KANDOO: A FRAMEWORK FOR EFFICIENT AND SCALABLE OFFLOADING OF CONTROL APPLICATIONS

SOHEIL HASSAS YEGANEH AND YASHAR GANJALI
UNIVERSITY OF TORONTO
{SOHEIL, YGANJALI}@CS.TORONTO.EDU

1. Introduction

Frequent network events can easily stress the control plane and hinder the scalability of Software-Defined Networks (SDNs) [**1**]. For instance, network-wide statistic collection and flow arrivals at a high frequency can over-consume the bandwidth allocated for the control channels with severe impact on other parts of the control plane. Thus, limiting the overhead of frequent events on the control plane is essential for scalable Software-Defined Networking.

Existing solutions try to address this problem by processing frequent events in the data plane. For instance, DIFANE [**3**] offloads part forwarding decisions to special switches, called authority switches, and DevoFlow [**1**] introduce new functionalities in the data plane to suppress frequent events and to reduce the load on the control plane.

To limit the load on the controller, an effective option is to handle frequent events close to the datapath, but preferably without modifying switches. Adding new primitives to switches comes at the cost of visibility in the control plane and necessitates changes to the southbound protocols (e.g., OpenFlow).

Kandoo. Taking an alternative route, we have proposed Kandoo [**4**], a framework for preserving scalability without modifying switches. Kandoo is a novel distributed control plane that offloads *local* control applications (i.e., applications that do not require the network-wide state), over available resources in the network to process frequent events at scale.

2. Design

As illustrated in Figure 1, a network controlled by Kandoo has multiple *local controllers* and a logically centralized *root controller* structured in a two layer hierarchy: (i) at the bottom layer, we have a set of controllers with no interconnection, and no knowledge of the network-wide state, and (ii) at the top layer, we have a *logically centralized* controller that maintains the network-wide state. Controllers at the bottom layer run only local control applications near datapath. These controllers handle most of the frequent events and effectively shield the top layer.

Kandoo is OpenFlow-compatible in a sense that it does not introduce any new data plane functionality in switches, and, as long as they support OpenFlow, Kandoo supports them, as well. To that end, all network functionalities are implemented as control applications that are automatically distributed by Kandoo without any manual intervention. In other words, Kandoo control applications are not aware of how they are deployed in the network, and application developers can assume their applications would be run on a centralized OpenFlow controller. The only extra meta-data Kandoo requires is a flag showing whether a control application is local or not.

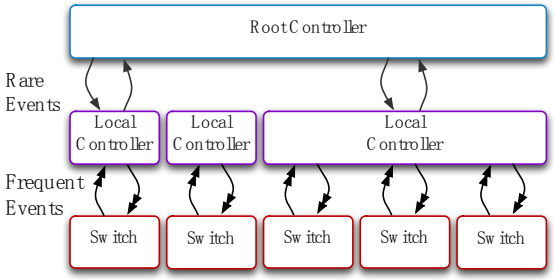

Figure 1. Two layers of controllers in Kandoo.

An Example. To shed more light on Kandoo's design, we show how Kandoo can be used to reroute elephant flows in a simple network of three switches (Figure 2). Our example has two applications: (i) App_{detect} that detects elephant flows, and (ii) $App_{reroute}$ that reroute the detected elephant flows. To detect elephant flows, App_{detect} constantly queries each switch. Once an elephant flow is detected, App_{detect} notifies $App_{reroute}$, which in turn may install or update flow-entries on network switches. Without modifying switches, it is extremely challenging, if not impossible, to implement this application in current OpenFlow controllers [**1**]. That is to say, collecting network-wide statistics from a (logically) centralized control would place a considerable load on control channels that results in profound degradation in the quality of service.

As shown in Figure 2, Kandoo replicates App_{detect} on processing resources close to the switches; hence, each application instance can frequently query each switch to detect an elephant flow without affecting the other parts of the control plane. Once an elephant flow is detected, App_{detect} notifies $App_{reroute}$ residing on the root controller. Since these events are significantly less frequent than statistic queries, Kandoo can scale considerably better than a normal OpenFlow network.

Kandoo local controllers alongside with the logically centralized root controller collectively form Kandoo's

distributed control plane. Each local controller can control multiple switches, but each switch is controlled by one and only one local controller. If the root controller needs to communicate with a switch, it delegates the request to the respective local controller. For high availability, the root controller can register itself as a slave controller in switches supporting OpenFlow 1.2 or higher.

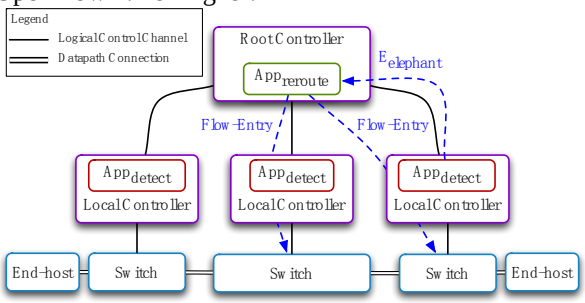

Figure 2. Rerouting elephant flows in Kandoo.

Deployment Model. Kandoo has an elastic architecture, and thus can be tailored based on the characteristics of a network. For in-software, virtual switches, local controllers can be directly deployed on the same end-host. Similarly, for programmable switches equipped with general purpose co-processors, we can deploy Kandoo directly on the switch. Otherwise, we deploy Kandoo local controllers on the processing resources closest to the switches (e.g., on an end-host directly connected to the switch). In such a setting, one should provision the number of local controllers based on the workload and available processing resources. We note that one can utilize a hybrid model in real settings.

3. Concluding Remarks

Kandoo is a highly configurable and scalable control plane, with a simple yet effective approach: it processes frequent events in highly replicated local control applications and rare events in a logically centralized controller. As confirmed by experiment [**4**], Kandoo scales considerably better than a normal OpenFlow implementation.

Although distinctive, Kandoo's approach is orthogonal to other distributed control plane. HyperFlow [**2**] and Onix [**5**] try to distribute the control plane while maintaining logically centralized, eventually consistent network state. These approaches can be used to realize a scalable *root* controller; the controller that runs non-local applications in Kandoo.

Moving forward, we are extending Kandoo's abstraction to include control applications that are not necessarily local but that have a limited scope and can operate by having access to the events generated by a part of the data plane. We note that this necessitates using a hierarchy of controllers (as opposed to the presented two-level hierarchy). We have also started porting Kandoo to programmable switches. When such switches are equipped with Kandoo, they can natively run local control applications.

References

[1] Jeffrey C. Mogul et al., "DevoFlow: cost-effective flow management for high performance enterprise networks," in *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, 2010, pp. 1:1--1:6.

[2] Amin Tootoonchian and Yashar Ganjali, "HyperFlow: a distributed control plane for OpenFlow," in *Proceedings of the 2010 internet network management conference on Research on enterprise networking*, 2010, pp. 3-3.

[3] Minlan Yu, Jennifer Rexford, Michael J. Freedman, and Jia Wang, "Scalable flow-based networking with DIFANE," in *Proceedings of the ACM SIGCOMM conference*, 2010, pp. 351-362.

[4] Soheil Hassas Yeganeh and Yashar Ganjali, "Kandoo: a framework for efficient and scalable offloading of control applications," in *Proceedings of the first workshop on Hot Topics in Software Defined Networks (HotSDN'12)*, 2012, pp. 19-24.

[5] Teemu Koponen et al., "Onix: a distributed control platform for large-scale production networks," in *Proceedings of the 9th USENIX OSDI conference*, 2010, pp. 1-6.

Soheil Hassas Yeganeh is PhD student at the University of Toronto. He received his bachelors and masters from Sharif University of Technology. His research interests include software-defined networking, network virtualization, and congestion control.

Yashar Ganjali is an associate professor of computer science at the University of Toronto. His research interests include packet switching architectures/algorithms, software defined networks, congestion control, network measurements, and online social networks. Dr. Ganjali has received several awards including an Early Researcher Award, Cisco Research Award, best paper award in Internet Measurement Conference 2008, best paper runner up in IEEE INFOCOM 2003, best demo runner up in ACM SIGCOMM 2008, and first and second prizes in NetFPGA Design Competition 2010.