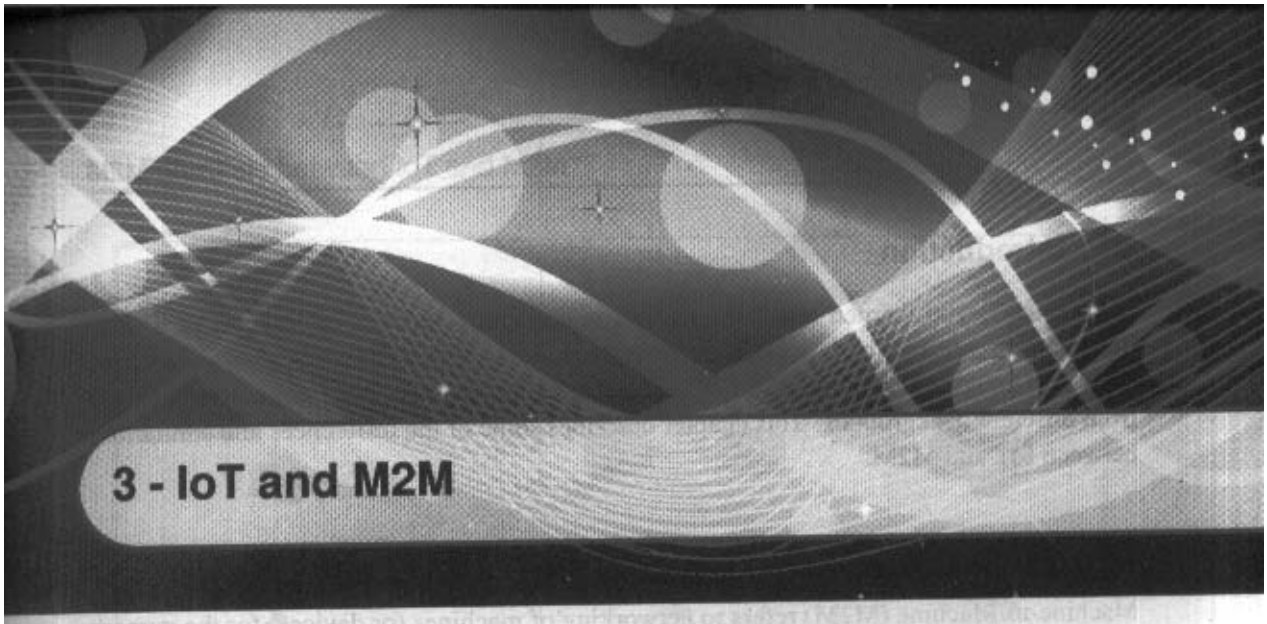


3 - IoT and M2M

This Chapter Covers

- M2M
- Differences and Similarities between M2M and IoT
- SDN and NFV for IoT

3.1 Introduction

In Chapter-1, you learned about the definition and characteristics of Internet of Things (IoT). Another term which is often used synonymously with IoT is Machine-to-Machine (M2M). Though IoT and M2M are often used interchangeably, these terms have evolved from different backgrounds. This chapter describes some of the differences and similarities between IoT and M2M.

3.2 M2M

Machine-to-Machine (M2M) refers to networking of machines (or devices) for the purpose of remote monitoring and control and data exchange. Figure 3.1 shows the end-to-end architecture for M2M systems comprising of M2M area networks, communication network and application domain. An M2M area network comprises of machines (or M2M nodes) which have embedded hardware modules for sensing, actuation and communication. Various communication protocols can be used for M2M local area networks such as ZigBee, Bluetooh, ModBus, M-Bus, Wirless M-Bus, Power Line Communication (PLC), 6LoWPAN, IEEE 802.15.4, etc. These communication protocols provide connectivity between M2M nodes within an M2M area network. The communication network provides connectivity to remote M2M area networks. The communication network can use either wired or wireless networks (IP-based). While the M2M area networks use either proprietary or non-IP based communication protocols, the communication network uses IP-based networks. Since non-IP based protocols are used within M2M area networks, the M2M nodes within one network cannot communicate with nodes in an external network. To enable the communication between remote M2M area networks, M2M gateways are used.

Figure 3.2 shows a block diagram of an M2M gateway. The communication between the M2M nodes and the M2M gateway is based on the communication protocols which are native to the M2M area network. M2M gateway performs protocol translations to enable IP-connectivity for M2M area networks. M2M gateway acts as a proxy performing translations from/to native protocols to/from Internet Protocol (IP). With an M2M gateway, each node in an M2M area network appears as a virtualized node for external M2M area networks.

The M2M data is gathered into point solutions such as enterprise applications, service management applications, or remote monitoring applications. M2M has various application domains such as smart metering, home automation, industrial automation, smart grids, etc. M2M solution designs (such as data collection and storage architectures and applications) are specific to the M2M application domain.

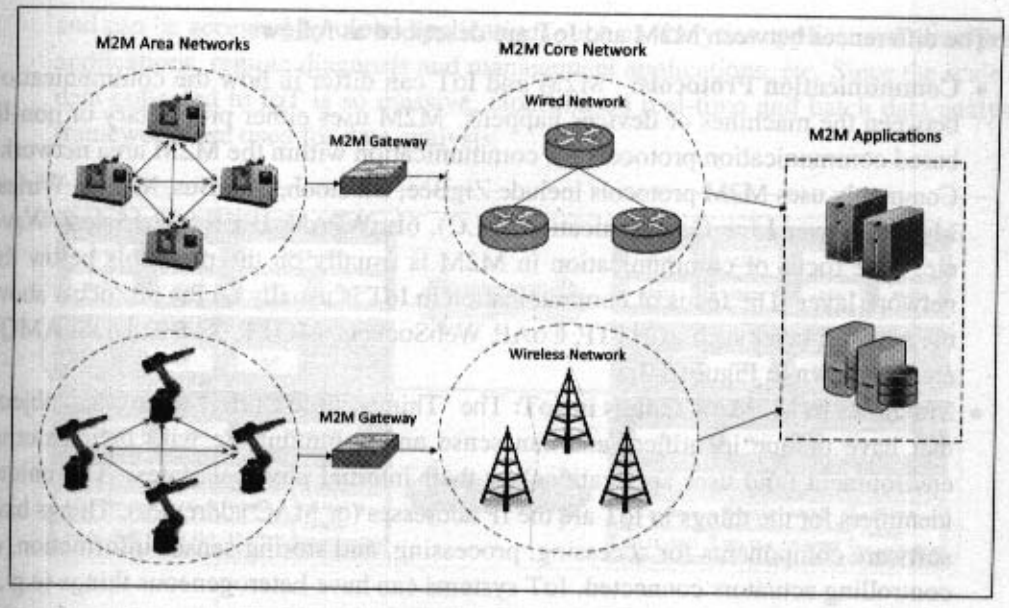


Figure 3.1: M2M system architecture

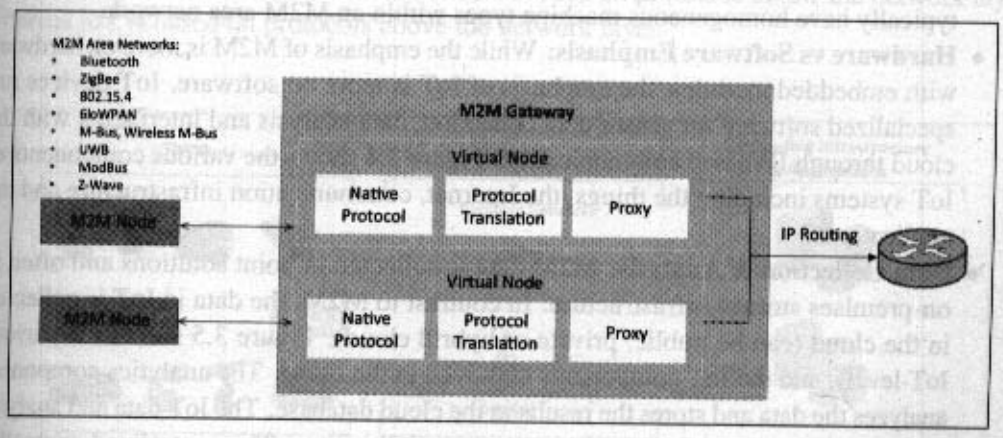


Figure 3.2: Block diagram of an M2M gateway

3.3 Difference between IoT and M2M

Though both M2M and IoT involve networking of machines or devices, they differ in the underlying technologies, systems architectures and types of applications.

The differences between M2M and IoT are described as follows:

- **Communication Protocols:** M2M and IoT can differ in how the communication between the machines or devices happens. M2M uses either proprietary or non-IP based communication protocols for communication within the M2M area networks. Commonly uses M2M protocols include ZigBee, Bluetooh, ModBus, M-Bus, Wirless M-Bus, Power Line Communication (PLC), 6LoWPAN, IEEE 802.15.4, Z-Wave, etc. The focus of communication in M2M is usually on the protocols below the network layer. The focus of communication in IoT is usually on the protocols above the network layer such as HTTP, CoAP, WebSockets, MQTT, XMPP, DDS, AMQP, etc., as shown in Figure 3.3.
- **Machines in M2M vs Things in IoT:** The "Things" in IoT refers to physical objects that have unique identifiers and can sense and communicate with their external environment (and user applications) or their internal physical states. The unique identifiers for the things in IoT are the IP addresses (or MAC addresses). Things have software components for accessing, processing, and storing sensor information, or controlling actuators connected. IoT systems can have heterogeneous things (e.g., a home automation IoT system can include IoT devices of various types, such as fire alarms, door alarms, lighting control devices, etc.) M2M systems, in contrast to IoT, typically have homogeneous machine types within an M2M area network.
- **Hardware vs Software Emphasis:** While the emphasis of M2M is more on hardware with embedded modules, the emphasis of IoT is more on software. IoT devices run specialized software for sensor data collection, data analysis and interfacing with the cloud through IP-based communication. Figure 3.4 shows the various components of IoT systems including the things, the Internet, communication infrastructure and the applications.
- **Data Collection & Analysis:** M2M data is collected in point solutions and often in on-premises storage infrastructure. In contrast to M2M, the data in IoT is collected in the cloud (can be public, private or hybrid cloud). Figure 3.5 shows the various IoT-levels, and the IoT components deployed in the cloud. The analytics component analyzes the data and stores the results in the cloud database. The IoT data and analysis results are visualized with the cloud-based applications. The centralized controller is aware of the status of all the end nodes and sends control commands to the nodes. Observer nodes can process information and use it for various applications, however, observer nodes do not perform any control functions.
- **Applications:** M2M data is collected in point solutions and can be accessed by on-premises applications such as diagnosis applications, service management applications, and on-premisis enterprise applications. IoT data is collected in the cloud

and can be accessed by cloud applications such as analytics applications, enterprise applications, remote diagnosis and management applications, etc. Since the scale of data collected in IoT is so massive, cloud-based real-time and batch data analysis frameworks are used for data analysis.

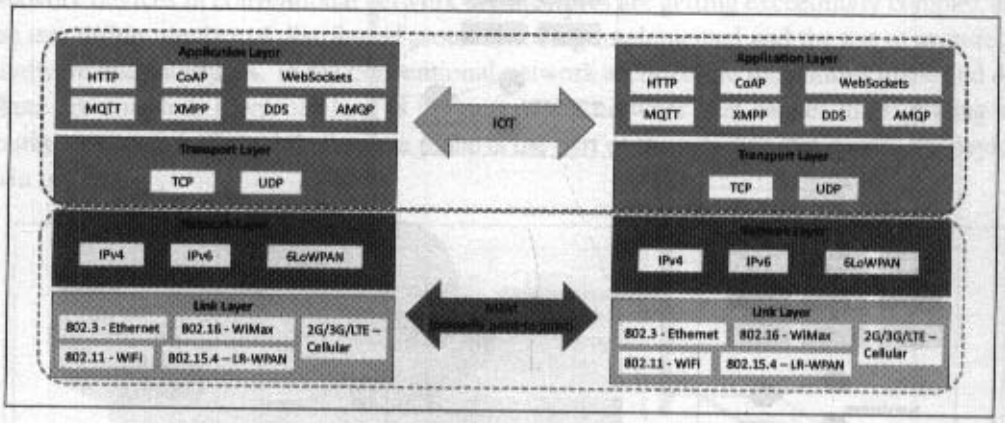


Figure 3.3: Communication in IoT is IP-based whereas M2M uses non-IP based networks. Communication within M2M area networks is based on protocols below the network layer whereas IoT is based on protocols above the network layer.

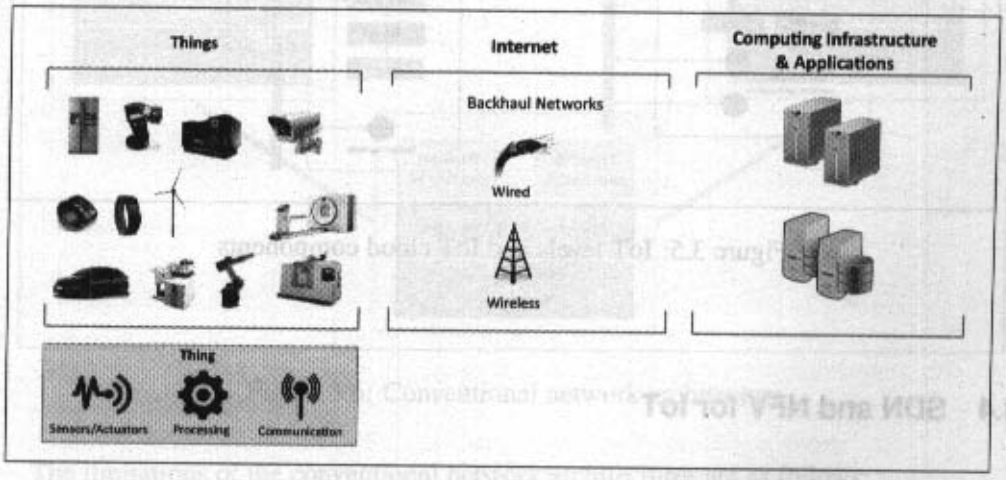


Figure 3.4: IoT components

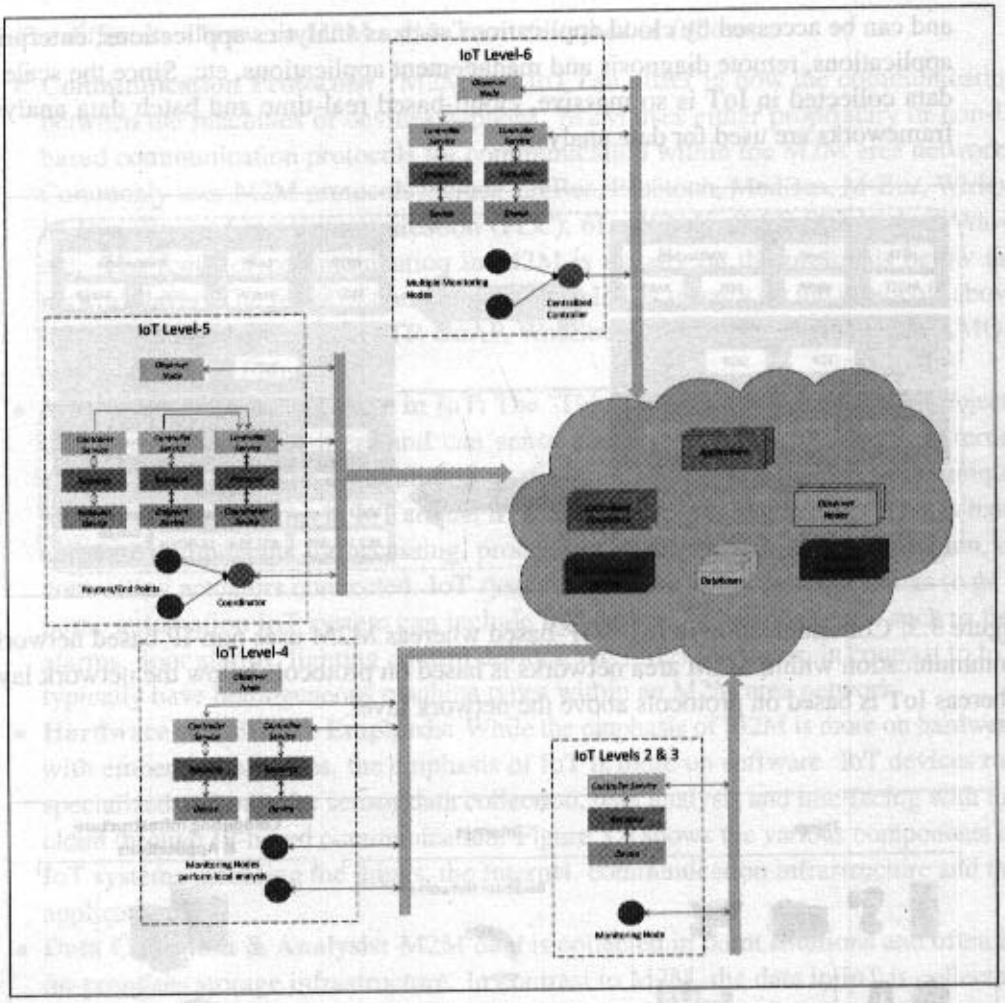


Figure 3.5: IoT levels and IoT cloud components

3.4 SDN and NFV for IoT

In this section you will learn about Software Defined Networking (SDN) and Network Function Virtualization (NFV) and their applications for IoT.

3.4.1 Software Defined Networking

Software-Defined Networking (SDN) is a networking architecture that separates the control plane from the data plane and centralizes the network controller. Figure 3.6 shows the conventional network architecture built with specialized hardware (switches, routers, etc.). Network devices in conventional network architectures are getting exceedingly complex with the increasing number of distributed protocols being implemented and the use of proprietary hardware and interfaces. In the conventional network architecture the control plane and data plane are coupled. Control plane is the part of the network that carries the signaling and routing message traffic while the data plane is the part of the network that carries the payload data traffic.

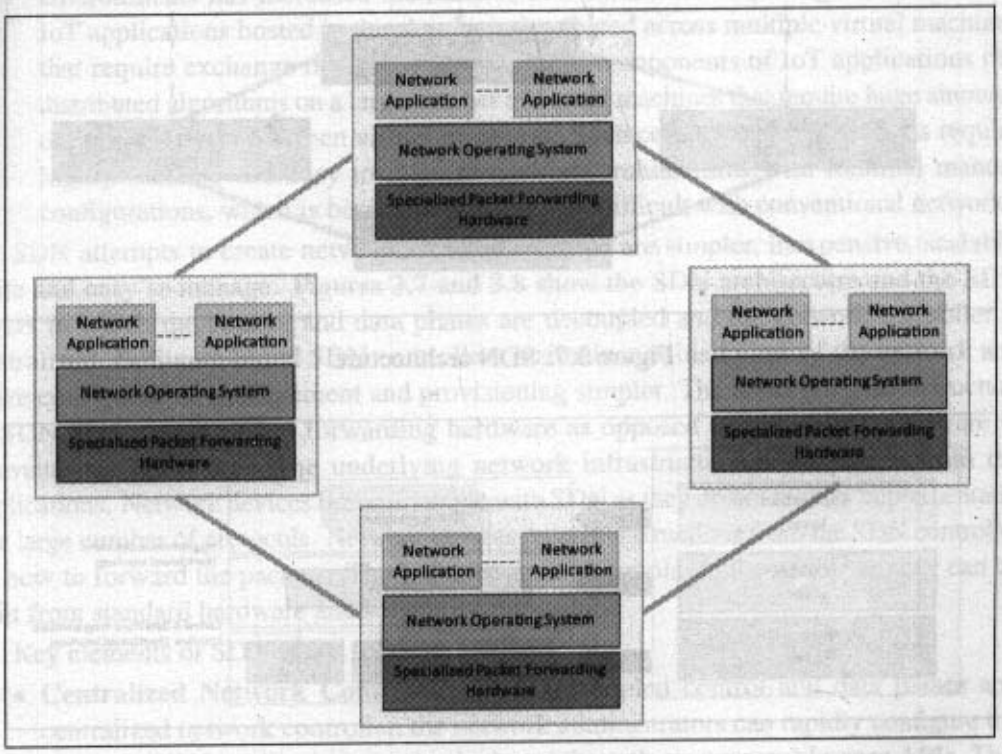


Figure 3.6: Conventional network architecture

The limitations of the conventional network architectures are as follows:

- **Complex Network Devices**: Conventional networks are getting increasingly complex with more and more protocols being implemented to improve link speeds and reliability.

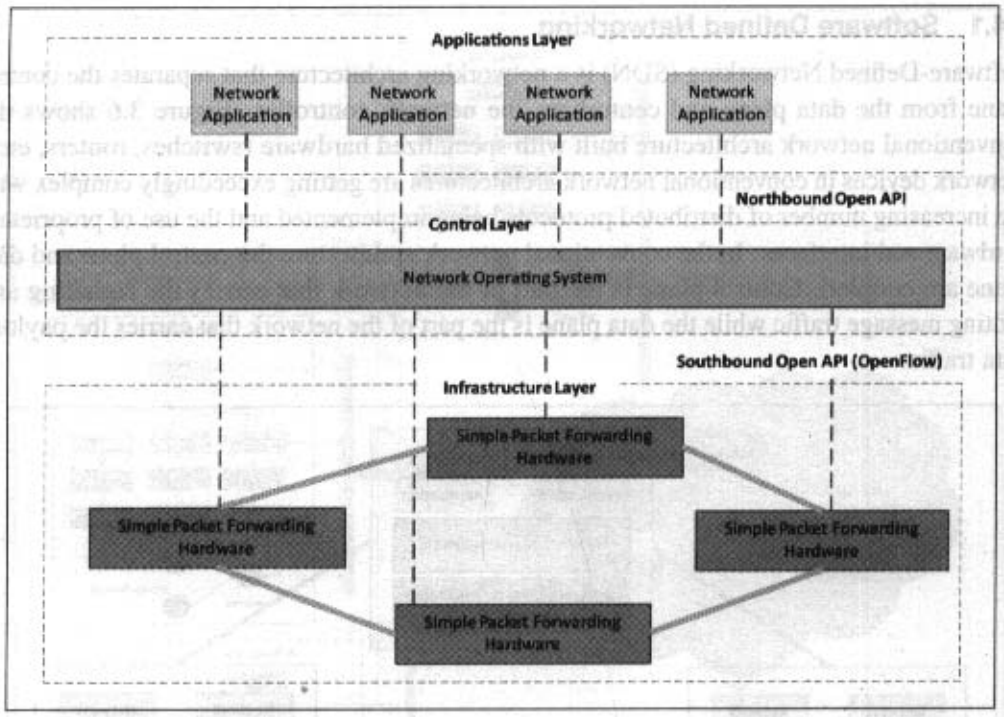


Figure 3.7: SDN architecture

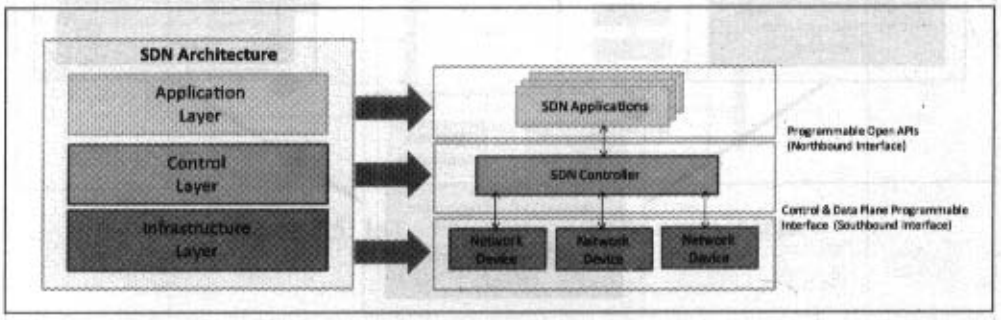


Figure 3.8: SDN layers

Interoperability is limited due to the lack of standard and open interfaces. Network devices use proprietary hardware and software and have slow product life-cycles limiting innovation. The conventional networks were well suited for static traffic

patterns and had a large number of protocols designed for specific applications. For IoT applications which are deployed in cloud computing environments, the traffic patterns are more dynamic. Due to the complexity of conventional network devices, making changes in the networks to meet the dynamic traffic patterns has become increasingly difficult.

- **Management Overhead**: Conventional networks involve significant management overhead. Network managers find it increasingly difficult to manage multiple network devices and interfaces from multiple vendors. Upgradation of network requires configuration changes in multiple devices (switches, routers, firewalls, etc.)
- **Limited Scalability**: The virtualization technologies used in cloud computing environments has increased the number of virtual hosts requiring network access. IoT applications hosted in the cloud are distributed across multiple virtual machines that require exchange of traffic. The analytics components of IoT applications run distributed algorithms on a large number of virtual machines that require huge amounts of data exchange between virtual machines. Such computing environments require highly scalable and easy to manage network architectures with minimal manual configurations, which is becoming increasingly difficult with conventional networks.

SDN attempts to create network architectures that are simpler, inexpensive, scalable, agile and easy to manage. Figures 3.7 and 3.8 show the SDN architecture and the SDN layers in which the control and data planes are decoupled and the network controller is centralized. Software-based SDN controllers maintain a unified view of the network and make configuration, management and provisioning simpler. The underlying infrastructure in SDN uses simple packet forwarding hardware as opposed to specialized hardware in conventional networks. The underlying network infrastructure is abstracted from the applications. Network devices become simple with SDN as they do not require implementations of a large number of protocols. Network devices receive instructions from the SDN controller on how to forward the packets. These devices can be simpler and cost less as they can be built from standard hardware and software components.

Key elements of SDN are as follows:

- **Centralized Network Controller**: With decoupled control and data planes and centralized network controller, the network administrators can rapidly configure the network. SDN applications can be deployed through programmable open APIs. This speeds up innovation as the network administrators no longer need to wait for the device vendors to embed new features in their proprietary hardware.
- **Programmable Open APIs**: SDN architecture supports programmable open APIs for interface between the SDN application and control layers (Northbound interface). With these open APIs various network services can be implemented, such as routing,

quality of service (QoS), access control, etc.

- **Standard Communication Interface (OpenFlow)**: SDN architecture uses a standard communication interface between the control and infrastructure layers (Southbound interface). OpenFlow, which is defined by the Open Networking Foundation (ONF) is the broadly accepted SDN protocol for the Southbound interface. With OpenFlow, the forwarding plane of the network devices can be directly accessed and manipulated. OpenFlow uses the concept of flows to identify network traffic based on pre-defined match rules. Flows can be programmed statically or dynamically by the SDN control software. Figure 3.9 shows the components of an OpenFlow switch comprising of one or more flow tables and a group table, which perform packet lookups and forwarding, and OpenFlow channel to an external controller. OpenFlow protocol is implemented on both sides of the interface between the controller and the network devices. The controller manages the switch via the OpenFlow switch protocol. The controller can add, update, and delete flow entries in flow tables. Figure 3.10 shows an example of an OpenFlow flow table. Each flow table contains a set of flow entries. Each flow entry consists of match fields, counters, and a set of instructions to apply to matching packets. Matching starts at the first flow table and may continue to additional flow tables of the pipeline [83].

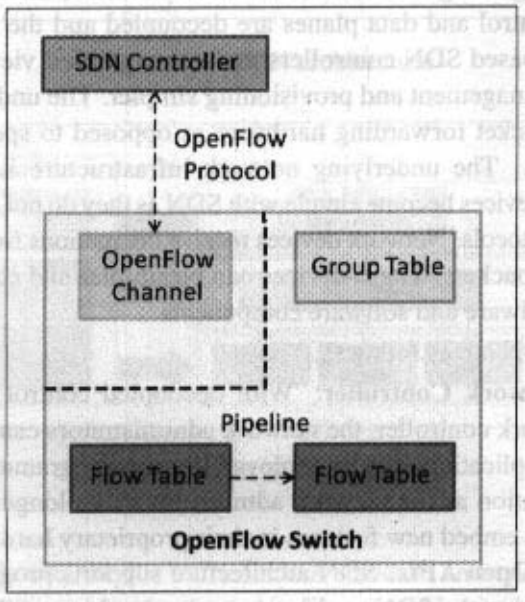


Figure 3.9: OpenFlow switch

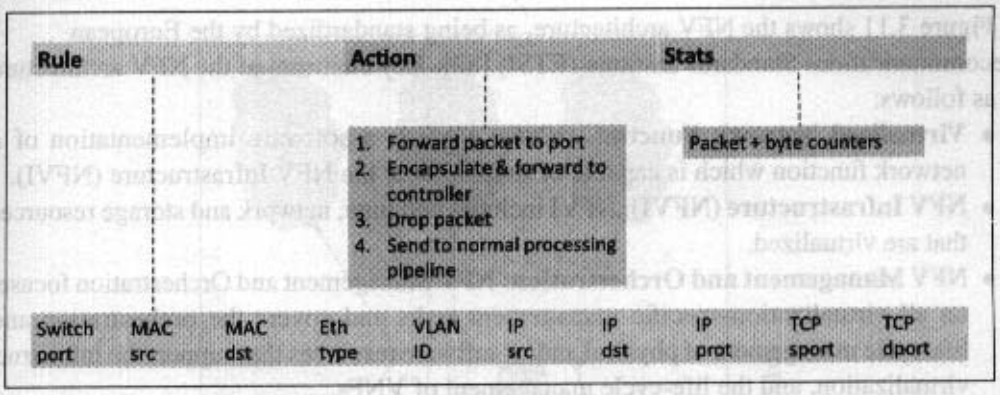


Figure 3.10: OpenFlow flow table

3.4.2 Network Function Virtualization

Network Function Virtualization (NFV) is a technology that leverages virtualization to consolidate the heterogeneous network devices onto industry standard high volume servers, switches and storage. NFV is complementary to SDN as NFV can provide the infrastructure on which SDN can run. NFV and SDN are mutually beneficial to each other but not dependent. Network functions can be virtualized without SDN, similarly, SDN can run without NFV.

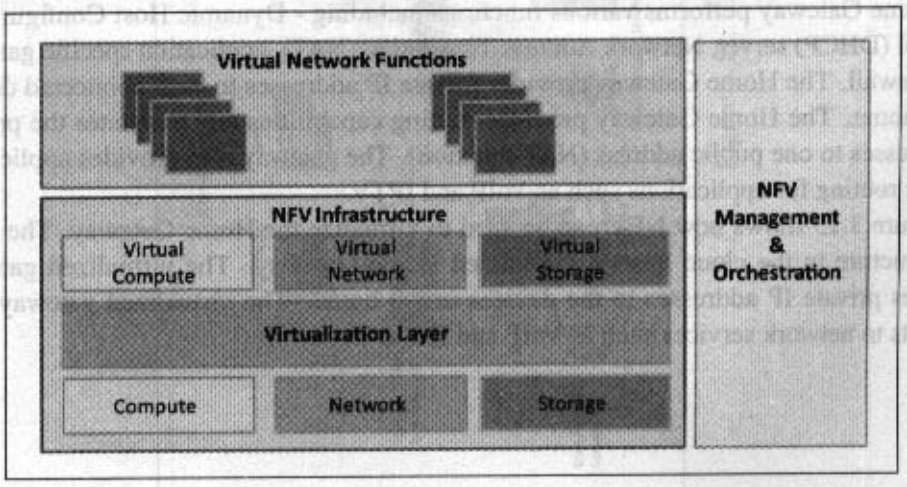


Figure 3.11: NFV architecture

Figure 3.11 shows the NFV architecture, as being standardized by the European Telecommunications Standards Institute (ETSI) [82]. Key elements of the NFV architecture are as follows:

- **Virtualized Network Function (VNF)**: VNF is a software implementation of a network function which is capable of running over the NFV Infrastructure (NFVI).
- **NFV Infrastructure (NFVI)**: NFVI includes compute, network and storage resources that are virtualized.
- **NFV Management and Orchestration**: NFV Management and Orchestration focuses on all virtualization-specific management tasks and covers the orchestration and life-cycle management of physical and/or software resources that support the infrastructure virtualization, and the life-cycle management of VNFs.

NFV comprises of network functions implemented in software that run on virtualized resources in the cloud. NFV enables separation of network functions which are implemented in software from the underlying hardware. Thus network functions can be easily tested and upgraded by installing new software while the hardware remains the same. Virtualizing network functions reduces the equipment costs and also reduces power consumption. The multi-tenanted nature of the cloud allows virtualized network functions to be shared for multiple network services. NFV is applicable only to data plane and control plane functions in fixed and mobile networks.

Let us look at an example of how NFV can be used for virtualization of the home networks. Figure 3.12 shows a home network with a Home Gateway that provides Wide Area Network (WAN) connectivity to enable services such as Internet, IPTV, VoIP, etc. The Home Gateway performs various functions including - Dynamic Host Configuration Protocol (DHCP) server, Network Address Translation (NAT), application specific gateway and Firewall. The Home Gateway provides private IP addresses to each connected device in the home. The Home Gateway provides routing capabilities and translates the private IP addresses to one public address (NAT function). The gateway also provides application specific routing for applications such as VoIP and IPTV.

Figure 3.13 shows how NFV can be used to virtualize the Home Gateway. The NFV infrastructure in the cloud hosts a virtualized Home Gateway. The virtualized gateway provides private IP addresses to the devices in the home. The virtualized gateway also connects to network services such as VoIP and IPTV.

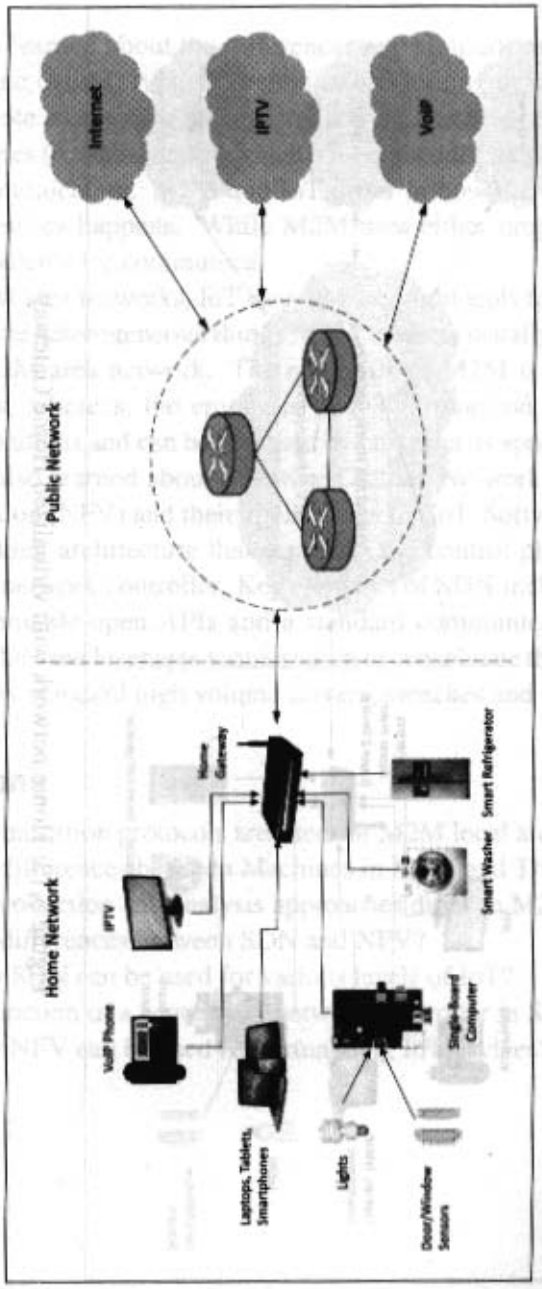


Figure 3.12: Conventional home network architecture

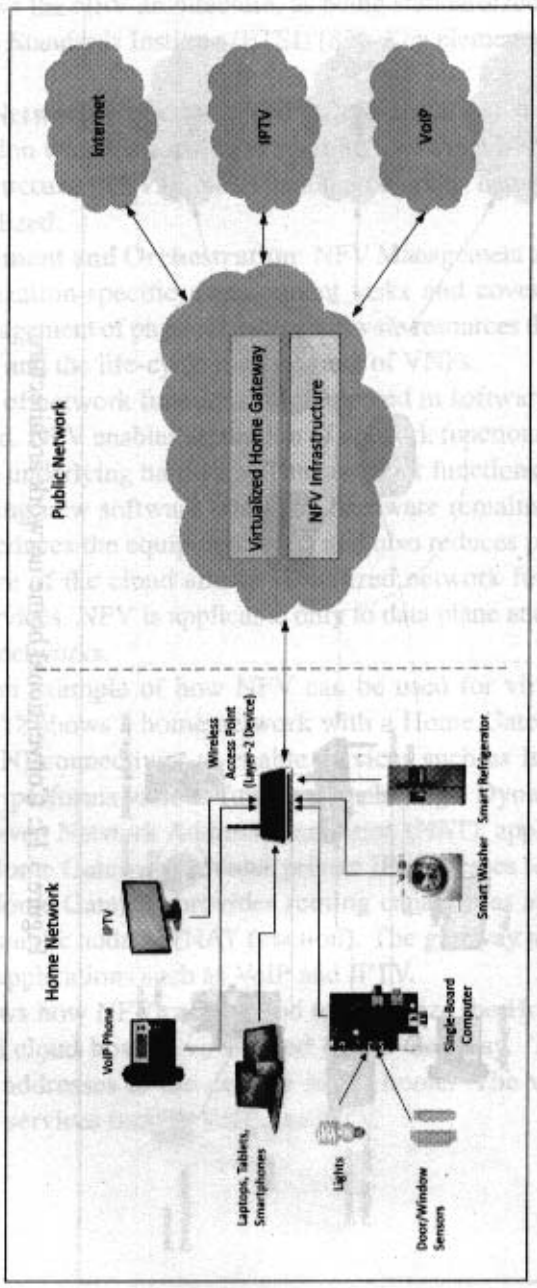
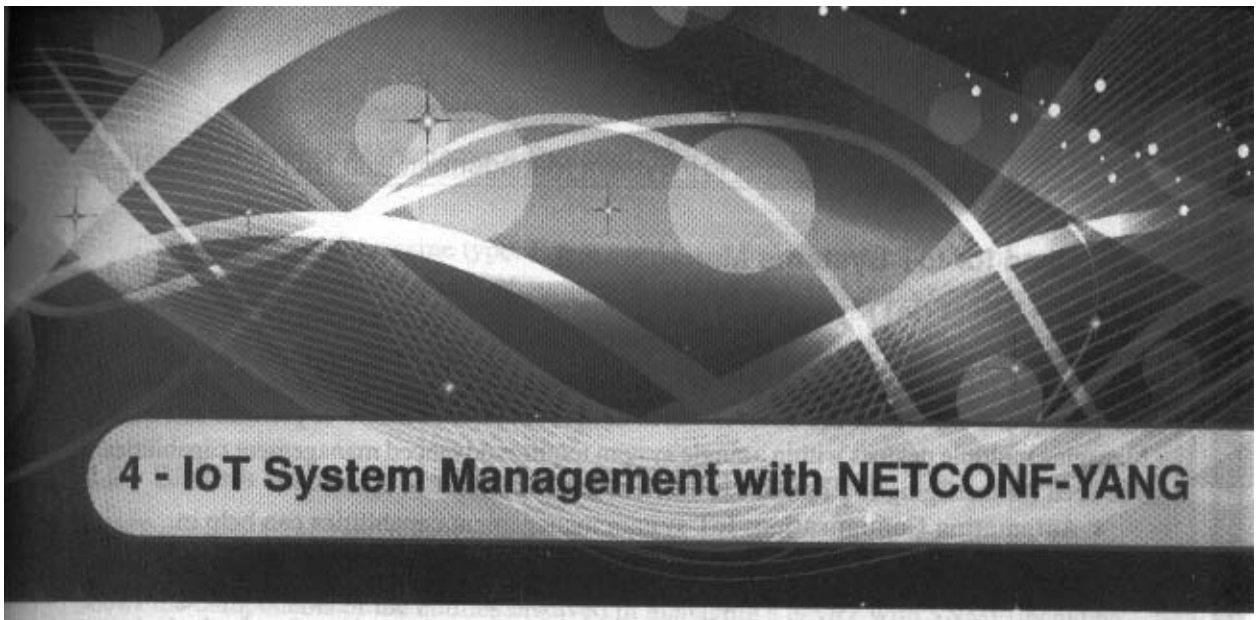


Figure 3.13: Home network with virtualized Home Gateway



4 - IoT System Management with NETCONF-YANG

This Chapter Covers

- Need for IoT Systems Management
- SNMP
- Network Operator Requirements
- NETCONF
- YANG
- IoT Systems Management with NETCONF-YANG

Summary

In this chapter you learned about the differences and similarities between IoT and M2M. Machine-to-Machine (M2M) typically refers to networking of machines (or devices) for the purpose of remote monitoring and control and data exchange. An M2M area network comprises of machines (or M2M nodes) which have embedded hardware modules for sensing, actuation and communication. M2M and IoT differ in how the communication between the machines or devices happens. While M2M uses either proprietary or non-IP based communication protocols for communication within the M2M area networks, IoT uses IP-based protocols for communication. While IoT systems can have heterogeneous things M2M systems usually have the same machine types within an M2M area network. The emphasis of M2M is more on hardware with embedded modules, whereas, the emphasis of IoT is more on software. M2M data is collected in point solutions and can be accessed by on-premisis applications. IoT is collected in the cloud. You also learned about Software Defined Networking (SDN) and Network Function Virtualization (NFV) and their applications for IoT. Software-Defined Networking (SDN) is a networking architecture that separates the control plane from the data plane and centralizes the network controller. Key elements of SDN include centralized network controller, programmable open APIs and a standard communication interface. NFV is complementary to SDN and leverages virtualization to consolidate the heterogeneous network devices onto industry standard high volume servers, switches and storage.

Review Questions

1. Which communication protocols are used for M2M local area networks?
2. What are the differences between Machines in M2M and Things in IoT?
3. How do data collection and analysis approaches differ in M2M and IoT?
4. What are the differences between SDN and NFV?
5. Describe how SDN can be used for various levels of IoT?
6. What is the function of a centralized network controller in SDN?
7. Describe how NFV can be used for virtualizing IoT devices?

4.1 Need for IoT Systems Management

Internet of Things (IoT) systems can have complex software, hardware and deployment designs including sensors, actuators, software and network resources, data collection and analysis services and user interfaces. IoT systems can have distributed deployments comprising of a number of IoT devices which collect data from sensors or perform actuation. Managing multiple devices within a single system requires advanced management capabilities. The need for managing IoT systems is described as follows:

- **Automating Configuration**: IoT system management capabilities can help in automating the system configurations. System management interfaces provide predictable and easy to use management capability and the ability to automate system configuration. Automation becomes even more important when a system consists of multiple devices or nodes. In such cases automating the system configuration ensures that all devices have the same configuration and variations or errors due to manual configurations are avoided.
- **Monitoring Operational & Statistical Data**: Operational data is the data which is related to the system's operating parameters and is collected by the system at runtime. Statistical data is the data which describes the system performance (e.g. CPU and memory usage). Management systems can help in monitoring operational and statistical data of a system. This data can be used for fault diagnosis or prognosis.
- **Improved Reliability**: A management system that allows validating the system configurations before they are put into effect can help in improving the system reliability.
- **System Wide Configuration**: For IoT systems that consist of multiple devices or nodes, ensuring system-wide configuration can be critical for the correct functioning of the system. Management approaches in which each device is configured separately (either through a manual or automated process) can result in system faults or undesirable outcomes. This happens when some devices are running on an old configuration while others start running on new configuration. To avoid this, system wide configuration is required where all devices are configured in a single atomic transaction. This ensures that the configuration changes are either applied to all devices or to none. In the event of a failure in applying the configuration to one or more devices, the configuration changes are rolled back. This 'all or nothing' approach ensures that the system works as expected.
- **Multiple System Configurations**: For some systems it may be desirable to have multiple valid configurations which are applied at different times or in certain conditions.
- **Retrieving & Reusing Configurations**: Management systems which have the capability of retrieving configurations from devices can help in reusing the configurations for

other devices of the same type. For example, for an IoT system which has multiple devices and requires same configuration for all devices, it is important to ensure that when a new device is added, the same configuration is applied. For such cases, the management system can retrieve the current configuration from a device and apply the same to the new devices.

4.2 Simple Network Management Protocol (SNMP)

SNMP is a well-known and widely used network management protocol that allows monitoring and configuring network devices such as routers, switches, servers, printers, etc. Figure 4.1 shows the components of the entities involved in managing a device with SNMP, including the Network Management Station (NMS), Managed Device, Management Information Base (MIB) and the SNMP Agent that runs on the device. NMS executes SNMP commands to monitor and configure the Managed Device. The Managed Device contains the MIB which has all the information of the device attributes to be managed. MIBs use the Structure of Management Information (SMI) notation for defining the structure of the management data. The structure of management data is defined in the form of variables which are identified by object identifiers (OIDs), which have a hierarchical structure. Management applications can either get or set the values of these variables. SNMP is an application layer protocol that uses User Datagram Protocol (UDP) as the transport protocol.

4.2.1 Limitations of SNMP

While Simple Network Management Protocol (SNMP) has been the most popular protocol for network management, it has several limitations which may make it unsuitable for configuration management.

- SNMP was designed to provide a simple management interface between the management applications and the managed devices. SNMP is stateless in nature and each SNMP request contains all the information to process the request. The application needs to be intelligent to manage the device. For a sequence of SNMP interactions, the application needs to maintain state and also to be smart enough to roll back the device into a consistent state in case of errors or failures in configuration.
- SNMP is a connectionless protocol which uses UDP as the transport protocol, making it unreliable as there was no support for acknowledgement of requests.
- MIBs often lack writable objects without which device configuration is not possible using SNMP. With the absence of writable objects, SNMP can be used only for device monitoring and status polling.
- It is difficult to differentiate between configuration and state data in MIBs.

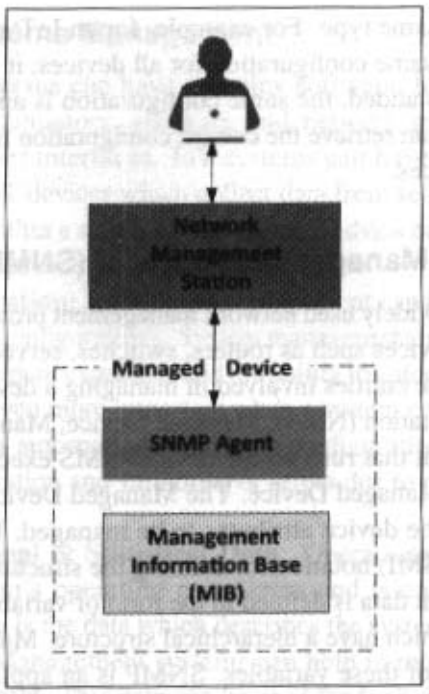


Figure 4.1: Managing a device with SNMP

- Retrieving the current configuration from a device can be difficult with SNMP. SNMP does not support easy retrieval and playback of configurations.
- Earlier versions of SNMP did not have strong security features making the management information vulnerable to network intruders. Though security features were added in the later versions of SNMP, it increased the complexity a lot.

4.3 Network Operator Requirements

To address the limitations of the existing network management protocols and plan the future work on network management, the Internet Architecture Board (IAB), which oversees the Internet Engineering Task Force (IETF) held a workshop on network management in 2002 that brought together network operators and protocol developers. Based on the inputs from operators, a list of operator requirements was prepared [122]. The following points provide a brief overview of the operator requirements.

- **Ease of use**: From the operators point of view, ease of use is the key requirement for

any network management technology.

- **Distinction between configuration and state data**: Configuration data is the set of writable data that is required to transform the system from its initial state to its current state. State data is the data which is not configurable. State data includes operational data which is collected by the system at runtime and statistical data which describes the system performance. For an effective management solution, it is important to make a clear distinction between configuration and state data.
- **Fetch configuration and state data separately**: In addition to making a clear distinction between configuration and state data, it should be possible to fetch the configuration and state data separately from the managed device. This is useful when the configuration and state data from different devices needs to be compared.
- **Configuration of the network as a whole**: It should be possible for operators to configure the network as a whole rather than individual devices. This is important for systems which have multiple devices and configuring them within one network wide transaction is required to ensure the correct operation of the system.
- **Configuration transactions across devices**: Configuration transactions across multiple devices should be supported.
- **Configuration deltas**: It should be possible to generate the operations necessary for going from one configuration state to another. The devices should support configuration deltas with minimal state changes.
- **Dump and restore configurations**: It should be possible to dump configurations from devices and restore configurations to devices.
- **Configuration validation**: It should be possible to validate configurations.
- **Configuration database schemas**: There is a need for standardized configuration database schemas or data models across operators.
- **Comparing configurations**: Devices should not arbitrarily reorder data, so that it is possible to use text processing tools such as $diff$ to compare configurations.
- **Role-based access control**: Devices should support role-based access control model, so that a user is given the minimum access necessary to perform a required task.
- **Consistency of access control lists**: It should be possible to do consistency checks of access control lists across devices.
- **Multiple configuration sets**: There should be support for multiple configurations sets on devices. This way a distinction can be provided between candidate and active configurations.
- **Support for both data-oriented and task-oriented access control**: While SNMP access control is data-oriented, CLI access control is usually task oriented. There should be support for both types of access control.

the role of a NETCONF client. For managing a network device the client establishes a NETCONF session with the server. When a session is established the client and server exchange 'hello' messages which contain information on their capabilities. Client can then send multiple requests to the server for retrieving or editing the configuration data. NETCONF allows the management client to discover the capabilities of the server (on the device). NETCONF gives access to the native capabilities of the device.

NETCONF defines one or more configuration datastores. A configuration store contains all the configuration information to bring the device from its initial state to the operational state. By default a <running> configuration store is present. Additional configuration datastores such as <startup> and <candidate> can be defined in the capabilities.

NETCONF is a connection oriented protocol and NETCONF connection persists between protocol operations. For authentication, data integrity, and confidentiality, NETCONF depends on the transport protocol, e.g., SSH or TLS. NETCONF overcomes the limitations of SNMP and is suitable not only for monitoring state information, but also for configuration management.

4.5 YANG

YANG is a data modeling language used to model configuration and state data manipulated by the NETCONF protocol [137, 124]. YANG modules contain the definitions of the configuration data, state data, RPC calls that can be issued and the format of the notifications. YANG modules defines the data exchanged between the NETCONF client and server. A module comprises of a number of 'leaf' nodes which are organized into a hierarchical tree structure. The 'leaf' nodes are specified using the 'leaf' or 'leaf-list' constructs. Leaf nodes are organized using 'container' or 'list' constructs. A YANG module can import definitions from other modules. Constraints can be defined on the data nodes, e.g. allowed values. YANG can model both configuration data and state data using the 'config' statement. YANG defines four types of nodes for data modeling as shown in Table 4.2.

Let us now look at an example of a YANG module. Box 4.1 shows a YANG module for a "network-enabled toaster". This YANG module is a YANG version of the toaster Management Information Base (MIB). We use the Toaster MIB since it has been widely used as an example in introductory tutorials on SNMP to explain how SNMP can be used for managing a network-connected toaster. A YANG module has several sections starting from header information, followed by imports and includes, type definitions, configuration and operational data declarations, and RPC and notification declarations. The toaster YANG module begins with the header information followed by identity declarations which define various bread types. The leaf nodes ('toasterManufacturer', 'toasterModelNumber' and

Operation	Description
connect	Connect to a NETCONF server
get	Retrieve the running configuration and state information
get-config	Retrieve all or a portion of a configuration datastore
edit-config	Loads all or part of a specified configuration to the specified target configuration
copy-config	Create or replace an entire target configuration datastore with a complete source configuration
delete-config	Delete the contents of a configuration datastore
lock	Lock a configuration datastore for exclusive edits by a client
unlock	Release the lock on a configuration datastore
get-schema	This operation is used to retrieve a schema from the NETCONF server
commit	Commit the candidate configuration as the device's new current configuration
close-session	Gracefully terminate a NETCONF session
kill-session	Forcefully terminate a NETCONF session

Table 4.1: List of commonly used NETCONF RPC methods

'toasterStatus') are defined in the 'toaster' container. Each leaf node definition has a type and optionally a description and default value. The module has two RPC definitions ('make-toast' and 'cancel-toast'). A tree representation of the toaster YANG module is shown in Figure 4.3.

Box 4.1: YANG version of the Toaster-MIB

```
module toaster {

  yang-version 1;

  namespace
   "http://example.com/ns/toaster";

  prefix toast;
```

Node Type	Description
Leaf Nodes	Contains simple data structures such as an integer or a string. Leaf has exactly one value of a particular type and no child nodes.
Leaf-List Nodes	Is a sequence of leaf nodes with exactly one value of a particular type per leaf.
Container Nodes	Used to group related nodes in a subtree. A container has only child nodes and no value. A container may contain any number of child nodes of any type (including leafs, lists, containers, and leaf-lists).
List Nodes	Defines a sequence of list entries. Each entry is like a structure or a record instance, and is uniquely identified by the values of its key leafs. A list can define multiple key leafs and may contain any number of child nodes of any type.

Table 4.2: YANG Node Types

```
revision "2009-11-20" {
 description
   "Toaster module";
}

identity toast-type {
 description
   "Base for all bread types";
}

identity white-bread {
 base toast:toast-type;
}

identity wheat-bread {
 base toast-type;
}

identity wonder-bread {
 base toast-type;
}

identity frozen-waffle {
```

```
 base toast-type;
}

identity frozen-bagel {
 base toast-type;
}

identity hash-brown {
 base toast-type;
}

typedef DisplayString {
 type string {
   length "0 ..  255";
 }
}

container toaster {
 presence
   "Indicates the toaster service is available";
 description
   "Top-level container for all toaster database objects.";
 leaf toasterManufacturer {
   type DisplayString;
   config false;
   mandatory true;
 }

 leaf toasterModelNumber {
   type DisplayString;
   config false;
   mandatory true;
 }

 leaf toasterStatus {
   type enumeration {
    enum "up" {
      value 1;
    }
    enum "down" {
      value 2;
    }
   }
   config false;
   mandatory true;
```

```
    }
  }

  rpc make-toast {
   input {
     leaf toasterDoneness {
      type uint32 {
        range "1 ..  10";
      }
      default '5';
     }

     leaf toasterToastType {
      type identityref {
        base toast:toast-type;
      }
      default 'wheat-bread';
     }
   }
  }

  rpc cancel-toast {
   description
     "Stop making toast, if any is being made.";
  }

  notification toastDone {
   leaf toastStatus {
     type enumeration {
      enum "done" {
        value 0;
        description "The toast is done.";
      }
      enum "cancelled" {
        value 1;
        description
          "The toast was cancelled.";
      }
      enum "error" {
        value 2;
        description
          "The toaster service was disabled or
               the toaster is broken.";
      }
     }
```

```
    }
  }
}
```

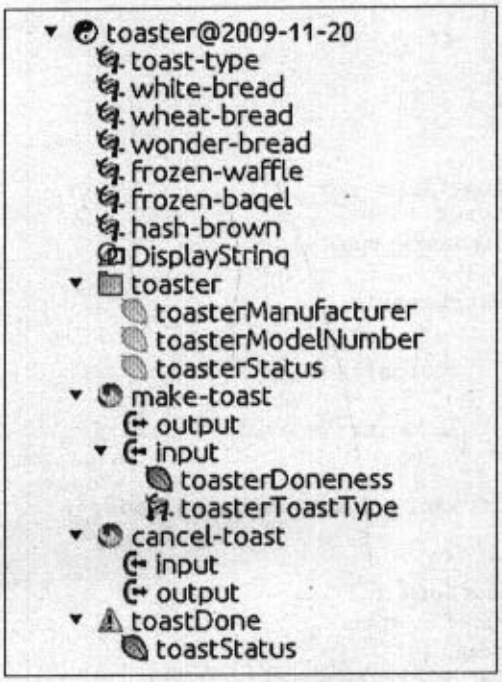


Figure 4.3: Visual representation of the Toaster YANG Module

Let us look at another example of a YANG module. Box 4.2 shows a YANG module for configuring a HAProxy load balancer for a commercial website. The module includes containers for global, defaults, frontend and backend sections of an HAProxy configuration. In the global container the leaf nodes for configuration data such as max-connections and mode are defined. In the defaults container the leaf nodes for configuration data such as retries, *contimeout*, etc. are defined. The front-end port bindings are defined in the frontend container. The backend container has definitions on the servers to load balance. A reusable tree structure called 'server-list' is used for the server definitions. The 'server-list' structure is defined using the 'grouping' construct in the module. A tree representation of the HAProxy YANG module is shown in Figure 4.4.

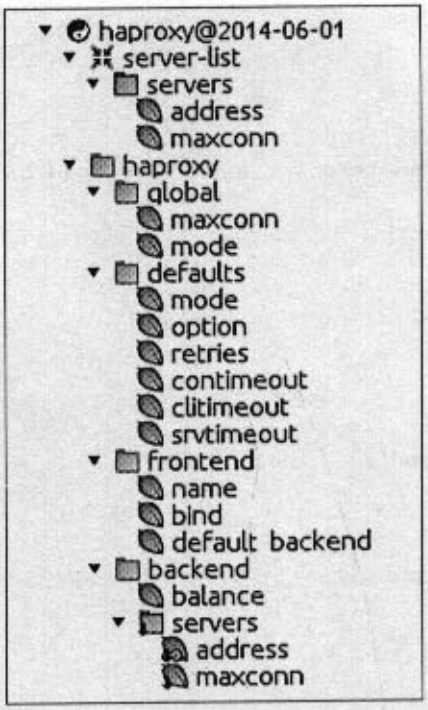


Figure 4.4: Visual representation of HAProxy YANG Module

Box 4.2: YANG Module for HAProxy configuration

```
module haproxy {
   yang-version 1;

   namespace "http://example.com/ns/haproxy";

   prefix haproxy;

   import ietf-inet-types { prefix inet; }

   description
   "YANG version of the ";

   revision "2014-06-01" {
      description
```

```
      "HAProxy module";
   }

container haproxy {
   description
   "configuration parameters for a HAProxy load balancer";

      container global {
         leaf maxconn {
         type uint32;
         default 4096;
         }

         leaf mode {
         type string;
         default 'daemon';
         }
      }

      container defaults {
         leaf mode {
         type string;
         default 'http';
         }

         leaf option {
         type string;
         default 'redispatch';
         }

         leaf retries {
         type uint32;
         default 3;
         }

         leaf contimeout {
         type uint32;
         default 5000;
         }

         leaf clitimeout {
         type uint32;
         default 50000;
         }
```

```
        leaf srvtimeout {
        type uint32;
        default 50000;
        }
    }

    container frontend {

        leaf name {
        type string;
        default 'http-in';
        }

        leaf bind {
        type string;
        default '*:80';
        }

        leaf default_backend {
        type string;
        default 'webfarm';
        }
    }

    container backend {
        leaf balance   {
        type string;
        default 'roundrobin';
        }
        uses server-list;
    }
}

    grouping server-list {
    description "List of servers to load balance";
        list servers {
        key address;
        leaf address {
            type inet:ip-address;
        }

        leaf maxconn {
            type uint32;
            default 255;
```

```
            }
        }
    }
}
```

4.6 IoT Systems Management with NETCONF-YANG

In this section you will learn how to manage IoT systems with NECONF and YANG. Figure 4.5 shows the generic approach of IoT device management with NETCONF-YANG.

Let is look at the roles of the various components:

- **Management System**: The operator uses a Management System to send NETCONF messages to configure the IoT device and receives state information and notifications from the device as NETCONF messages.
- **Management API**: Management API allows management applications to start NETCONF sessions, read and write configuration data, read state data, retrieve configurations, and invoke RPCs, programmatically, in the same way as an operator can.
- **Transaction Manager**: Transaction Manager executes all the NETCONF transactions and ensures that the ACID (Atomicity, Consistency, Isolation, Durability) properties hold true for the transactions. Atomicity property ensures that a transaction is executed either completely or not at all. Consistency property ensures that a transaction brings the device configuration from one valid state to another. Isolation property ensures that concurrent execution of transactions results in the same device configuration as if transactions were executed serially in order. Durability property ensures that a transaction once committed will persist.
- **Rollback Manager** : Rollback manager is responsible for generating all the transactions necessary to rollback a current configuration to its original state.
- **Data Model Manager**: The Data Model manager keeps track of all the YANG data models and the corresponding managed objects. The Data Model manager also keeps track of the applications which provide data for each part of a data model.
- **Configuration Validator**: Configuration validator checks if the resulting configuration after applying a transaction would be a valid configuration.
- **Configuration Database**: This database contains both the configuration and operational data.
- **Configuration API**: Using the configuration API the applications on the IoT device can read configuration data from the configuration datastore and write operational data to the operational datastore.

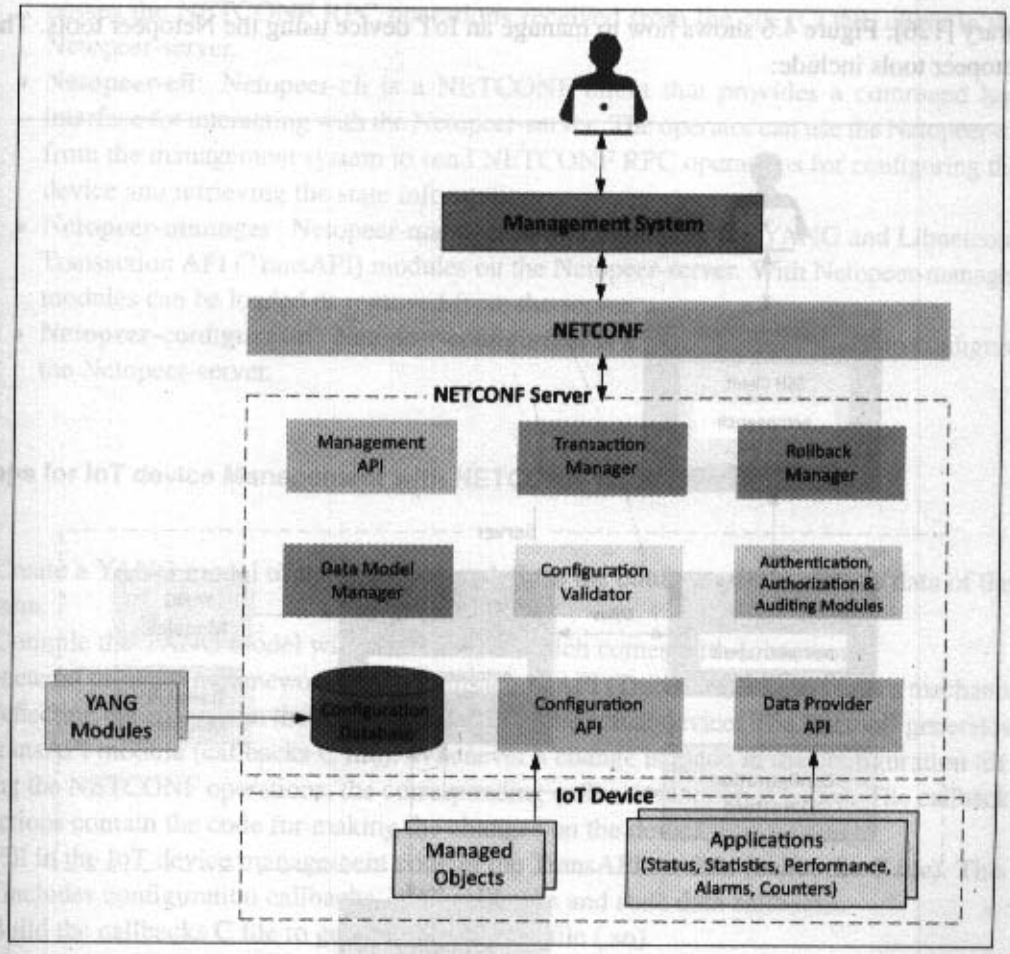


Figure 4.5: IoT device management with NETCONF-YANG - a generic approach

- **Data Provider API**: Applications on the IoT device can register for callbacks for various events using the Data Provider API. Through the Data Provider API, the applications can report statistics and operational data.

4.6.1 NETOPEER

While the previous section described a generic approach of IoT device management with NETCONF-YANG, this section describes a specific implementation based on the Netopeer tools [125]. Netopeer is set of open source NETCONF tools built on the Libnetconf

library [126]. Figure 4.6 shows how to manage an IoT device using the Netopeer tools. The Netopeer tools include:

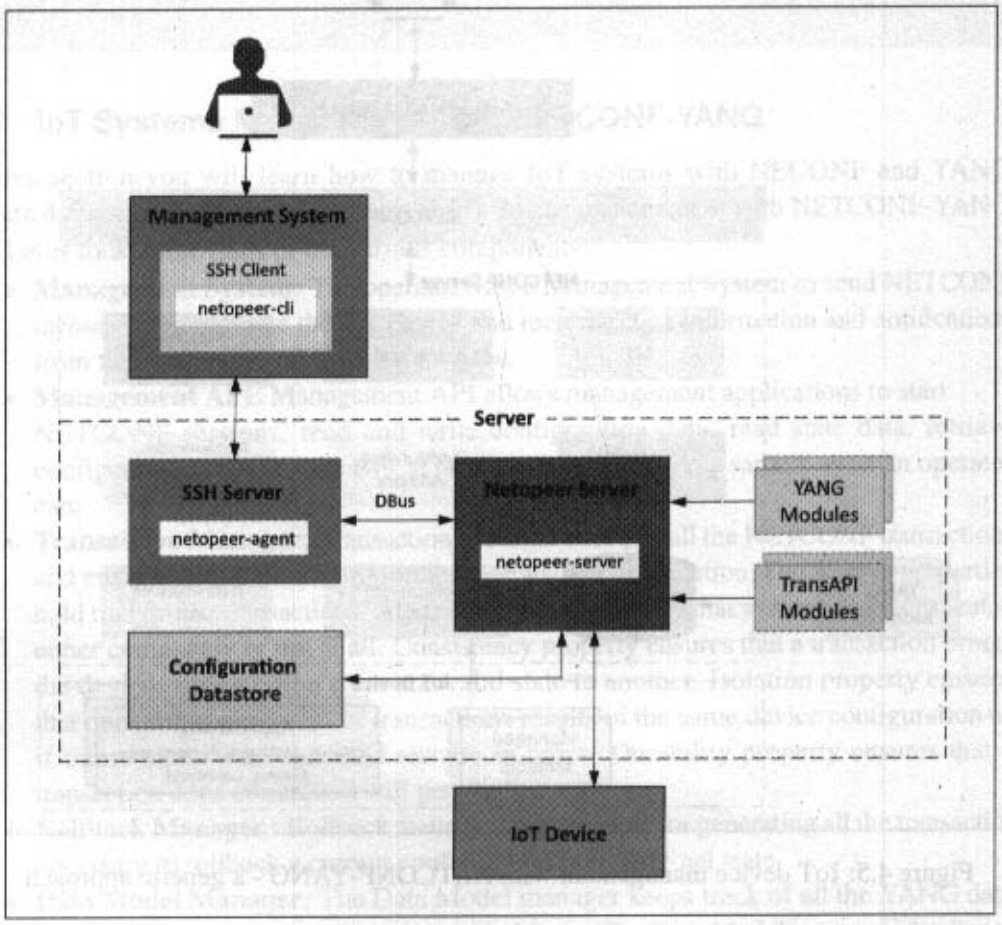


Figure 4.6: IoT device management with NETCONF - a specific approach based on Netopeer tools

- **Netopeer-server**: Netopeer-server is a NETCONF protocol server that runs on the managed device. Netopeer-server provides an environment for configuring the device using NETCONF RPC operations and also retrieving the state data from the device.
- **Netopeer-agent**: Netopeer-agent is the NETCONF protocol agent running as a SSH/TLS subsystem. Netopeer-agent accepts incoming NETCONF connection and

passes the NETCONF RPC operations received from the NETCONF client to the Netopeer-server.

- **Netopeer-cli**: Netopeer-cli is a NETCONF client that provides a command line interface for interacting with the Netopeer-server. The operator can use the Netopeer-cli from the management system to send NETCONF RPC operations for configuring the device and retrieving the state information.
- **Netopeer-manager**: Netopeer-manager allows managing the YANG and Libnetconf Transaction API (TransAPI) modules on the Netopeer-server. With Netopeer-manager modules can be loaded or removed from the server.
- **Netopeer-configurator**: Netopeer-configurator is a tool that can be used to configure the Netopeer-server.

Steps for IoT device Management with NETCONF-YANG

1. Create a YANG model of the system that defines the configuration and state data of the system.
2. Compile the YANG model with the 'lnctool' which comes with Libnetconf.
Libnetconf provides a framework called Transaction API (TransAPI) that provides a mechanism of reflecting the changes in the configuration file in the actual device. The 'lnctool' generates a TransAPI module (callbacks C file). Whenever a change is made in the configuration file using the NETCONF operations, the corresponding callback function is called. The callback functions contain the code for making the changes on the device.
3. Fill in the IoT device management code in the TransAPI module (callbacks C file). This file includes configuration callbacks, RPC callbacks and state data callbacks.
4. Build the callbacks C file to generate the library file (.so).
5. Load the YANG module (containing the data definitions) and the TransAPI module (.so binary) into the Netopeer server using the Netopeer manager tool.
6. The operator can now connect from the management system to the Netopeer server using the Netopeer CLI.
7. Operator can issue NETCONF commands from the Netopeer CLI. Commands can be issued to change the configuration data, get operational data or execute an RPC on the IoT device.

In Chapter 11, detailed case studies on IoT device Management using the above steps are provided.

Summary

In this chapter you learned about the need for IoT systems management. IoT system management capabilities can help in automating the system configurations. Management systems can collect operational and statistical data from IoT devices which can be used for fault diagnosis or prognosis. For IoT systems that consist of multiple devices, system wide configuration is important to ensure that all devices are configured within one transaction and the transactions are atomic. It is desirable for devices to have multiple configurations with one of them being the active and running configuration. Management systems which have the capability of retrieving configurations from devices can help in reusing the configurations for other devices of the same type. SNMP has been a popular network management protocol, however it has several limitations which make it unsuitable for IoT device management. Network Configuration Protocol (NETCONF), which is a session-based network management protocol, is more suitable for IoT device management. NETCONF works on SSH transport protocol and provides various operations to retrieve and edit configuration data from devices. The configuration data resides within a NETCONF configuration datastore on the server. The NETCONF server resides on the network device. The device configuration and state data is modeled using the YANG data modeling language. There is a clear separation of configuration and state data in the YANG models. You learned about a generic approach for IoT device management and the roles of various components such as the Management API, Transaction Manager, Rollback Manager, Data Model Manager, Configuration Validator, Configuration Database, Configuration API and Data Provider API. You learned about the Netopeer tools for NETCONF and the steps for IoT device Management using these tools.

Review Questions

1. Why is network wide configuration important for IoT systems with multiple nodes?
2. Which limitations make SNMP unsuitable for IoT systems?
3. What is the difference between configuration and state data?
4. What is the role of a NETCONF server?
5. What is the function of a data model manager?
6. Describe the roles of YANG and TransAPI modules in device management?