

MODULE-2

IOT and M2M

Introduction: M2M, Difference between IoT and M2M, SDN and NFV for IOT, IOT System Management with NETCONF-YANG, Need for IOT Systems Management, Simple Network Management Protocol (SNMP), Network operator requirements, NETCONF, YANG, IoT Systems Management with NETCONF-YANG.

Textbook: Arshdeep Bahga, Vijay Madisetti, “ Internet of Things- A Hands On Approach”, Universities press, 2014.

Ch. 3.1-3.4,4.1-4.6

2.1 M2M:

Machine-to-Machine (M2M) refers to networking of machines (or devices) for the purpose of remote monitoring and control and data exchange. Figure 2.1.1 shows the end-to-end architecture for M2M systems. It comprising of M2M area networks, communication network and application domain.

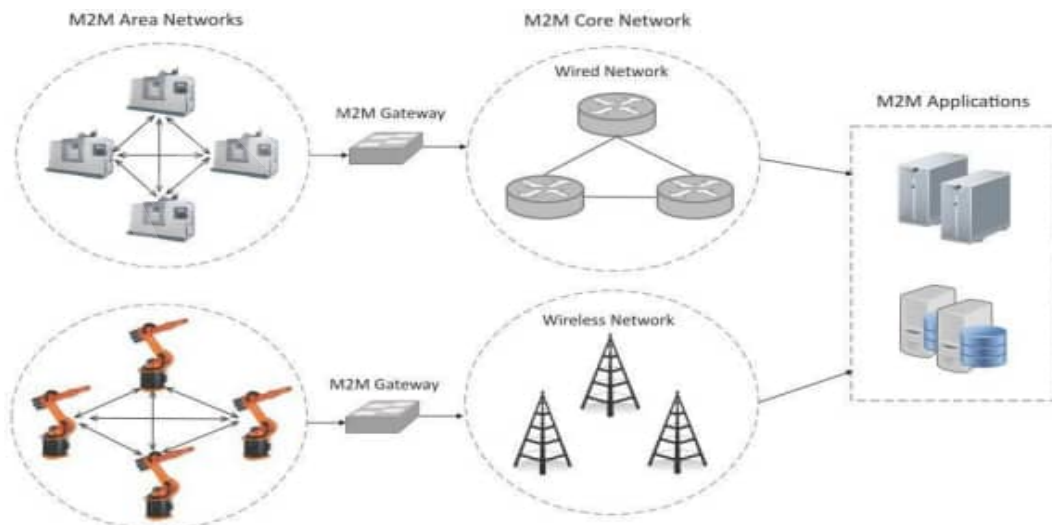


Figure 2.1.1 M2M system architecture

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, Bluetooth, ModBus, M-Bus, Wireless 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.

M2M gateways

Figure 2.1.2 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.

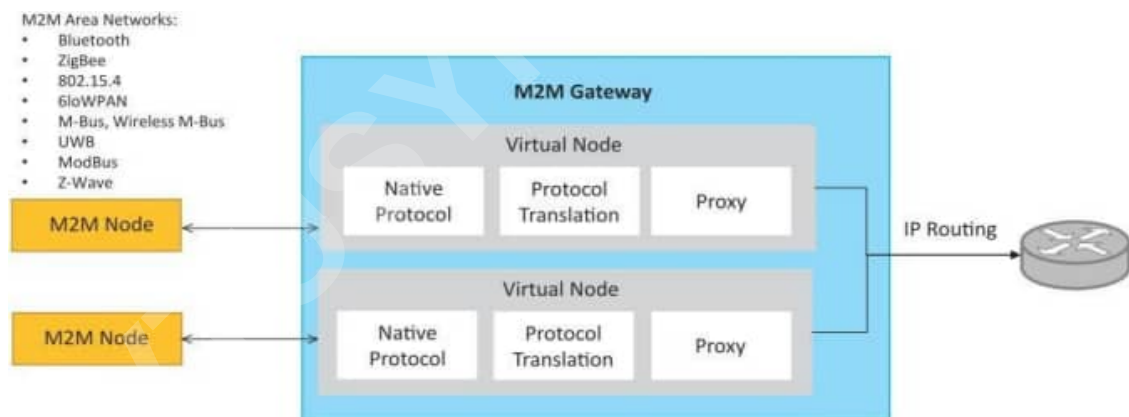


Figure 2.1.2 Block diagram of an M2M gateway

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.

Difference between IoT and M2M

- 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:

1) Communication Protocols:

Aspect	M2M (Machine-to-Machine)	IoT (Internet of Things)
Communication Basis	Proprietary or non-IP based protocols	IP-based protocols
Protocol Layer Focus	Focus below the network layer	Focus above the network layer
Common Protocols	ZigBee, Bluetooth, ModBus, M-Bus, Wireless M-Bus, Power Line Communication (PLC), 6LoWPAN, IEEE 802.15.4, Z-Wave	HTTP, CoAP, WebSockets, MQTT, XMPP, DDS, AMQP
Usage Context	Local, closed networks (device-to-device or device-to-gateway)	Global connectivity (device-to-cloud, device-to-application)
Design Goal	Reliable, energy-efficient device interconnection in constrained environments	Interoperability, scalability, and integration with internet services

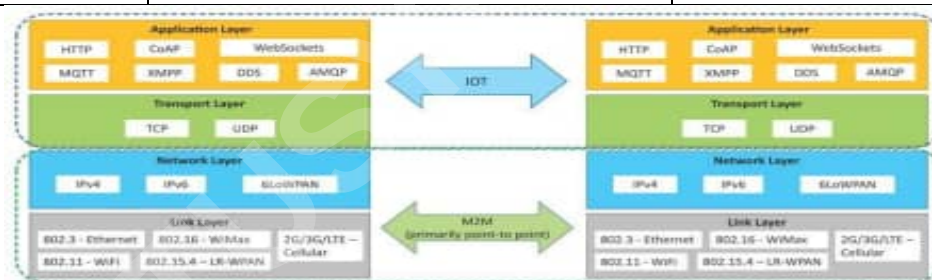


Figure 2.1.3: Communication in IoT and M2M

2) Machines in M2M vs Things in IoT:

Aspect	IoT (Internet of Things)	M2(Machine-to-Machine)
Definition of “Things” / Machines	Physical objects with unique identifiers that can sense, communicate, and interact with their environment or internal states	Machines/devices that communicate with each other within a closed area network
Unique Identifiers	IPaddresses,MAC addresses	Often proprietary IDs, not always IP-based

Capabilities	Sensing, processing, storing information, controlling actuators	Mainly focused on direct machine communication and control
Heterogeneity	Supports heterogeneous devices (e.g., fire alarms, door alarms, lighting control devices in home automation)	Typically homogeneous devices of the same type within the network
Software Components	Includes software for accessing, processing, storing, and sharing sensor/actuator data	Limited software, often tightly coupled to device function

3) Hardware vs Software Emphasis:

Aspect	M2M (Machine-to-Machine)	IoT (Internet of Things)
Emphasis	More on hardware with embedded modules	More on software for sensing, analysis, and cloud connectivity
Device Role	Machines communicate directly with each other	Devices interact with the Internet, cloud, and applications
Software Functionality	Limited, mainly supports communication between machines	Specialized software for sensor data collection, storage, analysis, and interfacing with cloud services
Communication	Often proprietary or non-IP based	IP-based communication (e.g., HTTP, MQTT, CoAP)
System Components	Machines/devices within a closed network	Things (devices), Internet, communication infrastructure, and applications

4) Data Collection and Analysis:

Aspect	M2M (Machine-to-Machine)	IoT (Internet of Things)
Data Collection	Collected in point solutions (device-specific)	Collected in the cloud (public, private, or hybrid)
Data Storage	Typically stored in on-premises infrastructure	Stored in cloud databases
Analytics	Limited or device-level	Performed in the cloud analytics component

Data Visualization	Usually device/application specific, not centralized	Visualized through cloud-based applications
Control	Localized control (device-to-device or gateway-based)	Centralized controller monitors all nodes and sends control commands
Observer Nodes	Not common in M2M	Present – can process information for applications, but do not perform control functions

5) Applications:

Aspect	M2M (Machine-to-Machine)	IoT (Internet of Things)
Data Collection	Point solutions	Cloud-based
Data Access	On-premises applications	Cloud applications
Typical Applications	Diagnosis applications, service management applications, on-premises enterprise applications	Analytics applications, enterprise applications, remote diagnosis and management applications
Data Scale	Relatively small/localized	Massive-scale (big data)
Data Analysis	Limited, local analysis	Cloud-based real-time and batch data analysis frameworks

2.2 SDN and NFV for IoT :

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 2.2.1 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.

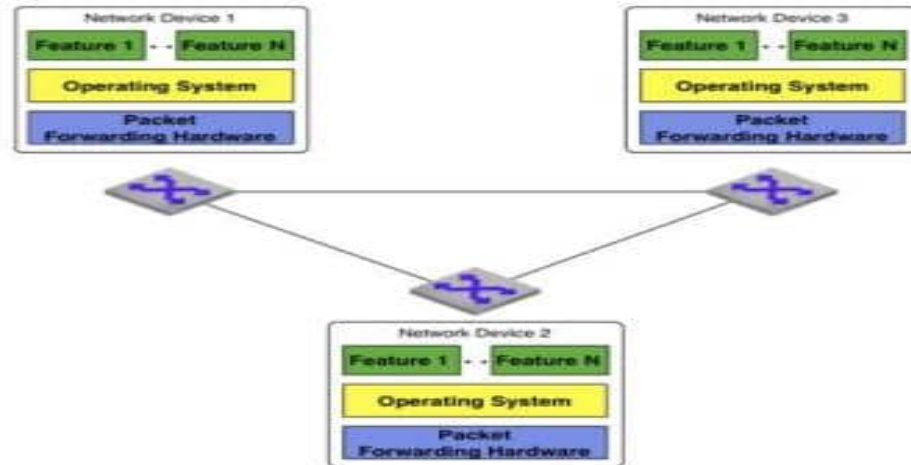


Figure 2.2.1 Conventional network architecture

Limitations of the conventional network architectures:

Complex Network Devices

- Conventional networks are getting increasingly complex with more and more protocols being implemented to improve link speeds and reliability.
- The conventional networks were well suited for static traffic patterns and had a large number of protocols designed for specific applications.
- IoT applications, the traffic patterns are more dynamic.
- In conventional network devices, making changes in the networks to meet the dynamic traffic patterns has become increasingly difficult.

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.
- This will be difficult with conventional networks.

Software Defined Networking(SDN):

- SDN is a networking architecture that separates the control plane from the data plane and centralizes the network controller.
- 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.
- Figure 2.2.2 shows the SDN architecture . 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.

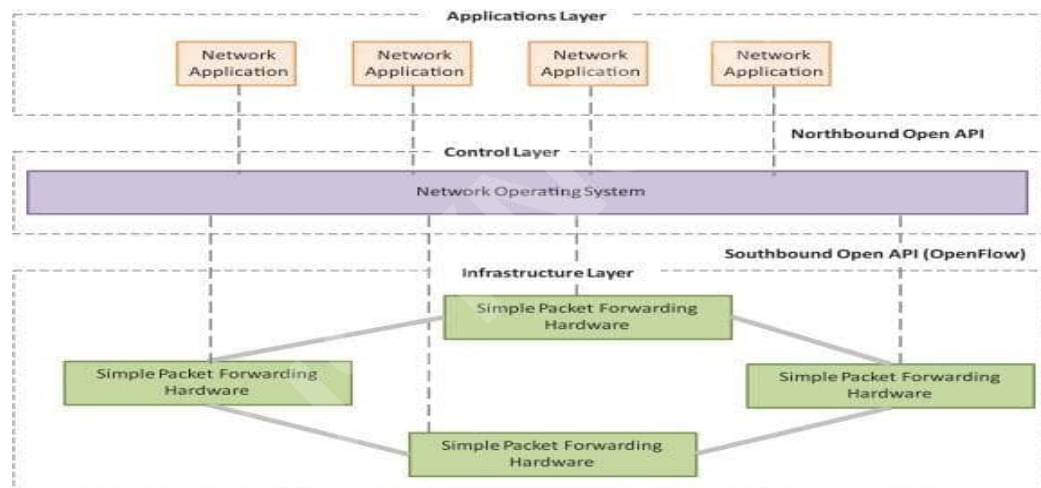


Figure 2.2.2 SDN architecture

Key elements of SDN:

Figure 2.2.3 shows the SDN layers

Centralized Network Controller

By decoupling the control and data planes and introducing a centralized network controller, administrators can configure and manage the network more quickly and efficiently. SDN applications can be deployed through programmable open APIs.

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.

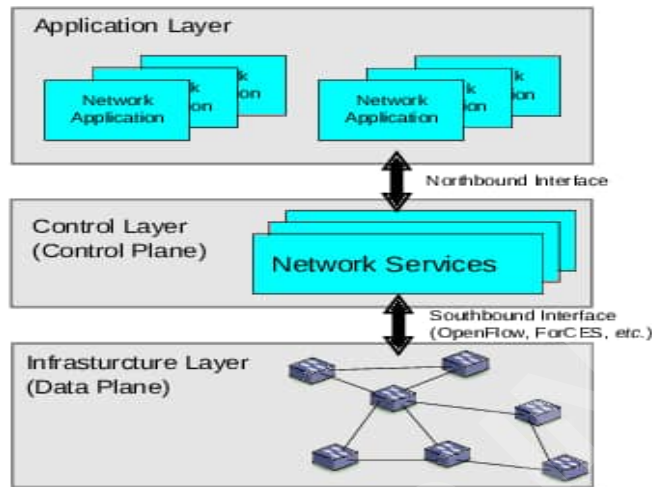


Figure 2.2.3 SDN Layers

OpenFlow enables direct access and control of the forwarding plane in network devices. It identifies network traffic using flows defined by specific match rules, which can be programmed either statically or dynamically through the SDN control software.

The figure 2.2.4 illustrates the components of an OpenFlow switch, which includes one or more flow tables and a group table responsible for packet lookup and forwarding, as well as an OpenFlow channel connecting to an external controller. The OpenFlow protocol operates on both sides of the interface between the controller and the network devices. Through this protocol, the controller manages the switch by adding, updating, or deleting flow entries in the flow tables

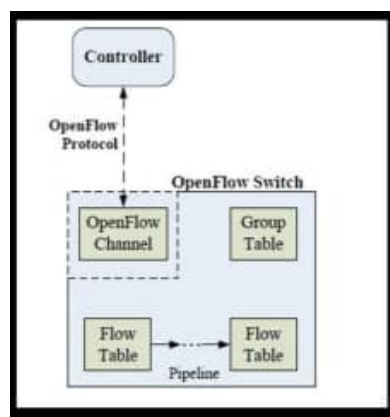


Figure 2.2.4 OpenFlow switch.

Figure 2.2.5 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.

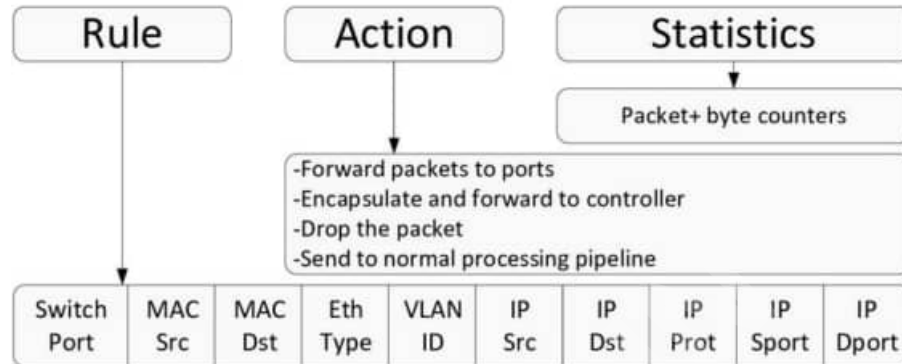


Figure 2.2.5 OpenFlow Flow Table.

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 2.2.6 shows the NFV architecture.

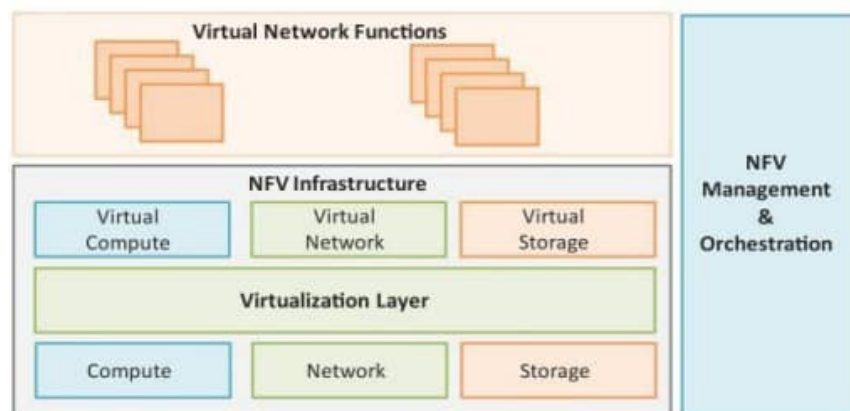


Figure 2.2.6 NFV Architecture.

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.

Network Function Virtualization (NFV) refers to implementing network functions as software running on virtualized cloud resources. It separates network functions from the underlying hardware, allowing them to be easily tested, upgraded, or replaced through software updates while using the same physical infrastructure. By virtualizing these functions, NFV lowers equipment costs, reduces power consumption, and leverages the multi-tenant nature of the cloud to share virtualized functions across multiple services. NFV is applicable only to data plane and control plane functions in fixed and mobile networks.

NFV Use Case:

Figure 2.2.7 shows a home network with a Home Gateway that provides Wide Area Network (WAN) connectivity to enable services such as Internet, IPTV, VoIP, etc.

NFV can be used to virtualize the Home Gateway.

The Home Gateway performs various functions including - Dynamic Host Configuration Protocol (DHCP) server, Network Address Translation (NAT), application specific gateway and Firewall.

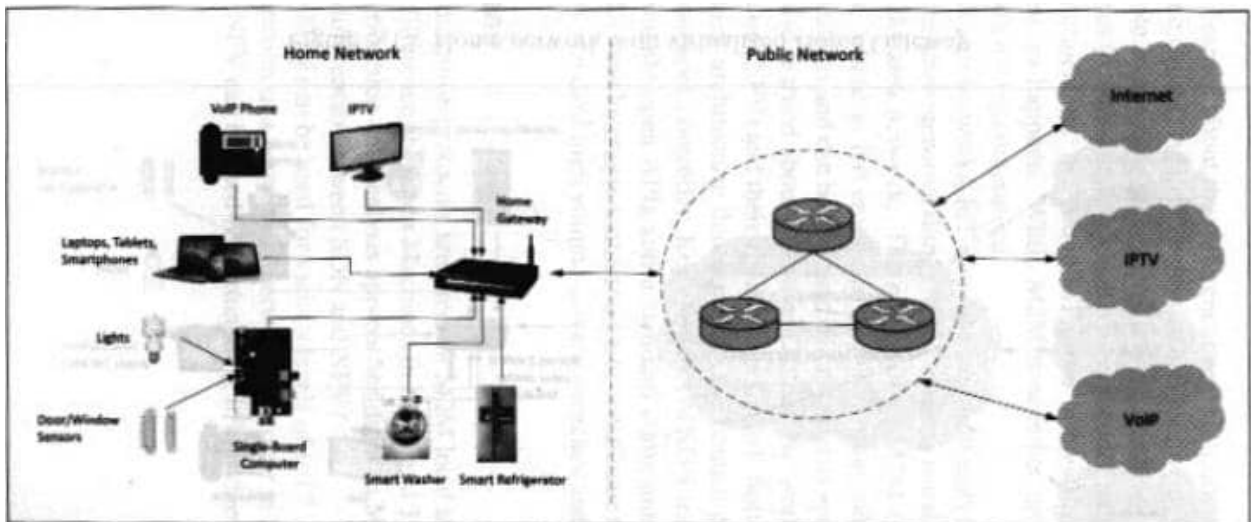


Figure 2.2.7 Conventional home network architecture

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 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 2.2.8 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.

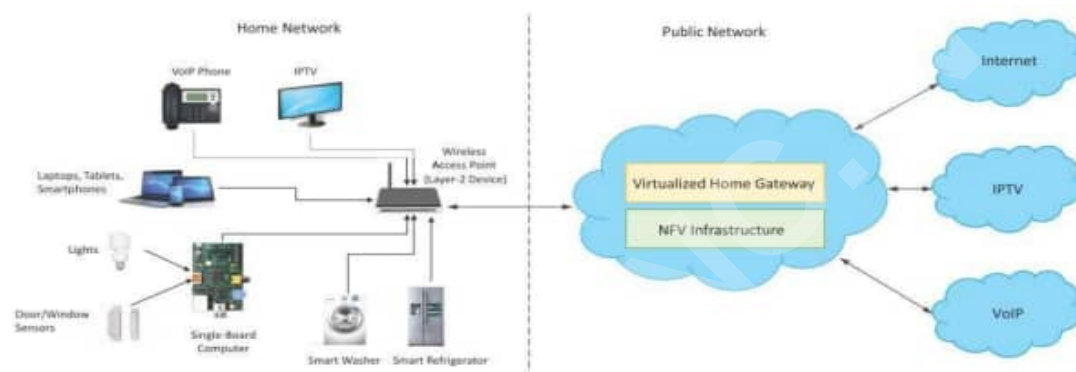


Figure 2.2.8: Home network with virtualized Home Gateway

2.3 IoT System Management

Internet of Things (IoT) systems involve complex combinations of hardware, software, and deployment architectures, including sensors, actuators, network resources, data collection and analysis services, and user interfaces. These systems are often distributed, consisting of numerous IoT devices that either gather sensor data or perform actuation tasks. Effectively managing such a large number of devices within a single system requires advanced management capabilities.

The need for managing IoT systems is described as follows:

- 1. 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.

2. 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.

3. Improved Reliability: A management system that allows validating the system configurations before they are put into effect can help in improving the system reliability.

4. System Wide Configurations: For IOT system 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. This 'all or nothing' approach ensures that the system works as expected.

5. 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.

6. 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 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.

Simple Network Management Protocol (SNMP)

Simple Network Management Protocol (SNMP) is a widely used network management protocol that enables monitoring and configuring devices such as routers, switches, servers, and printers. Figure 2.3.1 illustrates the key entities in SNMP-based device management: the Network Management Station (NMS), the Managed Device, the Management Information Base (MIB), and the SNMP Agent running on the device. The NMS issues SNMP commands to monitor and configure the Managed Device, which stores its attributes in the MIB. MIBs are defined using the Structure of Management Information (SMI) notation and organized hierarchically through object identifiers (OIDs). Management applications can retrieve or modify these variables as needed. Operating at the application layer, SNMP uses UDP as its transport protocol.

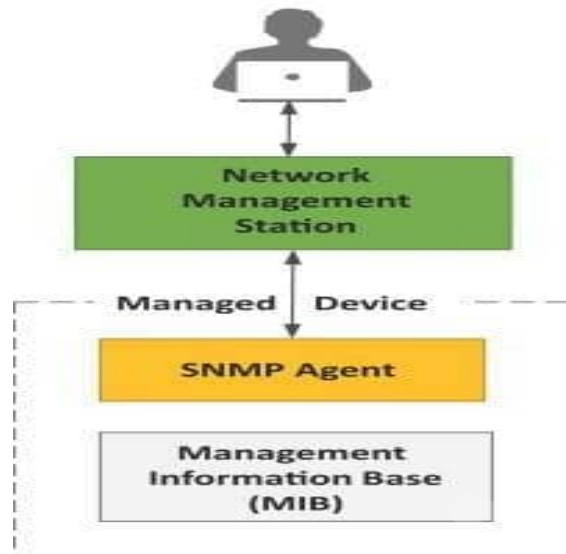


Figure 2.3.1 Managing a device with SNMP

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 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.
- 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.

Network Operator Requirements:

To overcome the limitations of existing network management protocols and guide future work, the Internet Architecture Board (IAB) identified set of operator requirements and, providing key insights for improved network management.

1. **Ease of use:** From the operators point of view, ease of use is the key requirement for any network management technology.
2. **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.
3. **Fetch configuration and state data separately:** 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.
4. **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.
5. **Configuration transactions across devices:** Configuration transactions across multiple devices should be supported.
6. **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.
7. **Dump and restore configurations:** It should be possible to dump configurations from devices and restore configurations to devices.
8. **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.
9. **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.
10. **Consistency of access control lists:** It should be possible to do consistency checks of access control lists across devices.
11. **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.
12. **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.

NETCONF:

Network Configuration Protocol (NETCONF) is a session-based network management protocol. NETCONF allows retrieving state or configuration data and manipulating configuration data on network devices

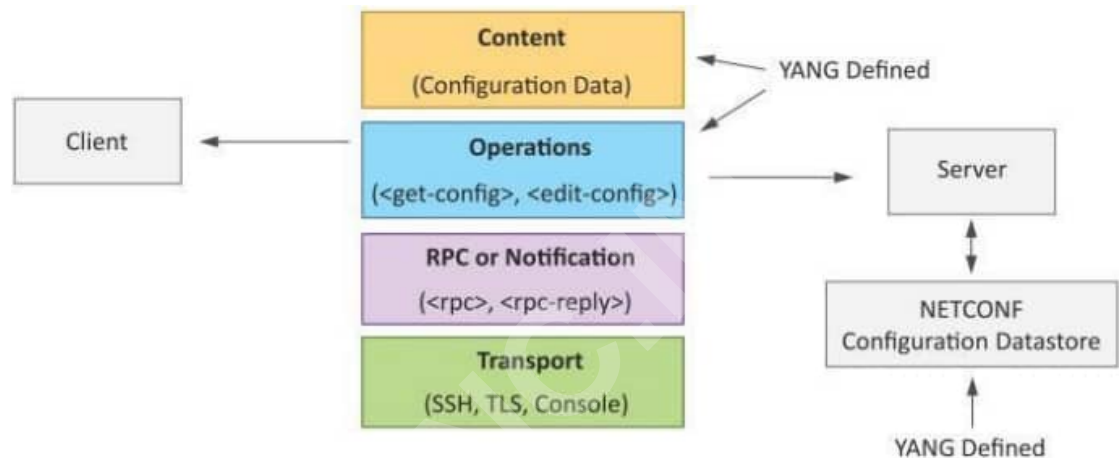


Figure 2.3.2 NETCONF protocol layer.

Figure 2.3.2 shows the layered architecture of NETCONF protocol. NETCONF works on SSH transport protocol. In addition to SSH, NETCONF implementations can support other transport mappings such as Blocks Extensible Exchange Protocol(BEEP). Transport layer provides end-to-end connectivity and ensure reliable delivery of messages. NETCONF uses XML-encoded Remote Procedure Calls (RPCs) for framing request and response messages. The RPC layer provides mechanism for encoding of RPC calls and notifications. NETCONF provides various operations to retrieve and edit configuration data from network devices. list of some commonly used NETCONF operations are listed in the table 2.3.1.

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 get-schema commit close-session kill-session 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 2.3.1: List of commonly used NETCONF RPC methods

The Content Layer consists of configuration and state data which is XML-encoded. The scheme of the configuration and state data is defined in a data modelling language called YANG. NETCONF provides a clear separation of the configuration and state data. The configuration data resides within a NETCONF configuration datastore on the server. The NETCONF server resides on the network device. The management application plays 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 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.

YANG:

YANG is a data modeling language used to model configuration and state data manipulated by the NETCONF protocol.

YANG modules contain the definitions of the configuration data, state data, remote procedure call (RPC) protocol 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 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.

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 2.3.2.

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 2.3.2: YANG Node Type

IoT Systems Management with NETCONF-YANG

Figure 2.3.3 shows the generic approach of IoT device management with NETCONF-YANG. The components are listed below.

1. 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.
2. 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.
3. 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.

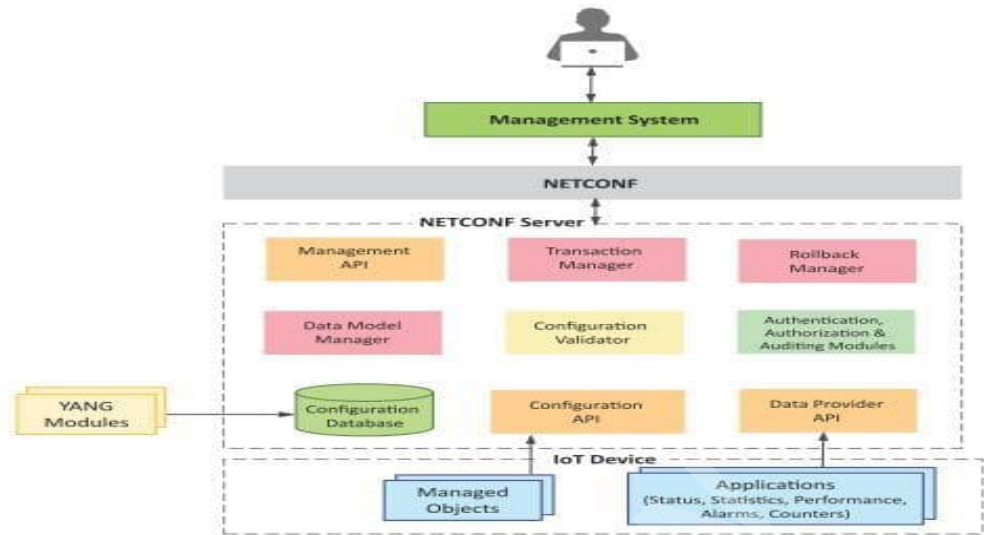


Figure 2.3.3: IoT device management with NETCONF-YANG - a generic approach

4. Rollback Manager: Rollback manager is responsible for generating all the transactions necessary to rollback a current configuration to its original state.
5. 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.
6. Configuration Validator: Configuration validator checks if the resulting configuration after applying a transaction would be a valid configuration.
7. Configuration Database: This database contains both the configuration and operational data.
8. 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 data store.
9. 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.

NETOPEER:

Netopeer is set of open source NETCONF tools built on the Libnetconf library. It allows operators to connect to their NETCONF-enabled devices as well as developers. Figure 2.3.4 shows how to manage an IoT device using the Netopeer tools. The Netopeer tools include:

1. 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.
2. 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.

3. 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

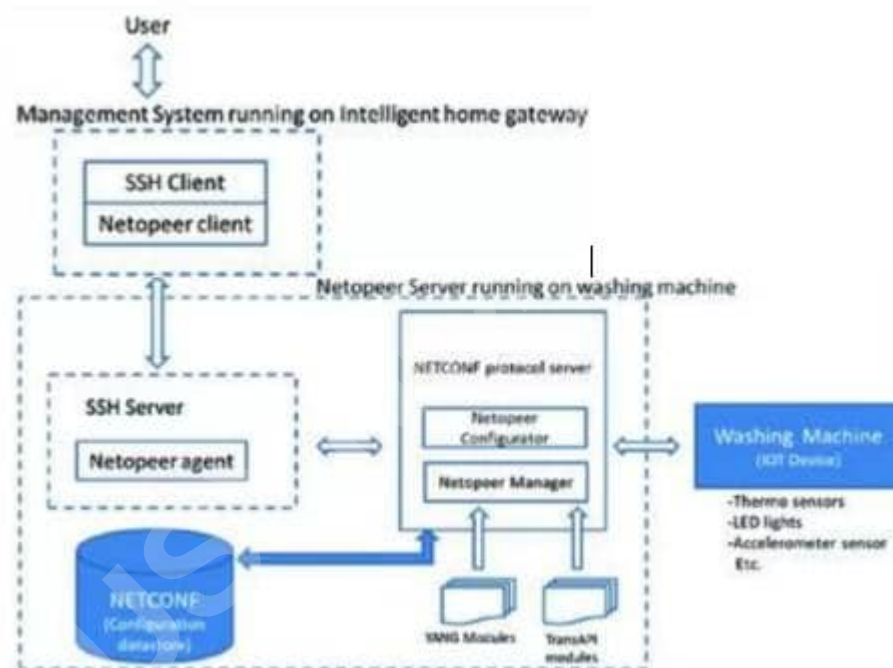


Figure 2.3.4: IoT device management with NETCONF - a specific approach based on Netopeer tools

4. 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.
5. 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 'Inctool' 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 'Inctool' 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.