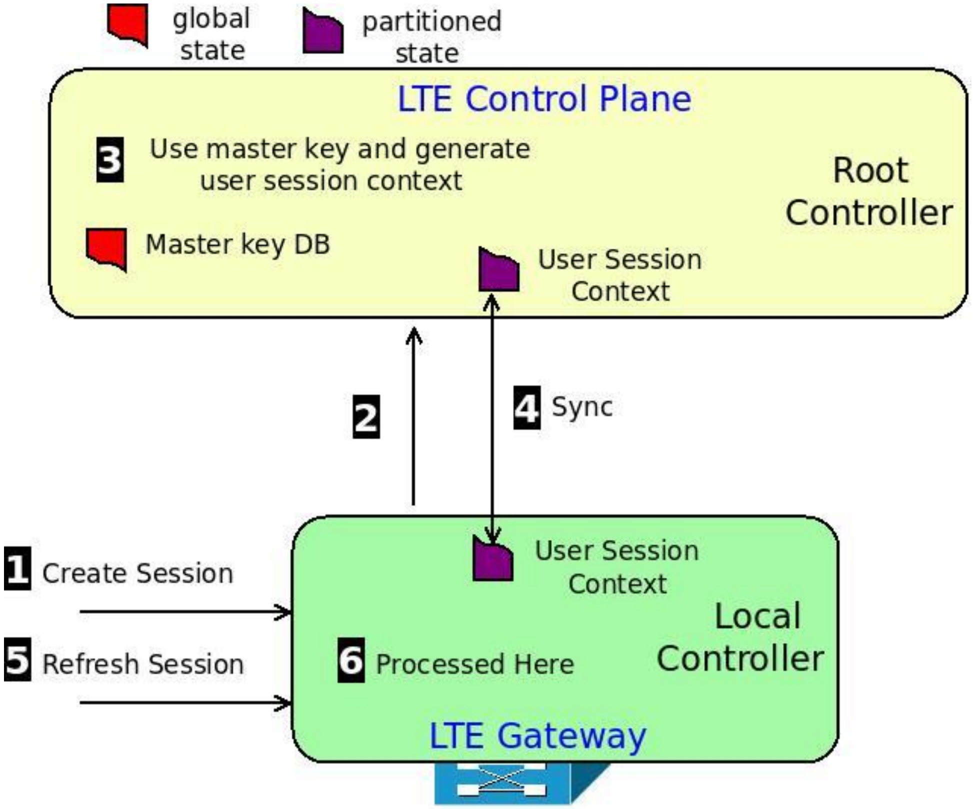
Our Key Idea



- **Increase extent of computation** at local controllers
- Achieved via increased amount of state offload
  - Break strict barrier between local and global state
  - **Partitioned state**

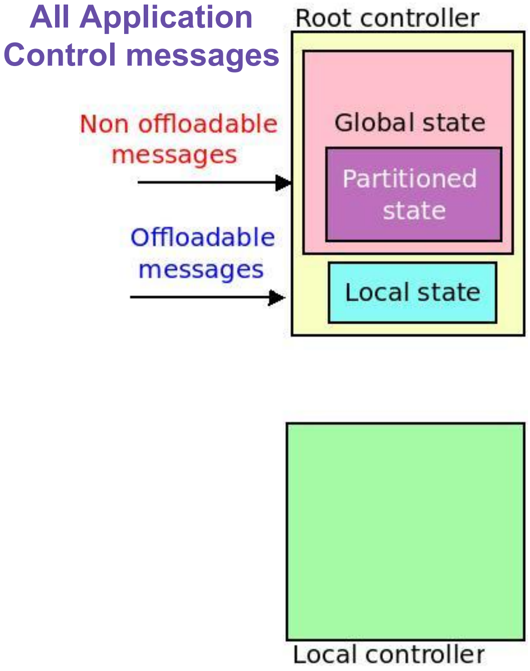
Increase in amount of computation offload => Improved performance

Partitioned state example: LTE packet core



- **Definition**
  - **Subset** of global state
  - **Accessed at one network location** at any point of time (like local state)
- **Pros**
  - Can be **cached** at local controllers temporarily
- **Cons**
  - Must be periodically **synchronized** with root controller
- **Partitioned state examples**
  - Any application specific session state
  - Route state like **flow-id : tunnel-id**

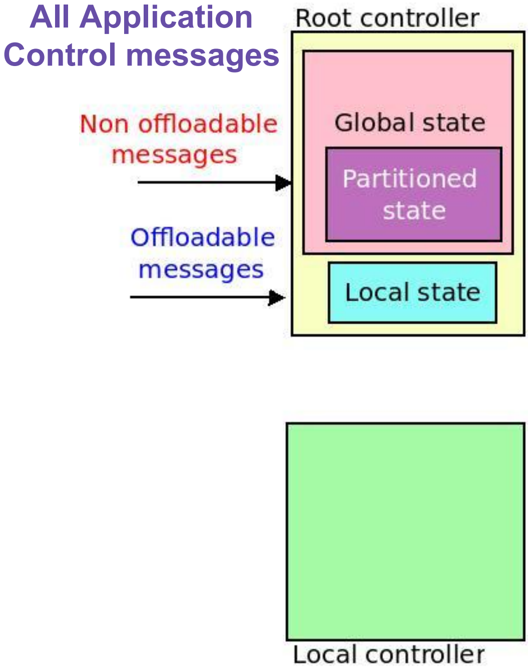
SDN Controller modes: Centralized mode



CENTRALIZED MODE

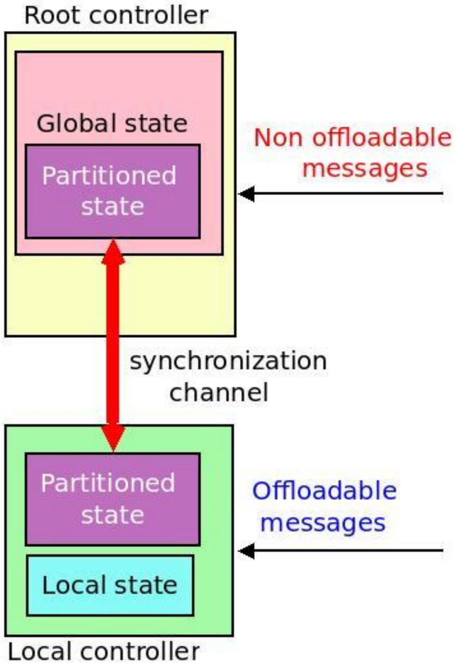
CONS: Single compute resource

SDN Controller modes: Offload mode (Proposed)



CENTRALIZED MODE

CONS: Single compute resource

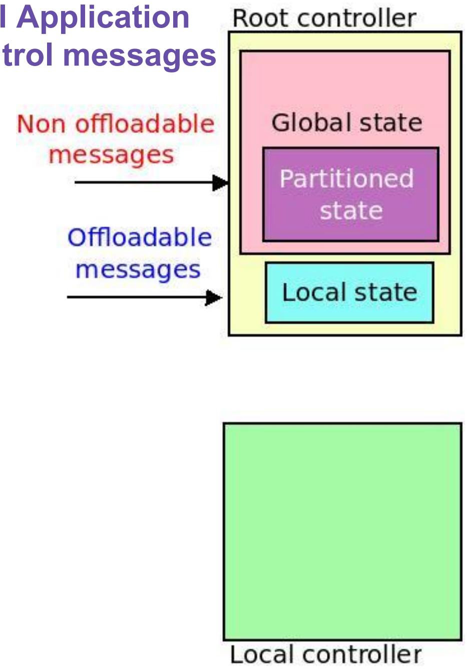


OFFLOAD MODE

PROS: Compute resource increases
CONS: Synchronization Overhead

SDN Controller modes

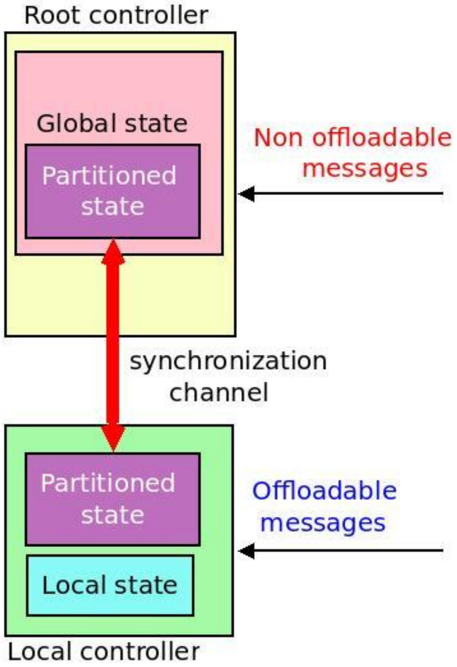
All Application Control messages



CENTRALIZED MODE

CONS: Single compute resource

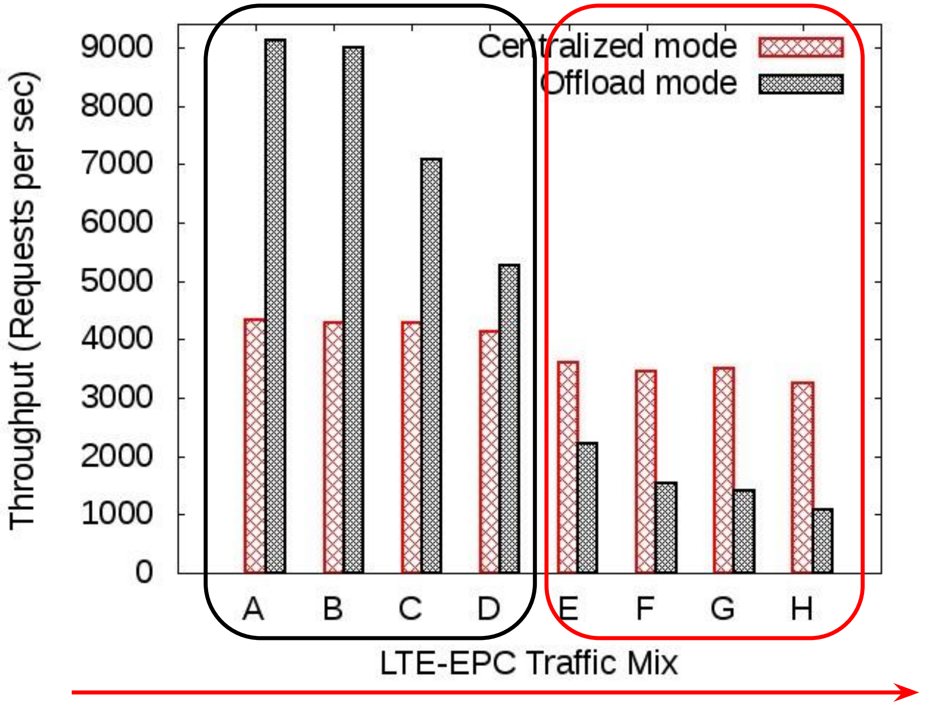
Which mode is better?



OFFLOAD MODE

PROS: Compute resource increases
CONS: Synchronization Overhead

Which Controller mode is better?

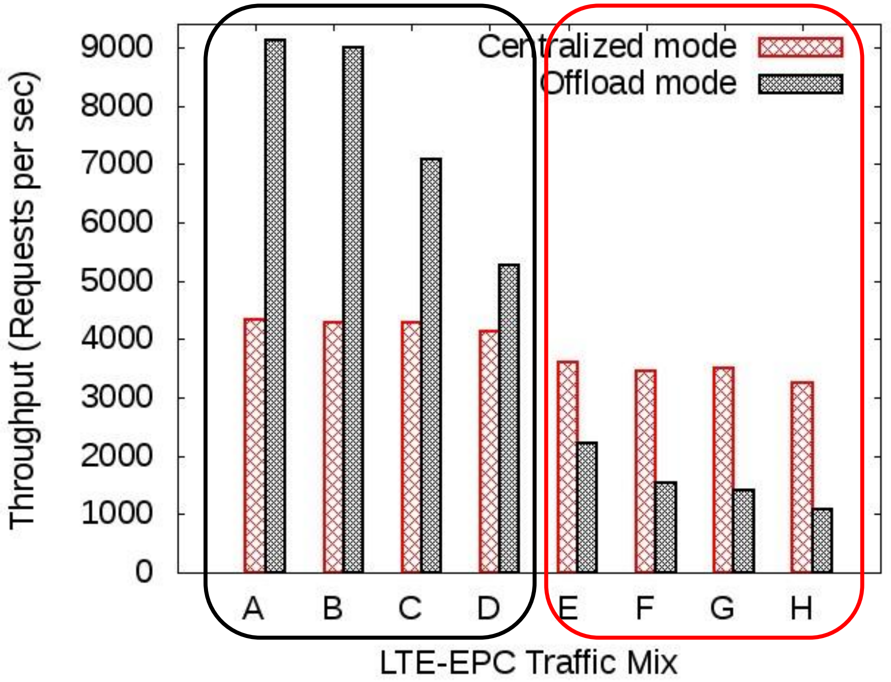


Amount of Synchronization traffic generated

Use case: SDN based application that performs subset of cellular network functionality
(SDN based LTE Evolved Packet core (EPC))

- ❑ **A to D: Offload mode**
- ❑ **E to H: Centralized mode**
- ❑ **Offload mode** performance depends on **synchronization cost** incurred

Which Controller mode is better?

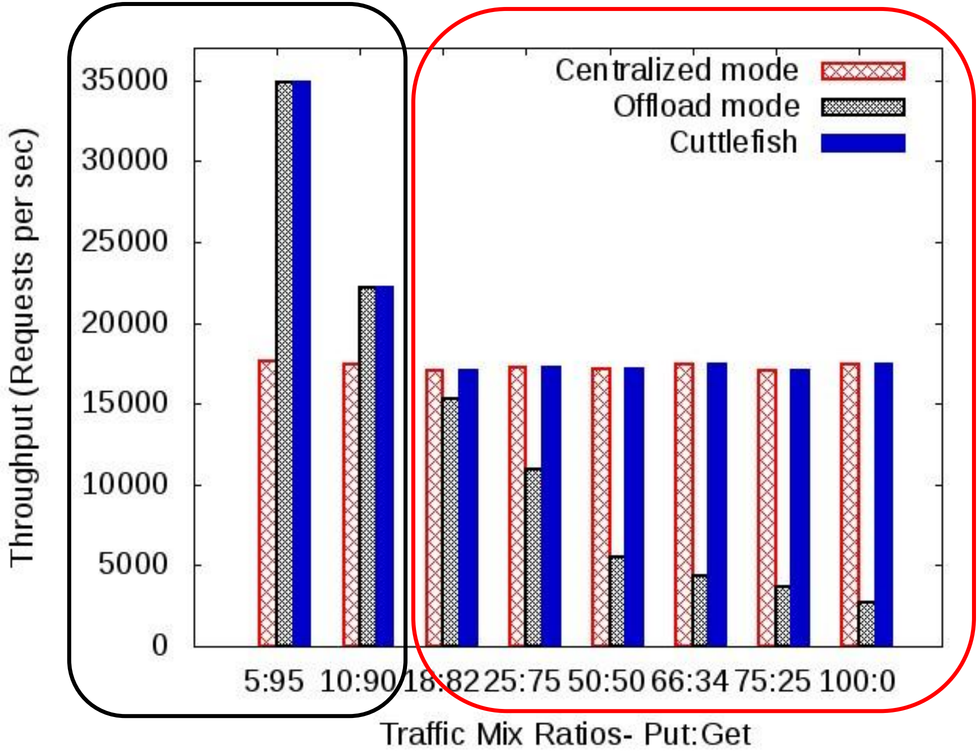


Use case: SDN based application that performs subset of cellular network functionality
(SDN based LTE Evolved Packet core (EPC))

- **A to D: Offload mode** ☆
- **E to H: Centralized mode** ☆
- **Offload mode** performance depends on **synchronization cost** incurred

Need for SWITCH between controller MODES, based on TRAFFIC MIX

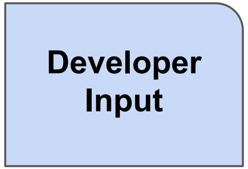
Cuttlefish: Adaptive Offload (Use case - KV store)



Cuttlefish matches the **BEST** Non-Adaptive mode

PUT @ Root (Non offloadable) : GET @ Local (Offloadable)

Cuttlefish Design: Developer input



Developer Input

Cuttlefish Design: Developer input example

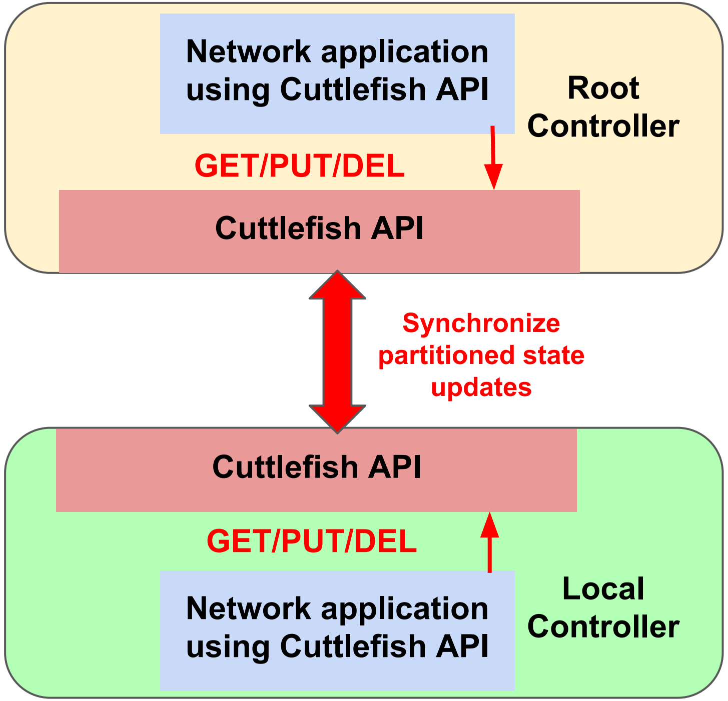
Developer Input

Example LTE-EPC Messages	**msg_Id**	**Offloadable**
Authentication Step 1	1	false
Authentication Step 3	2	false
NAS Step 2	3	false
Send APN	4	false
Send UE TeID	5	true
UE Context Release	6	true
UE Service Request	7	true
Context Setup Response	8	true
Detach Request	9	false

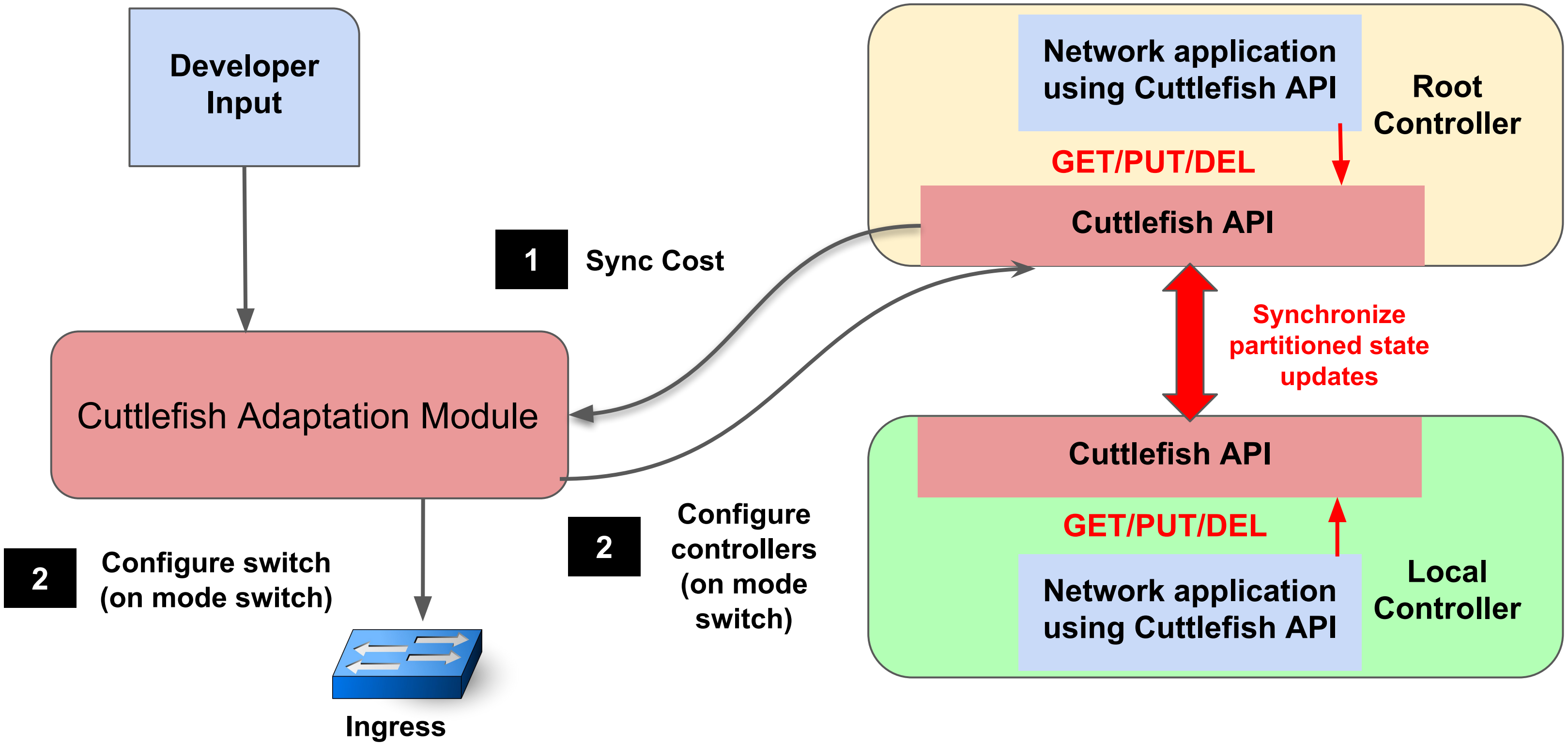
Example -
SDN Mobile Packet Core application

Cuttlefish Design: API

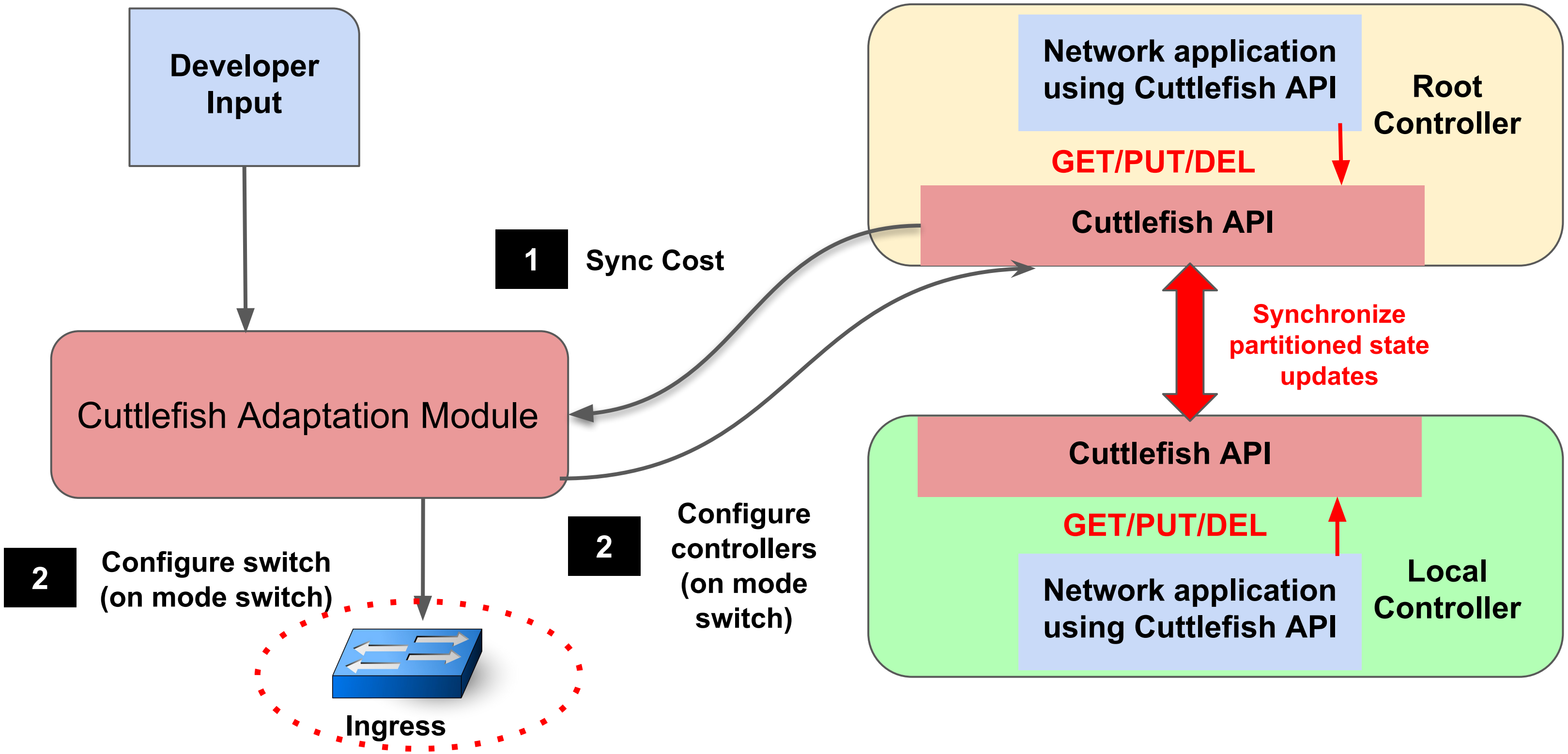
Developer Input


Network application using Cuttlefish API
Root Controller
GET/PUT/DEL
Cuttlefish API
Synchronize partitioned state updates
Cuttlefish API
GET/PUT/DEL
Network application using Cuttlefish API
Local Controller

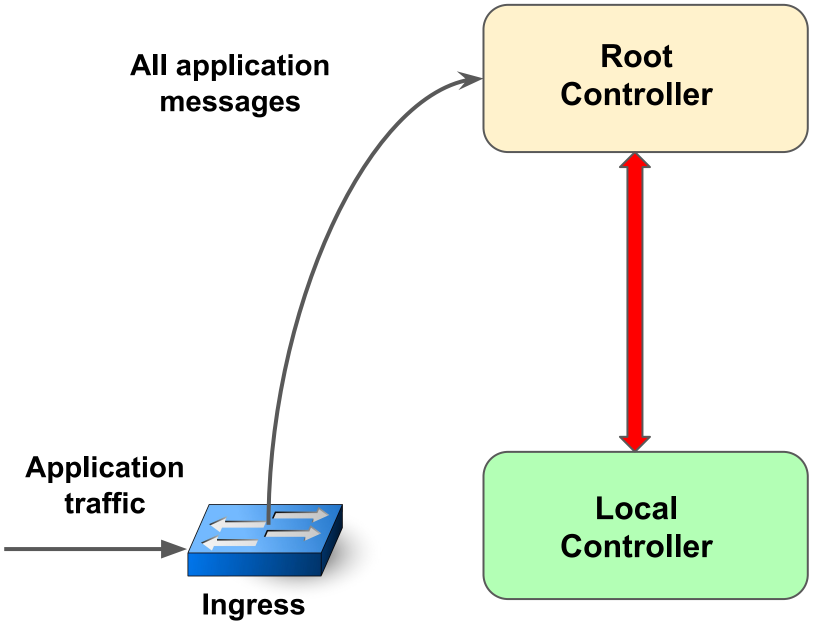
Cuttlefish Design: Adaptation module


Developer Input
Cuttlefish Adaptation Module
1
Sync Cost
2
Configure switch (on mode switch)
Ingress
2
Configure controllers (on mode switch)
Network application using Cuttlefish API
Root Controller
GET/PUT/DEL
Cuttlefish API
Synchronize partitioned state updates
Cuttlefish API
GET/PUT/DEL
Network application using Cuttlefish API
Local Controller

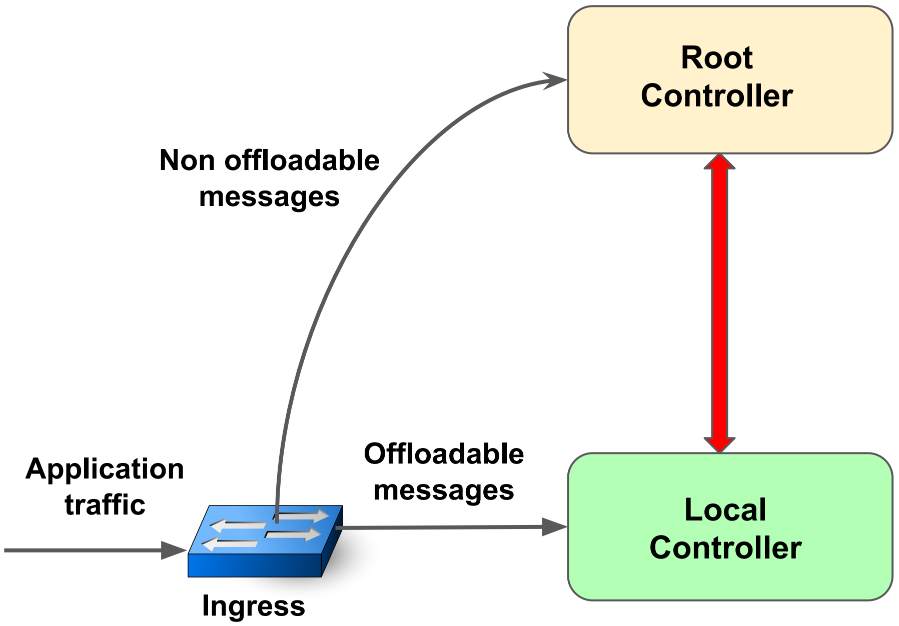
Cuttlefish Design


Developer Input
Cuttlefish Adaptation Module
1
Sync Cost
2
Configure switch (on mode switch)
Ingress
2
Configure controllers (on mode switch)
Network application using Cuttlefish API
Root Controller
GET/PUT/DEL
Cuttlefish API
Synchronize partitioned state updates
Cuttlefish API
GET/PUT/DEL
Network application using Cuttlefish API
Local Controller

Switch configuration

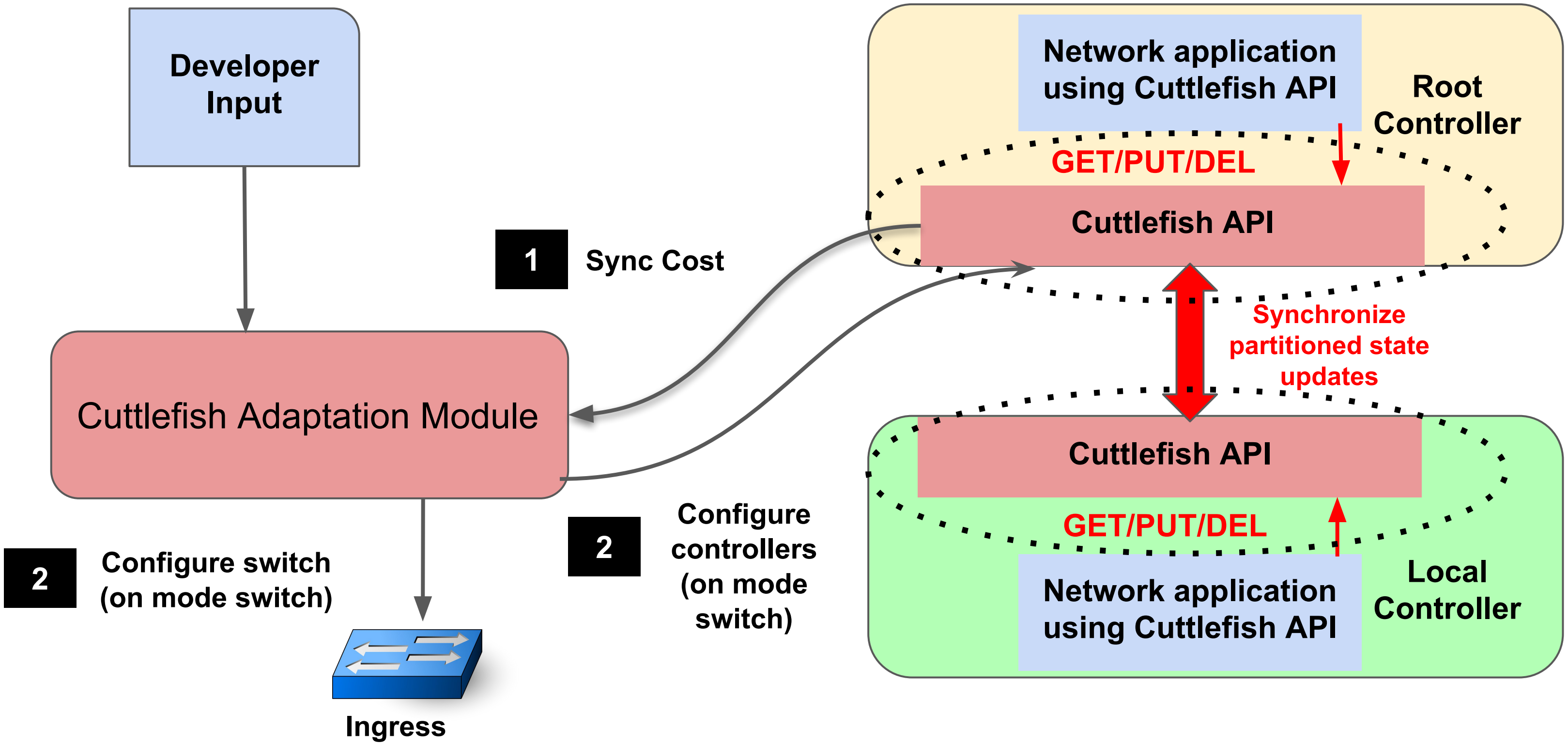


CENTRALIZED MODE

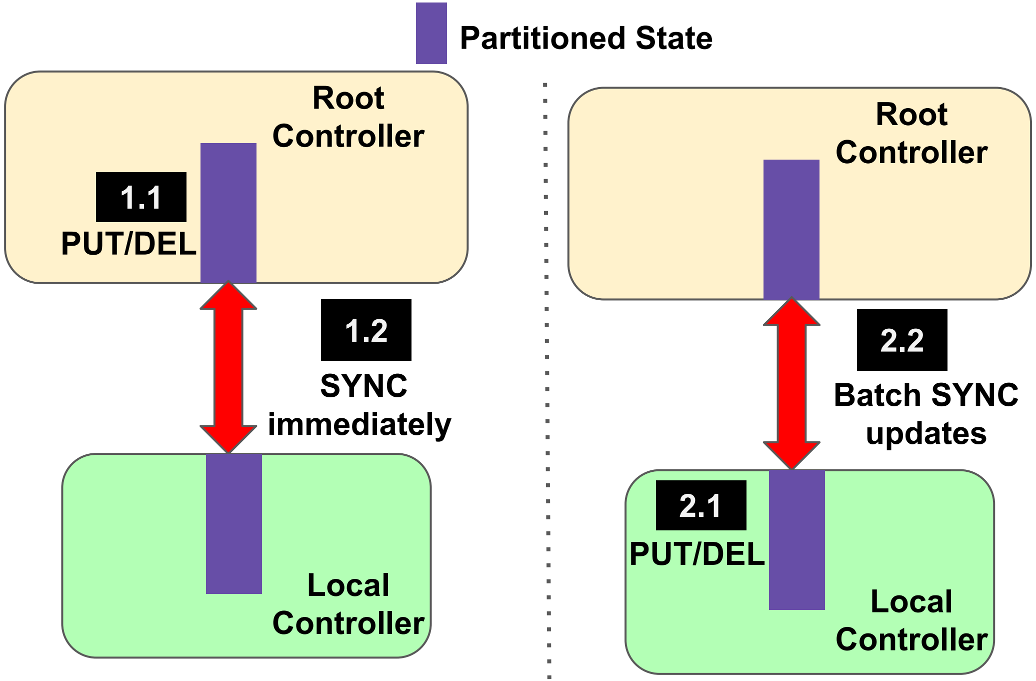


OFFLOAD MODE

Cuttlefish Design


Developer Input
Cuttlefish Adaptation Module
1
Sync Cost
2
Configure switch (on mode switch)
Ingress
2
Configure controllers (on mode switch)
Network application using Cuttlefish API
Root Controller
GET/PUT/DEL
Cuttlefish API
Synchronize partitioned state updates
Cuttlefish API
GET/PUT/DEL
Network application using Cuttlefish API
Local Controller

Synchronizing Partitioned State: Offload Mode



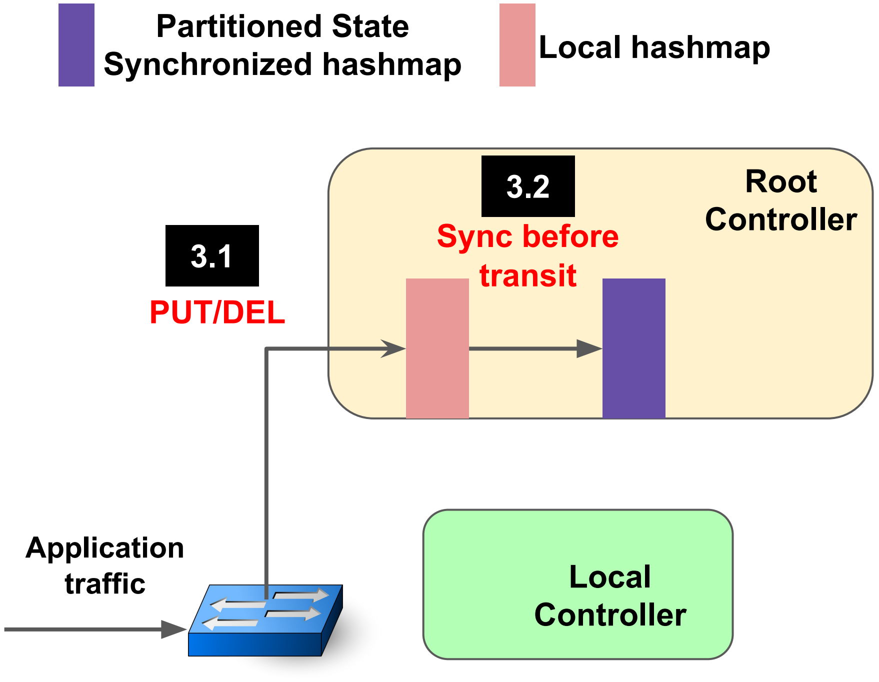
1.1	Update to partitioned state at root controller
1.2	Synchronize the partitioned state update immediately to the local controller
2.1	Update to partitioned state at local controller
2.2	Synchronize partitioned state updates in batches to root controller

Assumption:
Partitioned state **Get API** is called **only at Local controller**

During mode migration

- ❏ Synchronize all local controller state
- ❏ Gracefully transit to Centralized mode

Synchronizing Partitioned State: Centralized Mode



Partitioned State Synchronized hashmap

Local hashmap

Root Controller

3.2

Sync before transit

3.1

PUT/DEL

Application traffic

Local Controller

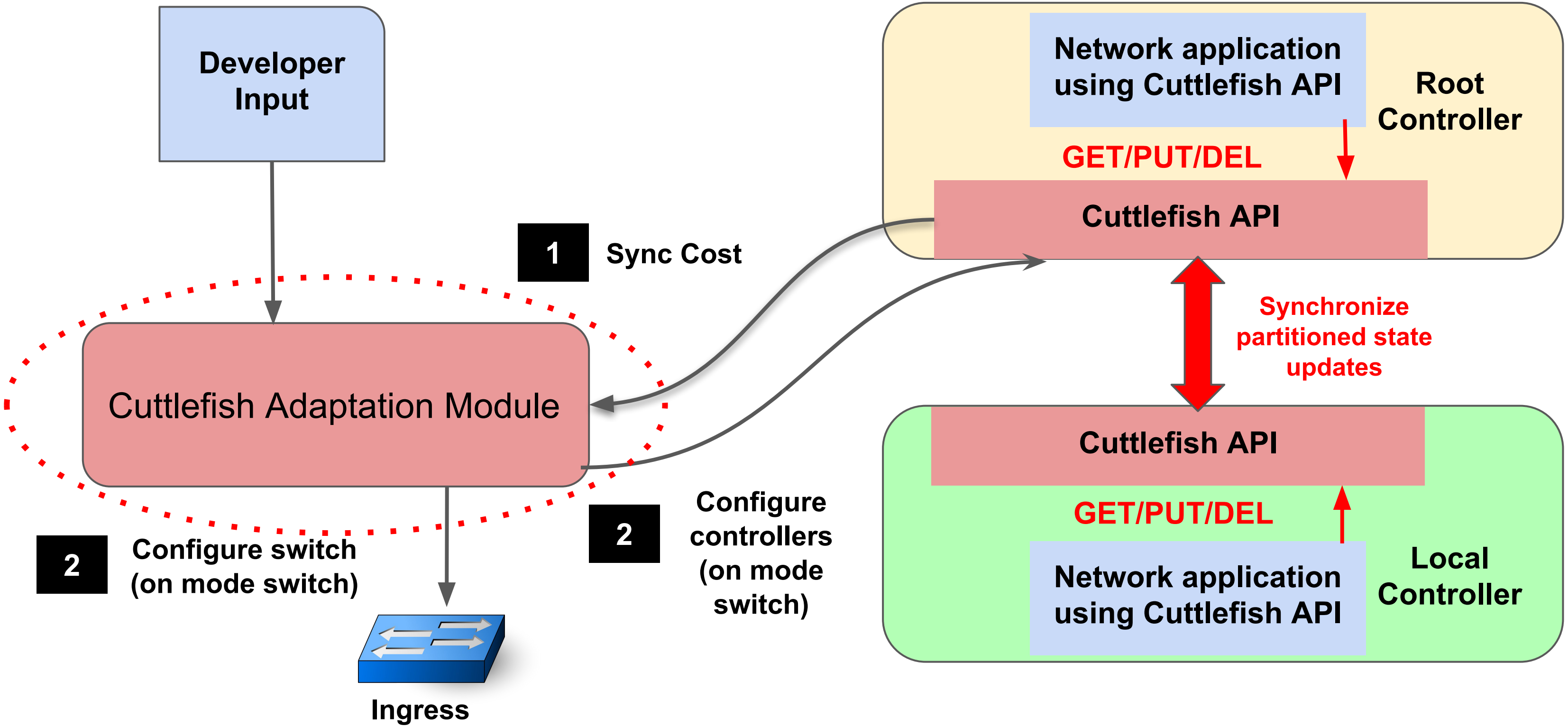
3.1 Partitioned state updates are done on local hashmap for better performance

2.2 Synchronize partitioned state updates from local hashmap to synchronized hashmap before mode switch

During mode migration

- ❏ Synchronize all local hashmap state
- ❏ Gracefully transit to Offload mode

Cuttlefish Design



Developer Input

Cuttlefish Adaptation Module

1 Sync Cost

2 Configure switch (on mode switch)

2 Configure controllers (on mode switch)

Ingress

Network application using Cuttlefish API

Root Controller

GET/PUT/DEL

Cuttlefish API

Synchronize partitioned state updates

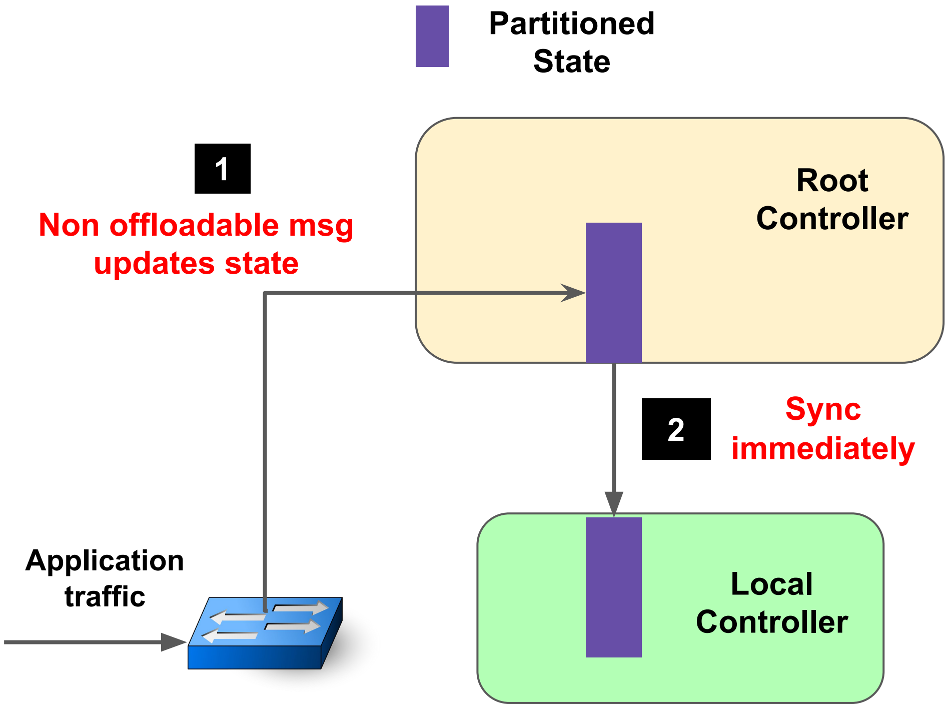
Cuttlefish API

GET/PUT/DEL

Network application using Cuttlefish API

Local Controller

Adaptation Module



Partitioned State

Root Controller

1

Non offloadable msg updates state

2

Sync immediately

Local Controller

Application traffic

- Monitor the **frequency of partitioned state updates** by **non offloadable** messages at the **root** controller

- This frequency acts as a **PROXY** to estimate the synchronization cost
  - **#Updates/sec**

- Switch the controller mode if **#Updates/sec crosses the threshold**
  - Threshold value is determined using our benchmark

Benchmark Parameters

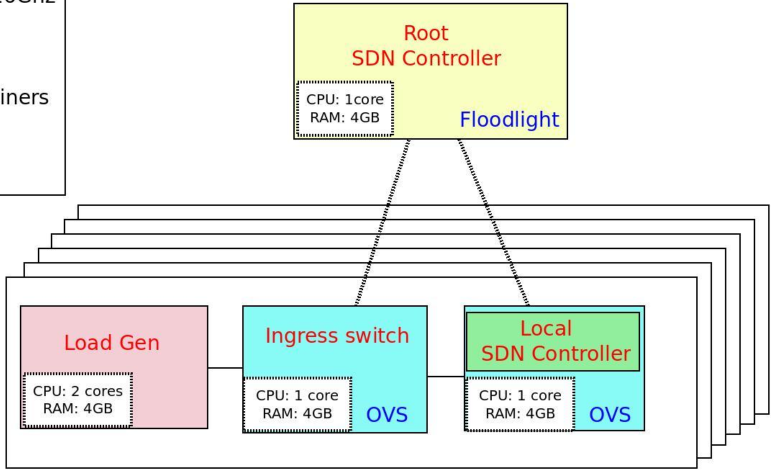
❑ sync CPU budget

❑ key-value size

Cuttlefish Evaluation

- Use cases
  - **Key-value store**
  - SDN based LTE EPC
  - Stateful Load Balancer
- Controller modes
  - Centralized mode
  - Offload mode
  - Cuttlefish adaptive offload mode
- Metrics measured
  - **Average throughput** - Average number of control plane messages processed per sec
  - **Average response latency** - Average time between request initiation and completion

Intel Xeon E312xx @ 2.16Ghz
16 cores
8 cores
Host: Ubuntu 14.04
Components: LXC Containers
Network: LXC Bridges
Floodlight v1.2
Openvswitch v2.3.2



Questions to be answered?

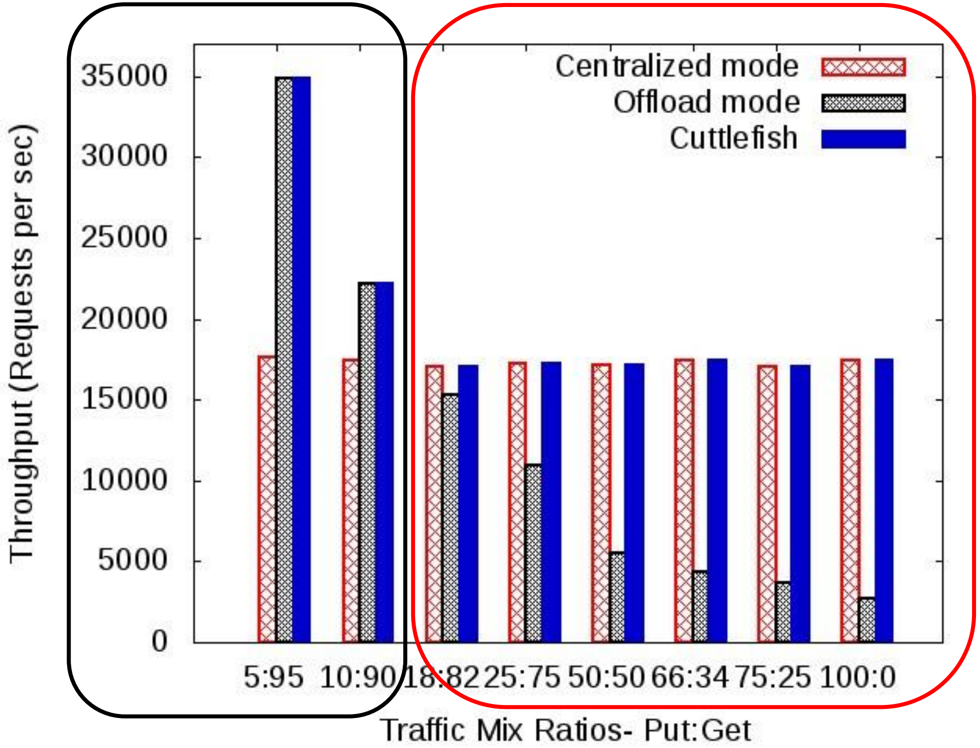
1. How does Cuttlefish **perform** compared to Centralized and Offload modes?

2. What is Cuttlefish efficacy?

    a. Can it take **correct switching decision**?

    b. How much **time is required** to implement the decision?

Performance of Adaptive Offload: Key value store

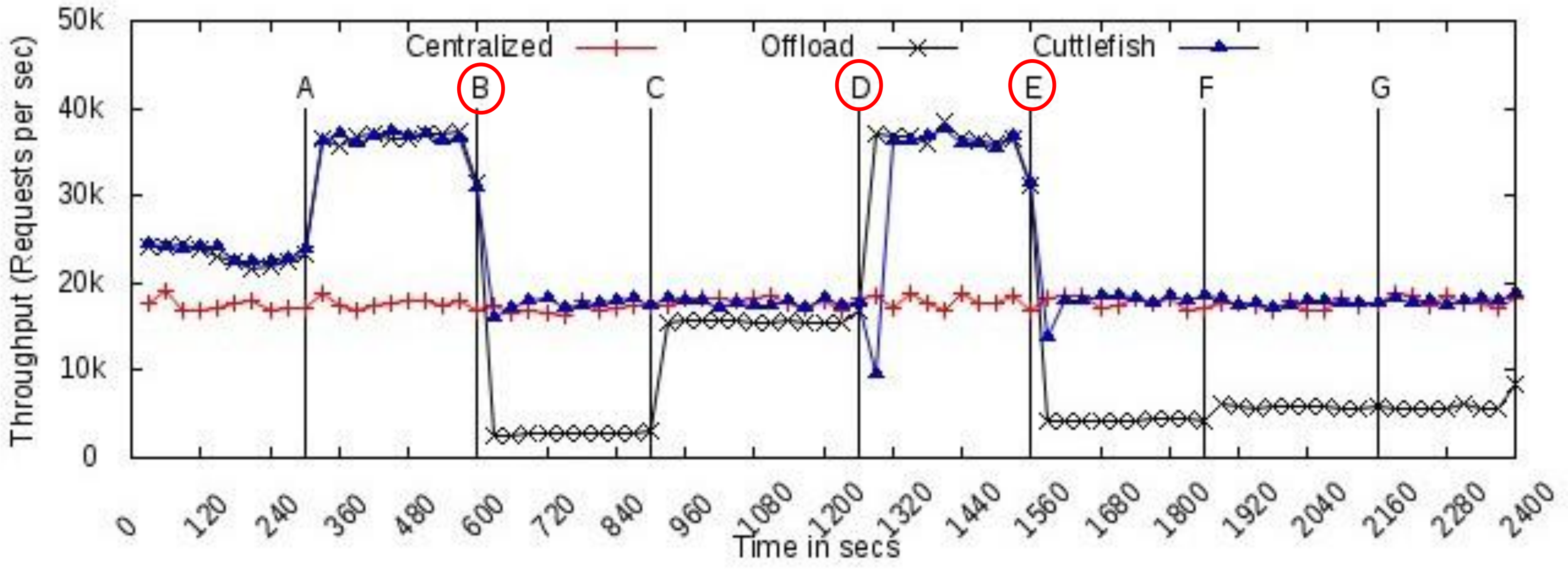


PUT @ Root (Non offloadable) : GET @ Local (Offloadable)

Cuttlefish matches the **BEST** Non-Adaptive mode

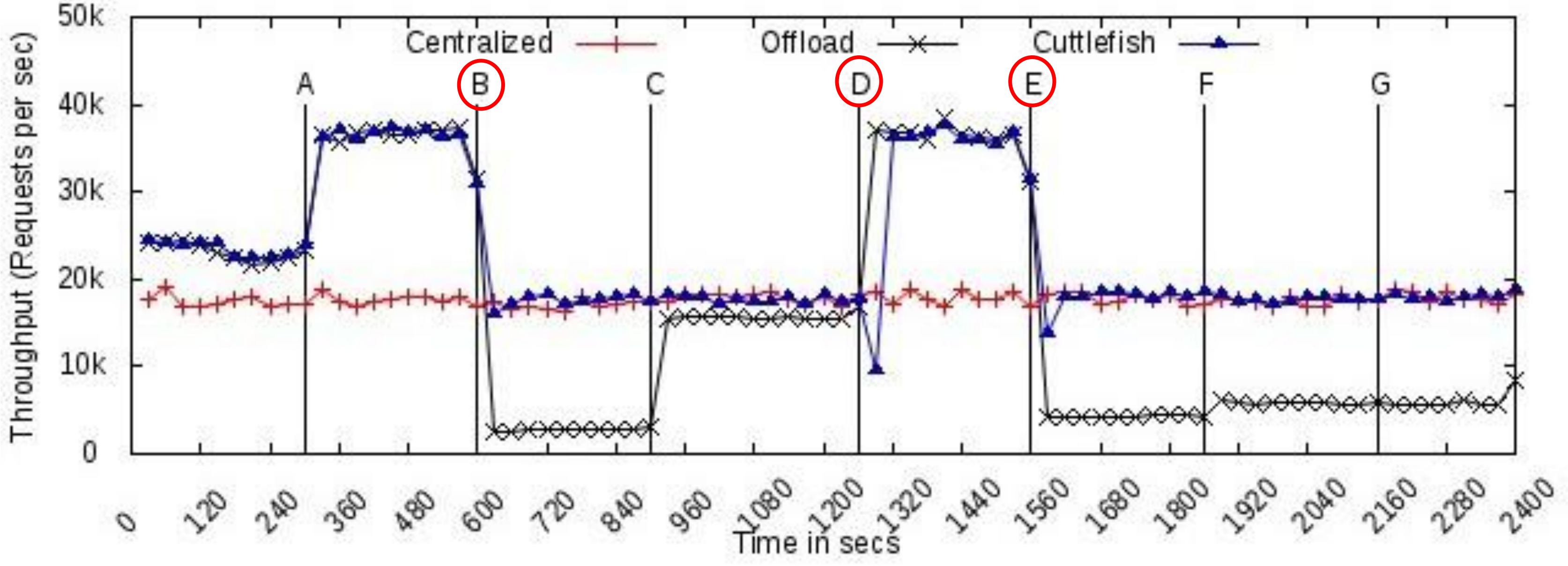
	Cuttlefish Throughput	Cuttlefish Latency Reduction
Centralized	**0.99x to 2x**	**0% to 50%**
Offload	**0.99x to 6.4x**	**-0.04% to 80%**

Cuttlefish Efficacy: Key Value store



Cuttlefish switches between controller modes to **MATCH the BEST PERFORMING** mode

Cuttlefish Efficacy: Key Value store



Cuttlefish switches between controller modes to **MATCH the BEST PERFORMING** mode

Cuttlefish takes **20-30** secs to switch between modes after traffic switch

Summary

- **New design** of Hierarchical Controller Framework
  - Concept of **Partitioned** State
- Design and Implementation of **Adaptive Controller**
  - Evaluation shows that Cuttlefish applications achieve **2x higher control plane throughput** and **50% lower control plane latency** as compared to the traditional SDN design.
- Cuttlefish **source code** is made available at
  - https://github.com/networkedsystemsIITB/cuttlefish