Educational Administration: Theory and Practice
2024, 30(5), 6484-6495
ISSN: 2148-2403
https://kuey.net/

Research Article

Managing Large Iot Network Using SDN And Hierarchical Tree Topology

Savita Vijay[1*], Dr M.k. Banga[2]

[1*,2]Dayananda Sagar University,Bangalore, India . Savita84.23@gmail.com, mkbanga-cse@dsu.edu.in

Citation: Savita Vijay, Dr M.k. Banga, (2024), Managing Large Iot Network Using SDN And Hierarchical Tree Topology, *Educational Administration: Theory and Practice*, 30(5), 6484-6495
Doi: 10.53555/kuey.v30i5.3969

ARTICLE INFO

ABSTRACT

The study covers the potential of managing large Internet of Things (IoT) networks, such as Smart City network using the Software-Defined Networking (SDN) approach. A hierarchical tree topology provides an efficient and scalable solution for connecting a vast number of IoT devices in a Wide Area Network (WAN). The tree topology makes it simple to manage and add or remove IoT nodes from the network due to its hierarchical structure. Additionally, the topology can be highly fault-tolerant, as each IoT node has redundant paths to the root node. By implementing a 1024-node IoT network with a hierarchical tree topology and leveraging the Ryu controller for centralized management, we aim to investigate the performance characteristics of such a network. Performance evaluation conducted for network size of 4,8,16,144 and 1024 hosts for TCP and UDP traffic using Mininet that is network emulation tool, Iperf used for bandwidth and Throughput analysis, and Ping for latency and packet loss measurements. The findings will offer valuable insights for optimizing large-scale IoT deployments.

Keywords—*Internet of Things IoT, Software Defined Networking, SDN, Large Scale Networking, WAN, IoT Nodes or hosts.*

1 Introduction

IoT networks of substantial size, which encompass a multitude of connected devices or "things," are known as large IoT networks. These devices gather and transmit data across the internet, and the scale of these wireless and wired networks can range from modest deployments within domestic or professional settings to vast networks that extend over entire cities, industries, or even nation-states.

Large-scale IoT networks possess several critical features and factors, including scalability, which enables the connection of thousands, millions, or even billions of devices, encompassing a wide range of capabilities, communication protocols, and data formats. These devices span across sensors, actuators, smart appliances, vehicles, and high-end GPU processors. The communication in IoT networks is typically facilitated through a diverse array of connectivity technologies, such as Wi-Fi, Bluetooth, cellular networks (LTE/3G/4G/5G), LPWAN technologies like LoRaWAN or NB-IoT, Zigbee, or Ethernet. The devices in these wireless networks may vary in their capabilities, communication protocols, and data formats.

Managing and safeguarding the copious amounts of data produced by IoT devices presents a substantial obstacle. The design of extensive IoT networks must consider the need for dynamic scaling as the number of connected devices increases. To facilitate growth and adaptation, scalable architectures, adaptable protocols, and modular solutions are necessary. It is also crucial to ensure interoperability and power management from a centralized location in large-scale deployments.

Effective monitoring and management tools are indispensable for ensuring the proper functioning, health, and security of IoT deployments. These tools include centralized dashboards, analytics platforms, and remote device management solutions that allow administrators to oversee, diagnose, and enhance the performance of IoT networks with ease.

To develop IoT network using the principles of Software-Defined Networking (SDN) to ensure efficiency, scalability, and flexibility by making network management programmable and supporting a wide range of IoT applications. Incorporate IoT devices into the SDN architecture through the utilization of IoT gateways or edge computing nodes. In this study as depicted in Fig. 1, we design an IoT network for 1024 devices using a tree topology on Mininet and assess its performance using Ping and Iperf as metrics.

The paper's structure is as follows: In Section 2, the Literature Review is discussed, where pertinent papers in this area are examined. This is followed by Section 2, which discusses the methodology. This section includes the methodology followed to perform the proposed research as depicted in Fig.1. mininet emulator is used to build up a custom Tree topology IoT network under the SDN environment. Followed by the implementation of the Ryu controller in the SDN. In the Mininet tree topology, the implementation of an Ryu controller was done. Once the network is successfully build, thereafter the data traffic is generated from source IoT node(hosts) to destination IoT node(hosts) using the Wireshark traffic analysis tool. Here Hosts and IoT nodes or devices terms can be used interchangeably. Lastly, the performance of the finally designed Tree topology IoT network using the Ryu controller is evaluated based on various metrics, Section 3 will provide the detailed study of Implementation and Result analysis will be done. Section 4 concludes the paper.

Literature Review

The traditional network architecture is deemed insufficient to serve the demands of Large IoT applications, as a significant portion of the network capacity is currently being utilized [1,2]. The majority of data traffic within the network is transmitted through the reliable Transmission Control Protocol (TCP), which offers several advantages such as congestion control, improved bandwidth, and reliable communication [3]. In light of the substantial demand for network usage, it is crucial to modernize the traditional network architecture for working on IoT networks like Smart cities [4]. This paper also discuss security of IoT networks.

In the traditional network architecture, the network node serves as the data and control plane, and end-to-end hosts working on Big Data processing like Hadoop tool [5]. As per the survey done by Farhady et al, the traditional network design combines plane in the same device [6,8]. However, there is no abstraction rule for the control plane across the entire network, It is impossible to visualize the global network perspective from centralized location and it requires manual configuration for each device including wearable sensors [7]. The traditional architecture of single controller has constraints such as limited scalability, reliability, and compromised security [8], this study includes multiple controllers to manage individual network domains and communicate with each other to provide end-to-end network services. This complexity in manual configuration is the primary reason for the limitations of the traditional network architecture. The survey done in [9] delves into key areas of SDN traffic engineering, including flow management, fault tolerance, topology updates, and traffic analysis. To overcome these shortcomings of network, Network programmers have proposed a new paradigm called software-defined networking (SDN), which provides facility for designing or configuring the large network easily and supporting programmable centralized controller by partially deploying SDN which manages the control plane with a global network perspective [10]. It proposes a novel and practical solution for cost-effective measurement systems in such deployments, along with a synchronization mechanism for aggregating traffic statistics from multiple controllers [11]. Here emulation is done on datacenter topology.

This paper introduces SEAL (Secure and Agile), a novel Software Defined Networking SDN-based framework designed to adaptively protect smart city applications from DDOS attacks, Collectively the framework's modular architecture ensures fault tolerance scalability and reliability, while experimental evaluations demonstrate its effectiveness in combating DDOS attack [12]. There are two types of application programming interface (API) operating for communication in SDN that is southbound and northbound API. The communication between the control plane and the data plane handled by the southbound API. Similarly northbound API is responsible for the communication between the control plane and management plane. This discusses the integration of SDN with IoT gateway nodes to enhance network management efficiency, implementing an Ethernet packet frame-based routing algorithm to improve Quality of Service [13] for the large network architecture. The SDN controller network operating system (NOS) operates within the control plane of the SDN environment, to manage network from the centralized location and provide software-based services and handle network traffic successfully. This paper proposes a reward based formal model, SDN R, to compute real time QoS, By separating time based reliability from other QoS parameters and utilizing Extended Time Automata [14].

This study proposes Software-defined architecture for Cyber-Physical Systems and IoT applications, focusing on scalability, flexibility, and cybersecurity. It uses smart agents, decentralized control, and in-network data processing [15]. Suarez-Varela et al proposed a scalable flow monitoring solution compatible with existing OpenFlow switches. Leveraging DPI and Machine Learning techniques, Here flows are classified ad data and enhancements you will ultimately determine than this one isn't as especially web and encrypted traffic, with two sampling methods tailored to OpenFlow features [16]. [17] In this Author explores the application of SDN in WSN, creating a Software-defined wireless sensor Network (SDWSN), Highlighting challenges in network management and configuration. The focus is on studying these challenges to enhance security efficiency, and reliability, with insights drawn from literature on SDN, WSN, and SDWSN architecture. [18] This study reveals the performance of various SDN controllers including NOX, Floodlight, ONOS, POX focusing on metrices like throughput and round-trip time. Using a Mininet emulator with an RYU controller, the research assesses parameters like bandwidth, throughput, round trip time, jitter, and packet loss, providing insights into the effectiveness of SDN architectures. The implementation of the SDN architecture is performed in the Mininet emulator. RYU controller is used for a custom-designed tree topology comprising network nodes

more than 1000 under one topology, and an OpenFlow switches. The proposed work aims to conduct an in-depth performance analysis of the SDN based IoT network Tree topology architecture for various parameters, such as the number of transmitted and received packets, minimum, average and maximum bandwidth, round trip time and throughput with and without acknowledgement and standard deviation in time(milli seconds) among others. The RYU controller in an SDN environment in case of UDP transmission is evaluated based on data traffic parameters, including throughput, packet loss, delay and bandwidth. The experimentation is performed in the Mininet emulator using OpenFlow switches. While some authors have investigated the SDN environment's performance for a default topology, deeper discussion of the SDN environment's performance for a custom topology is required because of dynamic IoT networks node addition and deletion. Queiroz et al. [19] paper proposes a Big data streaming approach to collect and process counter values, offering detailed insights into network resource utilization and enhancing TE capabilities, as validated by experimental results conducted by the author. Further the authors measure the usage of network resources in the SDN environment and suggested solutions for enhancements to improve SDN architecture performance. Badotra et al. [20] conducted a comparative study of the performance of two leading open-source SDN controllers, namely Open Networking Operating System(ONOS) and Open Daylight (ODL), using Mininet emulation and Wireshark analysis. Results indicate superior performance of the ODL controller over ONOS in terms of burst rate, throughput, bandwidth, round trip time, and data transfer.

In [21], Priya et al. discussed the functionality of the SDN controller within the control plane of the SDN environment. This controller depends on the Network Operating System (NOS) to deliver topology, traffic management services, virtual services, and network applications. The southbound API facilitates communication between the control plane and the data plane, while the northbound API enables communication between the control plane and the management plane. NOX/POX, OpenDayLight, beacon, floodlight, RYU are some of the various types of SDN controllers.. These controllers act as the "central processing unit" of the SDN architecture. The primary purpose of these controllers is to improve the resource utilization and enhance the network performance in the SDN environment.

The authors of paper [21] compared the performance of various OpenFlow controllers, such as NOX, POX, Ryu, FloodLight, and OpenFlow reference controller, based on packet handling capacity and parameters like packet size and traffic flow patterns. To measure the performance in terms of delay, jitter, throughput and packet loss, distributed internet traffic flow generator (D-ITG) tool was used.

The research indicated that the FloodLight controller demonstrated superior performance in both throughput and delay when compared to other controllers. The authors propose that future investigations expand on these findings by conducting a comparative analysis involving the OpenContrail controller and OpenDayLight controllers.

Wang et al. proposed a UDP-based reliable transmission framework aimed at improving the efficiency of TCP transmission on SDN-enabled networks. By leveraging SDN technology to customize flow rules and designate packet routes, the framework reduces TCP overhead significantly, ensuring reliable packet delivery with improved bandwidth usage [28].

In another study, Tootoonchian et al. [27] emphasized that controller responsiveness is the primary factor in deciding whether additional controllers should be deployed. While multiple controllers are necessary for high availability and maintaining low response times, it seemed feasible to maintain a consistent logically centralized view of the network across controllers. Finally, the study noted that understanding overall SDN performance remains an open research problem, and its single controller microbenchmarks are just a first step towards comprehending the performance implications of SDN.

In a separate study, Bhatia et al. [22] proposed a data-driven approach, integrating Software Defined Vehicular Networks (SDVNs) and machine learning, to predict vehicular traffic behavior accurately. Utilizing clustering algorithms and a long short-term memory neural network (LSTM-NN) architecture, the model achieves a high level of accuracy, equivalent to 97%, in real-time traffic density prediction. Additional investigation is required to evaluate the effectiveness of the suggested model in alleviating traffic congestion within vehicular networks.

Lu Yu et al. [29] introduced an alternative network architecture to counter vulnerabilities in destination IP prefix-based routing, proposing dynamic IP assignment via SDN to enhance security. By implementing three strategies through SDN, it enables scalable transformation of Internet addressing paradigms, offering flexible IP addressing and customizable network services. They suggested that future research work could be done to improve the security of the SDN architecture.

Amin et al. [23] did an extensive survey of several hybrid SDN environments and identified research gaps and existing solutions. They proposed the development of an automated and dynamic system for managing SDN networks.

A Study was conducted by Singh and Jha [24] to compare the algorithms for SDN control panel performance based on latency, jitter and Q factor values. It was concluded that a centralized programmed architecture outperformed other architectures. For further improvement, author recommended to do a load balancing between multi-controller arrangements in a SDN network.

Akyildiz et al. [25] outlined the state-of-the-art in traffic engineering for SDN, highlighting core aspects like flow management and fault tolerance, and discussing associated challenges. Moving forward, future steps may involve further exploration and development of novel traffic engineering solutions to address the evolving needs of SDN networks.
Bholebawa et al. [26] did a performance comparison of two SDN controllers, POX and Floodlight across different SDN topologies. They found that Floodlight showed improvement over POX in throughput and roundtrip time for single, linear, star and custom topologies.
As per available information, RYU stands out as a prominent SDN controller tailored for enhancing network agility in traffic engineering contexts [18]. However, a notable gap exists in research concerning the implementation of SDN architecture using a custom-designed topology and conducting a comprehensive analysis of diverse network performance metrics employing the RYU controller [18]. This study seeks to bridge this gap by focusing on deploying the SDN architecture with the RYU controller within the Mininet emulator, utilizing a specifically tailored tree topology. The investigation aims to assess various node-to-node performance metrics of the SDN network, including throughput, bandwidth, and round-trip time, leveraging the capabilities of the RYU SDN controller.

2 Problem Definition:

Table 1 highlights the problem statement through several key facets: firstly, it emphasizes the significance of the proposed investigation; secondly, it delineates the specific domain where the research problem manifests; and finally, it outlines the framework for presenting the research findings.

Table 1. Problem definition

Domain of Study	Research Focus	Methodology
Large IoT Networks	**Investigating the impact of SDN Ryu controller performance**	**Evaluation of host-to-host metrics with RYU SDN controller**
Specific Domains (e.g., Smart Healthcare, Smart City, Smart Transport)	**Identifying the need for high-performance controllers in diverse IoT applications**	**Examination of the significance of SDN controller performance in enhancing network capabilities**
Traffic Engineering for SDN	**Addressing the gap in research regarding traffic performance assessment with the RYU SDN controller**	**Proposing a framework for comprehensive traffic performance evaluation in SDN environments**

SDN allows for centralized management of network resources, which can be particularly useful in IoT deployments with a large number of devices. Through a centralized controller,dynamically allocate resources, monitor network traffic, and apply security policies, providing greater control and flexibility. Here experimentation is done for that

3 Methodology:

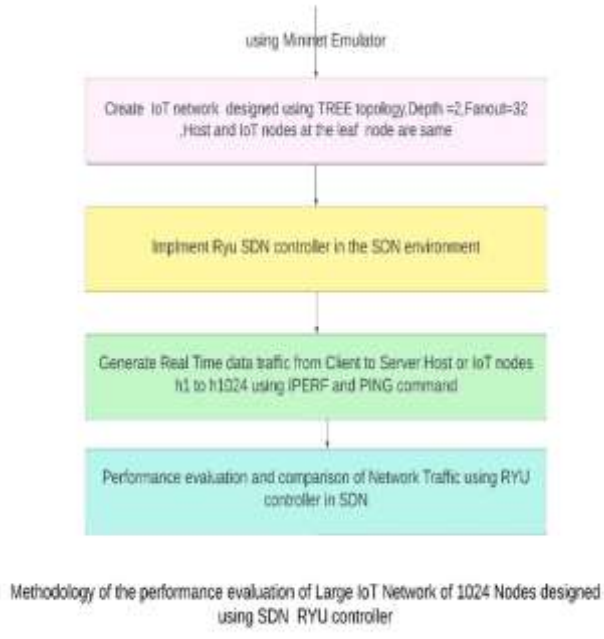

Fig 1. Methodology of Ryu SDN controller

Methodology as mentioned in Fig 1, is followed in the given below work.

3.1 Setup Mininet Environment:

Integration with Mininet Environment:

Set up the simulated network using Mininet, replicating the tree topology implemented and run as shown in Figure 2 below. Configure Mininet to emulate the behavior of switches, controller switch c0, and 1 Distribution switch connected with other 32 switches via its 32 ports. With every port of S2-S33 switch, 32 IoT devices or hosts or IoT nodes connected from port1 to port 32 within the virtualized environment. Integrate Mininet with an RYU controller to enable centralized control and management of the virtual network. Figure 4 shows all the nodes in IoT network implemented on Mininet.

Figure 2 is the creation of 1024 nodes on mininet emulation platform using Ryu controller and it handles here 1024 hosts using 32 switches directly . It is using tree topology structure .

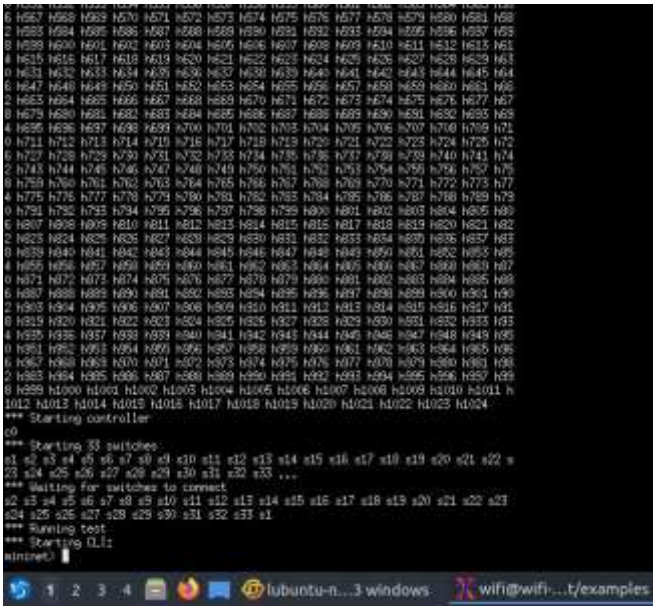

Fig 2. Implementation of Custom network of 1024 nodes using 33 switches using SDN running on Mininet and Ryu.

Traffic engineering and Quality of Service (QoS) policies can be implemented within the Ryu ccontroller to prioritize and optimize IoT traffic. Define rules for traffic shaping, prioritization, and routing based on the requirements of different IoT applications and services.

Setup Steps are given below-

Install Mininet on system.

Launch Mininet using the appropriate command.

Create the Tree Topology:

sudo mn –topo tree, depth=2,fanout=32

network = TreeNet(depth=2, fanout=32, host=HostV4,

switch=OVSSwitch, waitConnected=True).

Once the topology is created, assign unique IP addresses to each node in the network. To automate this process using Python scripts or manually assign IP addresses using Mininet's command-line interface is possible.

SDN Controller:

Implementing an SDN RYU controller (c0) at the root/core switch facilitates centralized management and control of the network. This controller communicates with all switches in the network using protocols such as OpenFlow to dynamically program forwarding rules and optimize traffic flow. RYU is an open-source and component-based SDN controller, licensed under Apache 2.0, and its name means “flow” in Japanese, reflecting its function of directing flow control for innovative network management. The RYU controller supports several protocols for managing the network, including the Network Configuration Protocol and OpenFlow protocol versions 1.0 to 1.5, and it is implemented in the Python programming language.

The RYU SDN controller's architecture is divided into three planes: the application layer, the network layer, and the physical layer [34]. The lowest layer is the physical layer, which includes various physical and virtual devices connected to the internet to communicate with one another. This layer is sometimes referred to as the infrastructure Layer .The structure in question comprises three tiers: a base layer, a middle layer, and a top layer. The base layer is made up of a variety of IoT devices and hosts that are positioned on the same level. The middle layer, often known as the control layer, manages the flow of data traffic between nodes to maintain

network stability without incurring excessive overhead. The interface between the physical layer and the application layer is facilitated through well-defined Application Programming Interfaces (APIs)[35] known as southbound interfaces, such as Network configuration protocol , OpenFlow, and OF-config. Finally, the topmost layer, known as the application layer, consists of end-user applications, including network and business logic. The objective of these applications is to centralize network intelligence at a single location, known as the controller.

3.2. IoT Network Topology:

Designing a large IoT network using a tree topology involves structuring the network in a hierarchical manner, with a root node (or core switch) at the top and multiple levels, here level 2, of branches (or distribution switches) connecting to edge devices (1024 IoT devices or hosts).
Here's a conceptual design of the network using a tree topology as shown in Figure 3:

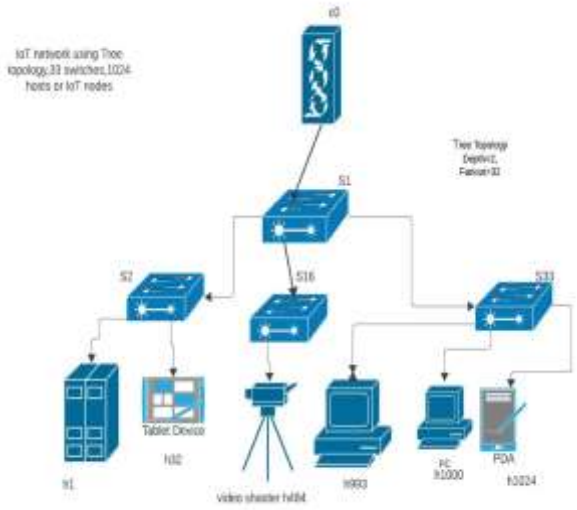

Fig 3. IoT Network using Tree Topology

Root/Core Switch:

At the top level, have a powerful root/core switch that serves as the central point of control for the entire network. We name that switch as controller c0.This switch connects to distribution switches at lower levels and provides high-speed connectivity and routing capabilities. Total 33 switches excluding controller used in this design.

Distribution Switches:

At the second level, the network is managed by S1 distribution switch. S1 distribution switch connects to the root/core switch and serves as a gateway for a specific subset of S2-S33 distribution switches. Distribution switches can be strategically placed to efficiently route traffic and minimize latency. here using tree topology, we have one switch S1 having 32 ports connected to other 32 Switches and one port connected to Root switch. Number of switches and their port and link connections are shown in Figure 4,5 and Figure 6.1,6.2.

IoT Devices(Hosts,1024) or Access Switches :

At the third level,1024 IoT devices or hosts encompass a variety of sensors, actuators, controllers, and other smart devices deployed throughout the network. These devices communicate with each other and with centralized controller to collect and transmit data. Sometimes it is possible to connect access switches to each distribution switch. Access switches provide connectivity to a group of IoT devices or end devices within a localized area or zone.

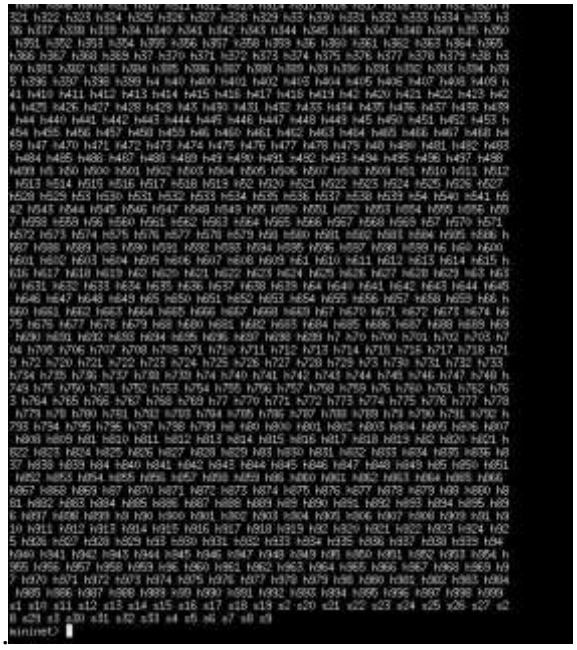

Fig 4. Nodes in Proposed IoT Network Topology

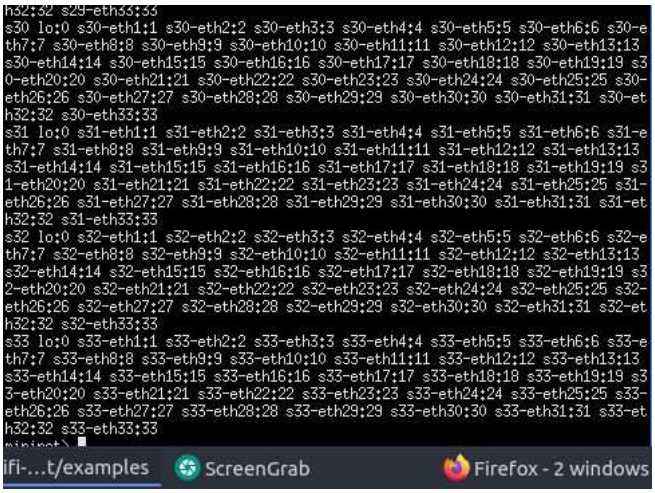

Fig 5. Ports in proposed IoT Network Topology

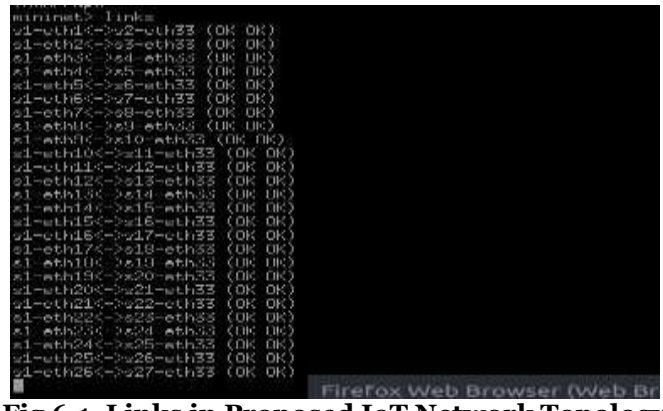

Fig 6.1. Links in Proposed IoT Network Topology

In Fig 5, its clearly define that switch S1, port number, 1-32, are connected to Switches S2-S33. The links as shown in Figure 6, in between switch S1 and switch numbers S2 -S33 are making tree topology. S1-eth33 port of Switch S1 is connected to controller c0.

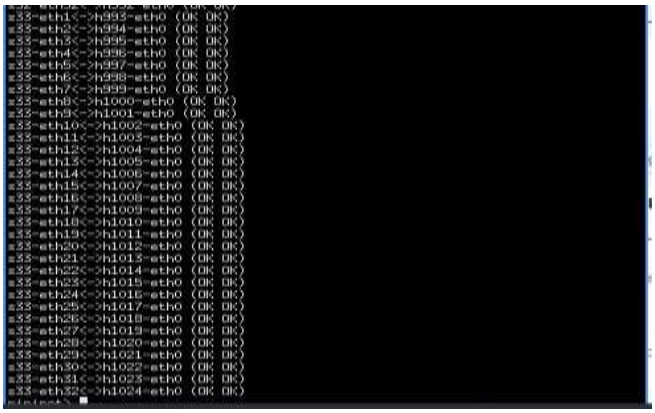

Fig 6.2. Links of Switch S33, in Proposed IoT Network Topology

the roles of core switches(S1), distribution switches(S2-S33) and Hosts(h1-h1024). Each distribution switch S2 is connected to 32 hosts and similarly S2-S33 switches connected to 32*32 that is 1024 hosts. Switch S1 is connected to Switch S2 to S33 as shown in Figure 6.1.

Performance Monitoring of Network:

This section mentions the results obtained for the SDN large IoT network performance evaluation through the RYU SDN controller. The performance parameters considered such as bandwidth , throughput(Gbps), round trip time(milli seconds)have been measured under TCP, UDP data traffic using benchmark tools like Iperf as well as Ping. Deploying monitoring tools Ping and Iperf to monitor the virtualized network environment and track the performance of IoT network. Utilize SDN-based analytics to gain insights into network traffic patterns, device behavior, and performance metrics is also possible.

Figure 7 presents the connections among different ports. Here, the host h1024 is connected to the TCP port 5001 via IP 10.0.4.0, and the default size of the TCP window taken is 178(no ack) and 85.3 Kb. The experimentation shows the interval between the client host and the server host, the transfer rate, and the bandwidth of the particular connection.

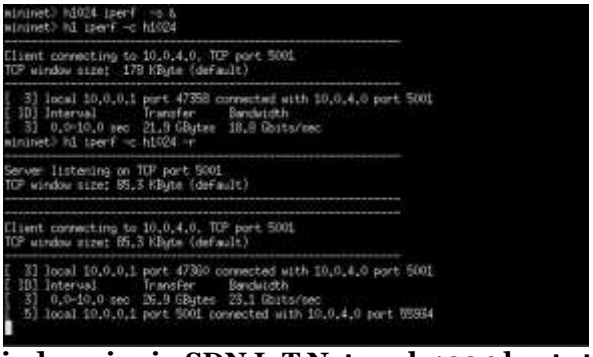

Fig 7. TCP window size in SDN IoT Network 1024 hosts tree topology

Throughput represents the actual amount of data traffic processed via controller(c0) between two nodes of the network(h1-h1024) in a second. In TCP traffic, one of the host act as a client(h1) and another act as a server(h1024). Iperf3 utility has been utilized to test the throughput of the controller. On the customer side, Iperf3 traffic has been generated in every 10 s, and the information about throughput has been gathered on another side of the network.

The throughput is in Gbps. It is 21.9 and 26.9 Gbps between h1 to h1024 respectively, without and with acknowledgement scenarios.

Bandwidth

To calculate the traffic transmitted between Host h1 and Host h1024 using TCP protocol, Iperf command is used.

Here Host 1024 behave as server and running in background. Host h1 as client node sends TCP SYN request packet to the Host h1024 node for the establishment of connection. Here S1 switch used for connection among client and server node. Iperf command executed in every 10 seconds between client and server for transmitting traffic. After performing this test Figure 7, shows the bandwidth of the proposed Tree network topology between the nodes. The maximum data transferred between h1 to h1024 is 18.8 and 23.1 Gbps.

Run Iperf UDP server in h1024:

Iperf server in Figure 8, is listening on the UDP port number 5001 receiving 1470 bytes of datagrams with the default buffer size of 208 Kbyte. Run Iperf UDP client in h1 and use Protocol, -u: UDP, -c: Client, 10.0.4.0: IP address of h1024, -b 10m: bandwidth internal time and transfer 10 Mbps data. If bandwidth is not specified it is by default taken as 1mbps. -P: creates 10 parallel connections and each connection will send 1mbps default value. Iperf client in Figure 8, is connecting to 10.0.4.0 (h1024) UDP port 5001.

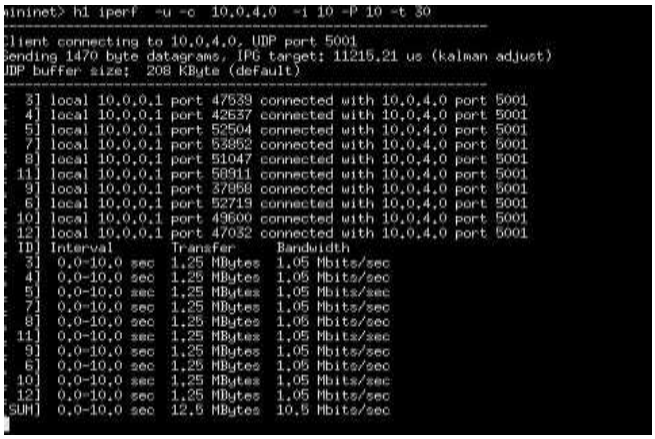

Fig 8. UDP window size in SDN IoT Network 1024 hosts tree topology

Monitoring Performance with Ping:

Using the ping command, one can monitor the network performance between IoT nodes. The source node, which can be either an IoT node (such as IoT node1), a client, or h1, should be selected, and each of its neighboring nodes (h2-h1024) should be pinged to measure latency and packet loss. It is recommended to perform these Ping tests periodically to monitor changes in performance over time. Additionally, the "Pingall" command can be used to check the connectivity of all nodes in the network.

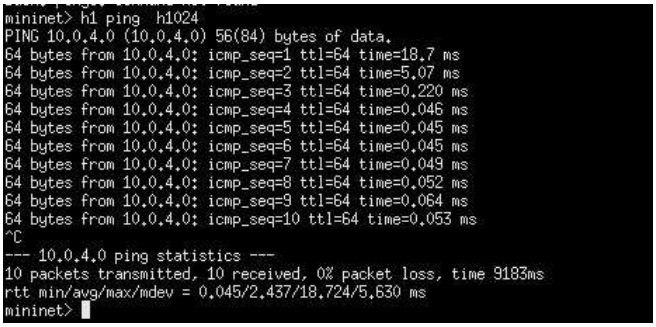

Fig 9. Ping Test Time or Round-Trip Time

Round trip time (RTT) is also known as a ping test time. RTT is the total time taken to by the data packet to reach from a specific source host h1 to the destination host h1024 and the acknowledgement packet to reach back to the source h1. Ping utility uses Internet Control Message Protocol (ICMP).
Figure 9 shows the calculation for 10 packets transmitted between h1 to h1024 and RTT for their path depicting minimum, maximum, and average values. The minimum /Average/Maximum RTTs of the proposed SDN topology is 0.045 /2.437/18.724 ms. Total RTT is 9183 ms.

Further, using the tree topology, different network setups were done using the mininet tool with the help of RYU controller starting from 4 hosts and scaled that to 16 hosts, 64 hosts, 144 hosts and finally to 1024 hosts. Performance of the TCP traffic transmission was observed while scaling up the network with more hosts. Below is the summary of the Iperf command run between two hosts to show throughput and bandwidth calculations between the extreme hosts using TCP traffic in the interval of 10 seconds (Table 2):

No. of switches	No. of nodes (hosts)	Throughput Gbps	Bandwidth Gbps	ACK Throughput Gbps	ACK Bandwidth Gbps
2	h1-h4	30.1	25.8	6.48	5.54
4	h1-h16	28.6	24.6	6.51	5.57
8	h1-h64	28.5	24.5	6.58	5.64
12	h1-h144	31.8	27.3	6.57	5.63
32	h1-h1024	45.8	39.3	6.27	5.36

Table 2. TCP traffic in the interval of 10 seconds

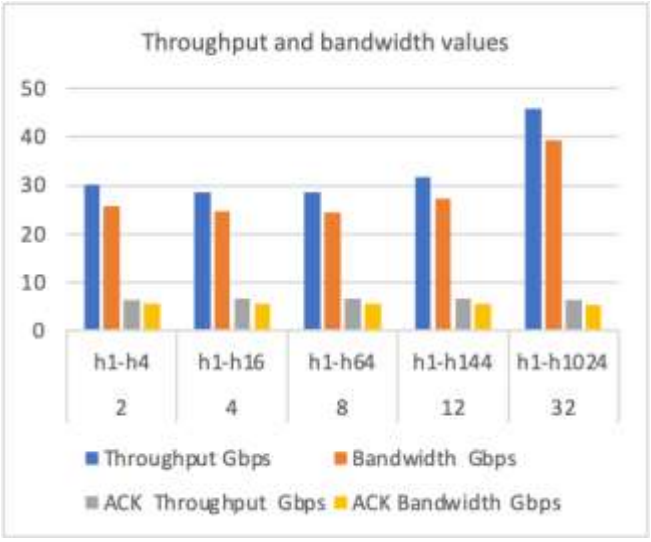

Fig. 10 – Throughput and bandwidth values

Based on the observed data:

The throughput values generally increase as the number of switches and nodes increases, which is expected as more network resources are available to handle data traffic.

However, there is a notable decrease in throughput observed when transitioning from 12 switches to 32 switches, which may indicate a bottleneck or saturation point in the network. Further investigation is needed to determine the cause of this decrease.

Similar to the throughput, the bandwidth values generally increase with the number of switches and nodes.

The bandwidth values closely mirror the throughput values, which is expected as throughput is a measure of the amount of data transferred over time, while bandwidth represents the maximum data transfer rate.

The ACK throughput and ACK bandwidth values remain relatively consistent across different configurations of switches and nodes.

This consistency suggests that the network is effectively handling acknowledgment packets, which are critical for ensuring reliable data transmission.

Overall, the provided values indicate that the network is performing reasonably well in terms of throughput, bandwidth, and acknowledgment packet handling. However, the observed decrease in throughput when transitioning to 32 switches warrants further investigation to identify and address any potential network bottlenecks or limitations.

Based on the provided data, the performance metrics such as throughput, bandwidth, ACK throughput, and ACK bandwidth demonstrate the effectiveness of the IoT network using SDN in handling TCP traffic. The increasing values of throughput and bandwidth with the number of switches and nodes indicate scalability and resource utilization improvements in the network. Additionally, the consistent values of ACK throughput and ACK bandwidth suggest that acknowledgment packets are being efficiently processed and transmitted across the network.

However, the notable decrease in throughput observed when transitioning from 12 switches to 32 switches warrants further investigation. This decrease could be indicative of network congestion or resource saturation, highlighting a potential bottleneck that needs to be addressed. Further analysis, such as identifying specific network components or configurations causing the bottleneck, is necessary to optimize network performance.

The data shows a notable increase in both average and maximum RTT as the number of switches and nodes in the network grows. This suggests that larger networks may experience higher latency and variability in communication, which could impact the overall performance and responsiveness of IoT applications.

The maximum deviation in RTT values across different configurations indicates the presence of significant variability in network latency

The observed spike in maximum RTT and deviation for the configuration with 32 switches and 1024 nodes highlights potential scalability challenges in large-scale SDN-enabled IoT networks. As the network size increases, managing and optimizing communication becomes more complex, leading to higher latency and greater variability.

Moving forward, future research could focus on several aspects:Identifying Bottlenecks To investigate the root cause of the throughput decrease observed with 32 switches. This could involve analyzing network traffic patterns, identifying congested links or nodes, and optimizing resource allocation strategies.

Table 3. – Summary of the outcomes of the ping command run between two hosts for 10 packets

No. of switches	No. of nodes (hosts)	Total RTT ms	Min RTT ms	Avg RTT ms	Max RTT ms	Max Deviation
2	h1-h4	9209	0.076	0.165	0.509	0.119
4	h1-h16	9213	0.086	0.167	0.446	0.095
8	h1-h64	9186	0.078	0.149	0.367	0.076
12	h1-h144	9198	0.119	0.156	0.325	0.056
32	h1-h1024	9354	0.138	4.14	20.88	7.739

Optimization Techniques: To develop and implement optimization techniques to improve network performance and scalability. This may include load balancing algorithms, traffic engineering strategies, or enhanced resource allocation mechanisms.
Dynamic Adaptation: To explore the feasibility of dynamic network adaptation mechanisms that adjust network configurations in real-time based on traffic patterns, resource availability, and performance metrics. This could involve leveraging machine learning algorithms or adaptive control mechanisms to optimize network efficiency.
Resilience and Fault Tolerance: To enhance the network's resilience and fault tolerance capabilities to mitigate the impact of failures or network disruptions. This could involve implementing redundancy mechanisms, failover strategies, or proactive fault detection mechanisms.
Security Considerations: To integrate robust security mechanisms into the SDN-based IoT network to protect against potential cyber threats and attacks. This may include encryption protocols, access control mechanisms, or intrusion detection systems.

Conclusion

This paper focuses on the evaluation of large IoT network performance through the Ryu Controller i.e. taking into consideration 1024 hosts and 33 switches using a tree topology on Mininet, ensuring optimal performance and monitoring for the IoT applications.
Results are obtained after the implementation is evaluated by various parameters like the bandwidth and Round-Trip Time (RTT). Iperf was used for testing the TCP/UDP server or client for host h1 and host h1024 to estimate the performance of the Ryu controller in mininet using the tree topology. It was observed that both throughput and bandwidth generally increased with the scaling of the IoT network however a decrease was observed in the throughput while scaling beyond a point. An increase in the average and maximum round trip time was observed however maximum deviation in RTT indicated that larger networks may experience network congestion or resource saturation leading to higher latency and variability in communication.
Overall, the future scope of research should focus on optimizing network performance, enhancing scalability and flexibility, improving fault tolerance, and addressing security challenges to ensure the reliable and efficient operation of SDN-enabled IoT networks and using deep learning using the different SDN Controllers for designing and monitoring large networks.

References

1. Aicha Idriss Hentati, Lamia Chaari Fourati, Lobna Richen, Ahmad Alaniz, "Multi-UAVs-based SDN, IoT, and Cloud Architecture for Hostile Areas Supervision," in 2023 15th International Conference on Developments in eSystems Engineering (DeSE), 2023, pp. 1-6, doi: 10.1109/APNOMS.2023.9874489.
2. Farhady, H., Lee, H., & Nakao, A. (2015). Software-defined networking: A survey. Computer Networks, 81, 79–95.
3. Sharif, A., Li, J. P., & Sharif, M. I. (2019). Internet of Things network cognition and traffic management system. Cluster Computing, 22(6), 13209–13217.
4. H. -K. Lim, J. -B. Kim, S. -Y. Kim and Y. -H. Han, "Federated Reinforcement Learning for Automatic Control in SDN-based IoT Environments," 2020 International Conference on Information and Communication Technology Convergence (ICTC), Jeju, Korea (South), 2020, pp. 1868-1873, doi: 10. 1109/ICTC49870.2020.9289245.
5. R. S. Alonso, J. Prieto, F. d. La Prieta, S. Rodríguez-González and J. M. Corchado, "A Review on Deep Reinforcement Learning for the management of SDN and NFV in Edge-IoT," 2021 IEEE Globecom Workshops (GC Wkshps), Madrid, Spain, 2021, pp. 1-6, doi: 10.1109/GCWkshps52748.2021.9682179.
6. N. Lo and I. Niang, "SDN-based QoS architectures in Edge-IoT Systems: A Comprehensive Analysis," 2023 IEEE World AI IoT Congress (AIIoT), Seattle, WA, USA, 2023, pp. 0605-0611, doi: 10.1109/AIIoT58121.2023.10174349.
7. Bhardwaj, S., & Panda, S. N. (2020). A study on noninvasive body wearable sensors. In Intelligent communication, control and devices (pp. 345–351). Springer, Singapore.

8. Phemius, K., Bouet, M., & Leguay, J. (2014). Disco: Distributed multi-domain sdn controllers. In 2014 IEEE network operations and management symposium (NOMS) (pp. 1–4). IEEE.
9. Akyildiz, I. F., Lee, A., Wang, P., Luo, M., & Chou, W. (2014). A roadmap for traffic engineering in SDN-OpenFlow networks. Computer Networks, 71, 1–30.
10. Agarwal, S., Kodialam, M., & Lakshman, T. V. (2013). Traffic engineering in software defined networks. In 2013 Proceedings IEEE INFOCOM (pp. 2211–2219). IEEE.
11. Tahaei, H., Salleh, R. B., Ab Razak, M. F., Ko, K., & Anuar, N. B. (2018). Cost effective network flow measurement for software defined networks: A distributed controller scenario. IEEE Access, 6, 5182–5198.
12. Bawany, N. Z., & Shamsi, J. A. (2019). SEAL: SDN based secure and agile framework for protecting smart city applications from DDoS attacks. Journal of Network and Computer Applications, 145, 102381.
13. Bevi, A. R., Shakthipriya, P., & Malarvizhi, S. (2019). Design of software defined networking gateway for the internet-of-things. Wireless Personal Communications, 107(2), 1273–1287.
14. Srivastava, V., & Pandey, R. S. (2020). A reward based formal model for distributed software defined networks. Wireless Personal Communications, 116, 691–707.
15. Freris, N. M. (2019). A software-defined architecture for control of IoT cyberphysical systems. Cluster Computing, 22(4), 1107–1122.
16. Suárez-Varela, J., & Barlet-Ros, P. (2018). Flow monitoring in Software-Defined Networks: Finding the accuracy/performance tradeoffs. Computer Networks, 135, 289–301.
17. Bhardwaj, S., & Panda, S. N. (2019). SDWSN: Software-defined wireless sensor networking. Internal Journal of Innovative Technology and Exploring Engineering, 8(12), 1064–1071.
18. Islam, M. T., Islam, N., & Al Refat, M. (2020). Node to node performance evaluation through RYU SDN controller. Wireless Personal Communications, 1–16, 15.
19. Queiroz, W., Capretz, M. A., & Dantas, M. (2019). An approach for SDN traffic monitoring based on big data techniques. Journal of Network and Computer Applications, 131(28–39), 16.
20. Badotra, S., & Panda, S. N. (2019). Evaluation and comparison of OpenDayLight and open networking operating system in software-defined networking. Cluster Computing, 1–11, 17.
21. Priya, A. V., & Radhika, N. (2019). Performance comparison of SDN OpenFlow controllers. International Journal of Computer Aided Engineering and Technology, 11(4–5), 467–479.
22. Bhatia, J., Dave, R., Bhayani, H., Tanwar, S., & Nayyar, A. (2020). Sdn-based real-time urban traffic analysis in vanet environment. Computer Communications, 149(162–175), 19.
23. Amin, R., Reisslein, M., & Shah, N. (2018). Hybrid SDN networks: A survey of existing approaches. IEEE Communications Surveys and Tutorials, 20(4), 3259–3306.
24. Singh, S., & Jha, R. K. (2019). SDOWN: A novel algorithm and comparative performance analysis of underlying infrastructure in software defined heterogeneous network. Fiber and Integrated Optics, 38(1), 43–75.
25. Akyildiz, I. F., Lee, A., Wang, P., Luo, M., & Chou, W. (2016). Research challenges for traffic engineering in software defined networks. IEEE Network, 30(3), 52–58.
26. Bholebawa, I. Z., & Dalal, U. D. (2018). Performance analysis of SDN/OpenFlow controllers: POX versus foodlight. Wireless Personal Communications, 98(2), 1679–1699.
27. Tootoonchian, A., Gorbunov, S., Ganjali, Y., Casado, M., & Sherwood, R. (2012). On controller performance in software-defined networks. In 2nd {USENIX} Workshop on Hot Topics in Management of Internet, Cloud, and Enterprise Networks and Services (Hot-ICE 12).
28. Wang, M. H., Chen, L. W., Chi, P. W., & Lei, C. L. (2017). SDUDP: A reliable UDP-Based transmission protocol over SDN. IEEE Access, 5(5904–5916), 23.
29. Yu, L., Wang, Q., Barrineau, G., Oakley, J., Brooks, R. R., & Wang, K. C. (2017). TARN: A SDN- based traffic analysis resistant network architecture. In 2017 12th international conference on malicious and unwanted software (MALWARE) (pp. 91–98). IEEE.
30. Khondoker, R., Zaalouk, A., Marx, R., & Bayarou, K. (2014). Feature based comparison and selection of software defned networking (SDN) controllers. In 2014 world congress on computer applications and information systems (WCCAIS) (pp. 1–7). IEEE.