Internet



of

Things

Contents

1. Software Defined Networking 203
   1. [Introduction 203](#_TOC_250015)
   2. [Limitations of Current Network 203](#_TOC_250014)
   3. [Origin of SDN 205](#_TOC_250013)
   4. [SDN Architecture 206](#_TOC_250012)
   5. [Rule Placement 207](#_TOC_250011)
   6. [OpenFlow Protocol 208](#_TOC_250010)
   7. [Controller Placement 209](#_TOC_250009)
   8. [Security in SDN 211](#_TOC_250008)
   9. [Integrating SDN in IoT 212](#_TOC_250007)
   10. [Software Defined WSN 214](#_TOC_250006)
   11. [SDN for Mobile Networking 215](#_TOC_250005)
   12. [Rule Placement at Access Devices 218](#_TOC_250004)
       1. [ODIN 218](#_TOC_250003)
       2. [Ubi-Flow 220](#_TOC_250002)
       3. [Mobi-Flow 221](#_TOC_250001)
   13. [Conclusion 222](#_TOC_250000)

**9**

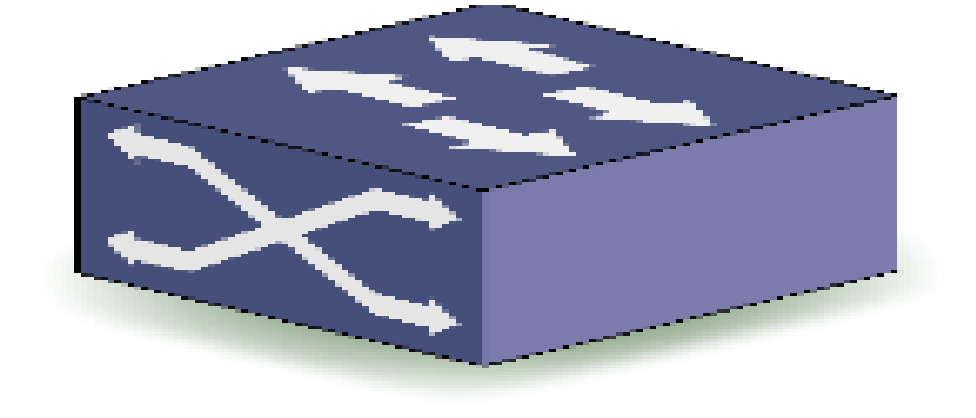
**SOFTWARE DEFINED NETWORKING**

##### 9.1. Introduction

Software Defined Networking (SDN) is for restructuring the current network infrastructure. SDN is an umbrella term encompassing several kinds of network technology aimed at making the network as agile and flexible as the virtualized server and storage infrastructure of the modern data center. The goal of SDN is to allow network engineers and administrators to respond quickly to changing business requirements. In an SDN, a network administrator can shape traffic from a centralized control console without having to touch individual switches, and can deliver services to wherever they are needed in the network, without regard to what specific devices a server or other hardware components are connected to. The key technologies for SDN implementation are functional separation, network virtualization and automation through programmability.

##### 9.2. Limitations of Current Network

Consider the scenario of current day networks. For this we are taking the example of a wired network with L3 switches. Figure 9.1 shows the network topology of current network.



User I

**Fig.9.1:** Network Topology of Current Network

User II

The data needs to be send from user 1 to user 2 and vice versa as shown in Figure 9.1. The data can be sent through several routes. The packet traverses through multiple L3 switches

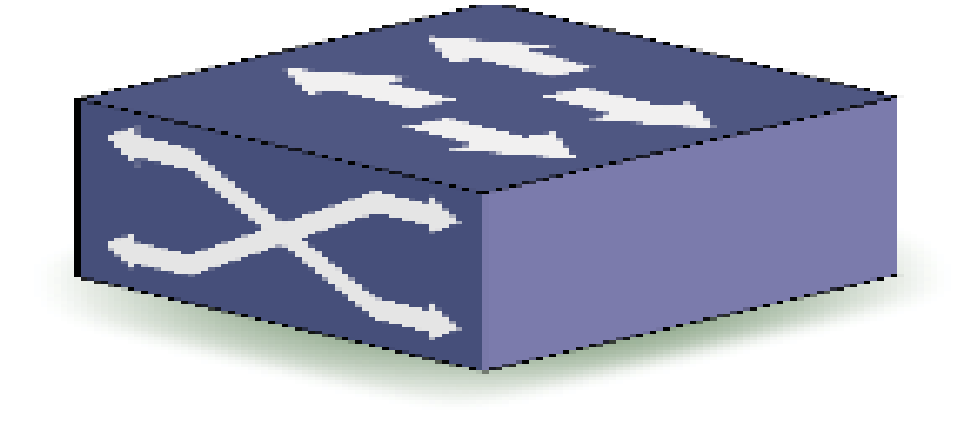
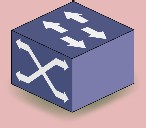
**204** | Internet of Things

before it reaches its destination. The protocol used for switching is the Open Shortest Path First (OSPF) protocol. Open Shortest Path First (OSPF) is a routing protocol for Internet Protocol (IP) networks. Each router knows to which router the packet needs to be sent based on the routing table it has. Each router doesn’t have a global view of the network as a whole.

Consider the situation where one of the routers or switches has been attacked as shown in Figure 9.2. The attacked switch will be down for one reason or another.

User I

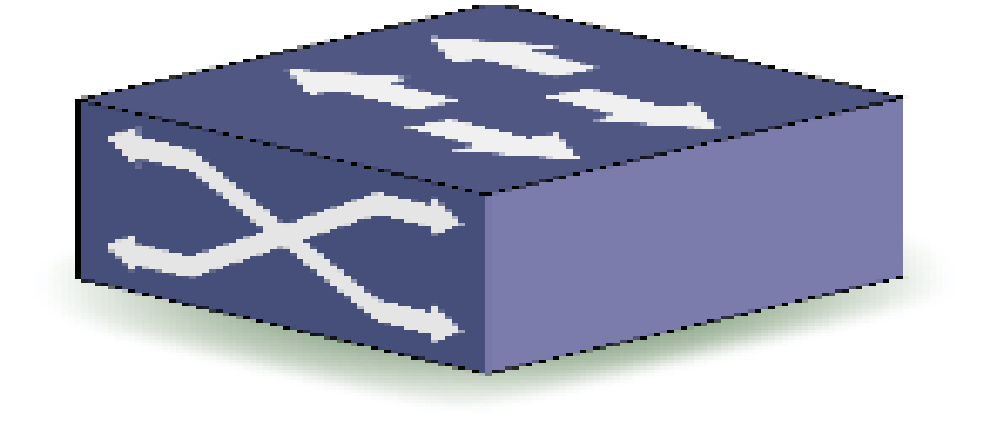
the switch has been attacked!



**Fig.9.2:** One of the Switches is Attacked

User II

In a traditional network, when a router is down, another route needs to be identified to transfer the packets. Since there is no centralized control over the routers this becomes a difficult task. So in a traditional network, when a switch is down, it can affect other switches also. Each router in a network has three layers comprising of hardware at the bottom layer, operating system running on top of the hardware and applications on the top most layer. The skeletal view of the traditional network with these layers is shown in Figure 9.3.



app OS

hardware

app OS

hardware

app OS

hardware

app OS

hardware

app OS

hardware

app OS

hardware

**Fig.9.3:** Skeletal View of the Traditional Network with Three Layers

There are several limitations in current network which is given below.

* Centralized control of the network is not feasible in traditional networks.
* Vendor-specific architecture of switches limits dynamic configuration according to application-specific requirements.

Software Defined Networking | **205**

* Each switch may have different operating system. Switches are required to configure according to the installed operating system.

In a traditional network, each router or switch will be running thousands of lines of code. Each router may comprise of millions of logic gates contributing to 10 GB RAM approximately, cost expensive and running various kinds of application programs such as routing, mobility management etc. Each of the routers in traditional network has specialized packet forwarding hardware, different operating system and different applications for routing, mobility management etc. The goal of the Software Defined Networking is to overcome the limitations or challenges of traditional network.

##### 9.3. Origin of SDN

The concept of SDN is to separate the hardware from the operating system and application programs as shown in Figure 9.4.

hardware

hardware

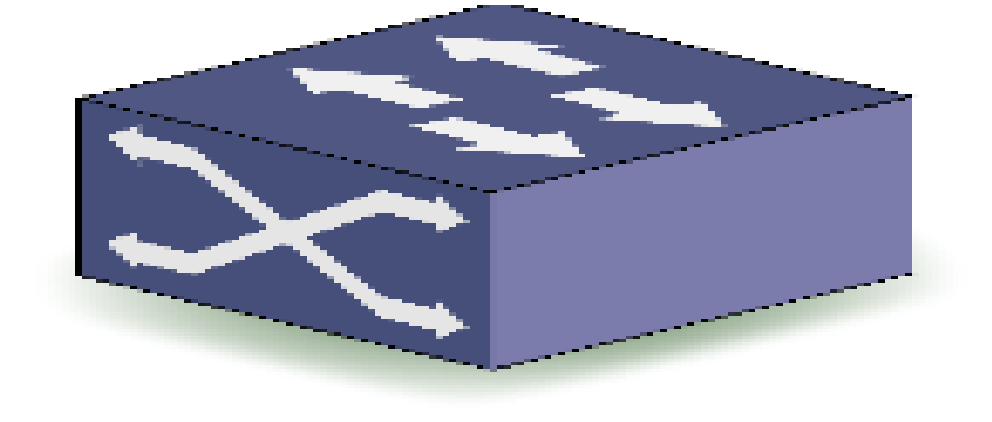
hardware

hardware

hardware

hardware

**Fig.9.4:** Concept of SDN



app OS

app OS

app OS

app OS

app OS

app OS

The history of SDN principles can be traced back to the separation of the control and data plane first used in the public switched telephone network as a way to simplify provisioning and management well before this architecture began to be used in data networks. The Internet Engineering Task Force (IETF) began considering various ways to decouple the control and forwarding functions in a proposed interface standard published in 2004 appropriately named “Forwarding and Control Element Separation” (ForCES). The ForCES Working Group also proposed a companion SoftRouter Architecture. Additional early standards from the IETF that pursued separating control from data include the Linux Netlink as an IP Services Protocol and A Path Computation Element (PCE)-Based Architecture.

These early attempts failed to gain traction for two reasons. One is that many in the Internet community viewed separating control from data to be risky, especially owing to the potential for a failure in the control plane. The second is that vendors were concerned

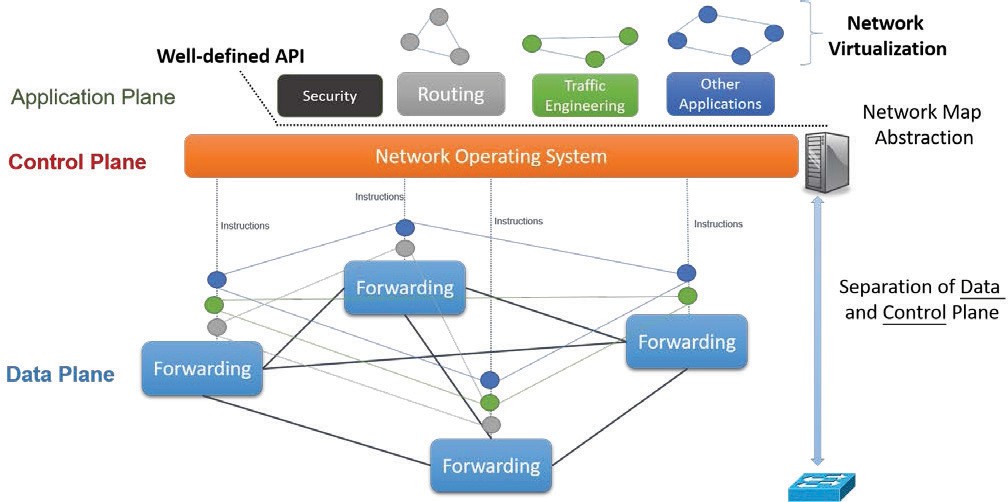
**206** | Internet of Things

that creating standard application programming interfaces (APIs) between the control and data planes would result in increased competition. The use of open source software in split control/data plane architectures traces its roots to the Ethane project at Stanford’s computer sciences department. Ethane’s simple switch design led to the creation of OpenFlow. An API for OpenFlow was first created in 2008. That same year witnessed the creation of NOX—an operating system for networks. Work on OpenFlow continued at Stanford, including with the creation of testbeds evaluating use of the protocol in a single campus network, as well as across the WAN as a backbone for connecting multiple campuses.

Beyond academia, the first deployments were by Nicira in 2010. The Open Networking Foundation was founded in 2011 to promote SDN and OpenFlow.

##### 9.4. SDN Architecture

Software Defined networking (SDN) is an approach to using open protocols, such as OpenFlow, to apply globally aware software control at the edges of the network to access network switches and routers that typically would use closed and proprietary firmware. SDN is meant to address the fact that the static architecture of traditional networks is decentralized and complex while current networks require more flexibility and easy troubleshooting. SDN suggests centralizing network intelligence in one network component by disassociating the forwarding process of network packets (Data Plane) from the routing process (Control plane). The control plane consists of one or more controllers which are considered as the brain of SDN network where the whole intelligence is incorporated. Figure 9.5 shows the architecture of SDN.



**Fig.9.5:** Architecture of SDN

The components or attributes of SDN are hardware switches, controller, applications, flow-rules and Application Programming Interfaces (API). The basic concept of SDN is to

Software Defined Networking | **207**

separate control logic from hardware switches. The control logic is defined in a centralized manner and controls the entire network including the switches. Communication between the application, control and data planes are done through APIs.

The Data plane contains various switches for forwarding. The Control plane contains the network operating system. At the top is the application plane which performs various functionalities like security, routing, traffic engineering and other applications. The concept of SDN is made possible through Network Function Virtualization. The interface between Application plane and Control plane is known as Northbound API whereas the interface between Control plane and Data plane is known as Southbound API. The protocol that supports Southbound API is the OpenFlow protocol. East-Westbound APIs are used to communicate among multiple controllers in the control layer.

The current status of SDN shows that companies such as Google have started to implement SDN at their data center networks. The current networks could be replaced with SDN in a phased manner. With SDN, operational cost and delay caused due to link failure can be significantly minimized. The two main challenges in the implementation of SDN are

(a) rule placement (b) controller placement.

##### 9.5. Rule Placement

Switches forward traffic based on a rule called “Flow Rule”. This is defined by the centralized controller. In the traditional network, there will be a routing table at every switch or router. Here in SDN, a Flow Table is maintained at every switch. Each entry in the Flow Table is the Flow Rule. Each Rule has a specific format, which is defined by a protocol. In SDN, it is OpenFlow protocol. Figure 9.6 shows an example for Flow Rules based on OpenFlow protocol.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Priority | Ingress Port | MAC  Source Address | MAC  Destination | Protocol | Vlan ID | IP  Source Address | IP Destination | Source Port | Destination Port | Instruction |
| 10000 | – | – | – | TCP | – | – | 10.1.1.20/32 | – | 80 | Forward to Port 1 |
| 5000 | – | – | – | – | – | – | 10.1.1.0/24 | – | – | Forward to Port 2 |
| 300 | – | – | – | – | 2600 | – | – | – | – | Send to Controller |
| 0 | – | – | – | – | – | – | – | – | – | OF Normal |

**Fig.9.6:** Example for Flow Rules based on OpenFlow Protocol

**Match SDN Applications First and Use Normal for Unmatched Packets (Hybrid Default Forwarding)**

Consider the first row of the Flow Table. When a packet arrives at a router, it will check for the priority. If the priority matches, the values for Ingress port, MAC source address, MAC destination can be anything because it is a wildcard. If the protocol is TCP, IP destination is 10.1.1.20/32 and destination port is 60, then the instruction is “Forward to port 1”. Likewise other rows (Flow Rules) of the Flow Tables are used for routing.

**208** | Internet of Things

The Flow Tables are stored at the switches. Flow Tables are stored in a specialized memory known as Ternary Content Addressable (TCAM) memory. The size of TCAM memory is limited and hence only a limited number of rules can be inserted. TCAM memories can do the processing very faster and that is why this memory is used at the switches. TCAM is very expensive also.

When a request is received for which no Flow Rule is present in the Flow Table, the switch send a PACKET-IN message to the controller. The controller then decides a suitable Flow Rule for the request and sends back to the switch. It is then inserted into the Flow Table. Whenever there is a new rule insertion, there will be a delay of 3-5 ms delay.

It is a challenging task to define or place the rules at switches while considering available TCAM. This is because TCAM is small and expensive and not easy to scale. Another challenge is how to define rules, so that less number of PACKET\_IN messages is sent to the controller.

##### 9.6. OpenFlow Protocol

OpenFlow protocol is the only one protocol that is available for rule placement. It has different versions like 1.0, 1.1, 1.2, 1.3 etc. to have different number of match fields. Different match fields include source IP, destination IP, source port, priority etc. One of the issues with OpenFlow protocol is how long a Flow Rule is to be kept at the switch. For addressing this issue, two strategies are used. The first one is the hard timeout and the other one is the soft timeout. In hard timeout, all the Flow Rules are deleted from the switch. This can be used to reset the switch. In a soft timeout, if no Flow is received associated with a Rule, then the Rule is deleted. This is used to empty the Rule space by deleting an unused Rule.

SDN and OpenFlow is not the same. SDN is a technology or concept while OpenFlow is a protocol used to communicate between data plane and control plane. We may have other protocols for this purpose. However, OpenFlow is the only protocol present today. There are various softwares associated with OpenFlow implementation. Some of them are given below.

* + - Indigo – Indigo is an open source project aimed at enabling support for OpenFlow on physical and hypervisor switches.
    - LINC – LINC is a pure OpenFlow software switch. It gives a lot of flexibility and allows quick development and testing of new OpenFlow features. It supports OpenFlow protocol 1.2, OpenFlow protocol 1.3 and OpenFlow protocol 1.4. It has a modular architecture which is easily extensible.
    - Pantou – Pantou turns a commercial wireless router or access point to an OpenFlow- enabled switch. OpenFlow is implemented as an application on top of OpenWrt. Pantou is based on the BackFire OpenWrt release (Linux 2.6.32).
    - Of13softswitch – This is an OpenFlow 1.3 compatible user-space software switch implementation. The code is based on the Ericsson TrafficLab 1.1 softswitch implementation, with changes in the forwarding plane to support OpenFlow 1.3.

Software Defined Networking | **209**

* + - Open vSwitch – This is an open source software which is the most popular one which is present today for SDN implementation.

##### 9.7. Controller Placement

Controllers define Flow Rules according to the application specific requirements. Control logic is taken care by the Control plane. The controller knows what has to be done with a particular Flow. The controllers must be able to handle all the incoming requests from the switches. Rules should be placed without incurring much delay. Typically, a controller can handle 200 requests in a second through a single thread. Currently multi-threaded applications are also possible in controllers.

The controllers are logically connected to the switches in one-hop distance. The switch thinks that controller is just a hop away from it. Physically, they are connected to the switches in multi-hop distance. If we have a very small number of controllers for a large network, the network might be congested with control packets i.e. PACKET-IN messages. Various architectures for Controller Placement are Flat architecture, Hierarchical (Tree) architecture, Ring architecture and Mesh architecture. Figure 9.7 shows a Flat architecture for Controller Placement.

Packet-IN Flow-Rule

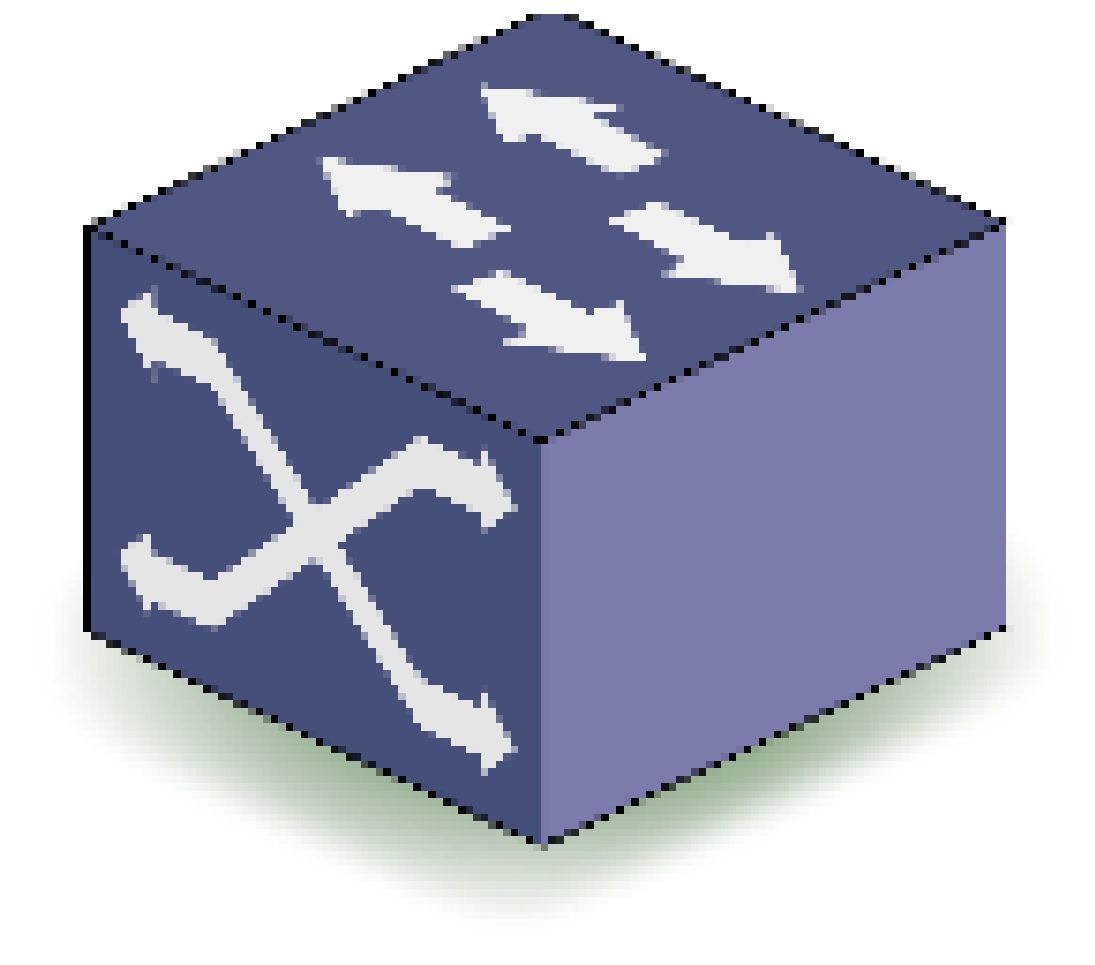


**Fig.9.7:** Flat architecture for Controller Placement

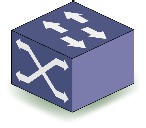
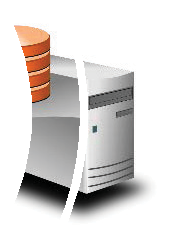
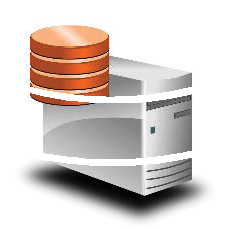
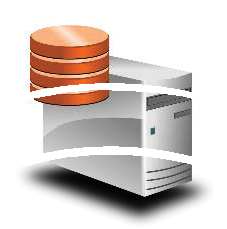
In Flat architecture, the Controller is logically one hop away from the switch. The switch sends the PACKET-IN message to the controller, if the Flow Rule for the particular Flow that it has received is not in the Flow Table. The Controller sends back the new Flow Rule to the switch. Figure 9.8 shows the Hierarchical (Tree) architecture for Controller Placement.

In a Hierarchical architecture, the controllers and switches are arranged like a tree structure. Figure 9.9 shows the Ring architecture for Controller Placement. In a Ring architecture, multiple controllers are arranged in a ring fashion. But a particular switch is connected to only one controller. A PACKET-IN message will be sent to a single Controller only and the Controller sends back the Flow Rule to that particular switch.

**210** | Internet of Things



**Fig.9.8:** Hierarchical (Tree) Architecture for Controller Placement

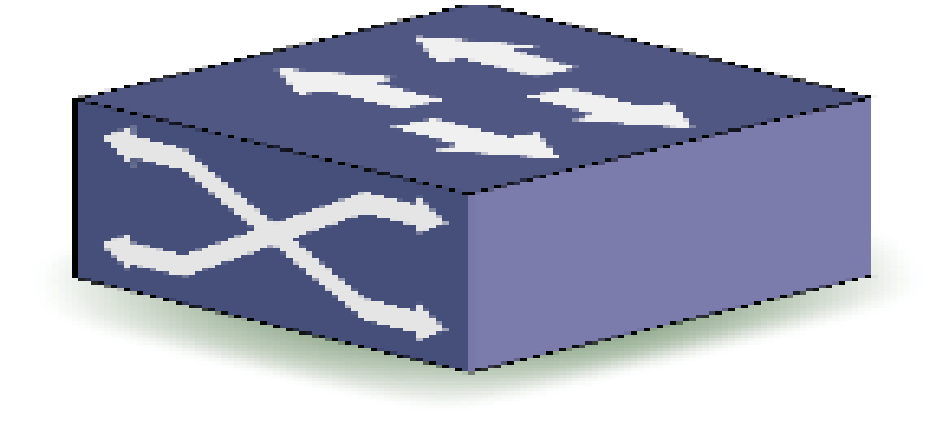


**Fig.9.9:** Ring Architecture for Controller Placement

Figure 9.10 shows the Mesh architecture for Controller Placement. Mesh architecture increases reliability and fault-tolerance. For example we can see two switches connected to the same controller. Even if a switch goes down, the other switch can take care of it.

User I

**Fig.9.10:** Mesh Architecture for Controller Placement



User II

Software Defined Networking | **211**

The control mechanisms can be either distributed or centralized. In a distributed control mechanism, the control decisions can be taken in a distributed manner. For example, each sub-network is controlled by a different controller. In a centralized control mechanism, the control decisions are taken in a centralized manner. For example, a network is controlled by a single controller. If the primary controller is down, a backup controller takes up the duties of the primary controller to have uninterrupted network service.

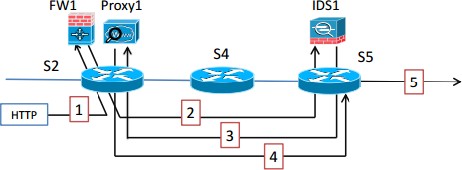
For experimenting with SDN, we need simulators or emulators. The emulators can be connected to the real network and can analyze the data. Simulators on the other hand, simulate the entire network including nodes and packet flows. The simulator or emulator takes care of infrastructure deployment and Controller Placement. Both must be supported with OpenFlow protocol. In Controller Placement, the controllers can be either remote or local. In a remote Controller, the Controller can be situated in a remote place and communicated using IP address and port number.

For switch deployment, Mininet software based on Python language can be used. This software can be used to create a virtual network with OpenFlow enabled switches. It supports both remote and local controllers. Various softwares are available for Controller configuration like Pox, Nox, FloodLight, OpenDayLight and ONOS. Among these OpenDayLight and ONOS are two popularly used softwares for Controller configuration.

Performance of SDN depends on Rule Placement and Controller Placement in the network. Control message overhead may be increased due to additional number of packets (PACKET-IN) messages. Unified network management is possible using SDN, while leveraging the global view of the network.

##### 9.8. Security in SDN

There is enhanced security while using SDN. The security is implemented using firewall, proxy, HTTP and Intrusion Detection System (IDS). Figure 9.11 shows a topology to implement security in SDN.



**Fig.9.11:** A Topology to Implement SDN

1. When an HTTP request comes, it is first forwarded to Firewall 1 (FW1).
2. From the Firewall 1, it is sent to Intrusion Detection System 1 (IDS1).

**212** | Internet of Things

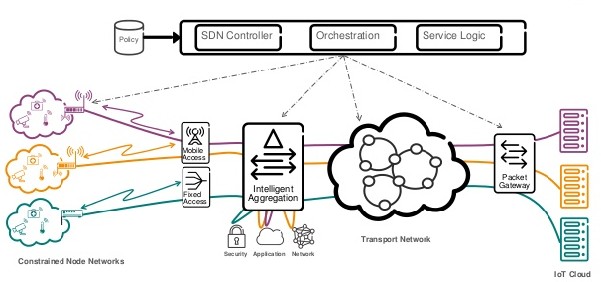
1. From (IDS1), it is sent to Proxy 1
2. Finally, it is sent from Proxy1 to the outside network.

##### 9.9. Integrating SDN in IoT

In this section we will see how SDN can be used with IoT. There are several benefits of integrating SDN with IoT.

1. With SDN, intelligent routing decisions can be deployed.
2. Simplification of information collection, analysis and decision making.
3. Visibility of network resources which includes network management simplification based on user, device and application specific requirements.
4. Intelligent traffic pattern analysis and coordinated decisions.

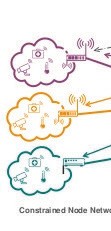
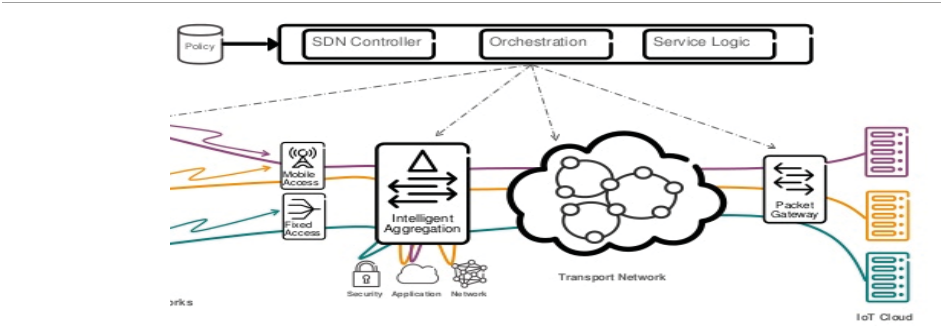
Consider the use of SDN for IoT as shown in Figure 9.12. In this there are several IoT devices in different sub-networks. The data from these devices through mobile access channels or fixed access channels are collected and transmitted to the data aggregator. The data aggregation from various IoT devices takes place at the data aggregator and then it passes through a transport network. From there it goes to different gateways and the packet segregation is done at the packet gateways. When we integrate SDN with IoT, we are going to integrate SDN controller with various devices, performs better orchestration of various devices and protocols that are running. Overall it will improve the service logic.



**Fig.9.12:** SDN for IoT

With the implementation of SDN, there is a centralized control of IoT devices like sensors, actuators, RFID tags etc. as shown in Figure 9.13.

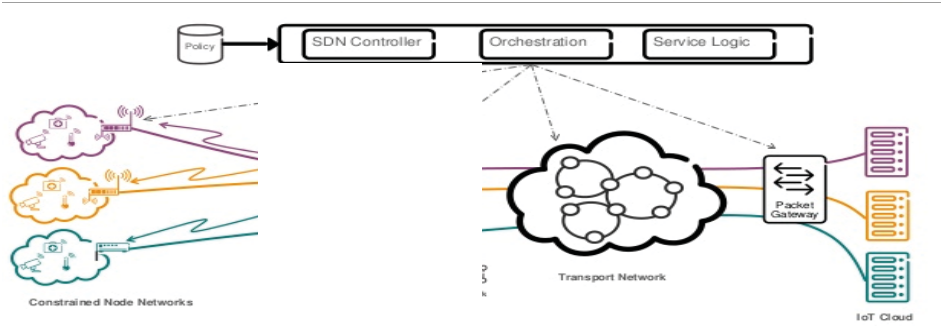
Software Defined Networking | **213**



Control of end-devices, such as sensors and actuators

**Fig.9.13:** Centralized Control of IoT Devices

The Rule-placement at access devices, while considering mobility and heterogeneity of end-users can be taken care by the fixed access channels, mobile access channels and intelligent aggregation as shown in Figure 9.14.

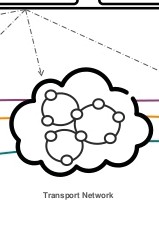
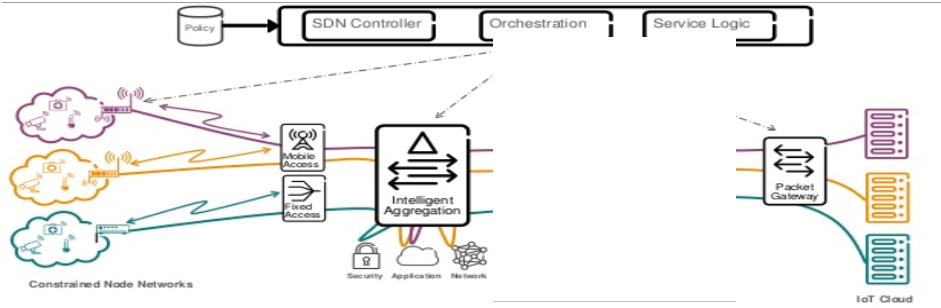


Rule-placement at access devices, while considering mobility and heterogeneity of end-users

**Fig.9.14:** Rule Placement at Access Devices

Rule Placement and traffic engineering at backbone networks is implemented by the Transport Network as shown in Figure 9.15.

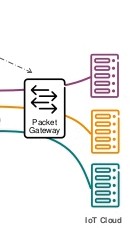
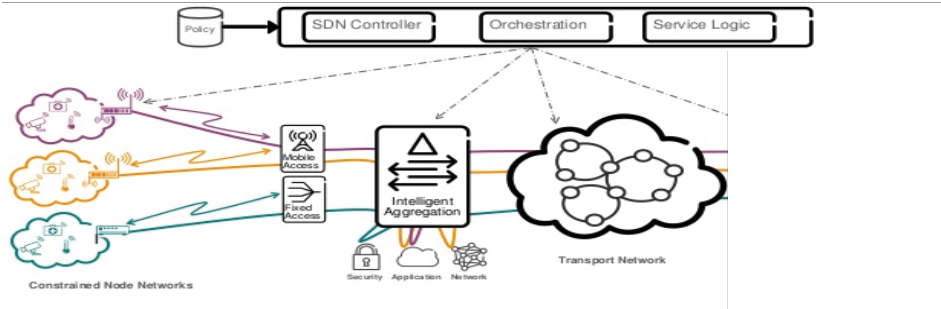
**214** | Internet of Things



Rule-placement and traffic engineering at backbone networks

**Fig.9.15:** Rule Placement and Traffic Engineering at Backbone Networks

Finally Flow classification and enhanced security at data center networks is done at packet gateways or data center networks as shown in Figure 9.16.



Flow classification and enhanced security at data center networks

**Fig.9.16:** Flow Classification and Enhanced Security at Packet Gateways

##### 9.10. Software Defined WSN

There are several challenges in WSNs. Real-time programming of sensor nodes is difficult. The sensor nodes and corresponding network follow vendor specific architecture. Each nodes made by different vendors have their own architecture and different layers implemented in them. Sensor nodes are resource-constrained i.e. heavy computations cannot be performed at the nodes. In addition, memory is also limited so that we cannot insert too many control programs. These challenges give the opportunity for the following.

* Can we program the sensor nodes in real-time?
* Can we change the forwarding path in real-time
* Can we integrate different sensor nodes in a WSN?

There are different solutions proposed for the above opportunities.

Software Defined Networking | **215**

##### Sensor OpenFlow

One solution is the Sensor OpenFlow protocol. This protocol takes care of data forwarding in two different ways. One is the value-centric data forwarding which forwards the sensed data, if it exceeds a certain value. The other one is the in-centric data forwarding which forwards the sensed data based on the ID of the source node. Real life implementation of such a method is not available. It still exists in its infant stage.

##### Soft-WSN

Soft-WSN is another protocol for sensor device management and topology management. The sensor device management comprises of a) sensor management b) delay management and c) active-sleep management. In sensor management, multiple sensors can be implemented in a single board and sensors can be used depending on application-specific requirements. In delay management, delay for sensing can be changed dynamically in real-time. In active-sleep management, the status of active and sleep mode can be changed dynamically.

Topology management comprises of (a) node-specific management and (b) network- specific management. In node-specific management, forwarding logic of a particular sensor can be modified. Network-specific management decides whether to forward all traffic of a node in the network or to drop all traffic of a node in the network.

The network performance can be improved using software defined WSN over traditional WSN. Packet delivery ratio in the network increases and number of replicated data packets is reduced using Soft-WSN compared to the traditional WSN. The number of control messages in the network is higher using Soft-WSN over the traditional WSN. This is due to the PACKET-IN message in the network. Each time a node receives a new packet, it asks the Controller for getting adequate forwarding logic.

##### SDN-WISE

A third protocol is SDN-WISE. SDN-WISE proposes a Software Defined Networking platform for WSN. Flow-Table for the Rule Placement is available at sensor nodes. Any programming language can be used through API to program the nodes in real-time. In SDN- WISE, the sensor node includes IEEE 802.15.4 protocol, Micro Control Unit (MCU) and IEEE 802.15.4 stack over which the SDN-WISE really functions. Forwarding layer consists of Flow Rules and In Network Packet Processing (INPP) process the packets in SDN-WISE.

##### 9.11. SDN for Mobile Networking

Meeting current market requirements is virtually impossible with traditional network architectures. Faced with flat or reduced budgets, enterprise IT departments are trying to squeeze the most from their networks using device-level management tools and manual processes. Carriers face similar challenges as demand for mobility and bandwidth explodes. Profits are being reduced by escalating capital equipment costs and flat or declining revenue.

**216** | Internet of Things

Existing network architectures were not designed to meet the requirements of today’s users, enterprises and carriers. Network designers are constrained by the limitations of current networks, which include the following.

1. **Complexity:** Networking technology to date has consisted largely of discrete sets of protocols designed to connect hosts reliably over arbitrary distances, link speeds and topologies. To meet business and technical needs over the last few decades, the industry has evolved networking protocols to deliver higher performance and reliability, broader connectivity and more stringent security. Protocols tend to be defined in isolation, however, with each solving a specific problem and without the benefit of any fundamental abstractions. This has resulted in one of the primary limitations of today’s network complexity. For example, to add or move any device, IT must touch multiple switches, routers, firewalls, Web authentication portals, etc. and update Access Control Lists (ACLs), Virtual LANs (VLANs), Quality of Services (QoS) and other protocol-based mechanisms using device-level management tools. In addition, network topology, vendor switch model and software version all must be taken into account. Due to this complexity, today’s networks are relatively static as IT seeks to minimize the risk of service disruption. The static nature of networks is in stark contrast to the dynamic nature of today’s server environment, where server virtualization has greatly increased the number of hosts requiring network connectivity and fundamentally altered assumptions about the physical location of hosts.

Prior to virtualization, applications resided on a single server and primarily exchanged traffic with select clients. Today, applications are distributed across multiple virtual machines (VMs), which exchange traffic flows with each other. VMs migrate to optimize and rebalance server workloads, causing the physical end points of existing flows to change (sometimes rapidly) over time. VM migration challenges many aspects of traditional networking, from addressing schemes and namespaces to the basic notion of a segmented, routing-based design. In addition to adopting virtualization technologies, many enterprises today operate an IP converged network for voice, data and video traffic. While existing networks can provide differentiated QoS levels for different applications, the provisioning of those resources is highly manual. IT must configure each vendor’s equipment separately and adjust parameters such as network bandwidth and QoS on a per-session, per-application basis. Because of its static nature, the network cannot dynamically adapt to changing traffic, application and user demands.

1. **Inconsistent policies**: To implement a network-wide policy, IT may have to configure thousands of devices and mechanisms. For example, every time a new virtual machine is brought up, it can take hours, in some cases days, for IT to reconfigure ACLs across the entire network. The complexity of today’s networks makes it very difficult for IT to apply a consistent set of access, security, QoS and other policies to increasingly mobile users, which leaves the enterprise vulnerable to security breaches,

Software Defined Networking | **217**

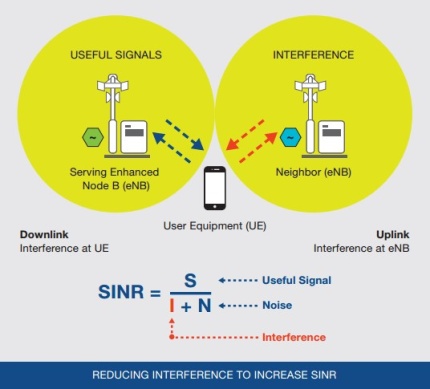
non-compliance with regulations and other negative consequences.

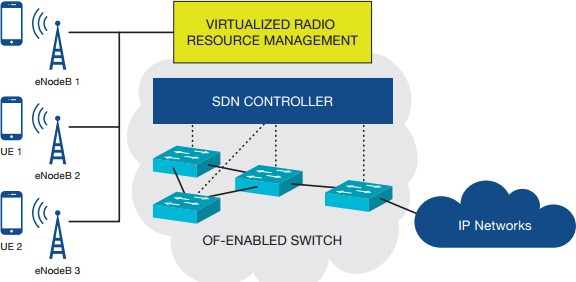
1. **Inability to scale**: The network becomes vastly more complex with the addition of hundreds or thousands of network devices that must be configured and managed. IT has also relied on over subscription to scale the network, based on predictable traffic patterns. However, in today’s virtualized data centers, traffic patterns are incredibly dynamic and therefore unpredictable.
2. **Vendor dependence**: Carriers and enterprises seek to deploy new capabilities and services in rapid response to changing business needs or user demands. However, their ability to respond is hindered by vendors’ equipment product cycles, which can range to three years or more. Lack of standard and open interfaces limits the ability of network operators to tailor the network to their individual environments. This mismatch between market requirements and network capabilities has brought the industry to a tipping point. In response, the industry has created the Software- Defined Networking (SDN) architecture and is developing associated standards.

The important aspects of SDN for mobile networking are

* Flow Table paradigm of SDN which is well suited for end –to-end communication over multiple technologies such as WiFi, 3G, 4G etc.
* Logically centralized control which is particularly useful for efficient base station coordination for addressing inter-cell reference.
* Path management in which data can be routed based on service requirements without depending on core routing policies.
* Network virtualization which abstracts the physical resources from the network services. This helps in providing seamless connectivity and service differentiation among users.

Figure 9.17 shows the diagram for interference management in Software Defined Wireless Mobile Network (SDWMN).





Signals of eNodeB 2 will not affect signals of eNodeB 3

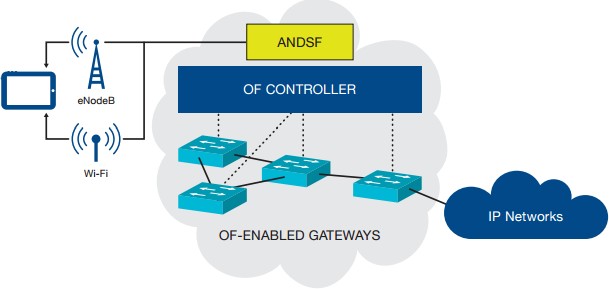
Traditional Mobile Network Software-Defined Mobile Network

**Fig.9.17:** Interference Management in SDWMN

**218** | Internet of Things

Consider a traditional mobile network. When mobile equipment operates, it connects with a base station or access point for sending data back and forth. At the same time, the equipment may get affected by another access point which is known as interference. SDWMN tries to address the problem of interference with the help of SDN. In the above Figure, we can see an SDN controller and Virtualized Radio Resource Management Unit which takes care of interference. For example, the signals from eNodeB2 will not affect the signals eNodeB3 as shown in the Figure 9.17.

Mobile traffic management in Software Defined Wireless Mobile Network (SDWMN) is shown in Figure 9.18.



**Fig.9.18:** Mobile Traffic Management in SDWMN

In Figure 9.18, the Access Network Discovery and Service Function (ANDSF) takes care of the mobile traffic management in SDWMN. When a user moves, he can be first connected to WiFi and then to the cellular network 3G, 4G and so on the SDN enabled wireless networks takes care of the traffic management and connectivity.

The key benefits of SDN in mobile networks are centralized control of devices manufactured by multiple vendors, higher rate of integration of new services and abstract network control and management which includes networks abstracted from the user.

##### 9.12. Rule Placement at Access Devices

The challenges in Rule Placement at access devices in mobile networks are General OpenFlow protocol is not supported by wireless network and a modified version of OpenFlow is required. Typically, users are mobile in nature and network is highly dynamic. Frequent changes in Rule Placement are also required. Another challenge is the presence of heterogeneous devices in the network and how to support such heterogeneous devices in a single platform. The approaches for Rule Placement at access devices are ODIN, Ubi-Flow and Mobi-Flow.

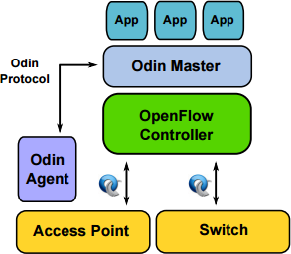
##### 9.12.1. ODIN

In Open Data centre with Interoperable Network (ODIN), an agent is placed at access points

Software Defined Networking | **219**

to communicate with controller. ODIN is an SDN framework programmable enterprise WLANs. It provides a platform for developing typical enterprise WLAN services such as mobility managers, and load balancers as “network applications”. Two components are present

– an ODIN agent and ODIN master. ODIN agent is placed with the physical devices and ODIN master is placed at the Controller end. Figure 9.19 shows the architecture of ODIN.



**Fig.9.19:** Architecture of ODIN

Odin’s architecture comprises of a single master and a set of agents that run on the (Access Points) APs. The master is implemented as an OpenFlow application on top of the Controller. The agents are implemented and run on the APs. The APs run Open vSwitch so that OpenFlow rules can be installed at each AP as well. Every Light Virtual Access Point (LVAP) has a set of OpenFlow rules attached to it, which enable faster mobility and can potentially be used to do access control as well.

An LVAP is basically a per-client, virtual AP. Every time a client performs a probe scan, a central controller spawns an LVAP on a physical AP close to the client. The LVAP is what responds to the client’s probe scan with a probe response. The client can then continue the regular 802.11 association handshake with the LVAP. Once this is done, an association is formed between a client device capable of mobility and an LVAP. At this point, the client is only concerned with getting ACKs for the data frames it generates and receiving beacons from its LVAP in a timely fashion. Just like virtual machines in a data center can be migrated to a different physical server, an LVAP can be migrated to another physical AP. If an LVAP is migrated to another physical AP fast enough such that it retains all the necessary state to maintain its associations and the client continues to receive ACKs and beacon frames, then the client can continue transmitting data frames without having to perform a re-association. Thus, an LVAP migration gives the effect of a handoff, without actually inducing a state machine change at the client and if this is done fast enough, the client won’t even notice any period of disconnectivity.

Odin applications run atop the master, and control the placement of LVAPs. For instance, a mobility manager would attempt to place the LVAPs as close to the client as possible, and a load balancer would move LVAPs between physical APs such that the APs have an even

**220** | Internet of Things

processing load. Applications written on top of Odin can be both reactive, and proactive. Reactive applications use a pub-sub mechanism to receive notifications for specific MAC layer events that they can subscribe to. Proactive applications merely run within a loop, cycling between sleeping and doing some work. The flow diagram for the communication between ODIN master and ODIN agent is shown below in Figure 9.20.

**ODIN Agent ODIN Master**

No

Client has LVAP

assigned?

No

Assign LVAP

Client has LVAP

assigned?

Yes

Receive 802.11 Mgt. Frame

Process and Respond

Forward Up

Client has LVAP

assigned?

No

Yes

Receive 802.11 Data Frame

Drop Frame

**Fig.9.20:** Flow Diagram for Communication between ODIN Master and ODIN Agent

##### 9.12.2. Ubi-Flow

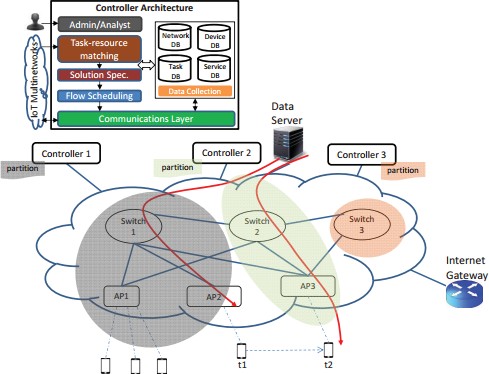
Ubi-Flow is the first software-defined IoT system for ubiquitous flow control and mobility management in urban heterogeneous networks. To achieve light-weight processing in IoT devices, in UbiFlow all jobs related to mobility management, handover optimization, access point selection and flow scheduling are executed by the coordination of distributed controllers. Specifically, UbiFlow adopts multiple controllers to divide an urban-scale SDN into different geographic partitions to achieve distributed control of IoT flows. A distributed hashing based overlay structure is proposed to maintain network scalability and consistency. Based on this UbiFlow overlay structure, relevant issues in mobility management such as scalable control, fault tolerance and load balancing have been carefully taken care of. The UbiFlow controller differentiates flow scheduling based on per-device requirements as well as whole-partition capabilities. Flow scheduling comprises of network partition, network matching and load balancing. Therefore, it can present a network status view for the optimized selection of access points in multi networks to satisfy flow requests, while guaranteeing the network performance in each partition. Figure 9.21 shows the architecture of Ubi-Flow.

The key contributions of UbiFlow are:

* + - * A novel overlay structure to achieve mobility management and fault tolerance in software-defined IoT.

Software Defined Networking | **221**

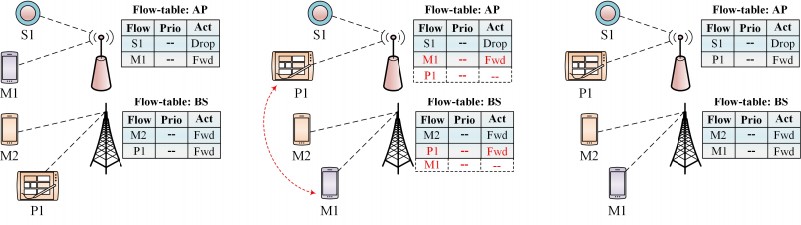
* + - * An optimal assignment algorithm for the controller to match the best available access points to IoT devices, with network status analysis and flow requests as inputs.
      * A load balancing scheme for distributed controllers by analyzing the variations in flow traffic characteristics.



**Fig.9.21:** Architecture of Ubi-Flow

##### 9.12.3. Mobi-Flow

Figure 9.22 shows the Mobi-Flow Rule Placement in Software Defined Internet of Things (SDIoT).



**Fig.9.22:** Mobi-Flow Rule Placement in SDIoT

**222** | Internet of Things

There are devices S1 and M1 connected to one Access Point (AP) and another two devices P1 and M2 connected to another Base Station (BS). The corresponding Flow Tables are also available. Imagine that the device P1 and M1 has interchanged its position. So a new Flow Rules need to be added in the respective Flow Tables. But the memory of Flow Tables is restricted. In Mobi-Flow, a proactive Rule Placement takes place depending on user’s movement in the network. The approach is to predict the location of end users at (t+1) time, while the users are at (t) time. Mobi-Flow place Flow Rules at the Access Points (APs) which can be associated to the users based on their predicted locations.

For location prediction, it uses Order-K Markov predictor which takes Kth location instances to predict the next location. For Flow Rule Placement, linear programming is used to select optimal Access Point (AP). Control message overhead and energy consumption can be minimized significantly using Mobi-Flow compared to the conventional Flow Rule placement schemes.

Anomalies can be monitored through OpenFlow. This can be done by monitoring each Flow in the network. It is also possible to collect the port statistics of the switches. If there is any anomaly, it may generate large number of packets in the network which leads to network congestion.

##### 9.13. Conclusion

This Chapter explains Software Defined Networking in detail. Limitations of current network, origin of SDN, SDN architecture, Rule Placement, OpenFlow protocol and Controller Placement are well explained. Security in SDN, integrating SDN in IoT, Software Defined WSN and SDN for mobile networking is described. Finally Rule Placement approaches at access devices including ODIN, Ubi-Flow and Mobi-Flow are clearly illustrated.