Chapter 4 System Analysis and Requirement

This chapter provides an in depth analysis of the UMJaNetSim network simulator. The chapter begins with the overview of UMJaNetSim architecture and the UMJaNetSim Application Programming Interface (API). The aim is to provide an explanation of the UMJaNetSim architecture.

The following section discusses the MPLS VPN architecture that run on MP-BGP. It is followed by an analysis of new components as well as the requirement to develop the MPLS VPN simulator.

The final section summarizes the details of this chapter. It summarizes the analysis of simulator as well as the MPLS VPN architecture that based on MP-iBGP.

4.1 UMJaNetSim Architecture

UMJaNetSim simulator is a flexible test bed for studying and evaluating the performance of MPLS network without the expense of building a real network. The simulator is written in JAVA Language whereby it is developed in object-Oriented programming approach. The UMJaNetSim architecture mainly consists of two important parts, namely the simulation engine and the simulation topology [42]. Figure 4.1 shows the overall view of the UMJaNetSim architecture.

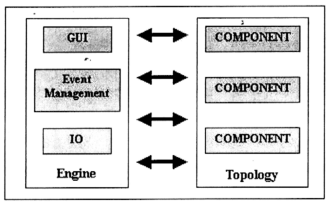

Figure 4.1 UMJaNetSim Overall Architecture

The simulation engine is the main controller of the entire simulation. It performs the event management task, GUI management as well as input and output processing such as topology saving and result logging.

The simulation topology consists of all the simulation objects, which are also referred to as simulation components. These simulation components are the main subject of a simulation scenario, and these simulation components typically consist of a group of interconnected network components such as router, switches, physical links and different type of source applications.

4.1.1 Event Management Architecture

JavaSim object is main object in the UMJaNetSim simulator. JavaSim object itself is the main component in the simulation engine. Under the event management architecture, the JavaSim object manages an event queue, an event scheduler, and a simulation clock. Figure 4.2 shows the event management architecture for the UMJaNetSim.

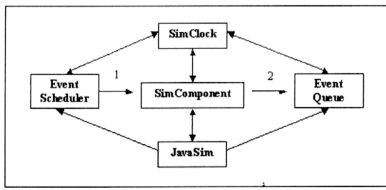

Figure 4.2 Event Management Architecture

Under the event management architecture, there are basically two main operations that are in process. First, a simulation component will schedule an event for a target component to be happening at a specific time using the enqueue operation. This target component can be others component or the component itself. Then, the events are stored inside a queue and are sorted by the event-firing time.

When a specific time is reached, the simulation engine will invoke the event handler of the target component. In this time, the event scheduler will fetch and remove the first event in the event queue. The target component will react to the event according to its behavior.

UMJaNetSim uses an asynchronous approach of the discrete event model, where any event can happen at any time. The SimClock object is the global time reference used by every component in the simulation and managed by the simulation engine. The simulation time in the UMJaNetSim is measured by tick and this tick can be converted to real time or vice versa.

4.1.2 GUI Management Architecture

The JavaSim object is the main controller under the GUI Management architecture. The functionality of the UMJaNetSim GUI management includes handling of user inputs; perform drawing as well as managing various on-screen windows. Besides that, there is a SimPanel object that keeps track of the latest set of simulation components and the interconnection among the components in order to present the simulation topology visually to the user. Thus, the new topology can be designed easily. Figure 4.3 shows the GUI management architecture for the UMJaNetSim.

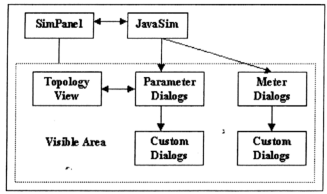

Figure 4.3 GUI Management Architecture

Under the GUI management architecture, there are two common types of dialog, namely the parameter dialogs and the meter dialogs. Each simulation component is associated with a parameter dialog that displays the list of parameter for the component. Meanwhile, a meter dialog is normally used for graphical display of a particular output value of a component such

as utilization as well as cell loss. Besides that, the two types of dialogs are also associated with one or many custom dialogs that show extra information about the information. These custom dialogs are normally the OSPF routing table, VPN detail and others information.

4.1.3 Simulation Components

The SimComponent object is the main simulation object in the UMJaNetSim. It is a well-defined base object with all the necessary interfaces that enable the interaction between the simulation engine and the component. There will be an event handler in every simulation component, which is invoked by the event scheduler in order to fire an event. All interactions between simulation components are achieved through the sending of messages in the form of a SimEvent.

Each of the SimComponent is associated with its properties that can be configured by using the parameter dialog. There are many types of parameters for each of the SimComponent. All of these parameters are objects that derived from a base object called SimParameter. By using the SimParameter, all type of values can be setup easily without any programming effort from the component designer.

4.2 UMJaNetSim API

After the study of the UMJaNetSim architecture, this section focuses on defining well-known interfaces for simulation objects. The UMJaNetSim API provides a consistent way of creating simulation components [42]. The discussion of the UMJaNetSim API will provide insights into some design issues of the simulator and prepare the ground for the actual creation of the MPLS VPN components in the next chapter.

4.2.1 JavaSim

The JavaSim object is the main object in the UMJaNetSim simulator. JavaSim object manages an event queue, an event scheduler and a simulation clock during the event management and provides all GUI functions together with SimPanel under the GUI management. The followings are some of the important method of the JavaSim object.

- java.util.List getSimComponents() – This method returns a list of all existing SimComponent.
- long now() – This method returns current simulation time (in tick).
- boolean isCompNameDuplicate(String name) – This method ensures no duplicated name for SimComponent.
- void notifyPropertiesChange(SimComponent comp) – This method executes whenever there are structural changes to the parameters.
- void enqueue(SimEvent e) – This method will enqueue an event that has been invoked.
- void dequeue(SimEvent e) – This method will dequeue an event when a specific time is reached for that event to be executed.

4.2.2 SimEvent

Every SimComponent communicates with each other by enqueuing SimEvent for the target component. For example, when component A wants to send a cell to component B, component A creates a SimEvent that specifies B as its destination, and enqueue the event. The SimEvent object also contains a time so that this event is fired at exactly the specified time.

There are two types of events, namely the public (well-known) events and private events. Public events are defined in the SimProvider object. This event can be enqueued for itself and or for another SimComponent. All private events are defined within the particular SimComponent source itself and can only be enqueued for itself. The followings are some of the important method of the SimEvent object.

- SimComponent getSource() – This method retrieves the source SimComponent.
- SimComponent getDest() – This method retrieves the destination SimComponent.
- int getType() – This method retrieves the event type.
- long getTick() – This method retrieves the event-firing time.
- Object [] getParams() – This method retrieves the event parameters.

4.2.3 SimClock

The SimClock object is the global time reference used by every component in the simulation and managed by the simulation engine. The followings are some of the important method of the SimClock object.

- static double Tick2Sec(long tick) – This method converts ticks to seconds.
- static double Tick2MSec(long tick) – This method converts ticks to milliseconds.
- static double Tick2USec(long tick) – This method converts ticks to microseconds.
- static long Sec2Tick(double sec) – This method converts seconds to ticks.
- static long MSec2Tick(double msec) – This method converts miliseconds to ticks.
- static long USec2Tick(double usec) – This method converts microseconds to ticks.

4.2.4 SimComponent

The SimComponent object is the main simulation object in the UMJaNetSim from the topology view. Every components such as link, broadband terminal equipment, application, switch and other network components must inherit this base class in order to obtain the capability to interact with the simulation engine.

Every SimComponent has a reference to the main object of the simulation engine, the JavaSim object, in order to access services provided by the simulation engine. Every SimComponent also maintains a list of all its external parameters and a list of all its neighbors. The neighbors of a component are all components that are directly connected to the component. The followings are some of the important properties and method of the SimComponent object.

- protected transient JavaSim theSim – This property is a reference to the main JavaSim object.
- protected java.util.List neighbors – This is a list of all (directly connected) neighbors of the SimComponent
- protected java.util.List params – This is a list of all external parameters of the SimComponent

- Object [] compInfo(int infoid,SimComponent source, Object [] paramlist) - This method provides a way for inter-component information exchange without sending run time events.
- boolean isConnectable(SimComponent comp) – This method is called by the simulation engine when a new component is about to be connected to this component. This method will verify whether the new component can be connected to this component.
- void addNeighbor(SimComponent comp) – This method is called by the simulation engine when a new component is connected to this component.
- void removeNeighbor(SimComponent comp) – This method is called by the simulation engine when a new component is disconnected to this component.
- void copy(SimComponent comp) - This method is used to copy parameter values from another SimComponent of the same type.
- void reset() – This method performs a reset operation in order to bring the status of the component back to the same status as if it is just newly created.
- void start() – This method performs any operations needed when the simulation starts.
- void resume() – This method perform any operations needed when the simulation need to be resume. One possible use is to capture any special changes /that have been done by the user during the pause period.
- void action(SimEvent e) - This is the event handler of this component, and will be called by the simulator engine whenever a SimEvent with this component as the destination fires.

4.2.5 SimParameter

Every SimComponent has internal parameters or externål parameters, which can be shown or accessible by users. All external parameters must inherit SimParameter. By extending SimParameter, the components obtain parameter logging and meter display features automatically. The parameter can simply holds one single value, such as an integer. It also can represent a complex piece of information, such as the entire routing table of a network router.

In some cases, the parameter itself can create and manage additional custom dialogs. A complex parameter type may just use a JButton that opens up new custom dialogs when invoked. The choice of components to use is dependent on the type of interaction needed by

the component designer. The followings are some of the important method of the SimParameter object.

- String getString() – This method returns a String representation of the parameter value. This is used for logging purpose.
- void globalSetValue(String value) – This method supports setting of the same parameter values for multiple components in one command.
- int getValue() – This method will read a value.
- void setValue(int val) – This method will write a value.
- JComponent getJComponent() – This method will return a Java swing component and its name.

4.3 MPLS VPN Architecture

In chapter two, a brief introduction on MPLS, VPN, MPLS VPN and MP-iBGP have been discussed. This section will provide a more detailed description on MPLS VPN that based on MP-iBGP. Besides that, it will cover the basic configuration steps that are necessary for MPLS VPN architecture.

One of the simplest VPN topologies using the MPLS VPN architecture is an Intranet between multiple sites that belong to the same organization. In order to provision the VPN service across the MPLS VPN backbone, a number of steps need to be followed. [1]

- Define and configure the VRFs.
- Define and configure route distinguishers.
- Define and configure the import and export policies.
- Configure the PE-CE links.
- Configure the MP-BGP.
- Packet Forwarding

4.3.1 Configuration of VRFs

The first step in provisioning a VPN service based on MPLS architecture is to define and configure the Virtual Routing and Forwarding Instances (VRFs). Under this process, a VRF name is defined for each VPN in a PE router. The VRF name is case sensitive. After the VRF is defined, further configuration is needed to provide routes for the tables and to create associated MPLS labels.

4.3.2 Configuration of Route Distinguishers

After the VRFs have been defined, the route distinguishers need to be configured. In the MPLS VPN architecture, each VPN must be capable of using the same IP prefixes as other VPNs as long as no communications between each other. Thus, it is necessary to prepend a route distinguisher to the Ipv4 address to make it unique. However, the route distinguisher mechanism doesn't allow multiple customers within the same VPN using the same addressing scheme within their sites.

The route distinguisher is a sequence of 64 bits and is different for each VPN. The route distinguishers are encoded as Figure 4.4. When the type field is zero, the value field will contain 2 bytes of administrator sub field and 4 bytes of assigned number sub field. The administrator sub field must contain the Autonomous System Number (ASN) and the assigned number sub field contains a number from a numbering space provide by the service provider. When the type field is one, the value field must contain 4 bytes of administrator sub field and 2 bytes of assigned number sub field. The administrator sub field must contain an IP address and the assigned number sub field contains a number from a numbering space provide by the service provider. Example of a route distinguisher based on type 0 is 100:26 [26].

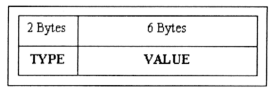

Figure 4.4 Route Distinguisher Structure

4.3.3 Configuration of Import and Export Policies

The routes are advertised across the MPLS VPN backbone through the use of MP-iBGP and any routes learned via MP-iBGP are placed into the VRFs of interested parties. Thus, extra information is needed by the PE router to process any route that it received.

Although the route target provides the mechanism to identify which VRFs should receive the routes, it does not provide a facility that can prevent routing loop. In order to prevent routing loop where routes learned from a site are advertised back to that site, the site of origin (SOO) identity is needed. Figure 4.5 illustrates the functionality of SOO. The figure shows that BPK PE-router receives an MP-iBGP update for 202.185.107.0 from the FSKTM PE-Router. This update contains a SOO of 1:13 that same as the one which configured for the UM VRF on the BPK router. Thus, the route is not advertised to the UM BPK Router.

The last step in the configuration of each VRF is the addition of import and export policies for the VRF to use. The route target must be configured to specify the routes. The import policies will import specific route target value into the VRF and the export policies will export or advertise the route target value to other PE routers.

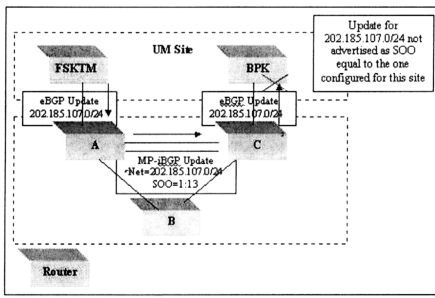

Figure 4.5 SOO BGP Extended Communities

The BGP extended community attribute contains the site origin and the export and import route target. Each of the extended community attributes is encoded in 8 bytes. The first two bytes define the attribute type and the next six bytes hold the value of attribute. The route target extended community has a type code of 0x0002 or 0x0102. The SOO has a type code of 0x0003 and 0x-0103. Figure 4.6 shows the structure of the BGP extended community attribute format.

4.3.4 Configuration PE-CE Links

To provide a VPN service, the PE router needs to be configured so that any routing information learned from a VPN customer interface can be associated with a particular VRF. If the routes learned across an interface are associated with the particular routing protocol, the routes are installed into the associated VRF. If the routes don't exist, the routes are placed in the global routing table.

Under the PE-CE link configuration, there are basically four types of configurations, namely the static routing, RIP version 2, standard BGP-4 and the OSPF. This project only concerns with the static routing configuration. The static routing is a good choice for deployment when the site only has one entry point into the service provider network. This is because there is little to be gained by dynamically learning the customer site via PE-CE link.

A static route for every network beyond the CE-router must be configured into the correct VRF rather than the global routing table on the PE-router that connects the site. This is to make sure that the address is not valid across multiple routing tables.

4.3.5 Configuration of MP-BGP

MP-BGP is an extension of the existing BGP-4 protocol to advertise customer VPN routes between PE routers that learned from connected CE routers. All MP-BGP sessions are internal BGP sessions because the sessions are between two routers that belong to the same autonomous system. This MP-iBGP update contains the VPN-IPv4 address, MPLS label information and the extended BGP communities.

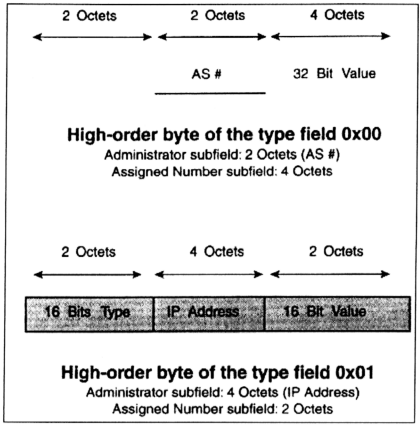

Figure 4.6 BGP Extended Community Attribute Format

When a BGP session is established between two peers, an OPEN message exchanges initial BGP parameters, such as the autonomous system number used by the BGP neighbors. The OPEN message can contain optional parameter. When the multiprotocol extension is introduced, two new optional attributes are introduced. The attributes are multiprotocol reachable NLRI and multiprotocol unreachable NLRI. The MP_REACH_NLRI announces new multiprotocol routes and it carries a set of reachable destinations together with the next hop information to be used for forwarding to these destinations. MP_UNREACH_NLRI revokes the routes previously announced by the MP_REACH_NLRI and it carries a set of unreachable destinations.

When a PE-router sends an MP-iBGP update to other PE-routers, the MP_REACH_NLRI will contains some attribute like addressing family information, next hopping information and NLRI. The address family information is set to AFI=1 and sub-AFI = 128 in MPLS VPN. The next hop information will be the next-hop address information of the next router on the path

to the destination. In MPLS VPN, the next hop information is the advertising PE-router. The NLRI will contain the length of label plus the RD, the 24 bits label and the route distinguisher plus the IPv4 prefix.

After the VRFs have been defined, each VRFs inject routes into the BGP and advertised this route through MP-iBGP across the MPLS VPN backbone. Before these routes are imported into any VRFs, the BGP has to make some decision. First, all the relevant routes are grouped to compare. Before the PE-router can select routes, it has to know which VPN routes exist and which of the routes should be comparable with each other by the BGP selection process. When the PE-router receives the MP-iBGP update message, it will take all routes with the same route target as any of the import statements within the VRF. Then, it will consider all routes that with the same route distinguisher as the one assigned to the VRF being processed. Finally, the PE router creates new BGP paths with a route distinguisher that is equal to the route distinguisher configured for the VRF that is being processed.

After the selection process is executed, the import process will import routes into all VRFs and filter any unwanted routing information from the particular VRFs. Figure 4.7 shows the MP-iBGP filtering. The PE router only accepts the route target that has been configured in the import route target list for that VRFs.

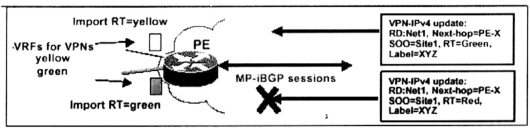

Figure 4.7 MP-iBGP Updates Filtering

4.3.6 Packet Forwarding

After all the VPN routes are propagated across the MPLS VPN backbone, the data packet can be forwarded to the destination based on VPN label and the MPLS label. PE routers now store two kinds of labels in their LFIB. The first labels are learned through the TDP/LDP protocol and assigned to IGP routes and the second labels learned through MP-BGP and assigned to

VPN routes. The first labels are called IGP Labels and the second labels normally called VPN Labels. Figure 4.8 shows how the IP packet is forward across the MPLS VPN backbone.

First, the ingress PE (PE1) will receive normal IP Packets from CE router (CE1). PE1 router does "IP Longest Match" from VRF and find iBGP next hop PE2 and impose a stack of labels which are the exterior label or VPN label and interior label or IGP label. Then, the IP packet will reach the P1 router based on the IGP label. When P1 router receives the IP packet, it will switch the packets again and swap to another IGP label to P2.

When P2 receive IP packet, it will lookup the IGP label and switch to the next router. P2 is the penultimate hop for the BGP next-hop. Thus, P2 will remove the top label and only left the VPN label. P2 will send the IP packet to the PE2 router. When PE2 router receives the IP packet, it will perform single lookup based on the label corresponding to the outgoing interface. Then PE2 will pop the VPN label and send the IP packet to the destination

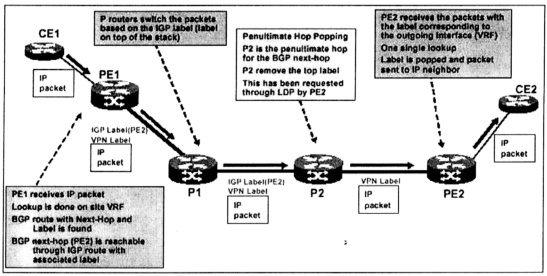

Figure 4.8 MPLS VPN IP Packet Forwarding Process

4.4 System Requirement

The system requirement to develop the MPLS VPN simulator is categorized into functional requirement and non-functional requirement. The following section will discuss the functional requirement and non-functional requirement of the MPLS VPN simulator.

4.4.1 Functional Requirement

This section describes the functional requirement of the MPLS VPN simulator.

- The simulator will allow the user to configure the VRFs. The user will be able to configure the route distinguisher, import and export policies.
- The simulator will allow the user to configure PE-CE links.
- The simulator will allow the user to configure MP-iBGP.
- Each PE-router will be able to setup many VRFs. The PE-router should be able to connect to multiple VPNs at one site.
- The simulator will run MP-iBGP during the advertising of the customer VPN routes.
- The simulator will support different types of VPN topologies, such as Intranet topology, Intranet and Extranet Integration topology and Central Services topology.
- P router will know how to perform label lookup as well as pop up label for MPLS VPN and non MPLS VPN.
- The simulator will perform label stacking.
- PE-Router will perform forwarding table lookup according to VPN and non-VPN.
- Logging Process
  - The system should be able to record the VRFs that have been setup.
  - The system should keep track or log all the process that are running while the simulator is running.

4.4.2 Non-Functional Requirement

This section describes the non-functional requirement of the MPLS VPN simulator.

- **Reliability**
  - The system should be reliable in performing its simulation functions and network operations. For example, whenever a button is clicked, the system should be able to perform some functionality or generate some message or animation to inform the user what is happening.

- **Usability**
  - The system should be user friendly. It will enhance and support rather than limit or restrict the understanding of MPLS VPN in MPLS network.
  - Human interfaces need to be intuitive and consistent with the user knowledge in order to let them gain some knowledge through the simulator.
- **Manageability**
  - The modules within the system should be easy to manage. This will make the maintenance and enhancement works simpler and not time consumed.
  - The system should not cause any damages to the current simulator after a new component has been added.
- **Flexibility**
  - The system should have the capabilities to take advantage of new technologies and resources. The system should be able to implement in a changing environment or platform.

4.5 Chapter Summary

This chapter covers the major analysis on the key features of UMJaNetSim simulator. The overall architecture of UMJaNetSim is analysed in order to find out how the new MPLS VPN simulator components can be integrated into the existing simulator without affecting the performance and the output of the simulator.

The process of configuring MPLS VPN network also covered in this chapter. It provides a good understanding of the MPLS VPN architecture as well as the steps required to configure the MPLS VPN.

This chapter concludes by presenting the functional requirements and non-functional requirements of the simulator. Details of system design will be discussed in the following chapter.