KANDOO
A FRAMEWORK FOR
EFFICIENT & SCALABLE OFFLOADING
OF CONTROL APPLICATIONS

Soheil Hassas Yeganeh
soheil@cs.toronto.edu

Yashar Ganjali
yganjali@cs.toronto.edu
EVENTS.

- Rare
  - Link state changes

- Frequent and Exhaustive
  - Network-wide stat collection
  - Packet-ins (if flow-entries are not installed proactively)
SCALABILITY ISSUES.

Frequent events stress the control plane.

Stress the control channels.

Stress controller's resources.
EXISTING SOLUTIONS.

Distributed Controllers:

- Consider this as an intrinsic limitation.
- HyperFlow, Onyx, Devolved Controllers, ...

Data Plane Extensions:

- Delegate more responsibilities to the data plane.
- DIFANE, DevoFlow, ...

Control Plane

Data Plane
EXISTING SOLUTIONS.

Still, high control channel consumption.

Comes at the cost of visibility.

Need to modify the data plane.
PROBLEM STATEMENT.

How to handle frequent events close to the metal without modifying OpenFlow?
THE IDEA.

OFFLOADING LOCAL CONTROL APPS TO LOCAL RESOURCES.

Applications that do not need the network-wide state.

Resources close to switches.
Local Apps.

- An **assumption** in distributed controllers:
  - All control apps require the network-wide state.

- But, there are many apps that are **local in scope**:
  - Applications that require only local switch state.

![Diagram of distributed controllers and local apps](image-url)
LOCAL APPS.

- Local applications:
  - Learning Switch
  - Local Policy Enforcer
  - Link Discovery

- Local components in control applications:
  - Elephant Flow Detection in an Elephant Flow Rerouting application.
LOCAL APPS.

• Local applications:
  • Learning Switch
  • Local Policy Enforcer
  • Link Discovery

• Local components in control applications:
  • Elephant Flow Detection in an Elephant Flow Rerouting application.

Local apps have implicit parallelism.
LOCAL RESOURCES.

We can offload local apps to computing resources next to switches.
LOCAL RESOURCES.

We can offload local apps to computing resources next to switches.

On the same hosts running software switches.

Hosts close to switches.

Inside programmable switches.
KANDOO.

- Two layers of controllers:
  - A logically centralized Root Controller.
  - Local Controllers.
KANOOO.

- **Two** layers of controllers:
  - A logically centralized **Root Controller**.
  - **Local Controllers**.

The root controller runs non-local apps.

Lightweight and easy to implement.

Local controllers run local apps.

Local controllers shield the root controller.
AN EXAMPLE: ELEPHANT FLOW REROUTEING.
AN EXAMPLE:
ELEPHANT FLOW REROUTEING.
AN EXAMPLE:
ELEPHANT FLOW REROUTEING.

Kandoo’s event channels.

Application-specific events.

Scales linearly with the number of switches.
SIMPLE, YET FLEXIBLE, ARCHITECTURE.
SIMPLE, YET FLEXIBLE, ARCHITECTURE.

Scale at the edge.

One local controller per switch!

Normal OpenFlow.
**EVALUATION SUMMARY.**

- **Implemented Kandoo:**
  - Handles 1.3 Mp/s on a single core of Xeon E7-4807.

- **Elephant Flow Rerouting:**
  - In an *emulated* environment.
  - More than 5x less channel consumption.
  - Significantly better scalability in regards to the network size.
EVALUATION SUMMARY.

• Implemented Kandoo:
  • Handles 1.3 Mp/s on a single core of Xeon E7-4807.

• Elephant Flow Rerouting:
  • In an emulated environment.
  • More than 5x less channel consumption.
  • Significantly better scalability in regards to the network size.
EVALUATION SUMMARY.

- Implemented Kandoo:
  - Handles 1.3 Mp/s on a single core of Xeon E7-4807.

- Elephant Flow Rerouting:
  - In an emulated environment.
  - More than 5x less channel consumption.
  - Significantly better scalability in regards to the network size.
FUTURE DIRECTIONS.

- A Generalized Hierarchy
  - Filling the gap between local and non-local apps.
  - Finding the right scope is quite challenging.
  - Finding the right scope is quite challenging.

- Porting Kandoo to Programmable Switches
  - Highly efficient and better resource utilization.
THANKS.

QUESTIONS?