A Road Map for SDN-Open Flow Networks

N.Venkata Ramana Gupta, Dr. M.V. Ramakrishna

***Abstract*— Software Defined Networking is the most recent advancement, extending the idea of programmable networks. Software-defined networking (SDN) has generated tremendous curiosity from both academia and industry. SDN framework decouples the network service from underlying implementation providing a novel approach for application to configure, operate and communicate with the network. SDN aims at simplifying network management while enabling researchers to experiment with network protocols on deployed networks. This article is a purification of the state of the art of SDN in the context of Computer networks. We extant a synopsis of the major design trends, SDN architecture, its current and future applications and highlight key differences between them. OpenFlow is the first standard interface designed specifically for SDN, providing high performance, granular traffic control across multiple networking devices. This Paper looks at the fundamentals of OpenFlow, as one of the early implementations of the SDN concept. Starting from OpenFlow switches and controllers up to the development of OpenFlow-based network applications (Net Apps) and network virtualization.**

***Index Terms*— Control plane, Data plane, Network Protocols , SDN, OpenFlow , Programmable Networks, Software Defined Networking.**

I. INTRODUCTION

In this article, we review the published research literature on the application of software-defined networking (SDN) ideas in computer networks. We begin with a brief overview of the history of SDN and what the term entails. Software-defined networking is a broad term that is applied to a variety of approaches to network design. Conventionally, packet-switched networks have consisted of nodes running distributed protocols to route packets. There is little, if any, control over the forwarding tables of a switch. The *control* of the path of a packet is given to individual routers making decisions according to some distributed algorithm. SDN is a network view that argues for a separation of routing intelligence from the device itself. In software, this would enable the management of the forwarding tables of these "dumb" switches by a possibly integrated controller. This parting of the control and data planes allows one to experiment with network protocols in an lucid manner than what is possible today.The vision is to have a common set of hardware, such as switches, that form a base over which various network functionalities are implemented in software. A rough analogy is how the x86 instruction set provides a common architecture for Personal computers over which many software implementations, providing different functionalities, are run. Figure 1 provides an illustration of the basic SDN architecture.The earliest efforts in SDN, such as Ethane [1], RCP [2], and 4D [3], arose as a protest against the ossification of networks. Researchers were being increasingly aware of the need to simplify the functions baked into network infrastructure and wished for the ability to extend network devices with desired functionality as and when needed.

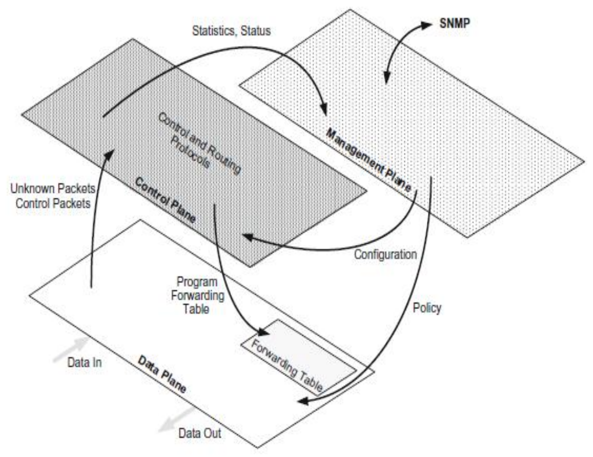


Fig. 1. Basic SDN architecture.

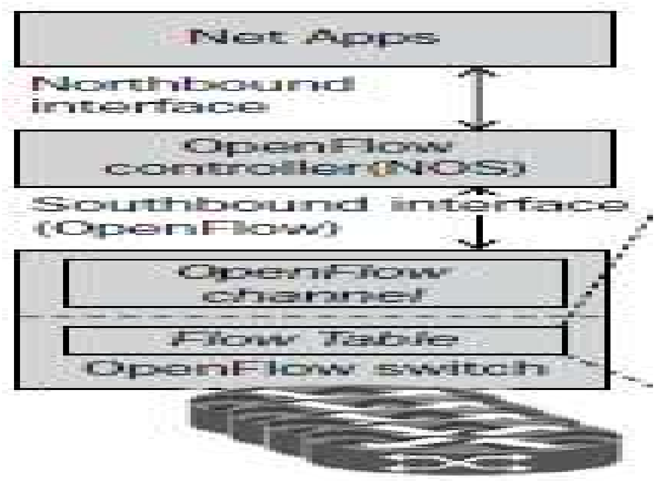
Ethane introduced a flow-based policy language to a network comprised of a simplified data plane and a centralized control plane. It demonstrated the feasibility of operating a centrally managed network. Open- Flow [4] built on this work by defining an open protocol that defines communication between the network controller and a network device such as a switch. Some of the ideas that characterize present-day SDN predate the coining of the term [5]. SDN originally referred to the OpenFlow project at Stanford. Since then, the term has evolved to encompass any network architecture possessing the following two properties [6]. First, the decisions on how to handle packet data are separated from the operations that carry out those decisions. This property is commonly known as *control and data plane separation*. The control and data planes interface with each other using a well-defined API such as OpenFlow. Second, the control plane allows the operation of a possibly disparate set of devices from a single vantage point. SDN uses packet-switched networks, adding features such as flow-based routing.

II. BUILDING BLOCKS & SDN OPERATION

The SDN switch (for instance, an Openflow switch), the SDN controller , and the interfaces present on the controller for communication with forwarding devices, generally southbound interface (OpenFlow) and network applications interface(northbound interface) are the fundamental building blocks of an SDN deployment as shown in the figure.

N. Venkata Ramana Gupta been working as Assistant Professor in the Department of Computer Science and Engineering Prasad V. Potluri Siddhartha Institute of Technology(Autonomous) Vijayawada, Andhra Pradesh

Dr. M.V. Ramakrishna has been working as Professor in the Department of Computer science and Engineering, Prasad V. Potluri Siddhartha Institute of Technology(Autonomous) Vijayawada, Andhra Pradesh



Switches in an SDN are often represented as basic forwarding hardware accessible via an open interface, as the control logic and algorithms are offloaded to a controller. OpenFlow switches come in two varieties: pure (OpenFlow-only) and hybrid(OpenFlow-enabled). Pure OpenFlow switches have no legacy features or on-board control, and completely rely on a controller for forwarding decisions.

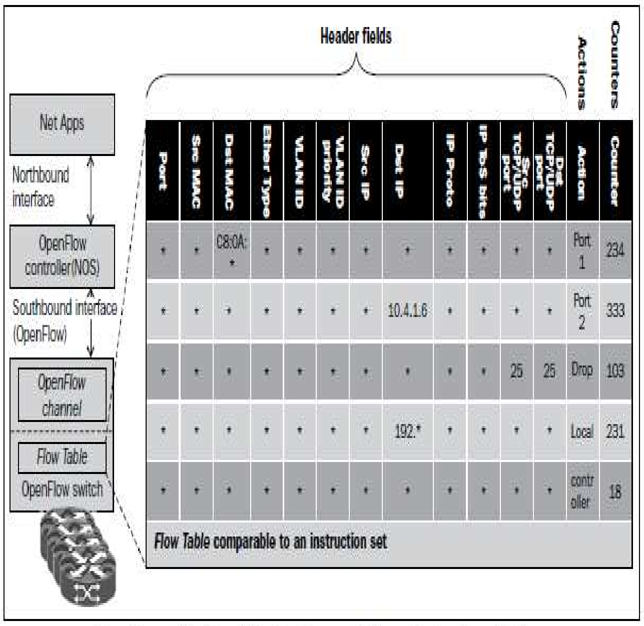
Hybrid switches support OpenFlow in addition to traditional operation and protocols. Most commercial switches available today are hybrids. An OpenFlow switch consists of a flow table, which performs packet lookup and forwarding. Each flow table in the switch holds a set of flow entries that consists of:

1. Header fields or match fields, with information found in packet header, ingress port, and metadata, used to match incoming packets.
2. Counters, used to collect statistics for the particular flow, such as number of received packets, number of bytes, and duration of the flow.
3. A set of instructions or actions to be applied after a match that dictates how to handle matching packets. For instance, the action might be to forward a packet out to a specified port.

The decoupled system in SDN (and OpenFlow) can be compared to an application program and an operating system in a computing platform. In SDN, the controller (that is, network operating system) provides a programmatic interface to the network, where applications can be written to perform control and management tasks, and offer new functionalities. A layered view of this model is illustrated in the following figure. This view assumes that the control is centralized and applications are written as if the network is a single system. While this simplifies policy enforcement and management tasks, the bindings must be closely maintained between the control and the network forwarding elements. As shown in the following figure, a controller that strives to act as a network operating system must implement at least two interfaces: a *southbound* interface (for example, OpenFlow) that allows switches to communicate with the controller and a *northbound* interface that presents a programmable API to network control and high-level policy applications/services. Header fields (match fields) are shown in the following figure.Each entry of the flow table contains a specific value, or ANY (* or wildcard , as depicted in the following figure), which matches any value.

If the switch supports subnet masks on the IP source and/or destination fields, these can more precisely specify matches.

The port field (or ingress port) numerically represents the incoming port of the switch and starts at 1. The length of this field is implementation dependent. The ingress port field is applicable to all packets.



Port	Src MAC	Dst MAC	Ether Type	VLAN ID	VLAN ID priority	Src IP	Dst IP	IP Proto	IP ToS bits	Src TCP/UDP port	Dst TCP/UDP port	Action	Counter
*	*	C8:0A:*	*	*	*	*	*	*	*	*	*	Port 1	234
*	*	*	*	*	*	*	10.4.1.6	*	*	*	*	Port 2	333
*	*	*	*	*	*	*	*	*	*	25	25	Drop	103
*	*	*	*	*	*	*	192.*	*	*	*	*	Local	231
*	*	*	*	*	*	*	*	*	*	*	*	controller	18

OpenFlow switch, Flow table, OpenFlow controller, and network applications.

The source and destination MAC (Ethernet) addresses are applicable to all packets on enabled ports of the switch and their length is 48 bits. The Ethernet type field is 16 bits wide and is applicable to all the packets on enabled ports. An OpenFlow switch must match the type in both standard Ethernet and IEEE 802.2 with a **Subnetwork Access Protocol** (**SNAP**) header and **Organizationally Unique Identifier** (**OUI**) of 0x000000. The special value of 0x05FF is used to match all the 802.3 packets without SNAP headers. VLAN ID is applicable to all packets with and Ethernet type of 0x8100. The size of this field is 12 bits (that is, 4096 VLANs). The VLAN priority (or the VLAN PCP field) is 3 bits wide and is applicable to all packets of Ethernet type 0x8100. The IP source and destination address fields are 32 bit entities and are applicable to all IP and ARP packets. These fields can be masked with a subnet mask.

The IP protocol field is applicable to all IP, IP over Ethernet, an the ARP packets. Its length is 8 bits and in case of ARP packets, only the lower 8 bits of the ARP op-code are used. The IP **ToS** (**Type of Service**) bits has a length of 6 bits and is applicable to all IP packets. It specifies an 8 bit value and places ToS in the upper 6 bits. The source and destination transport port addresses (or ICMP type/code) have a length of 16 bits and are applicable to all TCP, UDP, and ICMP packets. In case of the ICMP type/code only the lower 8 bits are considered for matching. Counters are maintained per table, per flow, per port and per queue. Counters wrap around with no overflow indicator. The required set of counters is summarized in the following figure. The phrase byte in this figure refers to an 8 bit octet. Duration refers to the time the flow has been installed in the flow table of the switch. The receive errors field includes all explicitly specified errors, including frame, overrun, and CRC errors, plus any others. Each flow entry is associated with zero or more actions that instruct the OpenFlow switch how to handle matching packets. If no forward actions are present, the packet is dropped. Action lists must be processed in the specified order.

However, there is no guaranteed packet output ordering within an individual port. For instance, two packets which are generated and destined to a single output port as part of the action processing, may be arbitrarily re-ordered. Pure OpenFlow switches only support the *Required Actions*, while hybrid OpenFlow switches may also support the **NORMAL** action. Either type of switches can also support the **FLOOD** action. The *Required Actions* are: Forward, All, Controller, Local, Table, IN_port, Drop

A flow entry with no specified action is considered as a Drop action.

- The *Optional Actions* are: Forward, Normal, Flood, Enqueue.
- associated data. This action is only applicable to IPv4 packets.

For each packet that matches a flow entry, the associated counters for that entry are updated. If the flow table look-up procedure does not result on a match, the action taken by the switch will depend on the instructions defined at the table-miss flow entry. The flow table must contain a table-miss entry in order to handle table misses. This particular entry specifies a set of actions to be performed when no match is found for an incoming packet. These actions include dropping the packet, sending the packet out on all interfaces, or forwarding the packet to the controller over the secure OpenFlow channel. Header fields used for the table lookup depend on the packet types.

For IP packets with nonzero fragment offset or more fragments bit set, the transport ports are set to zero for the lookup. Optionally, for ARP packets (Ethernet type equal to 0x0806), the lookup fields may also include the contained IP source and destination fields. Packets are matched against flow entries based on prioritization. An entry that specifies an exact match (that is no wildcards) is always the highest priority. All wildcard entries have a priority associated with them. Higher priority entries must match before lower priority ones. If multiple entries have the same priority, the switch is free to choose any ordering. Higher numbers have higher priorities. The following figure shows the packet flow in an OpenFlow switch. It is important to note that if a flow table field has a value of ANY (*, wildcard), it matches all the possible values in the header.

OpenFlow messages : The communication between the controller and switch happens using the OpenFlow protocol, where a set of defined messages can be exchanged between these entities over a secure channel. The secure channel is the interface that connects each OpenFlow switch to a controller. The **Transport Layer Security** (**TLS**) connection to the user-defined (otherwise fixed) controller is initiated by the switch on its power on. The controller's default TCP port is 6633. The switch and controller mutually authenticate by exchanging certificates signed by a site-specific private key.

Each switch must be user-configurable with one certificate for authenticating the controller (controller certificate) and the other for authenticating to the controller (switch certificate). In the case that a switch loses contact with the controller, as a result of an echo request timeout, TLS session timeout, or other disconnection, it should attempt to contact one or more backup controllers. If some number of attempts to contact a controller (zero or more) fail, the switch must enter *emergency mode* and immediately reset the current TCP connection. Then the matching process is dictated by the emergency flow table entries (marked with the emergency bit set). Emergency flow modify messages must have timeout value set to zero. Otherwise, the switch must refuse the addition and respond with an error message. All normal entries are deleted when entering emergency mode. Upon connecting to a controller again, the emergency flow entries remain. The controller then has the option of deleting all the flow entries, if desired. The controller configures and manages the switch, receives events from the switch, and sends packets out to the switch through this interface. Using the OpenFlow protocol, a remote controller can add, update, or delete flow entries from the switch's flow table. That can happen reactively (in response to a packet arrival) or proactively.

The OpenFlow protocol can be viewed as one possible implementation of controller switch interactions (southbound interface), as it defines the communication between the switching hardware and a network controller. The OpenFlow protocol defines three message types, each with multiple subtypes:

- Controller-to-switch
- Symmetric
- Asynchronous

At a conceptual level, the behavior and operation of a Software Defined Network is straightforward. In Figure 2.1 we provide a graphical depiction of the operation of the basic components of SDN: the SDN devices, the controller, and the applications. The SDN controller is responsible for abstracting the network of SDN devices it controls and presenting an abstraction of these network resources to the SDN applications running above. The controller allows the SDN application to define flows on devices and to help the application respond to packets that are forwarded to the controller by the SDN devices. In Figure 2.1 we see on the right side of the controller that it maintains a view of the entire network that it controls. Since one controller can control large number of network devices, these calculations are normally performed on a high-performance machine with an order-of-magnitude performance advantage over the CPU and memory capacity than is typically afforded to the network devices themselves. SDN applications are built on top of the controller.

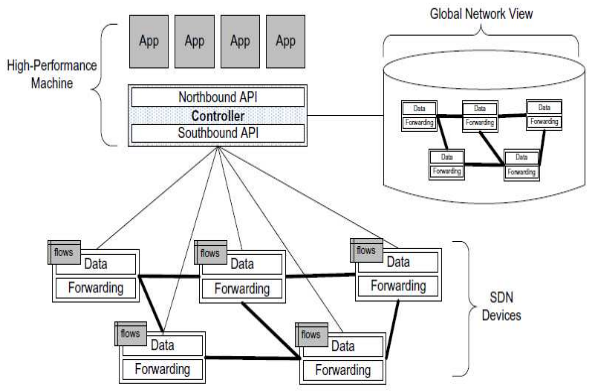


Fig 2.1 SDN operation overview.

Since SDN applications are really part of network layers two and three, this concept is orthogonal to that of applications in the tight hierarchy of OSI protocol layers. The SDN application interfaces with the controller, using it to set proactive flows on the devices and to receive packets that have been forwarded to the controller. Proactive flows are established by the application; typically the application will set these flows when the application starts up, and the flows will persist until some configuration change is made. This kind of proactive flow is known as a static flow. Another kind of proactive flow is where the controller decides to modify a flow based on the traffic load currently being driven through a network device.

In addition to flows defined proactively by the application, some flows are defined in response to a packet forwarded to the controller. Upon receipt of incoming packets that have been forwarded to the controller, the SDN application will instruct the controller as to how to respond to the packet and, if appropriate, will establish new flows on the device in order to allow that device to respond locally the next time it sees a packet belonging to that flow. Such flows are called reactive flows. In this way, it is now possible to write software applications that implement forwarding, routing, overlay, multipath, and access control functions, among others. There are also reactive flows that are defined or modified as a result of stimuli from sources other than packets from the controller. For example, the controller can insert flows reactively in response to other data sources such as intrusion detection systems (IDS) or the NetFlow traffic analyzer. Figure 2.2 depicts the OpenFlow protocol as the means of communication between the controller and the device.

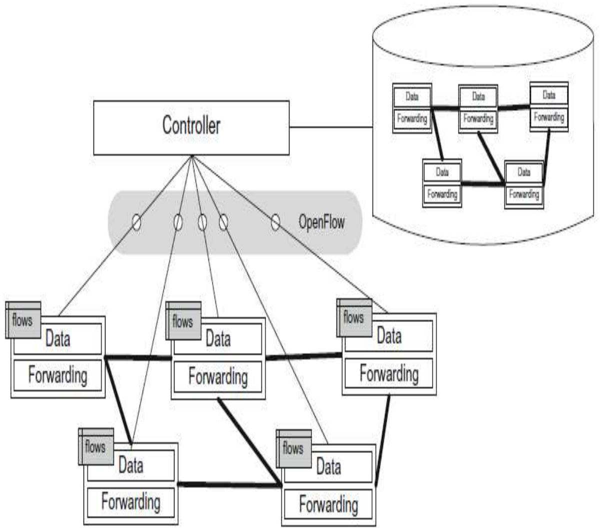


FIG 2.2 Controller-to-device communication.

2.1 SDN Devices

An SDN device is composed of an API for communication with the controller, an abstraction layer, and a packet-processing function. In the case of a virtual switch, this packet-processing function is packet processing software, as show in Figure 2.3. In the case of a physical switch, the packet-processing function is embodied in the hardware for packet-processing logic, as shown in Figure 2.4. The abstraction layer embodies one or more flow tables. The packet-processing logic consists of the mechanisms to take actions based on the results of evaluating incoming packets and finding the highest-priority match. When a match is found, the incoming packet is processed locally unless it is explicitly forwarded to the controller. When no match is found, the packet may be copied to the controller for further processing. This process is also referred to as the controller *consuming* the packet. In the case of a hardware switch, these mechanisms are implemented by the specialized hardware. In the case of a software switch, these same functions are mirrored by software. Since the case of the software switch is somewhat simpler than the hardware switch. Over time the actual packet-forwarding logic migrated into hardware for switches that needed to process packets arriving at ever-increasing line rates. More recently, a role has reemerged in the data center for the pure software switch. Such a switch is implemented as a software application usually running in conjunction with a hypervisor in a data center rack. Like a VM, the virtual switch can be instantiated or moved under software control. It normally serves as a virtual switch and works collectively with a set of other such virtual switches to constitute a virtual network.

2.2 SDN Controller

The controller maintains a view of the entire network, implements policy decisions, controls all the SDN devices that comprise the network infrastructure, and provides a northbound API for applications.

.

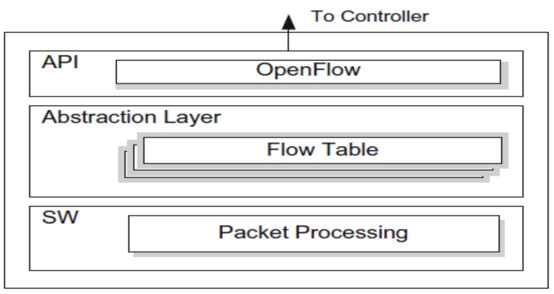


FIGURE 2.3 SDN software switch anatomy.

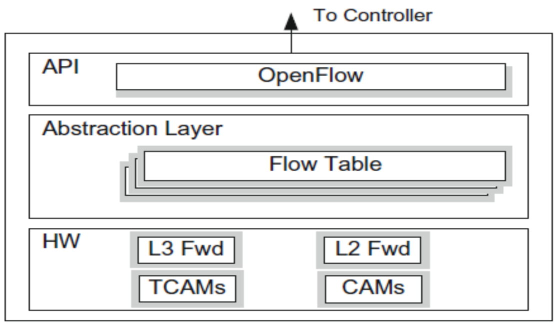


FIG 2.4 SDN hardware switch anatomy.

When we have said that the controller implements policy decisions regarding routing, forwarding, redirecting, load balancing, and the like, these statements referred to both the controller and the applications that make use of that controller. Controllers often come with their own set of common application modules, such as a learning switch, a router, a basic firewall, and a simple load balancer.

Figure 2.5 exposes the anatomy of an SDN controller. The figure depicts the modules that provide the controller's core functionality, both a northbound and a southbound API, and a few sample applications that might use the controller. As we described earlier, the southbound API is used to interface with the SDN devices. This API is OpenFlow in the case of Open SDN or some proprietary alternative in other SDN solutions.

It is worth noting that in some product offerings, both OpenFlow and alternatives coexist on the same controller. Early work on the southbound API has resulted in more maturity of that interface with respect to its definition and standardization. OpenFlow itself is the best example of this maturity, but de facto standards
Such as Cisco CLI and SNMP also represent standardization in the southbound-facing interface.

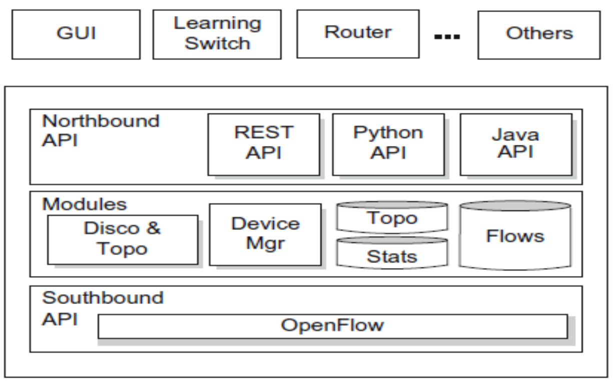


FIGURE 2.5 SDN controller anatomy.

OpenFlow's companion protocol, OF-Config, and Nicira's *Open vSwitch Database Management Protocol* (OVSDB) are both open protocols for the southbound interface, though these are limited to configuration roles. Unfortunately, there is currently no northbound counterpart to the southbound OpenFlow standard or even the de facto legacy standards. This lack of a standard for the controller-to-application interface is considered a current deficiency in SDN, and some bodies are developing proposals to standardize it.The absence of a standard notwithstanding, northbound interfaces have been implemented in a number of disparate forms. For example, the Floodlight controller includes a Java API and a *Representational State Transfer* (RESTful) API. The OpenDaylight controller provides a RESTful API for applications running on separate machines. The northbound API represents an outstanding opportunity for innovation and collaboration among vendors and the open source community.

2.3 SDN Controller Core Modules

The controller abstracts the details of the SDN controller-to-device protocol Figure 2.5 shows the API below the controller, which is OpenFlow in Open SDN, and the interface provided for applications. Every controller provides core functionality between these raw interfaces. Core features in the controller include : End-user device discovery, Network device discovery, Network device topology management, Flow management. The core functions of the controller are device and topology discovery and tracking, flow management, device management, and statistics tracking. These are all implemented by a set of modules internal to the controller. As shown in Figure 2.5, these modules need to maintain local databases containing the current topology and statistics. The controller tracks the topology by learning of the existence of switches (SDN devices) and end-user devices and tracking the connectivity between them. It maintains a *flow cache* that mirrors the flow tables on the various switches it controls. The controller locally maintains per-flow statistics that it has gathered from its switches.

2.3.1 SDN Controller Interfaces

For SDN applications, a key function provided by the SDN controller is the API for accessing the network.Figure 2.6 takes a closer look at how the controller interfaces with applications. The controller informs the application of *events* that occur in the network. Events are communicated from the controller to the application. Events may pertain to an individual packet that has been received by the controller or some state change in the network topology, such as a link going down. The applications may invoke methods independently, without the stimulus of an event from the controller, Such inputs are represented by the "Other Context" box in Figure 2.6.

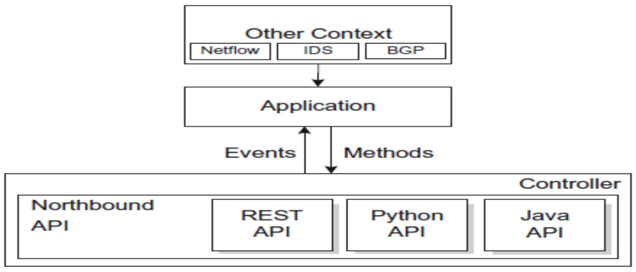


FIGURE 2.6 SDN controller northbound API.

Active networking is focused on improving the data plane functionality of the network. Certain variants specifically the capsule model, mandated that new data plane functionality be installed on network devices through code carried through data packets. Projects such as Planet Lab [7] feature the separation of traffic to different execution environments on the basis of packet headers. SDN efforts differed from active networking by focusing on problems of immediate import to network administrators.
Moreover, To increase the programmability of the control plane and together with commercial successes such as Nicira's Network Virtualization Platform [8], these efforts have led to significant industry attention being given to SDN. The application of these concepts in the context of wireless networks poses many challenges. Consider a WLAN. Each access point (AP) has to make decisions on its modulation format, power, and channel based on SINR estimates. In this case, a fully centralized network architecture imposes strict upper bounds on the latency between the controller and the AP. Thus, in wireless networks, it is not always clear as to which point in the design space one should operate.

III. WLAN

Much of the research in wireless SDN so far has focused on IEEE 802.11 networks. However, with the ubiquity of high-definition video content, there is considerable interest on the part of cellular companies in offloading data traffic to WiFi networks whenever possible. Thus, SDN efforts in WiFi and cellular are not entirely isolated from each other. There has been some work also on IEEE 802.16 (WiMax) networks, including a demonstration of the handover between WiFi and WiMAX on Openflow [15]. There is an ongoing effort to apply SDN in wireless backhaul network focusing on IEEE 802.16 [IEEE 2013], but it is recent and only beginning to be studied.

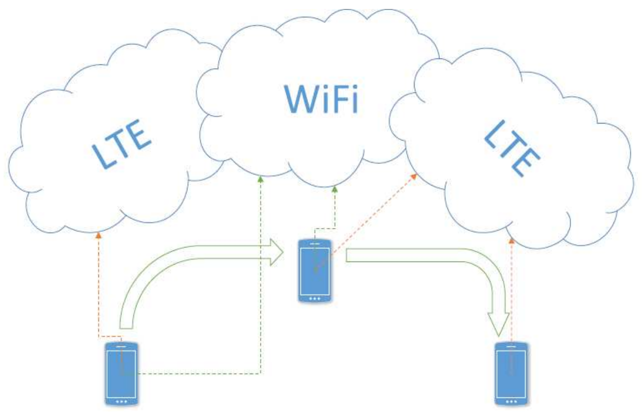


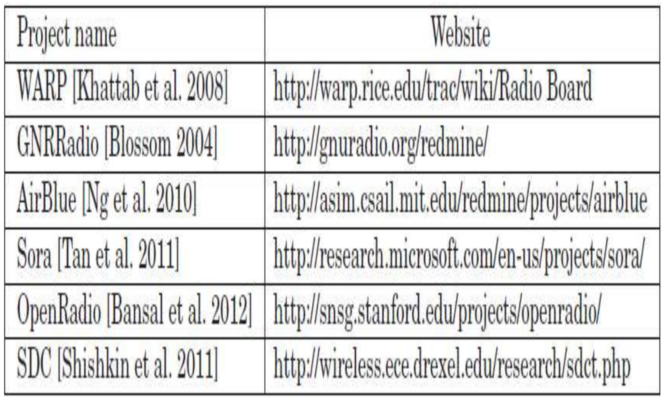
Fig. 3.1 Mobility across technologies.

3.1. OpenFlow Wireless

In campus Wi-Fi networks, network administration is mostly centralized. Commercial products from various companies [9]-[12] stand as a testimony to this fact. OpenFlow Wireless [13]-[15] aims to be an open alternative to the proprietary solutions currently being offered. Envisage a future where the end user is free from worrying about the details as to which wireless network she is getting service from. It offers a compelling vision, where a user roams freely between cellular and Wi-Fi networks taking advantage of seamless handovers (see Figure 3.1 for an illustration). Network operators benefit as well. The ubiquity of high-definition video content places enormous pressure on the cellular data infrastructure, which would gladly welcome the offloading of traffic to Wi-Fi. The realization of this vision would require decoupling of service providers and network owners. This decoupling already exists in the present day. H20 Wireless, Tesco Mobile and LycaMobile are mobile virtual network operators (MVNOs). An open research problem in this domain is the design of a mobility manager capable of servicing each customer.
An important feature of SDN-enabled WLANs is virtualization. The ability to slice the network, based on users, subnets, or traffic, allows many benefits. In OpenFlow-enabled networks, this virtualization is achieved using FlowVisor [13]-[15] to delegate the control of different slices to different controllers. The FlowVisor is essentially a proxy that forwards OpenFlow messages from different slices to the appropriate controllers. But it does not provide facilities for configuration of the network. Independent configuration per slice is achieved using a SNMPVisor. Home networks form a specific type of WLAN. In a home network, one of the primary objectives is to enable seamless sharing of data between various devices in the home while restricting access to other devices. One of the primary bottlenecks in realizing this goal is the need for network configuration by the end user. Another approach is to virtualize the AP by providing each user with their own personalized virtual access point [16]. However, it is not immediately clear how much of network management and control can be centralized in view of the fact that wireless channel conditions are variable the choice of taking a control decision centrally must be made after careful evaluation of the network parameters that the control decisions depend on.

3.2. Software-Defined Radios

The developments stated so far have focused on the network layer and above. OpenRadio[17] aims to provide programmability of the PHY and MAC layers by attempting to define a software abstraction layer that hides the hardware details from the programmer. Wireless protocols are decomposed to *processing* blocks and *decision* logic. Processing blocks operate on and manipulate the analog waveform, while the decision logic charts a path traversing different processing blocks at various times. Key advantages claimed by OpenRadio include modular programmability and the ability for real-time implementation on commodity digital signal processors.OpenRadio is not unique in offering programmable PHY and MAC layers. Other efforts in this direction include WARP [18]. See Table II for a listing of active projects. CloudMAC [19], CloudMAC uses virtual access points (VAPs).



Project name	Website
WARP [Khattab et al. 2008]	http://warp.rice.edu/trac/wiki/Radio Board
GNRRadio [Blossom 2004]	http://gnuradio.org/redmine/
AirBlue [Ng et al. 2010]	http://asim.csail.mit.edu/redmine/projects/airblue
Sora [Tan et al. 2011]	http://research.microsoft.com/en-us/projects/sora/
OpenRadio [Bansal et al. 2012]	http://snsg.stanford.edu/projects/openradio/
SDC [Shishkin et al. 2011]	http://wireless.ece.drexel.edu/research/sdct.php

Table II. Software-Defined Radio (SDR) Projects

Each physical access point, now termed as *wireless termination point (WTP)*, is "dumb" in that they do not generate their own MAC frames. WTPs are used only to send and receive MAC frames from an OpenFlow switch that they are connected to, in addition to the end user. The OpenFlow switch in turn is connected to a controller and to virtual machines that manage the VAPs. Thus, all MAC frames generated by the user will have to travel to the VAPs. This decreases delay performance considerably. Indeed, their implementation shows a three fold increase in round trip time (RTT). Ultimately, these efforts point to a future where we are able to modify the entire wireless stack to suit our requirements. There is a trade-off in delay as one moves more and more intelligence to the center of the network, which might not be acceptable as we move lower in the stack. The optimal operating point depends on the specific network requirements.

3.3. Odin

Odin [20] is an SDN framework that proposes to simplify the implementation of high-level enterprise WLAN services, such as authentication, authorization and accounting (AAA), by introducing light virtual access points (LVAPs). This approach is similar to the VAPs used in CloudMAC. Usually, the AP to which a user is connected to may change in accordance with local decisions made by the user. Thus, the last hop connecting the user to the WLAN infrastructure is not stable. Using the abstraction of LVAP, Odin gives programmers a virtual unchanging link connecting users to APs. To achieve this, each user is assigned a unique BSSID, giving the illusion that it has its own AP. This virtual user-specific AP is called an LVAP. Each physical AP will host multiple LVAPs, one corresponding to each client.

Architecturally, Odin consists of a single master, which is an application running on top of an OpenFlow controller, and multiple agents running on APs. The control channel is a TCP connection between the agent and the master. It is easier to implement mobility managers because the BSSID at the user does not change during a handover. Aeroflux builds on the aforementioned framework and proposes the division of the control plane into two tiers. The lower tier, managed by near-sighted controllers (NSCs), is responsible for events that do not require global state data or those events that occur very frequently. Events such as network monitoring or load balancing, which are global in nature, are handled by a global controller (GC).

IV. CELLULAR

There have been two main attempts at utilizing SDN concepts to improve cellular networks. SoftRAN [21] focuses on improving the design of the radio access network (RAN). RAN is the part of the cellular network architecture that is burdened with providing wide area connectivity to mobile devices. SoftRAN argues for improved management of radio resources to achieve this objective. CellSDN and SoftCell[20] constitute the major efforts toward using SDN concepts to improve the core cellular network. The following sections discuss the design principles of these works.

4.1. Core Network

Current cellular networks consist of base stations (BS) connected to server gateways (S-GW), which in turn are connected to packet gateways (P-GW). The S-GW acts as a mobility manager, maintaining state information for each user to ensure uninterrupted connectivity while the user travels across base stations. The P-GW is responsible for policy enforcement and accounting Traffic between users on the same network is needlessly routed through the P-GW, which, in addition to making the device more expensive, is slower. CellSDN [21] is to distribute the processing load over the switches and base stations while retaining centralized control over them using a controller. Implementing the architecture sketched earlier[21], however, would require extensions to both switches and base stations. Each of these devices should now run cell agents that enable remote control of resources. Moreover, Latency considerations would dictate which functionality can be handed over to the controller. Overall, the goal is to have a network operating system running over the cellular infrastructure, there enabling various network management requirements to be written as application modules. SoftCell [20] extends CellSDN by designing an architecture that supports fine-grained policies for mobile devices while still allowing the use of commodity switches and servers. This is achieved by employing fine-grained packet classification at the access switches in the base stations. The gateway switches perform only basic packet forwarding with the help of policy identifiers embedded directly in the IP packet headers. Support for such policies face the challenge of having to work with small switch tables. This obstacle is circumvented by aggregating traffic based on policy, location, and user equipment ID.

4.2. Radio Access Network

One way of dealing with the increasing mobile data traffic and the limited availability of spectrum resources is to bring the base station closer to the mobile client. This approach would mandate a smaller cell size, leading to a denser deployment. [23] argue that a denser deployment calls for an increase in coordination between neighboring base stations to improve interference management and load balancing. To this end, SoftRAN abstracts out all radio resources of a geographical area in a three-dimensional grid of base station index, time, and frequency slots. A *geographical area* is defined as a macro cell. The physical base stations are then viewed as individual radio elements that are controlled by a logically centralized controller. The radio elements and the controller communicate with each other using a suitable API. In view of the varying channel conditions, the controller is effectively working with a possibly outdated view of the network state. Consequently, not all the control plane functionality is given over to the controller. All control decisions that affect the neighboring radio elements are taken at the controller. The control decisions that depend on parameters that are known to vary frequently are taken locally at the radio elements. For example, handovers and power allocation fall under the purview of the controller, while the downlink resource block allocation is handled by the radio elements. The network state is maintained at the controller in the form of a database, known as RAN information base (RIB). The RIB maintains information required by the control such as frequency, in addition to the power allocation at the radio elements.

V. MULTIHOP WIRELESS NETWORKS

There have been multiple research efforts in multi hop networks that are in the spirit of SDN while differing in terminology. The following subsections provide an overview of these works.

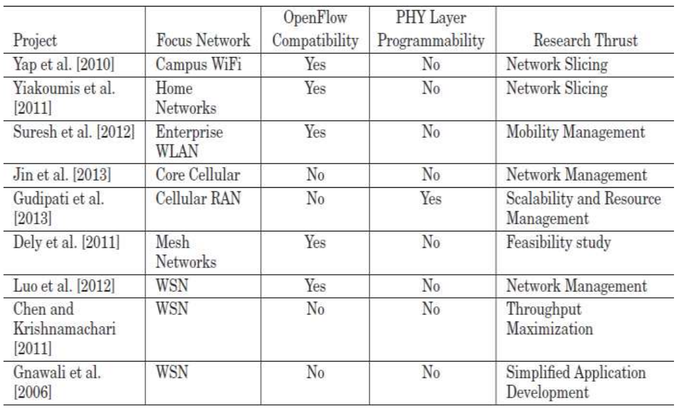
5.1. Mesh Networks

Feasibility study of using OpenFlow in wireless mesh networks was provided [24]. One of the primary challenges that mesh networks have to face, is frequent changes in topology. Nodes may be mobile, and nodes arrive and leave more frequently. The proposed network architecture is as follows. All nodes are OpenFlow-enabled mesh routers complying with the IEEE 802.11 standard. A single-channel single radio setup is assumed. Each wireless interface is used for both control and data traffic. Separation between the two is achieved using different SSIDs. A NOX controller is used for managing the forwarding tables of individual nodes and handling node mobility. Experiments conducted on the KAUMesh testbed show that on increasing the complexity of a rule, throughput suffers. A simple rule matches the port number of an incoming packet, whereas a complex rule consisted of matching the MAC and IP addresses for each packet. This was shown to lead to a throughput degradation of up to 15%. The control traffic was shown to be small, albeit linearly increasing with the number of rules to be installed as might be expected. It is not quite clear at what point the control traffic begins to pose a significant overhead.

5.2. Wireless Sensor Networks

A wireless sensor network (WSN) is another multihop wireless scenario where an SDN approach has been claimed to be beneficial [25]. WSNs are subject to a unique set of constraints. The nodes are typically low powered, small and are often deployed without any particular attention to the topology they form. In any case, a change in topology post deployment is quite common due to node failures or physical displacement of nodes. Nodes may also contain application specific components. [25]propose a SDN-based architecture that proposes to centralize network management and enable running different applications on a single WSN. This proposed architecture for WSNs is conceptually quite similar to the various wireless SDN efforts described previously. The control plane is decoupled from the data plane that runs on the sensor nodes. A centralized controller uses a customized version of OpenFlow to interact with the nodes. The nodes are modified to enable this centralized control of their flow tables. Various applications may be run on top of the controller. The primary contribution of the article lies in the specifics of extending OpenFlow to support WSN specific use cases, such as flow creation using sensor attributes.As control and data traffic share the same network, This increases the average latency of the control channel. Another problem that remains unaddressed is the reduced reliability of the control channel. Compared to the data network, the control network is typically subject to better standards of reliability. Particularly in WSNs, as node and link failures are much more common, sharing the network remains dubious. Other difficulties that need to be worked around include the need to minimize control overhead as communication is inherently costly.

There have been other efforts in centralizing control in WSNs that are in the same spirit as SDN, but the terminology pertaining to SDN has not been used. MCC [26] is multichannel time-scheduled protocol aimed toward real-time data collection in WSNs. MCC design features centralized channel allocation and time scheduling to combat co-channel interference, and routes from sources to the sink are centrally computed using a capacitated minimal spanning (CMS) tree heuristic. Thus, there is a centralized control plane that obtains information about the network topology to configure the transmission and routing decisions for all nodes to use for the distributed data forwarding plane. This makes the protocol design of MCC architecturally close to SDN. Tenet is another architecture design that decouples control from the sensor motes for a tiered architecture for WSNs. The lower tier consists of resource constrained motes, whereas the upper tier contains fewer but more capable nodes called *masters*. In the words of the authors, the Tenet design principle may be stated as: Multi-node data fusion functionality and multi-node application logic should be implemented only in the master tier. The cost and complexity of implementing this functionality in a fully distributed fashion on motes outweighs the performance benefits of doing so. In fact, Tenet represents an extreme design point that severely constraints even the type of communication allowed on the motes. Thus, the motes only provide a limited set of generic functions and applications, written in software in the master tier, combine this functionality to achieve network objectives. These properties are characteristic of SDN.



Project	Focus Network	OpenFlow Compatibility	PHY Layer Programmability	Research Thrust
Yap et al. [2010]	Campus WiFi	Yes	No	Network Slicing
Yiakoumis et al. [2011]	Home Networks	Yes	No	Network Slicing
Suresh et al. [2012]	Enterprise WLAN	Yes	No	Mobility Management
Jin et al. [2013]	Core Cellular	No	No	Network Management
Gudipati et al. [2013]	Cellular RAN	No	Yes	Scalability and Resource Management
Dely et al. [2011]	Mesh Networks	Yes	No	Feasibility study
Luo et al. [2012]	WSN	Yes	No	Network Management
Chen and Krishnamachari [2011]	WSN	No	No	Throughput Maximization
Gnawali et al. [2006]	WSN	No	No	Simplified Application Development

Table I. Focus Areas of Selected Papers

6. CONCLUSIONS

SDN opens many axes in the network design space. See Table I for a comparison of selected projects. The most important of these is the possibility of centralizing much of the intelligence in a network. For wireless networks, this move comes with obvious advantages. For example, the hidden terminal problem ceases to be an issue if transmission decisions are made centrally, based on a view of the entire network. Virtualization of network resources is now possible, enabling sharing of resources across vendors, thereby reducing cost. However, the increased channel variability and the latency sensitive nature of key network parameters, such as power allocation in a channel, increases the complexity of the design space. In fact, while research thus far has focused on enabling choice in different design axes, the larger question of how to make use of this new-found freedom optimally for different scenarios remains unanswered.

REFERENCES

[1] Martin Casado, Michael J. Freedman, Justin Pettit, Jianying Luo, Nick McKeown, and Scott Shenker. 2007.
Ethane: Taking control of the enterprise. *SIGCOMM Comput. Commun. Rev.* 37, 4 (Aug. 2007),12.DOI:http://dx.doi.org/10.1145/1282427.1282382.

[2] Matthew Caesar, Donald Caldwell, Nick Feamster, Jennifer Rexford, Aman Shaikh, and Jacobus van derMerwe. 2005.
Design and implementation of a routing control platform. In *Proceedings of the 2ndConference on Symposium on Networked Systems Design & Implementation - Volume 2 (NSDI'05)*.USENIX Association, Berkeley, CA, 1528.

[3] Albert Greenberg, Gisli Hjalmtysson, David A.Maltz, AndyMyers, Jennifer Rexford, Geoffrey Xie, Hong Yan, Jibin Zhan, and Hui Zhang. 2005. A clean slate 4D approach to network control and management. *SIGCOMM Comput. Commun. Rev.* 35, 5 (Oct. 2005), 41–54.DOI:http://dx.doi.org/10.1145/1096536.1096541

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

[5] TR10. 2009. Software Defined Networking. Retrieved from http://www.technologyreview.com/article/412194/tr10-software-defined-networking/.

[6] Nick Feamster, Jennifer Rexford, and Ellen Zegura. 2013. The road to SDN. *Queue* 11, 12 (Dec. 2013), 20 pages.

[7] Brent Chun, David Culler, Timothy Roscoe, Andy Bavier, Larry Peterson, Mike Wawrzoniak, and Mic Bowman. 2003. PlanetLab: An overlay testbed for broad-coverage services. *SIGCOMM Comput. Commun.Rev.* 33, 3 (July 2003), 3–12.

[8] Nicira. 2012. It'sTime to Virtualize the Network. Retrieved Jan 16,2014 from http://www.netfos.com.tw/PDF/Nicira/ItisTimeToVirtualizetheNetworkWhite Paper.pdf.

[9] Aruba. 2013. Aruba Networks. Retrieved from http://www.arubanetworks.com.
[10] Meraki. 2011. Meraki Whitepaper: Meraki Hosted Architecture. Retrieved Jan 21, 2014 fromhttps://meraki.cisco.com/lib/pdf/meraki_whitepaper_architecture.pdf.
[11] Meraki. 2013. CloudManagedWireless by Meraki.Retrievedfromhttp://www.meraki.com/products/wireless.
[12] Ericsson. 2012. The Cloud and Software Defined Networking. Retrieved Jan 16, 2014 fromhttp://www.ericsson.com/res/investors/docs/2012/ericsson-cloud-and-sdn.pdf.
[13] Rob Sherwood, Michael Chan, Adam Covington, Glen Gibb, Mario Flajslik, Nikhil Handigol, Te-Yuan Huang, Peyman Kazemian, Masayoshi Kobayashi, Jad Naous, Srinivasan Seetharaman, David Underhill, Tatsuya Yabe, Kok-Kiong Yap, Yiannis Yiakoumis, Hongyi Zeng, Guido Appenzeller, Ramesh Johari,Nick McKeown, and Guru Parulkar. 2010. Carving research slices out of your production networks with OpenFlow. *SIGCOMM Comput. Commun. Rev.* 40, 1 (Jan. 2010), 129–130..
[14] Kok-Kiong Yap, Masayoshi Kobayashi, Rob Sherwood, Te-Yuan Huang, Michael Chan, Nikhil Handigol, and Nick McKeown. 2010. OpenRoads: Empowering research in mobile networks. *SIGCOMM Comput. Commun. Rev.* 40, 1 (Jan. 2010), 125–126. Kok-Kiong Yap, Masayoshi Kobayashi, David Underhill, Srinivasan Seetharaman, Peyman Kazemian, andNick McKeown. 2009. The stanford openroads deployment. In *Proceedings of the 4th ACM International Workshop on Experimental Evaluation and Characterization (WINTECH'09)*. ACM, New York, NY, 59– 66.
[15] Kok-Kiong Yap, Rob Sherwood, Masayoshi Kobayashi, Te-Yuan Huang, Michael Chan, Nikhil Handigol,Nick McKeown, and Guru Parulkar. 2010. Blueprint for introducing innovation into wireless mobile networks. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Virtualized Infrastructure Systems and Architectures (VISA'10)*. ACM, New York,NY,25 -32..
[16] Yiannis Yiakoumis, Manu Bansal, Sachin Katti, and Nick McKeown. 2014. SDN for Dense WiFi Networks.USENIX Association, Santa Clara, CA.
[17] Manu Bansal, Jeffrey Mehlman, Sachin Katti, and Philip Levis. 2012. OpenRadio: A programmable wireless dataplane. In *Proceedings of the 1st Workshop on Hot topics in Software Defined Networks (HotSDN'12)*. ACM, New York, NY, 109–114.
[18] Ahmed Khattab, Joseph Camp, Chris Hunter, Patrick Murphy, Ashutosh Sabharwal, and Edward W.Knightly. 2008. WARP: A flexible platform for clean-slate wireless medium access protocol design. *SIGMOBILE Mob. Comput. Commun. Rev.* 12, 1 (Jan. 2008), 56–58.
[19] P. Dely, J. Vestin, A. Kassler, N. Bayer, H. Einsiedler, and C. Peylo. 2012. CloudMAC - An openflow based architecture for 802.11 MAC layer processing in the cloud. In *Proceedings of the 2012 IEEE GlobecomWorkshops (GC Wkshps'12)*.186–191.
[20] Xin Jin, Li Erran Li, Laurent Vanbever, and Jennifer Rexford. 2013. SoftCell: Scalable and flexible cellular core network architecture. In *Proceedings of the 9th ACM Conference on Emerging Networking Experiments and Technologies (CoNEXT'13)*. ACM, New York, NY,163–174.
[21] L. E. Li, Z. M. Mao, and J. Rexford. 2012. Toward software-defined cellular networks. In *Proceedings ofthe 2012 Eurpean Workshop on Software Defined Networking (EWSDN'12)*. 7–12.DOI:http://dx.doi.org/10.1109/EWSDN.2012.28.
[22] Lalith Suresh, Julius Schulz-Zander, Ruben Merz, Anja Feldmann, and Teresa Vazao. 2012. Towards programmable enterprise WLANS with Odin. In *Proceedings of the 1st Workshop on Hot Topics in Software Defined Networks (HotSDN'12)*. ACM, New York, NY, 115–120.
[23] Aditya Gudipati, Daniel Perry, Li Erran Li, and Sachin Katti. 2013. SoftRAN: Software defined radio access network. In *Proceedings of the 2nd ACM SIGCOMM Workshop on Hot Topics in Software Defined Networking (HotSDN'13)*. ACM, New York, NY, 25–30..
[24] P. Dely, A. Kassler, and N. Bayer. 2011. OpenFlow for wireless mesh networks. In *Proceedingsof 20th International Conference on Computer Communications and Networks (ICCCN'11)*.1-6.
[25] Tie Luo, Hwee-Pink Tan, and T. Q. S. Quek. 2012. Sensor OpenFlow: Enabling software-definedwirelesssensornetworks.*Communications Letters,IEEE* 16, 11 (2012), 1896–1899.
[26] Ying Chen and Bhaskar Krishnamachari. 2011. *MCC: A High-Throughput Multi-Channel Data Collection Protocol for Wireless Sensor Networks*. Technical Report, USC Computer Engineering, Los Angeles, CA.

N. Venkata Ramana Gupta been working as Assistant Professor in the Department of Computer Science and Engineering Prasad V. Potluri Siddhartha Institute of Technology(Autonomous) Vijayawada, Andhra Pradesh and is affiliated to JNTU-K, Kakinada,. He obtained M.Tech (Computer Science and Technology with specialization in computer Networks) from Andhra University College of Engineering, Visakhapatnam, AP. He had more than 15 Years of teaching experience. He had published 3 research papers in various International Journals. His areas of interest are Computer Networks, Distributed Systems. He is member of ACM and ISTE.,Ph.No:9849578292

Dr. M.V. Ramakrishna has been working as Professor in the Department of Computer science and Engineering, Prasad V. Potluri Siddhartha Institute of Technology(Autonomous) Vijayawada, Andhra Pradesh and is affiliated to JNTU-K, Kakinada,. He obtained B.Tech, M.Tech and Ph.D. from NIT- Suratkal, Jawaharlal Nehru Technological University, Hyderabad and Acharya Nagarjuna University, Guntur respectively. He has 23 Years of Teaching and 4 Years of Industry Experience. He had published 7 research papers in various International Journals and conferences. His areas of research include Computer Networks and Distributed Systems. He is a member of ACM and IEEE Computer Society. Ph.No: 9440672590