Software-Defined Networking in Mobile Access Networks

Yanhe Liu, Aaron Yi Ding, Sasu Tarkoma

Technical Report C-2013-1
University of Helsinki
Department of Computer Science

Helsinki, September 19, 2013

HELSINGIN YLIOPISTO — HELSINGFORS UNIVERSITET — UNIVERSITY OF HELSINKI

Tiedekunta — Fakultet — Faculty Faculty of Science	Laitos — Institution — Department Department of Computer Science

Tekijä — Författare — Author
Yanhe Liu, Aaron Yi Ding, Sasu Tarkoma

Työn nimi — Arbetets titel — Title

Software-Defined Networking in Mobile Access Networks

Oppiaine — Läroämne — Subject

Computer Science

Työn laji — Arbetets art — Level Technical Report C-2013-1	Aika — Datum — Month and year September 19, 2013	Sivumäärä — Sidoantal — Number of pages 29

Tiivistelmä — Referat — Abstract

Software-Defined Networking (SDN), a novel solution to network configuration and management, has shown great potential to simplify the existing complex and inflexible network infrastructure. For the design strength, SDN is gaining various investment from both industry and academia in terms of experimental implementation and deployment. In this report, we present an overview of SDN, covering the main concepts, features, and the latest proposals. Our focus is on the traffic management in mobile access networks. By illustrating the key challenges and benefits of utilizing SDN in such an environment, we discuss the potential research directions and share our perspectives in this domain.

Avainsanat — Nyckelord — Keywords
SDN, Traffic Management, Mobile Access Networks

Säilytyspaikka — Förvaringsställe — Where deposited
University of Helsinki / Kumpulan Kampuskirjasto

Muita tietoja — Övriga uppgifter — Additional information

Contents

1 Introduction

Computer networks, which form the critical infrastructure for our daily life, have become exceedingly complex nowadays. There are a large number of devices (e.g., switches and routers) and middleboxes (e.g., NAT, load balancer, and firewalls) integrated with various network protocols to implement a variety of basic forwarding policies. As a result, configuration and management of existing networks have become increasingly challenging. For instance, to build a campus network with authentication and traffic management on dozens of switches involves a large amount of manual configuration with low-level parameters (e.g., IP address, MAC address) and device-based interfaces, which is fault-prone and difficult to maintain.

Furthermore, the control logic for existing networks is integrated into every underlying device. While this can be beneficial to construct a small network with only one or two switches, it imposes a big problem for a large-scale network with hundreds of switches and thousands of hosts. The configuration is notably hard because nearly every device needs to be set up separately, not to mention to update the whole network for some new global policies. The integration of control logic and data forwarding also hinders the innovation of computer networks. For legacy networks, there is nearly no efficient way to experiment with new protocols in a real and large network environment without affecting original traffic.

In response, researchers are working hard to develop new architectures for networks, and Software-Defined Networking (SDN) is one of the emerging solutions which attracts much interest. SDN suggests separating the data and control plane with well defined programmable interfaces to provide a centralized global view of network and enable an easier way to configure and manage it. In SDN, network is managed by a central controller, and devices are only responsible for simple packet forwarding. This shift of network's logical brain provides good solutions to problems we mentioned at the beginning. First, it is easier to configure network policies by using a centralized controller with programmable interfaces and global view of the network, rather than setting up them on each underlying device with low-level commands. Second, it is easier to introduce new ideas to an existing physical infrastructure via software programs.

In this paper, we aim to present some basic architectures and practical features of SDN. To enhance our understanding of SDN, this paper first introduces some fundamental concepts and research projects like OpenFlow and NOX, and then explains several new extensions to those fundamental SDN solutions. In addition to the SDN principles, we also focus on wireless and mobile SDN systems, and try to explore how SDN can work for a better cellular and wireless network. The highlights of this paper could be summarized as follows:

- We introduce and explain some of latest SDN concepts and technologies, especially those on architecture and controller design
- We analyze applications of SDN in wireless and mobile networks

The rest of this paper is organized as follows. Section 2 gives a brief introduction on SDN, including the basic concepts and benefits. Section 3 describes some fundamental SDN systems and the typical architecture of SDN. Some new SDN research areas are explained in Section 4. Section 5 introduces mobile and wireless SDN systems. In section 6 we present some challenges and open questions to SDN, especially for cellular and wireless SDN, and also propose some solutions and future research directions.

2 Overview of Software-Defined Networking

2.1 SDN in A Nut Shell

Software-Defined Networking (SDN) is a new networking architecture which abstracts the logical part of computer networks to a centralized controller. The idea came from the work first done at Stanford University and UC Berkeley. It separates the control plane which decides where and how traffic is sent from the underlying devices (data plane) which simply forward data flows, and provides programmable interfaces to control network traffic.

Separating the control plane from the data plane is one of the most characterizing properties of SDN. This decoupling simplifies network management and configuration because there is need for administrators to specify hardware parameters in a low level. Programmability is another very important feature of SDN: the complex control logic can be defined by software programs, which is much easier to implement and maintain.

2.2 Motivation

In the early days of computers, people wrote programs using machine languages, which was extremely hard to understand and maintain because there were lack of abstractions. Today modern programming languages and operating systems have already overcome this difficulty by providing high-level abstractions and modulations for organizing information and resources. For example, you could use the logic name rather than MAC address (e.g. "eth0", "eth1") to refer to a network interface on a Linux system. Abstraction facilitates programmers to solve complicated problems easily and efficiently. However, networks are still configured and managed via low-level components. Though we use layers to represent the networks in an abstract way, we still need to know IP addresses or MAC addresses for setting up forwarding policies. What we expect is a new abstraction view of networks, in which we could easily represent all underlying networking components with human-readable notations and languages.

SDN can be considered as a new solution of abstraction for understanding and managing a network. It separates the control plane and data plane of a network, and provides a centralized software programmable controller with high-level abstractions instead of low-level configuration parameters (e.g. IP and MAC addresses). There are two main advantages of SDN:

- Independent evolution and deployment: By separating the control and data plane, the network logic is no longer bound by the capability of the software shipped with the hardware. Researchers can experiment with new ideas by adding new programs in the controller without affecting the underlying network structure, and administrators can easily deploy new policies into existing physical infrastructures by only configuring the controller.

- Control from high-level software program: Compared with the legacy architecture, SDN is much easier to configure. Policies do not have to be set up for every device with low-level device-dependent commands, but instead they could be configured with high-level programs only in the centralized controller.

2.3 SDN Architecture

SDN is a centralized paradigm where the logical brain of networks (control plane) is decoupled from underlying devices. Underlying packet forwarding is controlled by a global controller via programmable interfaces. A typical SDN architecture is shown in Figure 1. Programmable devices are in underlying networks as the data plane. Nowadays, OpenFlow is one of the most common SDN programmable interfaces for controlling packet forwarding. The upper SDN controller should provide an abstract and global view of resources and networks, which can also be considered as an "operating system". There are a variety of controllers, for example, NOX, Onix and Floodlight. The practical control programs written in high-level abstract languages are running on the network controller to implement different policies [9].

3 SDN Implementations and Applications

3.1 OpenFlow: A Widely Popular SDN Interface

As the network evolves with larger scale and new traffic patterns, it is realized that the current network architecture has been a barrier for innovation and network management. It is very difficult not only to experiment with new network protocols on the existing network infrastructure, but also to configure policies for nationwide facilities.

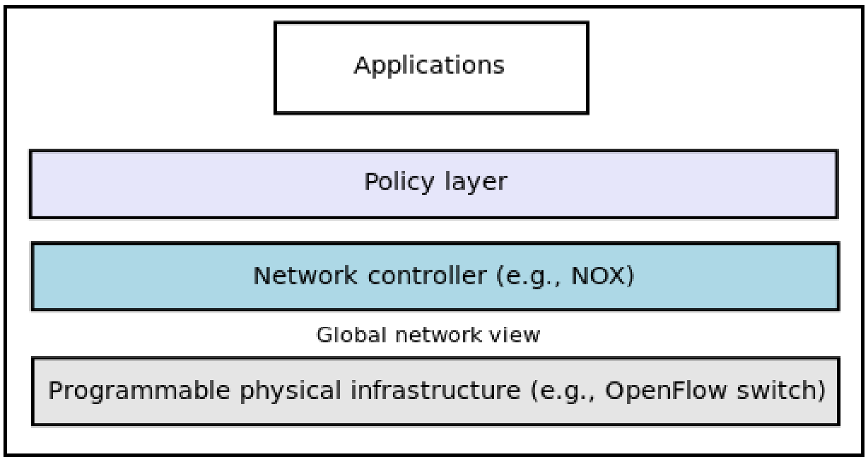

Applications
Policy layer
Network controller (e.g., NOX)
Global network view
Programmable physical infrastructure (e.g., OpenFlow switch)

Figure 1: Layers in Software-Defined Networks [9]

OpenFlow, considered as a short-term solution for the question and first published in 2008 [13], has been widely discussed and experimented with. It defines a protocol between the control plane and data plane in a SDN manner. The basic idea of OpenFlow is straightforward: we can access and manipulate the flowtables in Ethernet switches and routers to control network forwarding.

In the traditional network, the control plane and the data plane are coupled together within a switch (or a router), and the forwarding logic is defined by internal software and protocols. In contrast, OpenFlow separates this forwarding logic and move it to an external controller. Figure 2 shows an OpenFlow-enabled switch and the controller via OpenFlow Protocol.

Usually, an OpenFlow Switch consists of at least three parts [1]:

1. Flow Table: An OpenFlow switch is required to have at least one flow table to perform packet lookup and forwarding. Each flow table contains a set of flow entries with associated counters and actions indicating how to process defined flows. The flow table entry is like this:

Header Fields	Counters	Actions

2. A Secure Channel: The channel is used to connect a remote controller and the switch. Controlling packets can be sent in the channel with OpenFlow Protocol.

3. OpenFlow Protocol: The protocol is implemented between switches and their controllers to provide a standard and programmable way for the control plane and data plane to communicate.

Packets are processed by a OpenFlow switch according to its flow table:

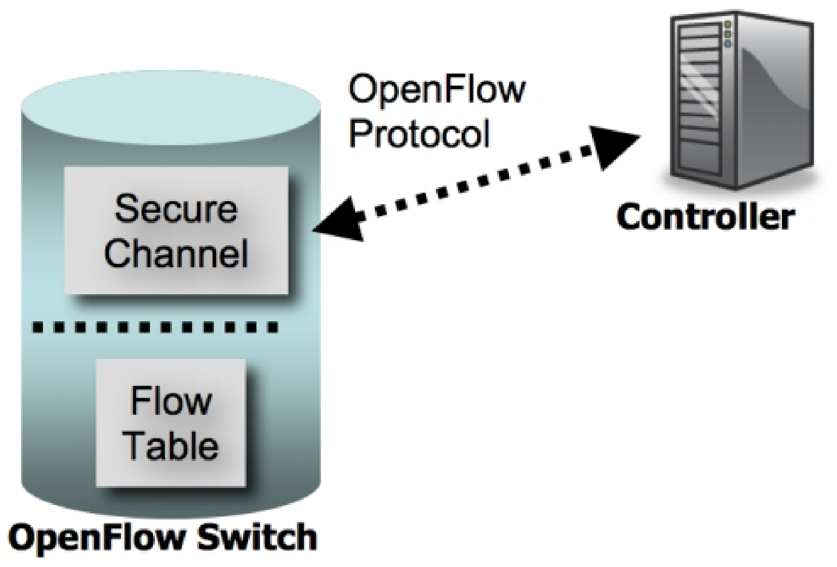


Figure 2: Structure of OpenFlow Network [1]

1. If a packet is matched to a flow entry, the actions for that entry will be performed on this packet.
2. If no match is found, the switch will forward the packet to the controller over the secure channel. The controller needs to determine how to process the packet following a programmable way, and inform the switch to update the flow table.

Figure 3 shows this process by using a flowchart.

3.2 NOX/POX: A Typical SDN Controller

NOX is a SDN controller ("SDN operating system") developed in C++ and Python [7], and POX is a Python version of NOX for rapid development. NOX presents networks with abstract and centralized views, and provides high-level programming interfaces to manage underlying devices.

Figure 4 shows major components in a NOX network: a number of OpenFlow switches and one NOX server running management applications with a centralized view of the network. The network view is collected and updated by using OpenFlow control channels, and different applications can use parts of the up-to-date view for their own controlling purposes.

NOX provides a set of simple programming interfaces which revolve with the concepts of events, network views and namespace abstractions.

- Events: NOX uses events to represent new or incoming changes happened in the whole system (e.g. users leave, links go up and down),

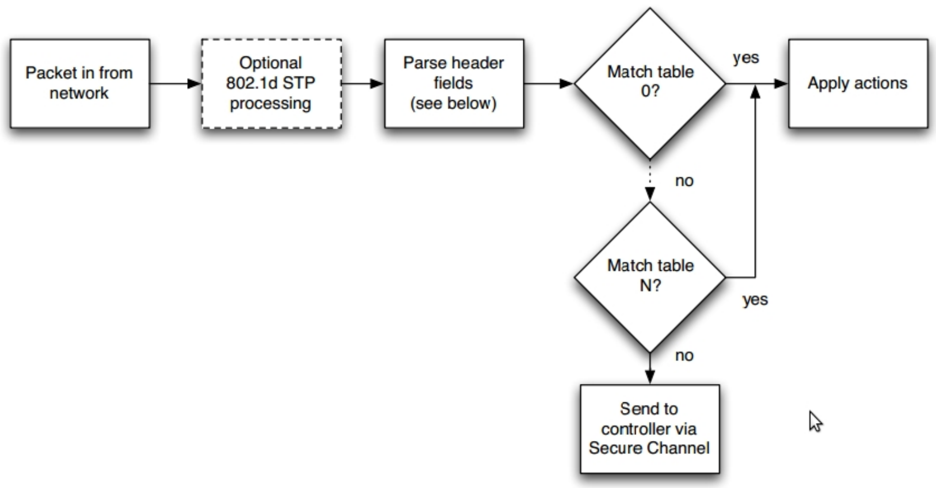

Packet in from network
Optional 802.1d STP processing
Parse header fields (see below)
Match table 0?
yes
Apply actions
no
Match table N?
yes
no
Send to controller via Secure Channel

Figure 3: Flowchart of OpenFlow [1]

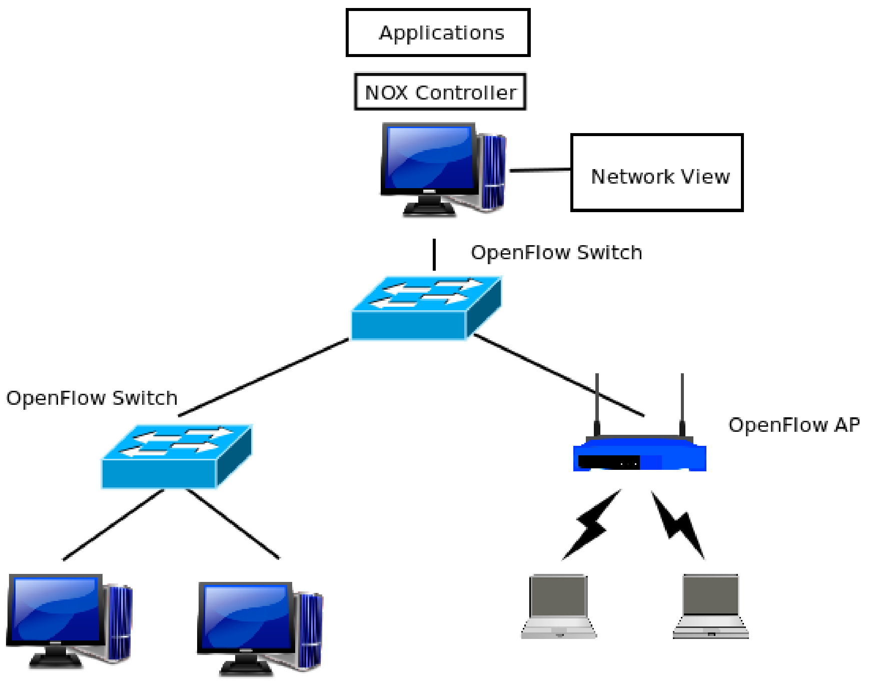


Figure 4: Network consisting of OpenFlow(OF) switches and a NOX controller [7]

and introduces event handlers for coping with these changes. Similar to the handlers used in Unix's system calls, developers could register specific handlers for particular events, and trigger certain control logic. Events could be generated either from specific OpenFlow messages or combination of some low-level events together.

- Namespace and Network View: NOX generates the centralized network view and constructs mappings of high-level namespace to low-level address, allowing developers to manage the network in a simple and topology-independent manner.

- Control: NOX uses the OpenFlow protocol to control and manage underlying devices. It can modify the flow tables of OpenFlow switches and collects the state of underlying traffic. For efficient implementation of common functions, NOX also provides a set of network libraries with common functionalities (e.g. routing, DHCP, DNS and ACL module).

3.3 SDN Deployment for Campus Network

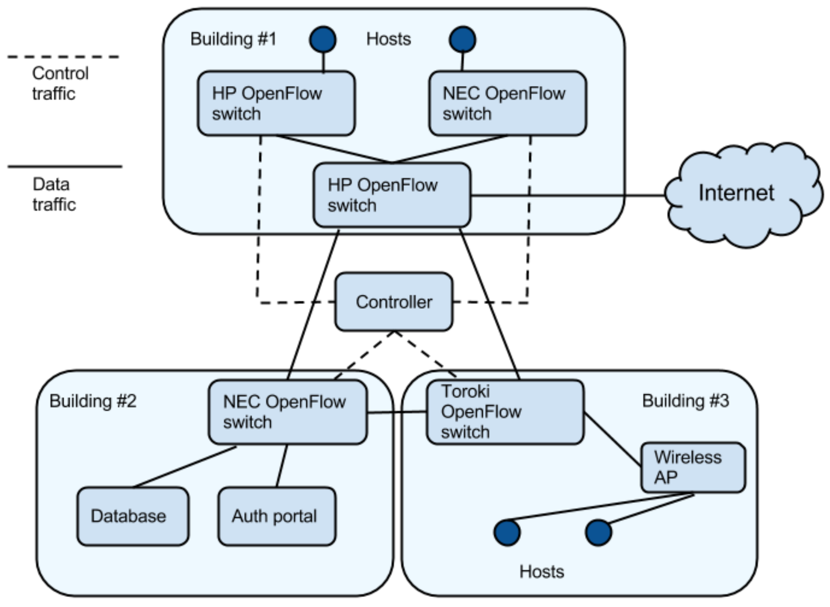


Figure 5: SDN Deployment Example in Georgia Tech [9]

Hyojoon Kim et.al [9] deploy Procera, a software-defined network control framework, at Georgia Institute of Technology. Because of the dynamic environment and complex policies in the campus network, Procera is considered to be a good example of SDN deployment.

As shown in Figure 5, there are three buildings connected with several switches, wireless access points and an authentication system in this network. To join into the campus network, Georgia Tech requires end hosts to first perform an authentication with a legal pair of username and password. After being successfully authenticated, the device needs to be scanned for possible vulnerabilities. Finally, the device could obtain access to the network. The campus network at Georgia Tech uses virtual LAN (VLAN) and several middleboxes to separate registered and unregistered devices. For consistent forwarding and management, switches in the campus network have to update their VLAN tables from a central server constantly.

To implement this kind of network in traditional ways, designers and administrators should set up all the network policies (e.g. VLAN, routing tables) in terms of low-level and device-based configuration. However, Procera simplifies the setting-up process by introducing SDN ideas: all the network configuration policies are moved a central controller, and administrators use software programs to dynamically control traffic forwarding and perform VLAN separation on underlying OpenFlow-enabled switches. In addition to traffic management, authentication has also been changed to a SDN application running on the central controller.

This deployment shows that SDN is a feasible solution for reducing complexity of network management by decoupling the control policies from underlying devices.

4 Extending SDN

To have a better view of SDN, we start to introduce and explain some new research areas and latest topics of SDN in this section.

4.1 Fabric: An Extended SDN Architecture

Although most of the current SDN techniques, taking OpenFlow as an example, achieve flexibility by separating the control and data plane, they are facing a main problem: core management applications running on central controllers are still heavily affected by underlying devices. In other words, current OpenFlow-based systems are far away from our desired networks which fully abstract low-level components and interfaces. For example, if a network starts to switch IPv4 addresses to IPv6 ones, all the matching rules using IP addresses in OpenFlow controllers should to be upgraded.

Martin Casado et al. [5] propose an extension, called Fabric, for today's SDN structures to avoid the limitation of insufficient abstraction. The Fabric architecture is shown in Figure 6. It divides the network into edge switches and core fabric switches. The edge and fabric switches are managed by different controllers. In this new topology, controllers are also distinguished by different purposes: the fabric controllers are only responsible for core forwarding logic, while the edge ones are configured for more detailed policies. To avoid influences caused by underlying changes (e.g.IP addresses changed), fabric controllers could use non-detailed contexts for forwarding, and there should also be some modules for mapping edge (using detailed address) and fabric contexts. The researchers suggest to use MPLS mechanism (labels) to map the edge and fabric context.

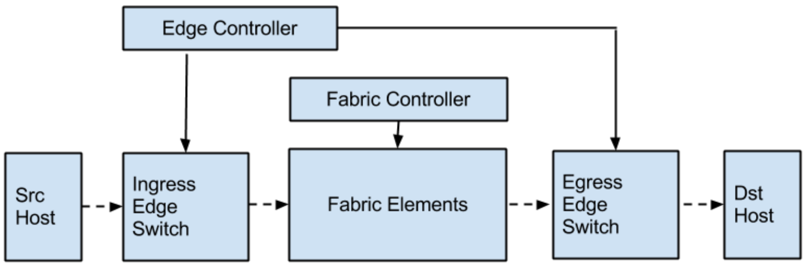


Figure 6: Basic Fabric Structure[5]

This two-level structure brings two main benefits. First, it allows the core and edge control plane to evolve separately. For example, different and more complex services such as security and mobility could be introduced to the edge network without affecting the core forwarding policies. Second, internal forwarding decisions are not dependent on specific addresses of the edge network, which could bring more flexibility and simplicity to SDN.

4.2 Scalable Controller Design

The initial design and implementation of SDN (OpenFlow, NOX) push all the control functionality to a centralized controller for simplicity and flexibility. However, they did not adequately address issues of scalability and reliability for networks. As the size of a network grows, it may be infeasible to use only one central controller to handle all incoming requests. In addition, some switches will encounter long latencies if they are far away from the center of the network, which affects efficiency of the whole network.

For scaling the SDN, researchers have proposed some new SDN structures. In this section we introduce and illustrate some of latest controlling structures for the scaling purpose.

4.2.1 Distributed Controllers

HyperFlow: Distributed Controllers Sharing the Same Global View of the Network HyperFlow [18] is a distributed implementation for OpenFlow, which allows any number of controllers to be deployed in a network. Every controller shares the same consistent network-wide view of the network and serves the incoming control requests for local switches to offer both scalability and centralization to SDN.

The HyperFlow network consists of several NOX servers as the distributed controllers, a set of OpenFlow switches as the data plane, and an event propagation system for synchronization of the network-wide view. Each controller uses the same controlling programs and parameters and runs as if it controls the whole network. Switches are controlled by a nearest controller, and if the controller crashes, they can be dynamically configured to another controller nearby. Figure 7 shows the structure of a HyperFlow network.

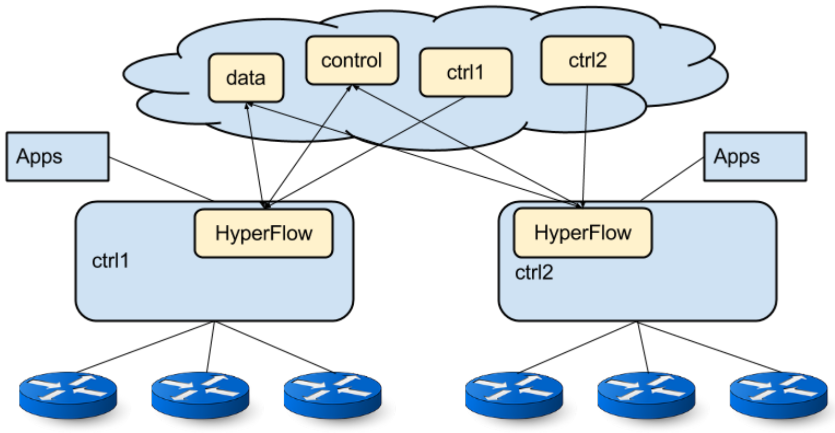


Figure 7: HyperFlow Networks [18]

For synchronizing the global views of a network, HyperFlow uses a publish/subscribe messaging paradigm called WheelFS to propagate controlling events. Here we briefly explain HyperFlow's most featuring functions. Besides the synchronization component, a module for health checking is also added into the original NOX controller.

- Publishing events: the HyperFlow uses the NOX handler mechanism to capture data plane events, and selectively publishes the ones which are generated locally but will affect the global controlling view.
- Replaying events: for synchronizing the control states, a HyperFlow controller replays the events published by other controllers and generates a consistent network-wide view.
- Health checking: the HyperFlow controller is required to send advertisements periodically to indicate its state and condition. If one controller does not publish itself for some intervals, it will be assumed to have crashed, and the switches which it controlled will be redirected to other healthy controllers.

Kandoo: Hierarchical Controllers Kandoo [8] is a hierarchical SDN controller for OpenFlow. It creates a two-level hierarchical SDN controlling system, and distinguishes local controllers and management applications from global ones. The local controllers manage a subset of switches and execute applications which do not need network-wide views, whereas the root controller is responsible for managing all the local controllers and implementing global policies. Figure 8 illustrates this two-level system.

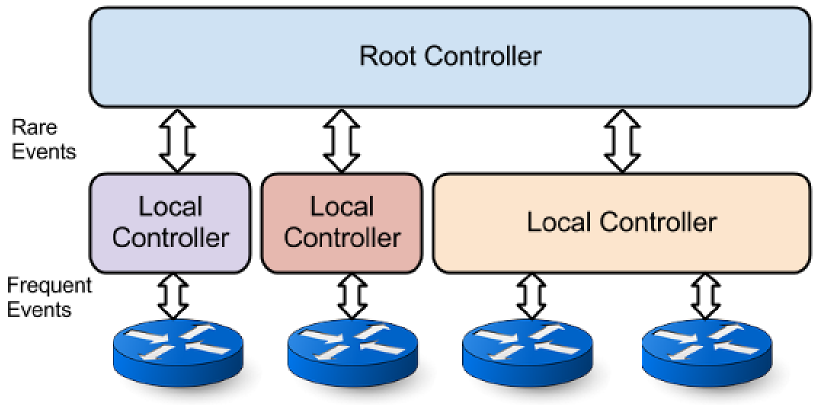
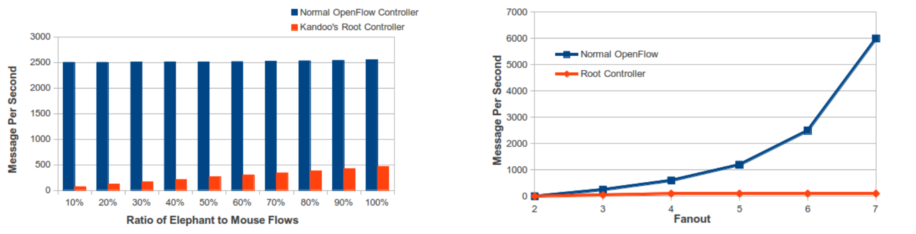


Figure 8: Kandoo's Two Levels of Controllers [8]

By using this hierarchical structure, a network can easily scale by deploying new local controllers. Controllers, organized in the two-level structure, are shield from too many management requests. In addition, the average communication latency between OpenFlow switches and corresponding local controllers can be reduced.

Soheil Hassas Yeganeh et al. evaluate Kandoo's performance by simulating a two-layer hierarchical system with a slightly modified version of Mininet [11]. To test the reduction of controlling interactions between the centralized controller and underlying devices, they design some "elephant" flows which need to be processed by the centralized controller instead of the local ones. First they use a topology of depth 2 and fanout 6, and send hundreds of UDP flows concurrently. The result is shown in Figure 9(a), where we can see a sharp drop of traffic to and from the central controller. Then they fix the ratio of the elephant flows at 20%, and test the controller's traffic with different numbers of fanouts. The result is displayed in Figure 9(b).

Onix: A Mixed Extension Onix [10] is another distributed control platform of SDN developed in C++, which introduces a distributed data structure called Network Information Base (NIB) to record the whole network graph. In NIB, each network element is represented as a specific entry (a set of key-value pairs, like port, link-speed, etc.) with a globally unique



(a) Average Load of Controller for Different Flows

(b) Average Load of Controller for Different Fanouts

Figure 9: Control Plane Load of Kandoo [8]

identifier. The NIB is a bit similar to the shared network view mentioned in HyperFlow. However, it also supports hierarchical topologies as Kandoo. There could be several controllers in an Onix network, and the control logic is distributed by using consistent distributed NIB records. Network applications are implemented by reading and modifying the NIB.

The initial aim of designing Onix is trying to provide a distributed SDN controller with better scalability. For this object, Onix uses some special mechanisms:

- Partitioning: For simplifying the control logic and reduce the redundancy, a specific Onix controller only keeps a subset of the latest NIB in memory. Each controller is only responsible for a part of the network, so it just maintains the NIB information of corresponding network elements.

- Aggregation: Onix supports to expose a collection of network devices in its NIB as one aggregated element to other Onix controllers. This is quite useful to simplify the NIB for hierarchical systems. For example, in a campus network with several controllers, each controller manages a couple of switches. For simplicity, the controller exposes this subset of switches as one switch to upper-level Onix controllers.

As a distributed SDN platform, one of the key points in Onix is to maintain a consistent NIB for the whole network. Based on the observation that different applications often have various requirements on consistency, the researchers design two kinds of mechanisms with different update speeds and consistency types. For applications with low changing frequency, Onix could implement a transactional consistent database by using a duplicated state server for sending required states with SQL-like APIs. For networks with high update rates, Onix provides an eventually-consistent DHT system.

So far, we have reviewed and introduced some SDN controllers. In Table 1, we summarize some key points of these SDN controllers and compare them with each other.

4.2.2 Datapath Extensions

DevoFlow and DIFANE DevoFlow [6] is a modification of OpenFlow, which extends functionalities of the data plane. It introduces new mechanisms to allow switches to make local forwarding decisions for some flows and provides efficient statistical methods to lower switch-controller interactions.

For devolving control to a switch, DevoFlow introduces rule cloning and local actions to a switch:

- Rule cloning: OpenFlow uses a wildcard rule to match a specific set of flows to reduce the number of flow table entries. However, this would aggregate all the flows of a given set into one group with the same counter and actions. In DevoFlow, the switch can locally clone the wildcard rule and replace the common fields with values of specific flows. By creating local rules, the switch can collect statistics for specific flows by only using one wildcard rule.
- Local actions: In DevoFlow, the switch can take a set of local actions without communicating with the controller.

In addition to the control logic, DevoFlow provides three new ways to lower the cost of statistics collection.

- Sampling: DevoFlow allows the switches to only collect the head info of the flows at a specific rate (e.g. 1/1000 packets).
- Triggers and reports: When a trigger condition is met, the switch will send a statistic message to the controller. A trigger condition can be a combination of some threshold values for specific flows or packets.
- Approximate counters: This is designed for wildcard rules. DevoFlow switches can count the top-k largest flows in a specific group based on rule cloning.

DIFANE [23] is another datapath extension system which follows a similar design idea as we mentioned in DevoFlow. It tries to use some special switches, called authority switches, to assist the centralized controllers to make forwarding decisions.

Table 1: Comparison of Different SDN Controllers

	NOX	HyperFlow	Kandoo	Onix
Architecture	Simply centralized	Flat-distributed, all the controllers share the same view of the network	Two-level Hierarchical Controllers	Distributed System supporting both flat and hierarchical structures by using NIB
Feature	Simple architecture and programming interfaces	Consistent global view of network	Controlling partition	Distributed network information database
Scalability	Centralized controller	No centralized controller for the whole network, one controller handles parts of devices	Divide the controllers into edge and core	Distributed controllers with well designed scalability strategies
Reliability	No mechanism designed	Designed mechanism for detecting a failed node and taking over the responsibilities of the failed instance quickly	No mechanism designed	Designed several mechanisms for handling different failures
Configuration and Update	Easy, only in the centralized controller	Easy, all the controller use the same parameters	Different devices use different configurations	Different devices use different configurations
Implementation	Real network implementation (C++ and Python)	Small-scale prototyping (C++)	Small-scale prototyping (C, C++, and Python)	Real network implementation (C++)
Evaluation	Evaluated by other researchers in different scenarios [17]	Simulation	Simulation in Mininet-like environment	Tested in small-scale network
Applicable Scenario	For general cases	For large scale network	For large scale network	For general cases with high requirements for scalability and reliability
Limitation	1) not suitable for very large networks 2)no reliable mechanisms	1) event propagation may be complicated and fragile	1) Consistency and logic partition may be challenging 2) Update may be complicated	1) Update may be complicated

4.3 SDN and Middlebox

Middleboxes have been a crucial part of today's network for providing a variety of functionalities and improving global performance. However, current management mechanisms of middleboxes are clumsy and complex. Administrators have to carefully plan the topology and set up different policies manually, which is fault-prone and difficult to update.

Recently some researchers attempt to integrate middleboxes into a software-defined network to simplify deployment and management. Since middleboxes may modify flow contexts, and have strong internal logic for high-level policies, existing SDN platforms and mechanisms are not very suitable for deploying them directly. The modifications caused by middleboxes make it difficult for SDN controllers to separate flows with desired policies. Here we introduce two solutions for deploying and managing middleboxes in a SDN system.

4.3.1 SIMPLE: A Middlebox Enforcement SDN

SIMPLE [14], an SDN-based enforcement layer for middleboxes, is designed to help SDN controllers to configure middleboxes and translate high-level policies into the data plane parameters efficiently.

Figure 10 shows the overall architecture of SIMPLE. It processes middlebox policy with high-level dataflow abstraction. The administrators do not need to worry about the middlebox location and the routing policies, they just specify the control logic. For example, they can first indicate the policy class like this: $\{src = prefix_1, dst = prefix_2, dstport = 80, proto = TCP\}$, and then specify the policy chain {NAT, Firewall}. To translate the logical policy to specific physical devices, SIMPLE maintains mappings which indicate the locations and capacities of each middlebox. Furthermore, because resources in a network are limited (e.g. device CPU, memory, bandwidth, available rules in OpenFlow switches), SIMPLE also designs algorithms for efficiently utilizing those constrained resources.

As shown in Figure 10 with red blocks, there are three main components in SIMPLE:

- Resource Manager: this component translates logical policies into specific requirements by taking into account both the resource constraints and the network topology.
- Dynamics Handler: since middleboxes may modify the header or the context of a packet, SIMPLE uses a lightweight correlation algorithm to map incoming and outgoing packets of one middlebox to provide a consistent view of the network.
- Rule Generator: this module generates the final configuration for the

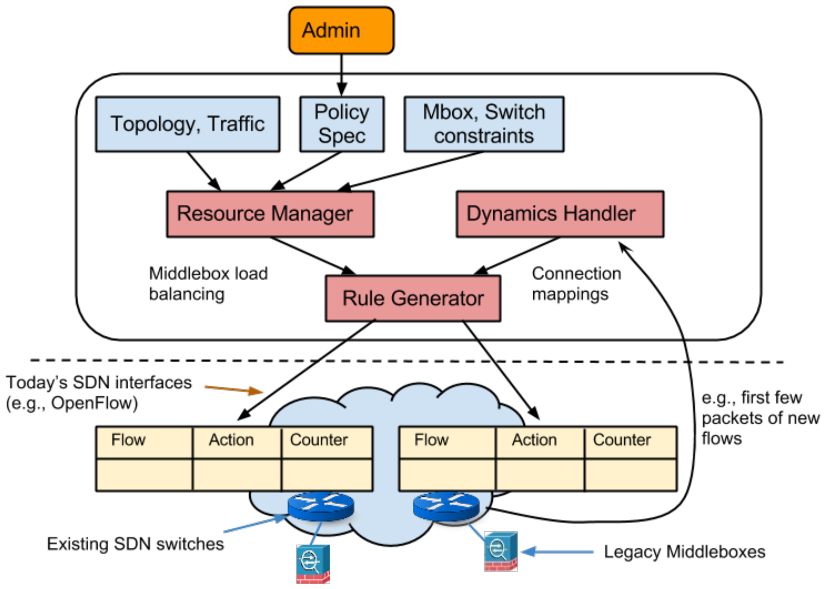


Figure 10: Architecture of SIMPLE [14]

data plane to forward flows via different middleboxes in right the sequence.

4.3.2 FlowTags

FlowTags [15] is an extension to current existing SDN architectures like OpenFlow. It mainly focuses on providing global visibility and management to middleboxes. FlowTags adds contextual information of middleboxes in terms of tags inside OpenFlow packet headers. All the downstream middleboxes in SDN can use these tags to make consistent decisions on packet forwarding and traffic statistics.

The key idea of FlowTags is to add tag info to the packets with middlebox context. Switches and middleboxes in FlowTags add or modify tags in the packet headers, and handle those packets with tag-related actions indicated by the controller. Because of header-size constraints, tags are encoded compactly in FlowTags.

4.4 SDN and Cloud

Have being attracted by the potential benefits of SDN on large-scale network management, some researchers try to apply SDN architectures into datacenters and integrating them with cloud services. For example, Arsalan Tavakoli et al. [17] demonstrated that NOX can provide a variety of datacenter functionalities. Google has already deployed OpenFlow in their datacenter [16].

To have a detailed understanding of SDN and the cloud, here we introduce a SDN platform called Meridian.

Meridian [4] is a SDN platform proposed for cloud network services. It aims to provide common service models and programming interfaces to cloud controllers and managers.

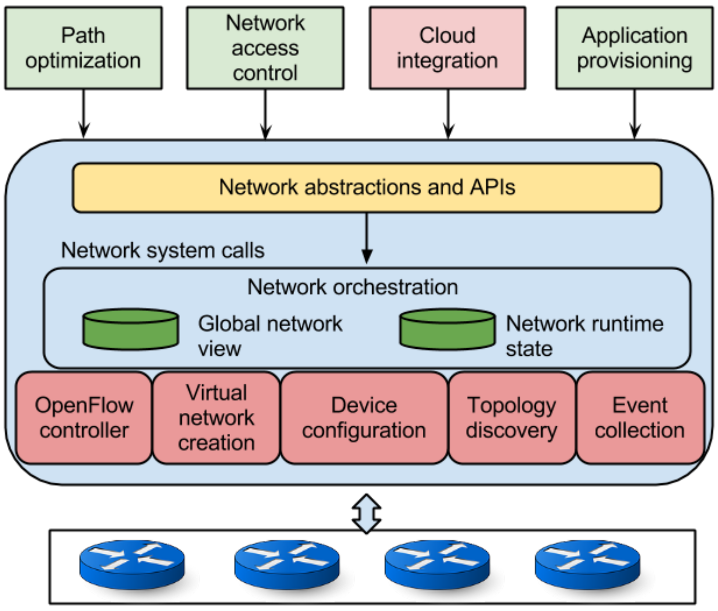


Figure 11: Meridian Architecture [4]

As shown in Figure 11, it is organized with three main logical layers: network abstraction, network orchestration, and network driver layer. The cloud applications and managers are at the top of the network abstraction layer. The abstract layer represents underlying networks in terms of logical topologies and provides interactive APIs to upper applications. The network orchestration layer performs a logical-to-physical translation and offers global views of underlying networks. The lowest layer of Meridian can be considered as the "drivers" of the underlying devices to provide unified control logic. For example, it can use OpenFlow to manage switches to create desired virtual networks.

5 SDN and Cellular Networks

Nowadays, more and more researchers start to focus on centralized control of wireless networks: a central controller manages a set of wireless access points for non-overlapping channel allocating, consistent user authentication and interference avoiding. Compared with legacy distributed wireless networks, a centralized system has better consistency since the controller has a global view of the network. For example, a mobile user can switch between different access points without changing the IP address in a centralized campus network, or forward messages across different wireless technologies (e.g. 3G, LTE, and WiFi). In this section, we move our attention to SDN and wireless network and introduce some of the latest work in this area.

5.1 OpenRoads: Wireless OpenFlow

OpenRoads [20] is an early attempt to develop a wireless SDN platform with OpenFlow and NOX by Stanford University. It separates the control plane from the datapath, and produces network slices by using FlowVisor to isolate different flows. The underlying infrastructure (e.g. power levels, wireless channels, and interface states) is configured with SNMPVisor, a command line interface for setting up datapath elements with the SNMP protocol. In other words, OpenRoads allows several different experiments and services to run simultaneously over one physical network.

In OpenRoads, the researchers add OpenFlow to WiFi access points and WiMAX base stations for traffic controlling, and use NOX as the network controller which can communicate with OpenFlow devices and provide global views of the network. FlowVisor can be considered as a transparent proxy for OpenFlow. It slices the network by selectively rewriting or dropping OpenFlow Messages to delegate control of different flows with different controllers. The basic structure of OpenRoads is shown in Figure 12: OpenRoads could successfully separated different users' traffic with different forwarding policies.

The researchers use 30 WiFi APs running Linux, 2 WiMAX base stations and 5 Gigabit Openflow-enabled switches in the initial deployment, and have created a software-defined network where different experiments can run in parallel in the same physical infrastructure.

5.2 SoftRAN

Currently the radio access network uses distributed algorithms to manage limited spectrum and enact handovers. While the decision in a sparse environment with few base stations could be straightforward to make, it would be more difficult to quickly choose the best candidate when the deployment scales. To handle growing mobile traffic and dense deployments of base stations, some researchers proposed SoftRAN [2], a centralized software-defined radio access network, for efficiently performing handovers and allocating spectrum resources as well as setting up transmit power values.

In SoftRAN, LTE networks are controlled in a centralized way: all the base stations are abstracted as a virtual element and managed by the logically central controller. The controller maintains the global states of the network and makes logical decisions. There are defined APIs for the control plane to

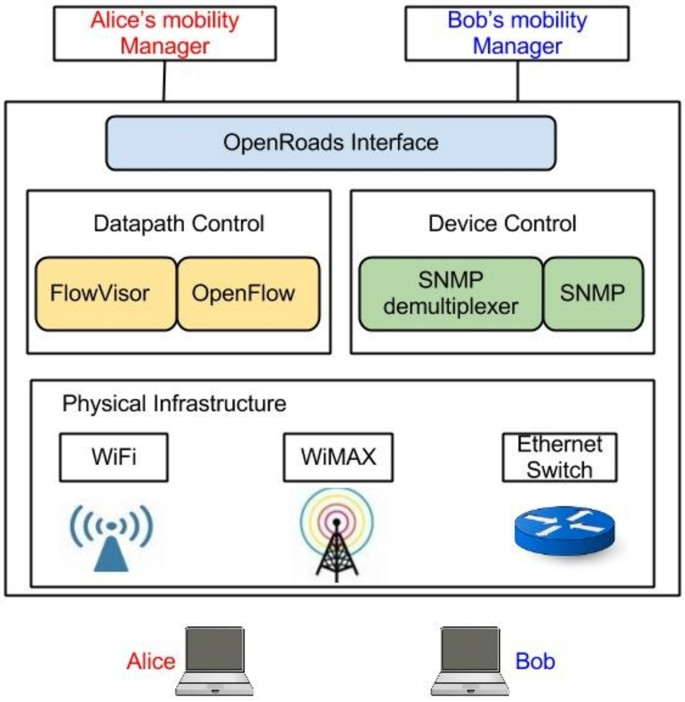


Figure 12: OpenRoads Architecture [19]

communicate with radio elements to update the global view of the network and configure every base station.

Figure 13 shows the architecture of SoftRAN. It collects states of base stations periodically and updates the global view within a database. The information stored in the database is utilized by the controller modules for future radio resource management.

For handling the inherent delay between the centralized controller and different base stations, some control tasks which are based on local network parameters could be moved to specific individual radio elements. SoftRAN has two main principles for separating the control plane. First, the control decisions affected by neighbouring radio elements should be made at the centralized controller, e.g. handovers, transmit powers setting. Second, decisions depending on rapidly varying parameters should be made locally by base stations preferably, e.g. resource block allocation.

5.3 SoftCell: Hierarchical SDN for Cellular Networks

Compared with wired networks, cellular networks have some unique features and face significant scalability challenges. For example, because users are always moving in cellular networks, there will be a large number of state updates generated from the data plane, which would create big pressure on

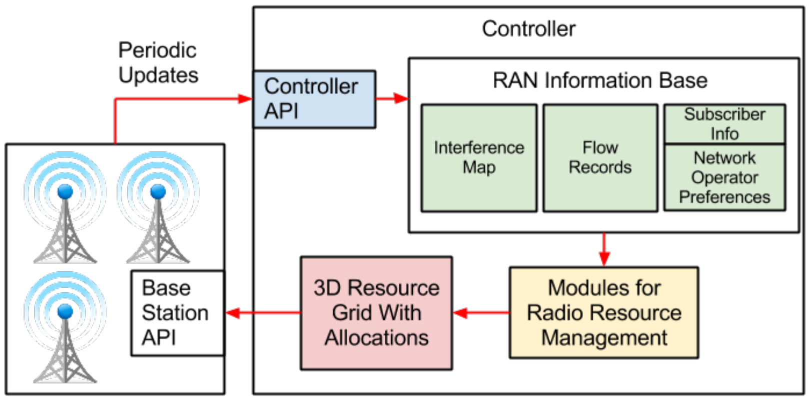


Figure 13: Architecture of SoftRAN [2]

a central controller. In addition, the average response latency would also increase sharply when a set of base stations communicate with one remote controller concurrently.

Li Erran Li et al. [12] propose a SDN architecture with local control agents, which is designed for handling this problem. This local software control agent can make simple decisions for a single base station. The cellular SDN architecture is illustrated in Figure 14. In addition to the local controller, traffic policies and cellular resources are represented with high-level abstraction rather than IP addresses or physical identities in SoftCell.

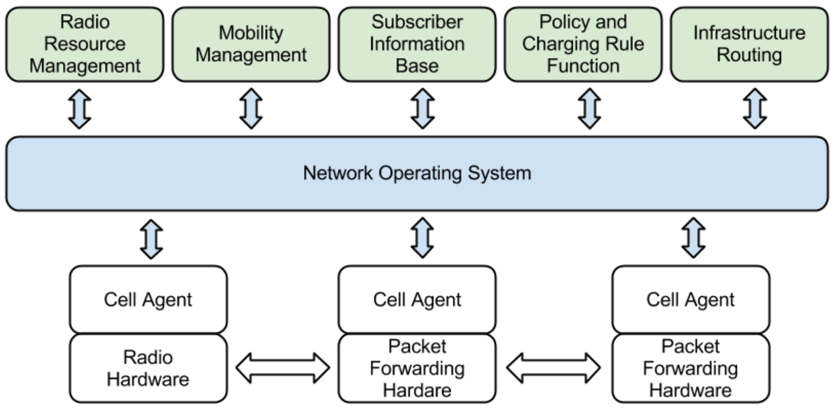

Radio Resource Management
Mobility Management
Subscriber Information Base
Policy and Charging Rule Function
Infrastructure Routing
Network Operating System
Cell Agent
Radio Hardware
Cell Agent
Packet Forwarding Hardare
Cell Agent
Packet Forwarding Hardware

Figure 14: Cellular SDN With Local Agents [12]

In Table 2, we summarize some key points of cellular SDN architectures and compare the three techniques mentioned before.

Table 2: Comparison of Different Cellular SDN Architectures

	OpenFlow Wireless (a.k.a. OpenRoads)	SoftRAN	SoftCell
Structure	Centralized controller	Local agent + Centralized controller	Local agent + Centralized controller
Feature	Provide resource virtualization, different policies can work together	1) Separation of local and core functionalities 2) Consistent centralized control	1) small switch tables by rule aggregation 2) Separation of local and core functionalities
Resource Virtualization	Using FlowVisor to slice the network, different configurations can be deployed in the same network together	No virtualization mechanism, resources are managed by the controller with a centralized 3D resource grid	Not mentioned
Scalability	Centralized controller makes all the decisions	Local agents make simple decisions to reduce the workload to the centralized controller	Similar to SoftRAN
Reliability	No mechanism designed	No mechanism designed	Controller failure is handled by maintaining a distributed, consistent copy of the control-plane state
Implementation	Real network implementation	small-scale prototyping	prototyping on top of the Floodlight OpenFlow controller
Evaluation	Tested in real network	Not fully tested	Trace-driven simulation
Applicable Scenario	For cellular and WiFi network	For cellular network	For cellular and WiFi network
Limitation	Not design any scalable and reliable network architecture for fine-grained policies	1) Only focus on radio resources, no mechanism designed for mobile flow traffic 2) Scalability and reliability are not considered	Consistent updating for the controllers could be a challenge

6 SDN-enabled Dynamic Mobile Traffic Management

6.1 Framework

After reviewing and introducing some SDN systems, we can easily find that current cellular network architectures are facing several challenges of openness and simplicity. The network infrastructure is close to innovation: each equipment inside the network has vendor-dependent controlling interfaces, which are difficult to configure and manage. As shown in Figure 15, the control plane and the data plane are coupled together in the current cellular network. For setting up such a system, administrators need to configure all the equipments separately.

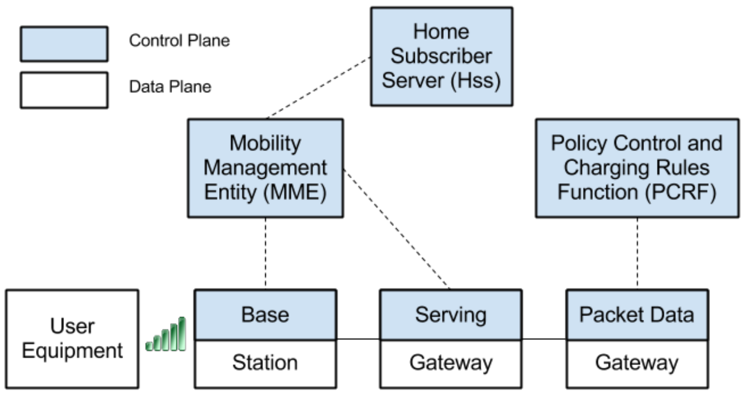

Control Plane
Data Plane
Home Subscriber Server (Hss)
Mobility Management Entity (MME)
Policy Control and Charging Rules Function (PCRF)
User Equipment
Base Station
Serving Gateway
Packet Data Gateway

Figure 15: Architecture of Legacy Cellular Networks [12]

As we mentioned in Section 5, recent work has recognized the problems of the architecture coupled with the data and control plane and suggested some new systems for a cellular network. Inspired by SoftRAN and SoftCell, we hope to develop a new SDN cellular platform for innovation, and plan to use a two-level hierarchical structure to solve the inherent delay between the central controller and the radio elements in the data plane.

- the local controller performs simple local actions and measurement
- the centralized controller has a global view of the network and uses some high-level abstract policies to manage all the elements in the network

Integrating hierarchical SDN controllers into wireless networks also raises many interesting research problems and open questions:

- How to partition functionalities between local and central controllers: the hierarchical two-layer SDN architecture could reduce the average

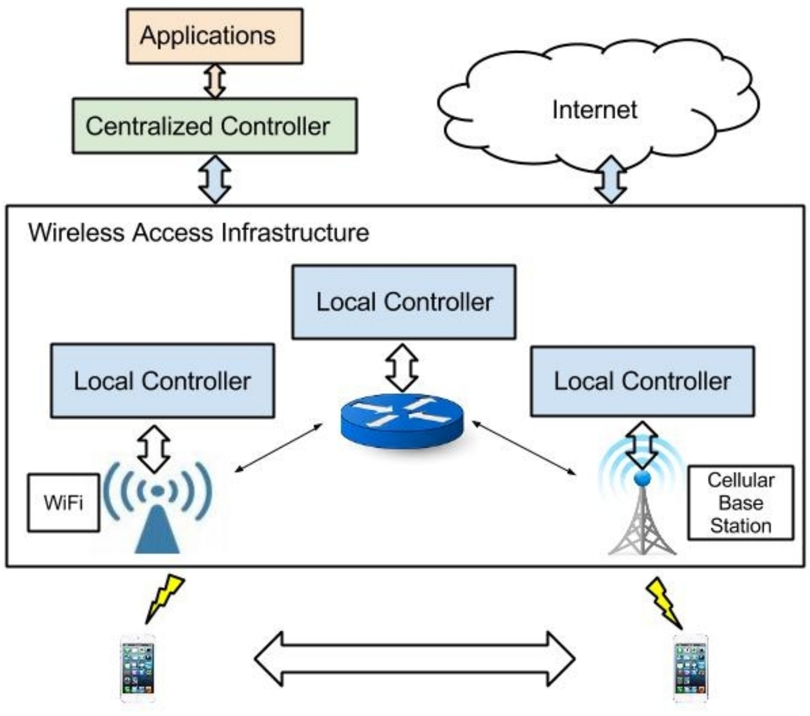


Figure 16: Architecture of Our Proposed System

workload and request latency of each controller. However, how to divide policies into different levels is still an open question today. SoftRAN and SoftCell have made some discussion about it, but do not explore it to all kinds of applications. Aurojit Panda et al. [3] analyze the problem in theory, but not in practical work. We plan to explore the hierarchical structure and local partition techniques in our future work.

- How many controllers do we need and where to place them: besides partitioning, we also plan to explore this practical question about deployment based on testing and simulation.
- Interaction between multiple types of access technologies (e.g. cellular, wifi): one of our goals is to use SDN techniques to implement seamless mobility between different wireless access technologies (e.g. 3G, LTE, WiFi), which requires a large amount of work in centralized controlling, mobility prediction and billing.
- Consistency: with the increasing of controllers, consistent management and update will also be challenging in the SDN system. Since there have already been several projects in this area, we will try to utilize previous works, like HotSwap, and apply them in our framework.
- Smart usage of context from mobile users' devices: nowadays, we are experiencing an explosion of mobile apps and services, which generate a large amount of user data (e.g., sensing context, WiFi-related data and GPS info) that could be used to identify traffic patterns and predict user and environment status. SDN leads to a very good opportunity to utilize these data because of its more powerful monitoring tools and high-level network views. However, there is still no standard mechanism for using these contexts in a software-defined system.

6.2 Monitoring

While many researchers have focused on network controller and management of SDN, few efforts are put into monitoring and measurement of software-defined network. Existing methods, such as OpenFlow, are not suitable for large-scale measurement tasks. They are constrained by limited hardware memory and communication overhead. Compared with wired networks, it could be more difficult to monitor wireless and cellular networks because users and traffic patterns are always changing.

FlowSense [21] is one of the latest proposals to monitor SDN. It uses the controller to analyze and estimate traffic utilization. The devices in the data plane are only responsible for sending necessary information messages for new and expired flows, but not collecting the counter. FlowSense can reduce the workload of the data plane, and the overhead for controlling. However, it does not fully utilize switches and equipments in underlying networks. OpenSketch [22] is another new SDN-based measurement solution which separates the measurement control from basic measurement functionalities in the data plane. It provides flexible programming interfaces for a variety of monitoring tasks, but is a bit heavy-weight for some simple measurement jobs.

In our future work, we plan to explore some of the latest SDN measurement solutions, like OpenSketch, and extend them for cellular networks. The monitoring tool should be accurate and lightweight with fewer resource constraints. The main challenge is to handle radio resources and mobility in cellular network. Two-level hierarchical system should be a good choice to reduce the overhead and workload to each controller, but we need to design solutions for handling frequent changes of states in a highly dynamical network with users moving.

6.3 Prediction

As one of the hottest topics in wireless and cellular networks, predicting user mobility and traffic patterns has been studied for over a decade, and several different algorithms are developed and published in this area. However, few of these algorithms are deployed in practical network because of the complexity of both the algorithms and real mobile users.

In our future work, we plan to utilize and extend previous work in network prediction for wireless SDN framework. Thanks to global views and efficient monitoring tools of SDN, we may have an opportunity to improve the reliability and availability of existing predicting solutions. However, how to accurately predict is still a big challenging problem.

6.4 Management

6.4.1 Resource Management

Because of the centralized control of network equipments, a SDN controller could have a global view of the network, which is very beneficial and efficient to allocate limited resources (e.g., wireless radio spectrum). However, management solutions based on flow table in OpenFlow switches are not suitable for controlling radio resources. We need some new abstracted way to represent traffic policies for wireless networks. In our future work, we plan to extend our platform for efficient management of cellular and wireless resources, including radio spectrum, transmission power and traffic flows.

6.4.2 Mobility

Today's cellular network does not provide optimal mechanisms for handover across different wireless technologies (e.g., 3G, WiFI and LTE). It is complex and prone to lead to higher packet loss rate and longer latency if users always switch among different technologies. However, SDN would be a good way to control and perform seamless handover across different protocols because of its global view of the network. Although some researchers suggest this will be a promising research area, we do not see practical implementations on different-protocol handover. We plan to explore this area and try to design some new algorithms to make mobility management much easier in our SDN platform. This work is also partially dependent on the prediction algorithms.

6.4.3 Run-time Adaptive Configuration

As mentioned before, we hope to develop some prediction algorithms for wireless traffic and user mobility with the help of SDN measurement, and the prediction result, therefore, could be used to reconfigure cellular networks in run time. Besides load balance, it is possible to design some mechanisms for better utilization of the wireless resources. As far as we are concerned, there is still no application or implementation integrating SDN prediction and dynamical configuration for mobile networks.

References

[1] The openflow switch specification. `http://OpenFlowSwitch.org`.

[2] Gudipati Aditya, Perry Daniel, Erran Li Li, and Katti Sachin. Softran: software defined radio access network. In *Proceedings of the second ACM SIGCOMM workshop on Hot topics in software defined networking*, pages 25–30. ACM SIGCOMM, 2013.

[3] Panda Aurojit, Scott Colin, Ghodsi Ali, Koponen Teemu, and Shenker Scott. Cap for networks. In *Proceedings of the ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN 2013)*, August 2013.

[4] Mohammad Banikazemi, David Olshefski, Anees Shaikh, John Tracey, and Guohui Wang. Meridian: an sdn platform for cloud network services. *IEEE Communications Magazine*, 51:120–127, 2013.

[5] Martin Casado, Teemu Koponen, Scott Shenker, and Amin Tootoonchian. Fabric: a retrospective on evolving sdn. In *Proceedings of the first workshop on Hot topics in software defined networks*, HotSDN '12, pages 85–90, New York, NY, USA, 2012. ACM.

[6] Andrew R. Curtis, Jeffrey C. Mogul, Jean Tourrilhes, Praveen Yalagandula, Puneet Sharma, and Sujata Banerjee. Devoflow: scaling flow management for high-performance networks. *SIGCOMM Comput. Commun. Rev.*, 41(4):254–265, August 2011.

[7] Natasha Gude, Teemu Koponen, Justin Pettit, Ben Pfaff, Martín Casado, Nick McKeown, and Scott Shenker. Nox: towards an operating system for networks. *SIGCOMM Comput. Commun. Rev.*, 38(3):105–110, July 2008.

[8] 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, pages 19–24, New York, NY, USA, 2012. ACM.

[9] Hyojoon Kim and Nick Feamster. Improving network management with software defined networking. *IEEE Communications Magazine*, 51(2):114–119, 2013.

[10] Teemu Koponen, Martin Casado, Natasha Gude, Jeremy Stribling, Leon Poutievski, Min Zhu, Rajiv Ramanathan, Yuichiro Iwata, Hiroaki Inoue, Takayuki Hama, and Scott Shenker. Onix: a distributed control platform for large-scale production networks. In *Proceedings of the 9th USENIX conference on Operating systems design and implementation*, OSDI'10, pages 1–6, Berkeley, CA, USA, 2010. USENIX Association.

[11] Bob Lantz, Brandon Heller, and Nick McKeown. A network in a laptop: rapid prototyping for software-defined networks. In *Proceedings of the 9th ACM SIGCOMM Workshop on Hot Topics in Networks*, Hotnets-IX, pages 19:1–19:6, New York, NY, USA, 2010. ACM.

[12] Erran Li Li, Mao Z. Morley, and Rexford Jennifer. Toward software-defined cellular networks. *European Workshop on Software Defined Networking (EWSDN)*, 0:7–12, 2012.

[13] Nick McKeown, Tom Anderson, Hari Balakrishnan, Guru Parulkar, Larry Peterson, Jennifer Rexford, Scott Shenker, and Jonathan Turner. Openflow: enabling innovation in campus networks. *SIGCOMM Comput. Commun. Rev.*, 38(2):69–74, March 2008.

[14] Zafar Ayyub Qazi, Cheng-Chun Tu, Luis Chiang, Rui Miao, Vyas Sekar, and Minlan Yu. Simple-fying middlebox policy enforcement using sdn. In *Proceedings of the ACM SIGCOMM 2013 conference on SIGCOMM*, SIGCOMM '13, pages 27–38, New York, NY, USA, 2013. ACM.

[15] Fayazbakhsh Seyed, Sekar Vyas, Yu Minlan, and Mogul Jeff. Flowtags: Enforcing network-wide policies in the presence of dynamic middlebox actions. In *Proceedings of the ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN 2013)*, August 2013.

[16] Jain Sushant, Kumar Alok, Mandal Subhasree, Ong Joon, Poutievski Leon, Singh Arjun, Venkata Subbaiah, Wanderer Jim, Zhou Junlan, Zhu Min, Zolla Jon, Hölzle Urs, Stuart Stephen, and Vahdat Amin. B4: experience with a globally-deployed software defined wan. In *ACM SIGCOMM 2013 Conference, SIGCOMM'13, Hong Kong, China, August 12-16, 2013*, pages 3–14. ACM, 2013.

[17] Arsalan Tavakoli, Martin Casado, Teemu Koponen, and Scott Shenker. Applying nox to the datacenter. In *HotNets*. ACM SIGCOMM, 2009.

[18] 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*, INM/WREN'10, pages 3–3, Berkeley, CA, USA, 2010. USENIX Association.

[19] Kok-Kiong Yap, Masayoshi Kobayashi, Rob Sherwood, Te-Yuan Huang, Michael Chan, Nikhil Handigol, and Nick McKeown. Openroads: empowering research in mobile networks. *SIGCOMM Comput. Commun. Rev.*, 40(1):125–126, 2010.

[20] Kok-Kiong Yap, Rob Sherwood, Masayoshi Kobayashi, Te-Yuan Huang, Michael Chan, Nikhil Handigol, Nick McKeown, and Guru Parulkar.

Blueprint for introducing innovation into wireless mobile networks. In *Proceedings of the second ACM SIGCOMM workshop on Virtualized infrastructure systems and architectures*, VISA '10, pages 25–32, New York, NY, USA, 2010. ACM.

[21] Curtis Yu, Cristian Lumezanu, Yueping Zhang, Vishal Singh, Guofei Jiang, and Harsha V. Madhyastha. Flowsense: monitoring network utilization with zero measurement cost. In *Proceedings of the 14th international conference on Passive and Active Measurement*, PAM'13, pages 31–41, Berlin, Heidelberg, 2013. Springer-Verlag.

[22] Minlan Yu, Lavanya Jose, and Rui Miao. Software defined traffic measurement with opensketch. In *Proceedings of the 10th USENIX conference on Networked Systems Design and Implementation*, nsdi'13, pages 29–42, Berkeley, CA, USA, 2013. USENIX Association.

[23] Minlan Yu, Jennifer Rexford, Michael J. Freedman, and Jia Wang. Scalable flow-based networking with difane. In *Proceedings of the ACM SIGCOMM 2010 conference*, SIGCOMM '10, pages 351–362, New York, NY, USA, 2010. ACM.