TEAM 14

ECE/CSC 792 - Linux Networking

Project Milestone 3

Software Defined -Wide Area Networks for IoT

Aishwarya Subramanian
Chandana Veeraneni
Shruti Ashok Dalvi
Nikhila Nathani

Introduction:

SD-WAN for IoT

Software Defined Networking in Wide Area Network (SD-WAN) replaces the traditional WAN routers by decoupling data forwarding plane from control plane and applications in order to simplify management and operation of long distance networks. It also allows any combination of transport technologies such as Broadband Internet, MPLS or LTE as well as load sharing among multiple WAN connections, which make it more efficient.

Virtual Private Cloud topology is built based on the requirements given by the tenants. It facilitates the logical division of a service provider's multi-tenant cloud architecture. So that tenants can experience the benefits of a private cloud and exercise control over their virtual networks. They can utilize the provider's public resources and also maintain isolation from the other tenants.

Objective:

SD-WAN can be deployed for a wide range of use cases, including many IoT applications. It can provide the reliability, quality of service demanded by many critical IoT applications used in healthcare, retailers, smart cities and public safety. In the present world where enterprises are adding millions of IoT sensors and devices to their networks, organizations may find implementing and integrating IoT systems to be challenging due to the complexity of networking, securing and managing a diverse range of IoT devices. Considering the growth of the IoT sector, these enterprises will need a centralized mechanism to be able to manage all the devices, cloud storage and the networks they are part of.

The objective of this project is to provide an SD-WAN solution that extends support to important IoT requirements such as support for mobile connectivity of devices and support for device status and management. By implementing features such as traffic prioritization and location support for moving devices, we aim to ensure critical IoT applications can utilize prioritized WAN bandwidth. In summary, SD-WAN creates a more efficient traffic flow between offices, IoT devices and data centers, while maintaining security and fluid scalability.

Implementation Architecture:

In this project, we have a container-based environment. The infrastructure consists of two hosts, where both the hosts represent two different provider environments. The tenant is provided with the ability to design their own VPC topology through input json file.

Containers provide isolated systems on a single server or host OS. It is especially useful when software has to run predictably when moved from one server environment to another. Containers sit on top of a physical server and its host OS, for example, Linux or Windows. Each container shares the host OS kernel and, usually, the binaries and libraries, too. Shared components are read-only. Containers are thus exceptionally light. They are only megabytes in size and take just seconds to start, versus gigabytes and minutes for a VM.

Containers also reduce management overhead. Because they share a common operating system, only a single operating system needs care and feeding for bug fixes, patches, and so on. This concept is similar to what we experience with hypervisor hosts: fewer management points but slightly higher fault domain. In short, containers are lighter weight and more portable than VMs.

Dockers were used to manage the creation and deletion of containers. We built the docker container for the project with an image named “all_in_one” as it has all the basic files which a container uses to establish connectivity between other containers, hosts and other L2 devices.

The key aspects provided in this overview topology: L3 Isolation using Container, L2 Isolation using Bridges in Route mode within a Tenant’s subnets, GRE connectivity to provide tunneling across containers in different subnets and in different hypervisors and a management network and controller containers for management activities.

L3 Isolation Among Tenants: To provide L3 isolation for each tenant, two dedicated containers are created. All network bridges of the same tenant are connected to the first container. GRE tunneling for L3 connectivity implemented between containers across hypervisors.

L2 Isolation Among Tenants: To provide L2 isolation between subnets of the same tenant, a dedicated bridge in forward mode is used for each subnet of the tenant.

GRE Connectivity: GRE tunnel interfaces are created inside containers that provide L3 connectivity among different subnets of the same tenant across the hypervisors. This allows the tenant to preserve their private IP addresses by encapsulating the traffic in the tenant's provider-side IP address while the traffic goes through the provider’s infrastructure. GRE tunnel interfaces are also created in the hypervisors to preserve the IP addresses as the traffic goes out of the provider’s infrastructure.

Management Network and Controller: A dedicated management network and a Controller container is created with respect to each tenant. Each tenant container has two interfaces - one of them is for hosting its applications/services and the other interface is solely for the purpose of management activities. All the containers, Provide Edge Containers and Customer Edge Containers within a Tenant can be reached from the Controller container over the management network, where it communicates with them via a Controller Bridge.

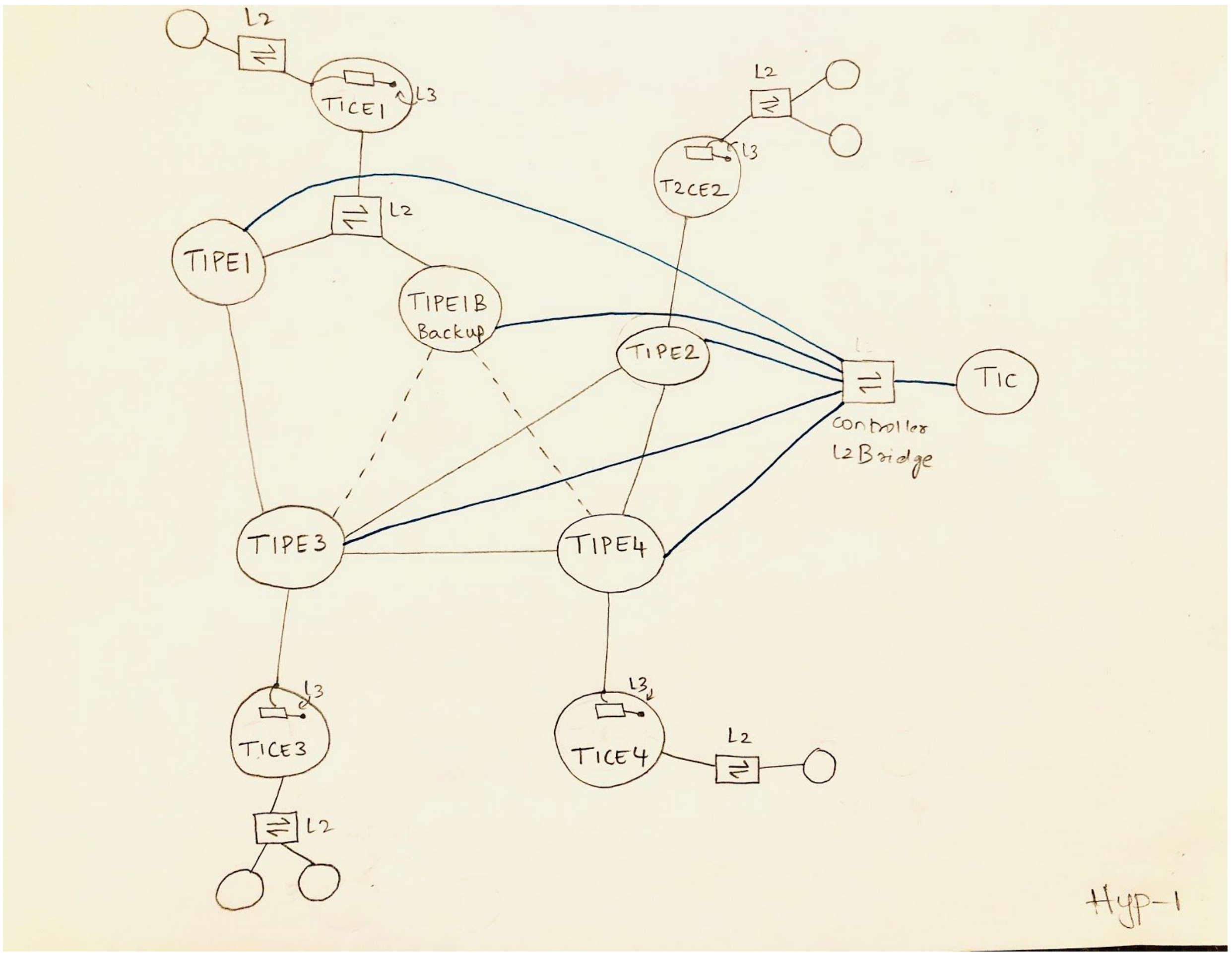
Separation of Logic Layer and Southbound Layer:

In the NorthBound layer, We are receiving input from the tenants in JSON format in the Northbound layer. The requirements for creating a VPC are specified by the tenant. Then the automation scripts are run accordingly to create the VPC as customized by the tenant.

In the Logic layer, we are organising the input we get from the tenant in a logical way and call the ansible scripts from the southbound layer, wherever they are required. We parse the variable in an argument we have written in the southbound as per tenant requirement. For example, we create the containers based on the VPC and subnet requirement.

In the SouthBound layer, we have separately written the ansible scripts for all the functions that must be called for every functionality required by the user and to create isolation between the tenants irrespective of the hypervisors and the subnets. It has scripts for creating and deleting individual components linke bridges, networks etc.

Infrastructure Topology:



A few words about our topology:

As the tenant requests for a vpc, we are creating a container with all_in_one image and connecting the container with one L2 bridge as we are considering one subnet. We are connecting the L2 bridge to a Customer Edge container and providing L3 connectivity inside the container with the help of Route mode bridge and assigning the default gateway IP address.

We are attaching the Provider Edge containers to the Customer Edge inorder to connect the Provider network to the tenant requested topology as we must enable communication between multiple sites wherever the tenant holds his container.

Controller implements control path tasks via a controller bridge. As seen in the above topology, the dark links represent the control path links between different provider edge containers. The remaining links represent the data path links between all the containers and devices in the system.

Infrastructure Automation:

We've used a combination of Python script and Ansible to parse the JSON input file and set up the containers, bridges, networks and connections between the different components using veth peer.

We have split our automation in three parts on calling a single python script:

- Creating infrastructure
- Assigning IP addresses
- Providing tunneling functionality

Details on the python script:

- We are running a python script named "new_builder.py" that in turn calls the "tenant.json" which is used to get input information from the tenant such as VPC name, Tenant ID, Number of subnet the tenant wants in the VPC.
- On calling the new_builder.py, the following infrastructure is getting created:
  1)Customer Edge Container
  2)Provider Edge Container
  3)Provider Edge Backup Container (for providing location support functionality)
  4)Subnet loop : Creation of L2 bridge , Creation of L3 bridge inside the CE container, Providing dnsmasq functionality inside the CE to the L3 bridge and assigning default gateway to L3 bridge inorder to support multiple subnet communication.
  5)We are assigning IP addresses to the Customer Edge and the Provider edge containers. We are setting the backup Containers down as it will come up automatically only if the Regular PE container goes down and moves away from its location.
  6)We are providing GRE tunneling functionality in the topology for the multi location communication support. The GRE tunneling has been done both in CE and PE.
  For Provider Edge, we have created a shell script with hard coded addresses as it is under provider control. We are calling pe_container.sh to provide this functionality.
  7) There is a tunnel connected to all the backup PE's as well but it will come up only if the Actual PE fails.

We have split our southbound automation into following playbooks:

- Create_container.yml- to create a container with an already available image named all_in_one.
- Create_veth_container.yml- to create veth peer between any networking devices in the topology
- Attach_veth_to_l2bridge_container.yml- to create connection between containers and L2 bridge
- Attach_veth_to_container.yml- to attach one end of the veth to Container
- Attach_veth_to_routebr.yml- to attach the end connected to the Container to the L3 bridge inside the container.
- Create_gre_container- to provide tunnel functionality based on tenant requirements.

  We have all the containers to detach/destroy the functionality as well.

Functional Features:

TRAFFIC PRIORITIZATION FOR SD-WAN

The controller has visibility over the entire network thereby allowing the controller to pass messages, dictating data flows between different provider edge containers as well as customer edge containers. Hence it has the capacity to configure priorities on routes to different devices and direct the traffic accordingly. We are using iptable rules to implement this feature. In this case, when traffic is going to two destinations, one of them can be given preference of the other. The packets destined for the less preferred destination are dropped.

TRAFFIC RATE LIMITER

We are implementing Traffic Prioritization in our SD-WAN using ‘tc’ command for Traffic Control. We are specifically using the Token bucket filter queueing discipline (tc qdisc - tbf) to slow down the traffic on an interface to which a specific PE is connected to. Originally we are not setting any data limit on the traffic being sent from controller. Assume one PE is sending ping packets to two other PEs simultaneously. Ping packets are received in almost same amount of time at each PE.

Now, assuming the first has informed the controller that the rate at which it was originally receiving traffic should be reduced (as it is busy). We implemented a lower rate on the interface through which traffic is sent to this PE using tc qdisc rules. We are able to stagger the packets going towards the first PE, while the second PE receives them at a normal rate. It can be observed that it takes more time to receive all the ping packets at the first PE as compared to the second PE. Thus traffic is being sent at a set limit rate towards a particular destination.

Functional Features (IoT):

LOCATION SUPPORT FOR IoT DEVICES

Due to the mobile nature of IoT devices, there is a need to keep checking the status of these devices as they might not be used all the time; they might be powered off or something could go wrong and they could lose connectivity. To save bandwidth and other resources, when devices are powered off, the traffic they were intended to receive could be routed to other locations. Sometimes the provider itself can switch over traffic intending to one PE to another PE depending on the traffic it receives during that period of time. Tenants can also need to move their containers and devices from one location to another due to various requirements. So we are providing connectivity between different provider and container edge containers. We plan to turn off devices at random intervals to test the routing of traffic.

MANAGEMENT FEATURES

1) High Availability:

This feature is to ensure that the routing is done at all times, and it is to retain the connectivity between the devices, through the provider side routing devices. It can be done by duplicating another provider edge container, with same tunnelling as used in the original, and to populate the routing tables such that, the duplicate one is up and running, whenever the original provider edge container, goes down, or is not giving any connectivity. For example, we are providing a backup PE which will be populated with the necessary routes so that it can reach the current transit PE. So that in the event that any PE which we are using goes down, we can immediately start using the backup, without any interruption to our services.

2) Logging:

While the controller is communicating with the other provider containers, these actions are collected and stored as log files, keeping the stats in check, and monitoring it, as a checking mechanism. With this, we can track what actions are taking place and it is helpful in troubleshooting if some problems arise.