Design of Metro Ethernet Customer Edge using Virtual LAN (VLAN)

M.Tech Dissertation

Submitted in partial fulfillment of the requirements
for the degree of

Master of Technology

in
Communication Engineering

by

Hemant Kumar Rath
Roll No: 02307404

under the guidance of

Prof. Abhay Karandikar

Department of Electrical Engineering,
Indian Institute of Technology - Bombay,
Mumbai, 400076,
July - 2004

Acknowledgment

I would like to express my sincere gratitude towards my guide **Prof. Abhay Karandikar** for his motivation, guidance and ideas he provided during the project. I would also thank **Prof. R. N. Biswas** and UACT, IIT-Madras, for sponsoring me for the M.Tech program at IIT-Bombay. I thank the entire Eisodus team and especially Mr. Abhijit Gadgil, Mr. Praveen, Mr. Balaji, Mr. Zainul and Ms. Ranoo for their help during the project. Also, I would like to thank my professors, library staff, my friends at Infonet Lab and my batch-mates who directly or indirectly helped me in the course of entire project period. Above all, I thank my parents and my family members for their support and encouragement I needed for this.

Hemant Kumar Rath
02307404

Abstract

Metro Ethernet Network is a network that provides Ethernet services over a Wide Area Network (WAN). It comprises of a Customer Edge (CE) device and a Provider Edge (PE) device.

The CE device in our design is a *uClinux* based Embedded Platform. Here, we have implemented Generic Attribute Registration Protocol (GARP)/ GARP VLAN Registration Protocol (GVRP) to dynamically assign Virtual LANs (VLANs) to customers. GARP/GVRP implementation in our project includes timers, registrar and applicant state machine, VLANid manipulation and event scheduler etc.

Also, we have discussed various Layer-2 Virtual Private Network (L2VPN) services such as Virtual Private Wired Services (VPWS) and Virtual Private LAN Services (VPLS) in this report. Both VPLS and VPWS are defined in Pseudo Wire Emulation Edge-to-Edge (PWE3) framework. We have included the signaling techniques between PE and CE, end-point detection, forwarding schemes, Attachment Circuit provisioning for both VPWS and VPLS in our discussion in this report. Further, we have implemented the signaling between PE and CE and encapsulation techniques for both the PE-bound and CE-bound traffic in a PWE3 framework. Towards the end of this report, we have discussed various test methods for our CE device. A CE device like ours can be used as a building block to provide various L2VPN services.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Metro Ethernet Network [1] is a network that provides Ethernet services over a Wide Area Network (WAN). Since, it uses Ethernet as the core protocol, it requires redesign of access and core network. Access network is the part of the communication infrastructure that connects the service provider's central office (CO) to a business or residential subscriber whereas core network is the service provider's network connecting COs over an IP or Multi Protocol Label Switch (MPLS) backbone. Use of Ethernet as the core protocol can provide *fast, simple, reliable, inexpensive, inter-operable, and ubiquitous* connectivity to the users as well as to the service providers. As a result of this, though the servers and users are physically in geographically diverse locations, they appear to reside on the same LAN [2].

Metro Ethernet Network services can be classified into point-to-point, point-to-multipoint or multipoint-to-multipoint type. These kind of services are Layer-2 Virtual Private Network (L2VPN) services. To provide L2VPN services, service providers use IP or MPLS in the core network. To make these services scalable, they use Ethernet's Virtual Local Area Network (VLAN) concept, where logically distinct partitioning of user groups over the same physical network is possible. But, there can be only 4096 VLAN tags [3] possible and is insufficient for a large service provider. As a solution to this, they use multiple VLAN tags (VLAN stacking). This is known as hub and spoke architecture in L2VPN.

There are different approaches followed by various service providers to provide L2VPN services. But, it needs standardisation of the framework and the service-specific techniques for different services. Pseudo Wire Emulation Edge-to-Edge (PWE3) work group is defining these kind of services. *Virtual Private Wired Services (VPWS)* and *Virtual Private LAN Services (VPLS)* are such kind of services defined by PWE3. VPWS provides point-to-point services, whereas VPLS provides point-to-multipoint services. We have discussed these services in detail in our report

1.1 Definitions

1. **Attachment Circuit (AC):** This is the physical or virtual circuit attaching a Customer Edge device to a Provider Edge device [4]. An AC can be a Frame Relay DLCI, an ATM VPI/VCI, an Ethernet port, a VLAN, a HDLC link, a PPP connection on a physical interface, a PPP session from an L2TP tunnel or an MPLS-LSP.

2. **Customer Edge (CE):** The device whose one end originates and/or terminates a particular service (connected to customers [1]) and other end connects to the Provider Edge device (connects to service provider) to emulate the above service is known as a Customer Edge device. It is a customer located device [2].

3. **Generic Attribute Registration Protocol (GARP):** GARP implements a service that provides a generic Layer-2 (L-2) transport mechanism to propagate information throughout a switching domain [3]. It supports mechanisms for providing dynamic group filtering and registration [5].

4. **GARP Information Declaration (GID):** GID is the declaration service offered by GARP for declaring the membership information within a bridged LAN.

5. **GARP Information Propagation (GIP):** GIP is the propagation service offered by GARP for propagation of membership information within a bridged LAN.

6. **GARP VLAN Registration Protocol (GVRP):** GVRP is a GARP application that allows bridges to dynamically register and propagate VLAN membership information throughout a switching domain. GVRP relies upon the services provided by GARP and 802.1Q VLAN tagging [3].

7. **Provider Edge (PE):** The device whose one end connects to a CE to provide the services needed by CE and the other end is a part of the service provider's network is known as a Provider Edge device. This is the interface between the edge and the service provider network.

8. **Packet Switched Network (PSN):** It is a network designed to carry data in the form of packets (variable/fixed length),e.g., IP and MPLS network.

9. **Pseudo-Wire (PW):** It is a mechanism that carries the essential elements of an emulated circuit from one PE to another PE over a Packet Switched Network (PSN).

[1] customers are essentially end users and not in commercial sense
[2] can be placed by customer or by service provider
[3] broadcast domain

10. **Pseudo-Wire PDU:** A Protocol Data Unit (PDU) sent on a PW containing data and control information required to emulate the desired service is known as Pseudo-Wire PDU.
11. **Virtual LAN (VLAN):** It is a group of devices on different physical LAN segments, which can communicate with each other, as if they are in the same physical LAN.
12. **Virtual Private LAN Services (VPLS):** It is a point-to-multipoint VPN service where we can emulate the full functionality of a traditional LAN. A VPLS makes it possible to interconnect several LAN segments over PSN to form a single LAN.
13. **Virtual Private Wired Services (VPWS):** It is a point-to-point virtual circuit (link) connecting two remote CE devices. The CE connects to PE through an AC, whereas PEs are connected through PWs.

1.2 PWE3 Protocol

Pseudo Wire Emulation Edge-to-Edge (PWE3) is a mechanism that emulates the essential attributes of telecommunication services (viz., E1 or T1) over PSN [6]. It is intended to emulate the wire with minimum necessary functions of a particular service. This can be achieved by some switching and translating function at the junction between the PSN and the emulated service network (e.g., PE). The switching function in this case is carried out by a forwarder and the translating function is carried out by a packet translator. For achieving emulation of wire, PWE3 defines Pseudo Wires (PWs) between PEs which generates a feel of un-sharing link in a shared link.

1.2.1 Components of PWE3

PWE3 distinctly divides the L2VPN services into access and core network. The access network is in between the customer end through CE and PE. The core network is between PEs. The service-specific PDUs of customers flow through an MPLS tunnel or an IP path between PEs (core network). The main components of PWE3 are CE and PE devices. The physical or virtual connections between the CE and PE (ACids) and the pseudo wires (PWs) between PEs are also members of PWE3.

1.2.2 CE Device and AC

CE device originates and/or terminates an emulated service. Though the users use emulated services, they are not aware that they are using emulated services rather than native services. CEs send and/or receive control plane messages over the PWs with

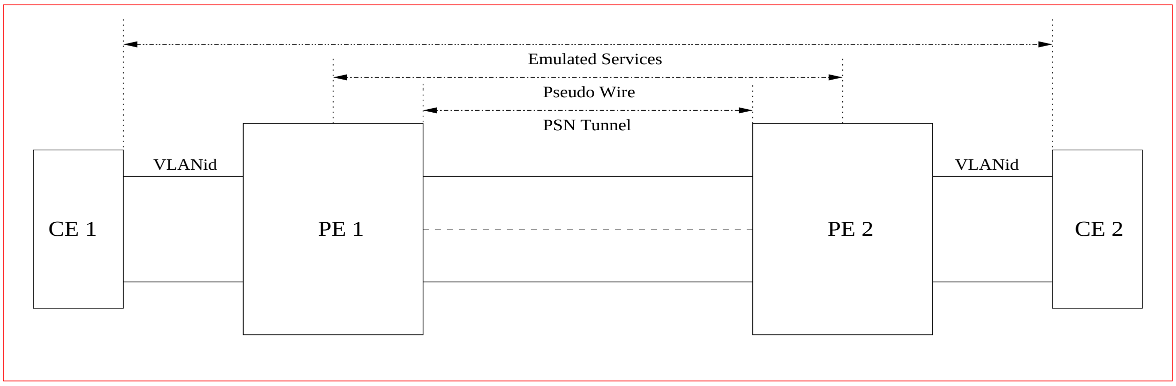

Figure 1.1: PWE3 Reference Model

the help of PEs to emulate the service. The ACs between CEs and PEs are numbered with ACids, which are unique for a PE and CE pair. The ACids are assigned by PEs by consultation with the CEs. There can be multiple PEs connected to one CE and it is not restricted by PWE3.

1.2.3 PE Device and PW

In PWE3 protocol, PE devices are the interfaces between the metro and the provider's network. This is the point where the service specific PDUs generated by CEs are translated to packets of the provider's network. Similarly at the other end of the provider's network these packets are again translated into the service specific PDUs needed by the remote CE. It uses PWs for transmitting between PEs. The PEs also take part in deciding the particular PW to be used for the service specific PDUs and hence act like forwarders for the service specific PDUs. The PWs are either point-to-point or point-to-multipoint or multipoint-to-point in nature. The function of PEs can be divided into two parts, (a) *Pre-processing* (b) *Forwarding*. PWs carry packets between PEs. Setting up and maintaining the PWs is the job of the PEs. State information for a particular PW is maintained at the two PEs.

The forwarder at PE works differently for PW-bound traffic and AC-bound traffic. For the PW-bound traffic the forwarder selects a PW (translates the ACid into PWid) and for AC-bound traffic, it selects an AC (translates the PWid into ACid). This selection process can be made static or dynamic.

The Pre-processing job of PE is handled by Native Service Processing (NSP).

NSP

NSP performs the data/address translation required for L2VPN services. Its functions includes packet processing required for the Ethernet packets which are forwarded to the PW from AC. Such functions are *stripping, overwriting* or *adding VLAN tags,* *physical port multiplexing* and *de-multiplexing, L-2 encapsulation, shaping, policing* etc. NSP helps by simplifying the design of PWs by restricting the PWs to homogeneous operations. The position of NSP with respect to PWE3 is shown in Figure 1.2.

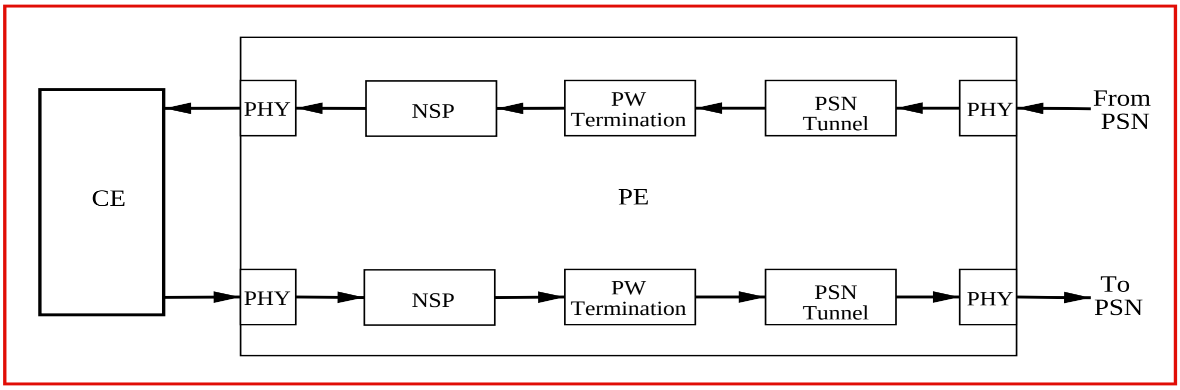

Figure 1.2: PW Reference Diagram at PE

1.2.4 ACid/PWid Encapsulation

In PWE3, there is a need of translation of the ACids into PWids and vice-versa at the PEs. PE removes the ACids and inserts the appropriate PWids and reforms the packets before sending through the PWs for the packets entering into PE from CE. Similarly, for the packet entering PE from PW, PE removes the PWid and inserts the appropriate ACid and recreate the service specific PDUs before sending through the specific AC to the CE.

The PWids are normally MPLS labels and are assigned by the core network. So when one PE sends an L-2 PDU to the remote PE, it first pushes a PWid on its label stack and hence create a packet specific to the core network. The remote PE needs this PWid before sending the packet to the remote CE. It encapsulates the ACids,(i.e., it translates the PWid to ACid), which are normally VLANids, assigned for the specific ACs.

1.3 Network Architecture

The network architecture in our project is mainly based on the PWE3 reference model [6]. The PWE3 reference model described in Figure 1.1, shows the Attachment Circuits (ACs), pseudo-wires (PWs), PSN tunnels and the Emulated LAN etc. In our architecture, we assume the CEs and PEs are VLAN-aware devices. We consider the ACids as VLANids and the PWids as MPLS labels. So, the packets in the AC are VLAN tagged. For VLAN tag (Figure 2.4), we have only 16 bits [3] field available. Out of those 16 bits, 3 bits goes for the priority setting and 1 bit goes for canonical format indication (CFI). So, we have only 12 bits available and hence we get maximum of 4096 numbers for VLANids. So, the ACid can be any number between 0 and 4095. Practically, the number of VLANids limits to 4094, as 0 and 4095 are reserved. The assignment of ACid (VLANid) can be achieved either manually by static configuration or dynamically by using GARP/GVRP [5, 3]. The ACids are mapped to PWids for the PW formation between a pair of PEs.

The mapping of ACid to PWid is different for different types of services (viz., VPLS and VPWS). The native data units coming from CEs to PEs are encapsulated in a PW PDU and are carried across the PSN tunnel to the other PEs, where it is to be de-encapsulated and forwarded to the remote CEs. The protocol model described in Figure 1.4 illustrates the basic idea about tunneling, pseudo-wire and emulated services.

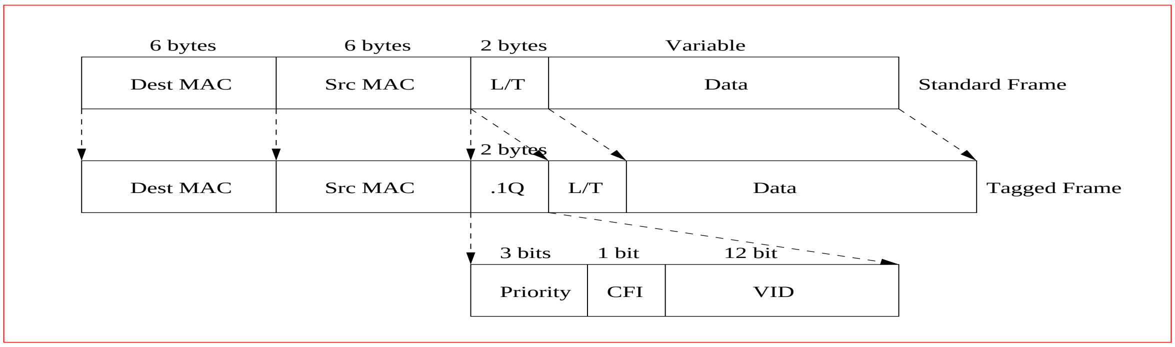

Figure 1.3: Frame Structure of Standard and Tagged Frame

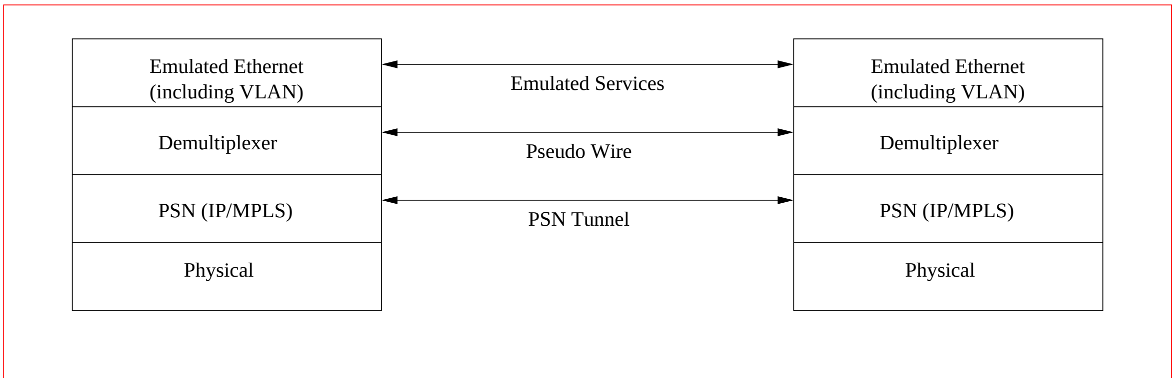

Figure 1.4: PWE3 Protocol Model

1.3.1 Hub and Spoke Architecture in L2VPN

Based on the PWE3 reference model, we can connect customers to PE either directly through the CE or we can connect several customers as a group to a local device (which acts like data/voice concentrator) and connect that to CE device. The later case needs separate VLANids for the traffic from the concentrator to CE and from CE to PE. The VLANids assigned for concentrator to CE traffic are known as the customer VLANids and the VLANids assigned for CE-PE traffic are known as provider VLANids. This principle is known as hub and spoke architecture in L2VPN.

1.4 Goal of the Project

Our main aim is to provide Metro Ethernet services based on VLAN. So, we have considered L2VPN services like VPWS and VPLS. Here we are looking into the access part of these services. In this project we are dealing with the design of CE device for Metro Ethernet services. The design and implementation of dynamic VLAN assignment process, the signaling between PE and CE for an L2VPN service and encapsulation techniques at PE are the essential contributions of this project.

1.5 Organisation of the Report

In the first chapter we have described the PWE3 protocol and the network architecture for our project. In Section 2.1, we have discussed the components of VPWS and the signaling and data flow diagram of VPWS. Further, we have considered the implementation of VLAN services and VLAN stacking in that chapter. In Section 2.6, we have discussed VPLS. There, we have looked into the various issues, the protocol structure, the signaling and data flow in VPLS etc. We have discussed the implementation details of PWE3 protocol in Chapter 3. In Chapter 4, the implementation of GARP/GVRP in an Embedded Platform is described. Chapter 5 lists out the testing scenarios for GARP/GVRP, VPLS and VPWS. Finally we have discussed the implementation issues and future work in this direction in Chapter 6. The pseudo-code for our implementation is described in Appendix-A.

Chapter 2

VPWS and VPLS

2.1 What is VPWS ?

VPWS is a type of L2VPN [7] service where point-to-point connectivity can be achieved between two customer sites (CEs in our case). To achieve this, we need to form a dedicated virtual circuit between two CEs. In this case the service provider network can be thought of emulating a set of wires between customers. To provide VPWS, we need to establish pseudo-wires (PWs) between the PEs for each pair of CEs. So, for PW-bound traffic[1], the forwarder at PE binds exactly one AC to exactly one PW and for the AC-bound[2] traffic, the forwarder at the PE binds exactly one PW to exactly one AC. Since the services are being emulated, the PEs are transparent to the customers and hence the users see a point-to-point connection between them. A particular combination of <AC, PW, AC> forms a *virtual circuit* between two CE devices. There are two such virtual circuits needed for achieving full point-to-point connection. The two virtual circuits may not belong to same service provider.

2.2 Components of VPWS

Figure 2.1 shows the point-to-point L-2 connection between the customer sites. CEs are connected to PEs through ACs and are numbered based on the mutual understanding between CEs and PEs. Since, CEs and PEs are VLAN-aware devices, we can assign VLANids as ACids. For each point-to-point virtual connection between the CEs in VPWS, we need separate ACs and hence ACids. In Figure 2.1 we have explained this with different ACids for different point-to-point connections. Assigning ACids is handled by the signaling module of VPWS.

The PEs act like forwarders for CEs. So, the packets entering to PE from CE should be properly encapsulated (Figure 1.2) and should be forwarded in a separate PW. In

[1] traffic from AC to PW
[2] traffic from PW to AC

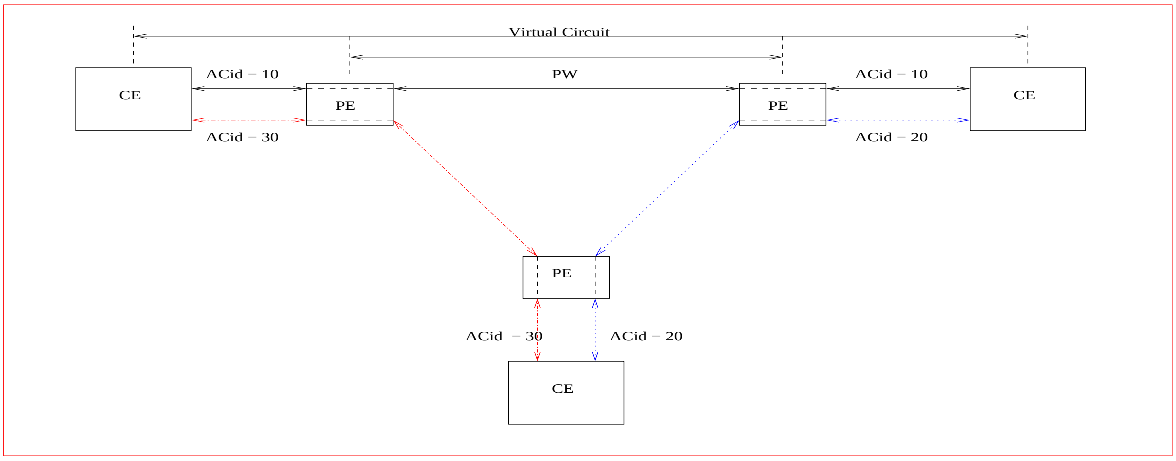

Figure 2.1: Point-to-point Virtual Circuits in VPWS

VPWS, a forwarder binds exactly one PW to exactly one AC and vice versa. At the egress PE these packets should be de-encapsulated and forwarded to the destination CE through a separate AC.

2.2.1 Attachment Circuit and Pseudo-wire

When a packet traverse from a CE to a PE through an AC, we call that AC as *ingress AC* and, when a packet traverse from a PE to a CE we call that AC as *egress AC* with respect to the packet. In VPWS, separate ACs are required for each point-to-point connection between a pair of PE and CE. This is explained in Figure 2.1.

In VPWS, PWs are point-to-point and bi-directional in nature. Any given packet travels first on its ingress AC, then on a PW, then on its egress AC. So, Ethernet PW emulates a single Ethernet link between two endpoints. The PW terminates at a logical port within PE as shown in Figure 1.2.

2.3 Signaling in VPWS

The signaling in VPWS [8] is used for setting up and provisioning the virtual circuit and maintaining of the virtual circuit. We have discussed the end-point detection mechanism, provisioning of ACs and PWs in the following subsections.

2.3.1 End-Point Detection

This includes end-point identification and provisioning of ACs at both sides. If we use *PWid FEC element* [9] as the forwarding element, then only one forwarder identifier is sufficient. It is used only when both forwarders have the same identifier. If we use *the Generalised Id FEC element* [9] as the forwarding element, then two forwarder identifiers, one for each of the two forwarders are required. Each identifier is known as an *Attachment Identifier (AI)*. In this case the signaling message carries both a *Source Attachment Identifier (SAI)* and a *Target Attachment Identifier (TAI)*. The Generalised ID FEC element also provides some additional structuring of the identifiers. SAI and TAI sometimes have a common part, called the *Attachment Group Identifier (AGI)* (Figure 2.2). Hence, SAI and TAI can each be thought of as the concatenation of the AGI with an *Attachment Individual Identifier (AII)*. Hence the pair of identifiers (SAI, TAI) are encoded in three fields: AGI, Source AII (SAII), and Target AII (TAII). In our network architecture, the AGI is a VLANid assigned by the CE for the virtual connection (if we use VLAN stacking). The AGI field may be empty (null). The SAII and the TAII are the VLANids assigned by PE.

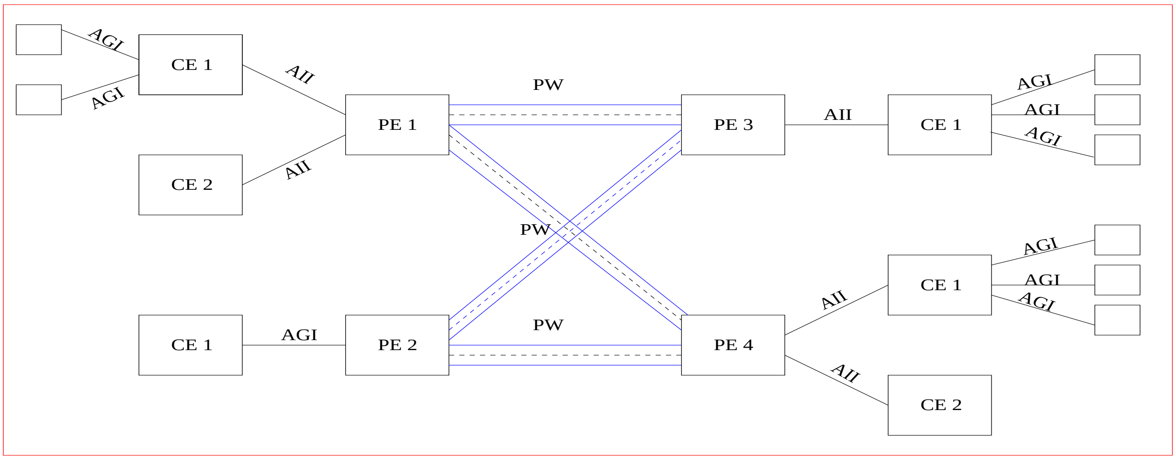

Figure 2.2: VPWS Provisioning

2.3.2 VPWS Provisioning

Provisioning of VPWS [7] can be divided into three logical steps. They are: (a) *Provisioning of ACs*, (b) *PW setup for the AC* and (c) *PW configuration*. The ACs need to be individually provisioned on PE and/or CE. The PW provisioning can be done either by *two-sided provisioning* or by *single-sided provisioning*. In two-sided provisioning, each PE at the end point of a PW is provisioned with information like the SAI to PWid mapping at the local PE, PW type parameters, address of the remote PE, the PWid to TAI mapping at the remote PE. This is required for forwarding at both local and remote PE and CE. In single-sided provisioning the PE at one end of a PW is provisioned with the following information: identifier of the local AC to which the PW is to be bound, PW type and parameters and the globally unique identifier of remote AC. We are using the two-sided provisioning method for VPWS provisioning.

2.4 Signaling Steps in VPWS

We have considered a simple example (Figure 2.3) as described below to show the signaling flow in VPWS. In this example, we are interested for providing a VPWS service between CE-1 and CE-2. The following method is used to find the ingress PE and egress PE [10], SAI and TAI etc and other parameters.

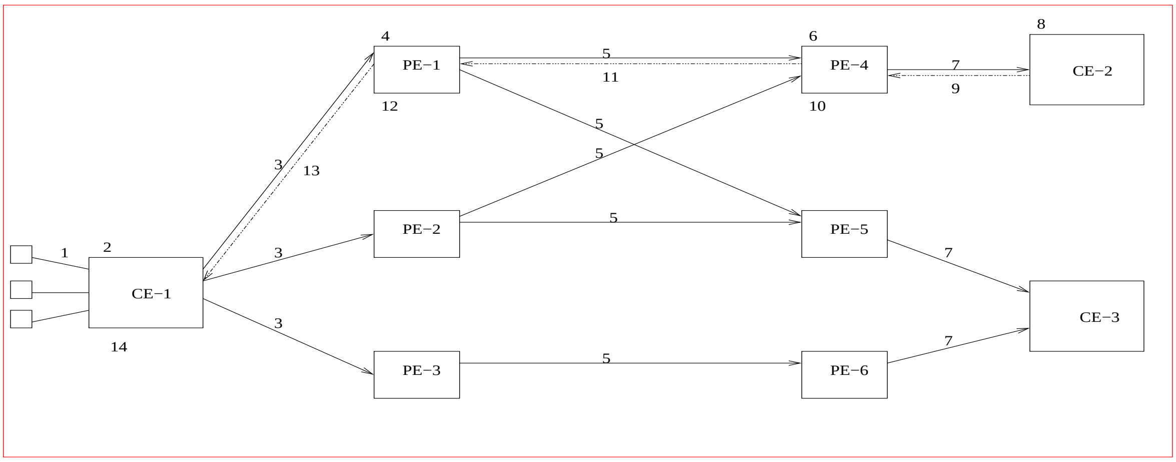

Figure 2.3: VPWS Signaling

1. CE-1 receives a packet from its customer with source MAC address and destination MAC address.
2. CE-1 searches its own database to find a match for the destination MAC address to <SAI, PW, TAI>, i.e., to know the local and target ACids and ingress and egress PE's address.
3. If CE-1 does not find a match, then it will send a broadcast packet to all PEs.
4. PEs search for this in their own databases.
5. If PEs do not find this information in their databases, then they forward to all the PEs to which they are connected.
6. Step 4 is repeated at remote PE.
7. If the remote PEs do not find the match, then they forward to all CEs to which they are connected.
8. The remote CE (CE-2 in our example) finds the destination machine.
9. CE-2 passes this information to PE-4.
10. PE-4 learns the above address information.

11. PE-4 passes this information to PE-1.

12. PE-1 learns this address information, i.e., the remote PE for that destination MAC address and PWid for that ACid.

13. PE-1 passes this information to CE-1.

14. CE-1 learns this information, i.e., <SAI, PW, TAI> for that destination MAC address.

Now CE-1 knows the addresses of the local PE (PE-1) and remote PE (PE-4). Based on the two-sided provisioning method, CE-1 gets an ACid to connect to PE-1. In our design, we insert a VLAN tag (SAI) for the traffic entering the ingress AC. At the PE, NSP finds out the PWid from its forwarding table and forms the PW between the local and the remote PE (PE-1 and PE-4) for forwarding the traffic to the egress AC. At the remote PE (PE-4), NSP again finds out the ACid (TAI) and overwrites PWid with ACid or strip-off the PWid and insert the remote ACid before forwarding to the target CE (CE-2). Like-wise packets can reach the destination machine and point-to-point virtual circuits can be defined between CEs of same VPN.

2.5 Data Flow in VPWS

In hub and spoke architecture, the packets entering into the CEs are VLAN-tagged (AGI). So, the packets leaving CE for PE are tagged with two VLANids (AGI and SAII) (Figure 2.4) [3] resulting to a 32-bit forwarding identifier. If AGI is null, the forwarding identifier is of 16 bits only.

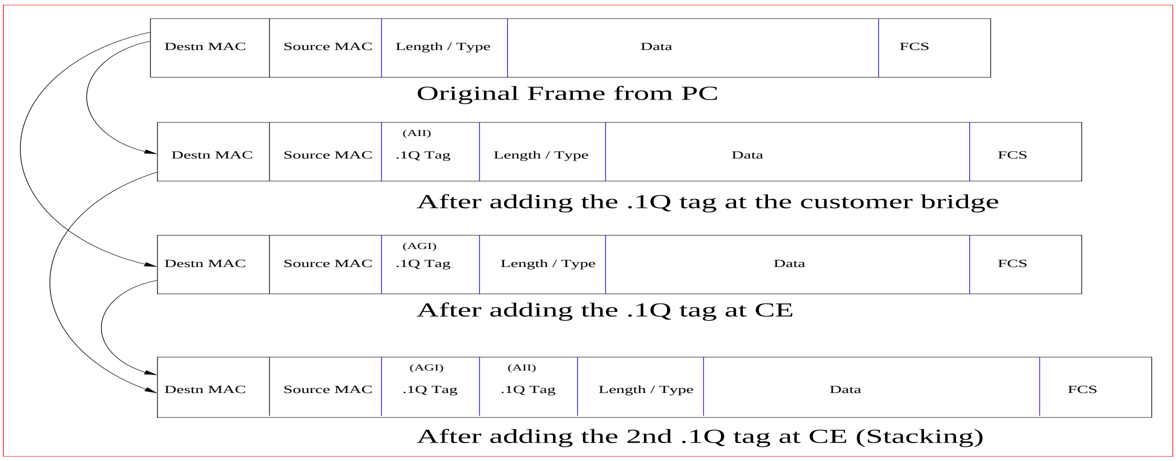

Figure 2.4: VLAN Tag Insertion with(out) Stacking

2.5.1 Function of PE

In our architecture PWs are being formed between PEs [9, 8] as a part of the point-to-point virtual circuit formation between two CEs in VPWS. So, when a packet enters PE and is PW-bound, the NSP of PE modifies the received packet [11] as follows:

1. Strip-off the preamble (if, any) and FCS.
2. Insert the control word, if necessary.
3. Strip-off the ACid and store (SAI).
4. Insert the PWid (else, re-write the the ACid with PWid).
5. Insert the Tunnel-id and de-multiplexer.
6. Transmit the resulting packet to the other PE through the PW.

When a packet enters PE and is AC-bound, the NSP modifies the received packet [11] as follows:

1. Strip-off the Tunnel-id and de-multiplexer.
2. Strip-off the PWid, insert/re-write the PWid with the ACid (TAI).
3. Strip-off the control word, if present.
4. Insert the preamble (if, required) and FCS.
5. Transmit the resulting packet to the CE through the AC.

The exact packet structure at different point of the virtual circuit between two CEs in VPWS is shown in Figure 2.5.

2.5.2 Forwarding

In VPWS, a forwarder binds a pseudo-wire to a single AC, such that packets received on the one are sent on the other, and vice-versa. In our network architecture we define two forwarders, viz., at CE and at PE. The forwarder at CE works differently for customer-bound and AC-bound traffic. It learns the mapping for the destination MAC address to <SAI, PW, TAI> in a dynamic manner as discussed in VPWS signaling (Figure 2.3) [10, 8]. The forwarder at PE also works differently for PW-bound and AC-bound traffic. The PE also learns the information about the local CE and the remote CE dynamically [10] and updates this forwarding information to the local CE. Both forwarders at CE and PE can be manually configured and administered. Also, the forwarding at PE and at CE follows some ingress, egress rules for forwarding the

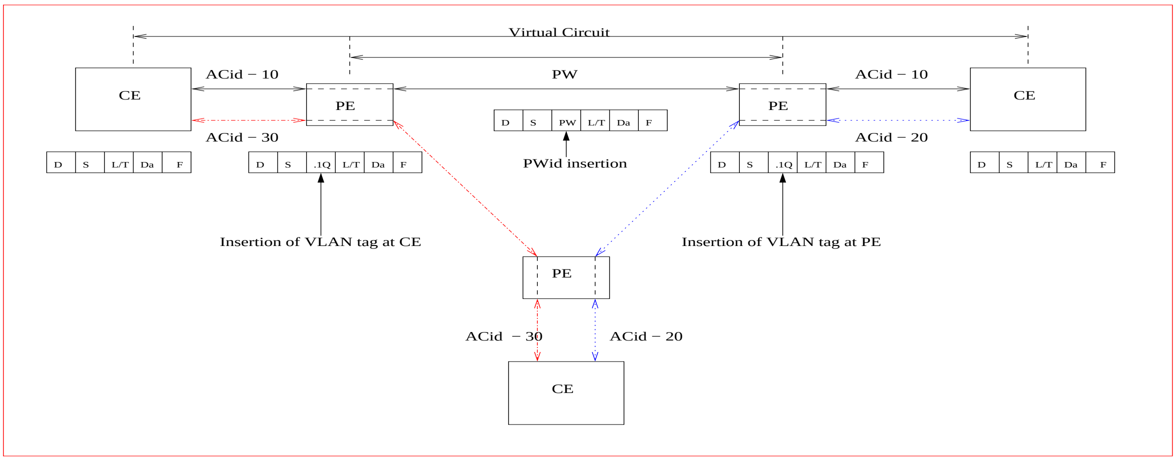

Figure 2.5: Packet Structure at Different Points of VPWS

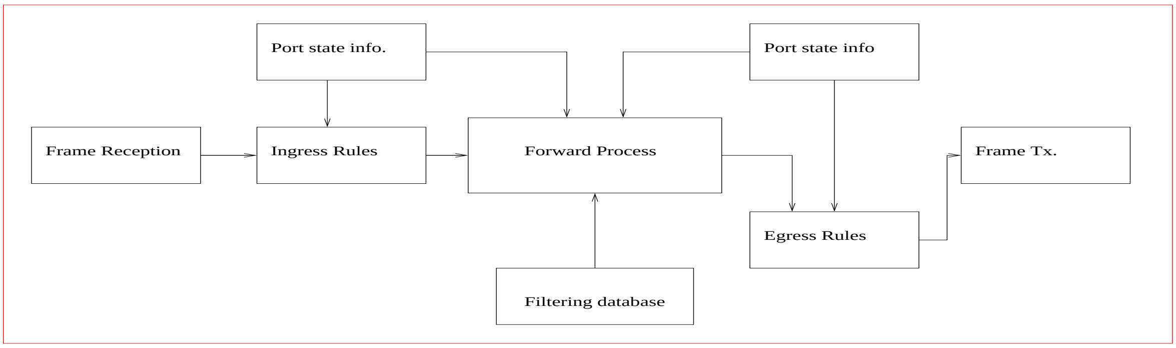

Figure 2.6: Forwarding at PE

packets. Here the SLAs and QoS mechanisms can be implemented. The interaction among all these processes is shown in Figure 2.6.

Any change at CE should reflect the forwarding table at both remote and local PE and remote CE, to which the virtual circuit exists. Also, if a new CE gets added up, then all CEs of that VPN should form separate virtual circuits with the new CE (individually). In this case new sets of $<$SAI, PW, TAI $>$ are to be provisioned. This helps a fully meshed VPWS connection among the CEs of same VPN.

2.6 What is VPLS ?

VPLS is a type of L2VPN service (a point-to-multipoint connection) and can be achieved between customer sites (CEs) of same VPN. In a VPLS, each CE device has one or more LAN interfaces that lead to a *virtual backbone*. Two CEs are connected to the same virtual backbone, if and only if, they are members of the same VPLS instance (i.e., same VPN). As shown in the reference model of VPLS (Figure 2.7) [7], PE devices that are VPLS-capable provide a logical interconnect, such that CE devices belonging to a specific VPLS appear to be on a single bridged Ethernet. A VPLS can contain a single VLAN or multiple tagged VLANs. A CE device attaches, through an access network, to the *bridge* module of a PE. Within PE, the bridge module attaches, through an *Emulated LAN Interface* to an Emulated LAN. For each VPLS, there is one Emulated LAN instance. The Emulated LAN consists *VPLS forwarder modules* (one per VPLS-PE per VPLS-instance known as Virtual Switch Instance (*VSI*)) connected by PWs, where the PWs may be traveling through PSN tunnels over a routed backbone.

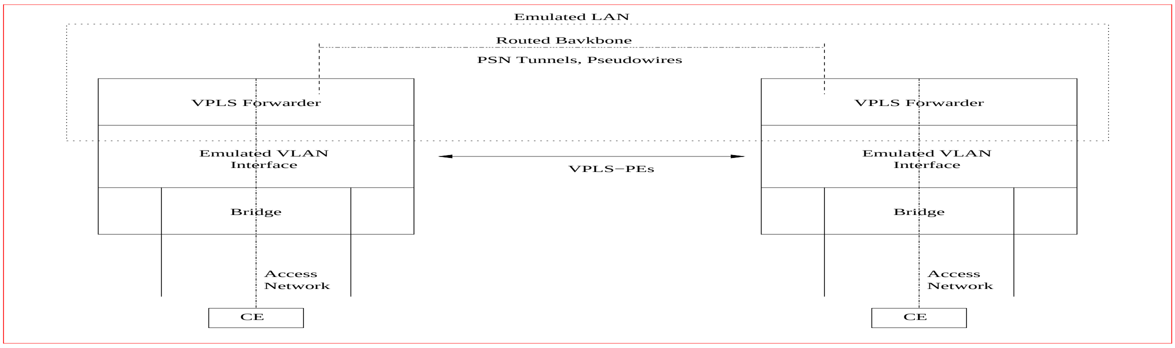

Figure 2.7: Reference Model in VPLS

When PE receives a packet from AC, it examines the destination MAC address field in order to determine how to forward the packet. Thus the PE functions as a bridge. As indicated in Figure 2.8, if a set of PEs support a common VPLS instance, then there is an Emulated LAN, corresponding to that VPLS instance, to which each of those PE bridges attaches (via an emulated interface). From the perspective of a CE device, the virtual backbone is the set of PE bridges and the Emulated LAN on which they reside. Thus to a CE device, the LAN which attaches it to the PE is extended transparently over the routed MPLS and/or IP backbone. So, PE bridge function treats the Emulated LAN as it would do to any other LAN to which it has an interface. Forwarding decisions are made in the manner which is normal for bridges, based on MAC address learning [5].

2.7 Components of VPLS

In simple VPLS, a forwarder (VSI) binds exactly one AC to multiple PWs connecting to the VSIs of other PEs of same VPN. CEs are connected to PEs through an AC and are numbered based on the mutual understanding between CE and PE. We assign the ACids as VLANids and are different for different ACs of different VPN. It is because the ACid is unique between a pair of PE and CE for a VPLS. Separate ACs and hence

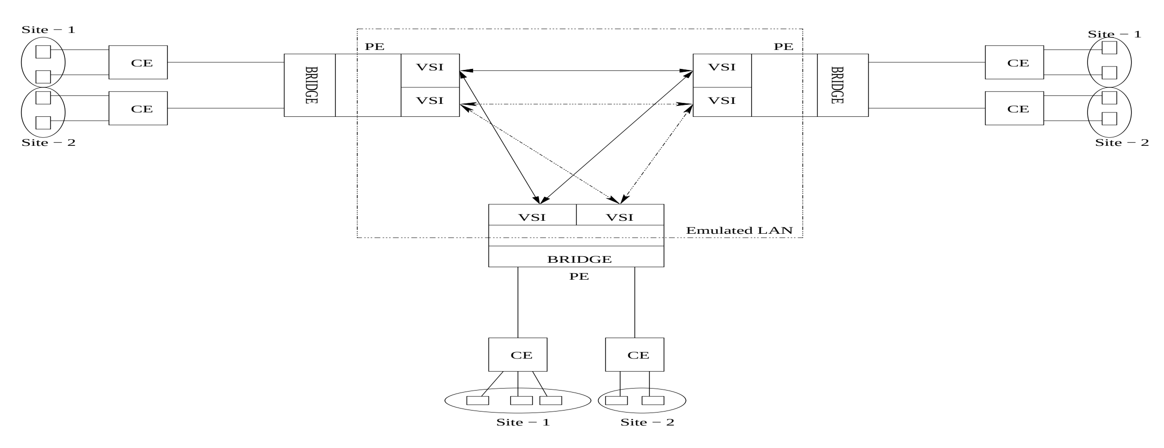

Figure 2.8: Point-to-multi-point Virtual Connections in VPLS

ACids between a pair of CE and PE is required, if the same CE is a part of different VPN or the same CE is part of different VLAN of same VPN.

As discussed previously, PE in VPLS can be divided into three logical parts, viz., (a) *VPLS forwarder per VPLS (VSI)*, (b) *Emulated LAN interface* and (c) *Ethernet bridge*. Each PE is configured with the VPNs in which it participates. For each VPN, PE has a list of CEs, which are members of that VPN. For each CE, PE knows the ACid (through signaling). The PEs act like the forwarder for the CEs. The packets entering into PE from CE should be properly encapsulated and should be forwarded to the remote PE on the PW [12].

2.7.1 Attachment Circuit and Pseudo-wire

The Attachment Circuit in VPLS is similar to that of VPWS. The only difference between the two is the number of ACs and the ACids. In VPLS, there is only one AC between a pair of CE and PE and hence one ACid per VPN, whereas in VPWS, separate ACs are required for each virtual circuit in a VPN.

In VPLS we need separate PWs between a pair of VSIs [8, 13]. Each PE device may have multiple VSIs for multiple VPNs (Figure 2.8). Hence a single ACid can be mapped to multiple PWids and the traffic from a single AC can be forwarded to different PEs through different PWs resulting in point-to-multi point virtual circuits. Like VPWS, PWs in VPLS too terminate at a logical port (NSP) within the PE.

2.8 Signaling in VPLS

2.8.1 End-Point Detection

The end-point detection in VPLS is similar to that of VPWS. In VPLS, a VSI binds a set of pseudo-wires to a set of ACs; when a packet is received from any member of that set, a MAC address table is consulted [5] to determine the member or members of that set on which the packet is to be transmitted. The Ethernet bridge in PE learns the source MAC address of the packets coming in the AC and VSI learns the source MAC address of the packets coming in the PWs. Though logically these are two different modules, one can think of making a single module for doing both the jobs and hence detect the end points.

2.8.2 VPLS Provisioning

Each VPLS must have a globally unique identifier, which is known as a VPNid. Every VSI must be configured with the VPNid of the VPLS to which it belongs. The VPNid in our case is a simple VLAN tag (16 bit) or stacked VLAN tag (32 bit). The provisioning of ACs in VPLS is similar to that of VPWS. A VSI for that VPLS must be provisioned on the PE, and the local ACs of that VPLS must be associated with that VSI. The VSI must be provisioned with the identifier of the VPLS to which it belongs. It can be achieved through auto-discovery [7] or through static configuration. So, PE can map the ACid to the set of PWs and hence can map to the set of remote VSIs of that VPLS. If AGI is not empty (nun null), then the connectivity between VSIs can be limited so that two VSIs are connected, only if, they have a common VLAN. In that case, each VSI would need to be provisioned with one or more VLANids, and the auto-discovery scheme would need to map a VPLS identifier into pairs of <PE, VLANid>.

2.9 Signaling Steps in VPLS

We have considered a simple example (Figure 2.9) to show the signaling flow in VPLS. Here the function of VSI, Ethernet bridge [12] in PE for MAC address learning [10], PWid identification and ACid (VLANid) identification etc. are described.

1. CE transmits the packet to PE with source MAC address and the destination MAC address.
2. The Ethernet bridge in PE learns the source MAC address of the packet.
3. Ethernet bridge passes the packet to VSI (in that PE) of that VPLS.
4. VSI searches the remote PE's address and the PWid for the destination MAC address from its own database.

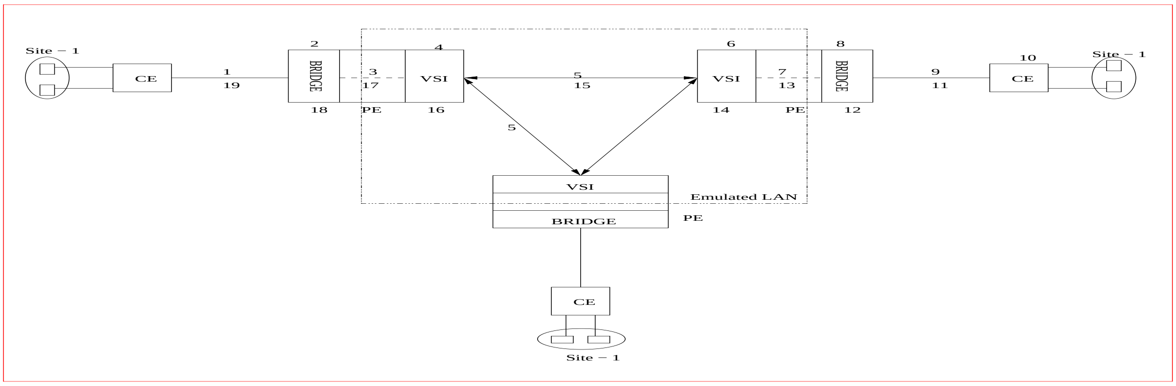

Figure 2.9: VPLS Signaling

5. If it does not find a match, then it broadcasts in all the PWs to all the PEs of that VPLS.
6. VSI at the remote end learns the source MAC address of the received broadcast packet.
7. It passes that packet to the Ethernet bridge.
8. Ethernet bridge searches its own database for the destination MAC address (i.e., the CE's address for that MAC address).
9. If the searching fails, it broadcasts to all the CEs of that VPLS connected to it.
10. CE finds out the machine for that MAC address.
11. CE replies back to the Ethernet bridge.
12. Ethernet bridge learns the source MAC address.
13. Ethernet bridge forwards that packet to VSI.
14. VSI replies to the broadcast packet it received from the other VSI.
15. VSI, which sent that broadcast packet, receives the acknowledged packet.
16. VSI learns the source MAC address. Now it is in a position to map the destination MAC address (packets received from Ethernet bridge) to PWid and PE.
17. It passes this to Ethernet bridge.
18. Ethernet bridge finds out the CE's address.
19. It passes this information to CE.

2.10 Data Flow in VPLS

2.10.1 Function of PE

The function of PE in VPLS is almost similar to that of VPWS for data flow. When a packet enters PE and is PW-bound, the NSP of PE performs the following actions on the received packet [11].

1. Strip-off the preamble (if any) and FCS.
2. Prepend the control word, if necessary.
3. Strip-off the ACid and store (SAI).
4. If the packet is unicast in nature then, PE will have to find out the PWid for that destination MAC address.
   (a) Insert the PWid (else, re-write the the ACid with PWid).
   (b) Prepend the tunnel id and de-multiplexer.
   (c) Transmit the resulting packet to the other PE through the PW.
5. If the packet is broadcast or multicast in nature, then PE has to forward that packet in all the PWs to all the PEs of that VPLS.

When a packet enters PE and is AC-bound, NSP of PE does the following things to the received packet[11].

1. Strip-off the tunnel id and de-multiplexer.
2. Strip-off the PWid, insert/re-write the PWid with the ACid, which is the VPNid or the VLANid of the VPLS.
3. Strip-off the control word, if present.
4. Insert the preamble (if required) and FCS.
5. Transmit the resulting packet to the CE through the AC.

2.10.2 Forwarding

In VPLS, Ethernet bridge at PE learns the source MAC addresses of the packets entering into it from CE [7]. It does not learn the destination MAC address from these packets. But, VSI learns the source MAC address of the packets entering into it from the other VSI through the PW. So, VSI can map the destination MAC address of the packets it receives from the Ethernet bridge to a PW, i.e., it can map the ACid to a

PWid and can forward the traffic accordingly. It also ages out this information periodically. The Ethernet bridge on the other hand inspects the destination MAC address of the packets received by the VSI and forwards it to the specific CE after re-writing the PWid with the VLANid.

The CE also learns the MAC address from the customer facing ports and the ACs [14]. When a new MAC address is learned, the MAC address gets associated with the virtual interface or ports where the packet arrives. When a packet with the cached MAC address is received, the CE knows which virtual interface or port to forward the packet to. When a packet with a new MAC address is received, a CE floods the packet in all other ports or virtual interfaces, except the interface where the packet is received from. It follows the IEEE 802.1Q [3] and IEEE 802.1D [5] standards for the ingress, egress and forwarding process (Figure 2.6).

Chapter 3

PWE3 Implementation

Implementing PWE3 for VPLS and VPWS deals with the PE and CE configuration, AC provisioning, grouping based on VLAN, PW setup, OAM issues and SLA issues, if any. In this project, we are dealing with CE configuration, AC provisioning, PE configuration for Native Service Processing (NSP) and forwarding. The set-up and maintenance of the PW and the tunnel [15] are the service provider's job. The implementation for VPLS and VPWS are almost similar with some differences arising due to the specifications of the services.

3.1 CE Device

The CE in our case is a *uClinux* [16] based Embedded Platform. This is a manageable L-2 Ethernet, GVRP enabled switch. Hence it is capable of assigning VLANs dynamically to the customers and can register for the VLANs assigned by the PE device dynamically. It deals with the customers as well as with the PE device for assigning with the ACid (both AII and AGI) for an Attachment Circuit. Since PEs are not GVRP enabled in our case, VLAN assignment for the CEs are static. So, the assignment of the ACid (AII) in our case is done through selecting a VLANid from a pool of VLANids statically. Since in the up-link ports[1] the egress tagging is disabled, the CE can send both tagged and un-tagged traffic. In the down-link ports the ingress tagging is enabled and hence all the packets entering into the CE are VLAN tagged. These VLANs are assigned by CE using GVRP techniques. So, the forwarding component in CE uses *Generalised ID FEC Element*. If the packets are not tagged, then CE inserts the default VLAN 4092 into them as port VLANid (PVID) [3]. In our implementation, we included the encapsulation for the packets entering to CE from customers and vice versa. This is implemented at CE. Also, for the PE-bound[2] packets we incorporated the encapsulation

[1]discussed in the next chapter
[2]packets leaving CE and are AC-bound

method at CE. So, any AC-bound packet has both AGI and AII in its packet header. Similarly, any customer-bound packet has AGI in its header.

3.2 PE Device

PE device in our implementation is a Linux machine with MPLS routing enabled in it. It is VLAN enabled and accepts the tagged packets through its Network Interface Card (NIC). Since it is MPLS enabled, it can create MPLS LSPs to control the other PEs. These MPLS LSPs form the PWs. The MPLS Label acts as PWid in our case.

PE acts differently for AC-bound and PW-bound traffic. When it receives a packet at the ingress AC, it needs to forward the packet with proper encapsulation [11] in the MPLS-LSP [17, 18] to the remote PE.

3.2.1 Native Service Processing (NSP)

This is the core part of PE. As discussed earlier, it translates ACid into MPLS Label and vice-versa. Also, it regenerate the *FCS* related to the AC-bound and PW-bound traffic after the translation of the data and addresses. In the following sections, we have discussed the structures that we have used for NSP implementation.

3.2.2 Encapsulation and de-encapsulation for CE-PE Traffic

As discussed above, the PE needs to translate the VLANid into a MPLS Label for the PW-bound traffic and MPLS Label to VLANid for the AC-bound traffic. MPLS Label is a 20-bit label, whereas the VLANid is a 12-bit id. In our implementation, we use hub and spoke architecture and hence opted for VLAN stacking by using both customer and provider VLANids.

```
struct pe_pwbound  {
   uint8 dest_hw_addr[6];       //Destination eth addr
   uint8 src_hw_addr[6];        //Source ether addr
   uint8 mpls_label[3]          //20-bit MPLS Label
   uint8 length[2];
   uint8 dsap[1];               //can be null
   uint8 ssap[1];               //can be null
   uint8 control[1];            //can be null
};

struct pe_acbound  {
   uint8 dest_hw_addr[6];      //Destination ether addr
   uint8 src_hw_addr[6];       //Source ether addr
```

```
   uint8 acid1[2];             //AGI (customer VLANid)
   uint8 acid2[2];             //AII (provider VLANid)
   uint8 length[2];
   uint8 dsap[1];              //can be null
   uint8 ssap[1];              //can be null
   uint8 control[1];           //can be null
};
```

We used RAW socket method to receive (and send) packets from (to) the PWs and from (to) the CEs. The packet structure in both AC-bound and PW-bound traffic is shown below.

```
struct cebound_packet {         //AC-bound traffic
   uint8 packet_header[21];     //Ethernet header (maxm)
   uint8 packet_message[1493];  //data
   uint8 packet_fcs[4];
};

struct pwbound_packet {         //PW-bound traffic
   uint8 packet_header[20];     //Ethernet header (maxm)
   uint8 packet_message[1494];
   uint8 packet_fcs[4];
};
```

3.2.3 CE-PE Signaling

The CE-PE signaling includes ACid assignment, creating the forwarding component at PE for both AC-bound as well as PW-bound traffic. In our implementation, we maintain the forwarding table at PE, which finds out the MPLS Label for the destination MAC address and finds out the VLANid for the destination MAC address. Also, it learns and maintains the source MAC address and the ACid pair from the PW-bound traffic and source MAC address and PWid from the AC-bound traffic. Since we use a homogeneous network at both ends of the PEs, there is no need for the protocol conversion component in our implementation.

Chapter 4

GARP/GVRP Implementation in the Embedded Platform

In our Network Architecture (Section 1.3), we have assumed that the CEs and PEs are VLAN-aware devices. In the case of hub and spoke architecture (ACids are the combination of AGIs and AIIs), the AGIs are assigned by CEs after consulting with the end users and AIIs are assigned by PEs after consulting with the CEs. In the other case (single VLANid), the AGIs are null and hence the ACids constitute with AIIs only. These assignment processes can be static or dynamic in nature. In the static assignment, each port of the PE device can be assigned with a specific VLANid in a static manner, either by the administrator or by some rule. Such assignments are appropriate on ports where the configuration of the attached devices is fixed, or where there is a need of administrative boundary. This method is good if we deal with lesser number of VLANs and lesser number of attached devices. But, as the number grows, it becomes complicated and in turn may lead to the VLANid duplication problem. So, we investigated the dynamic method of assignment of VLANids. The dynamic assignment process needs implementation of GARP/GVRP at PE and CE. In this project, we have implemented GARP/GVRP at CE.

4.1 ***uClinux*** based Embedded Platform

The CE device in our project is an Embedded Platform with *Infineon* made switch-on-chip running with *uClinux*[16] in the core. This is a configurable L-2 switch on a chip. It has 48 Full Duplex Fast Ethernet (FE) ports, which supports 10/100 Mbit/s data rate and 2 Gigabit Ethernet (GE) ports which supports 1000 Mbit/s data rate. The on-chip integration of a 64 bit RISC CPU controls the management of the complex network and the on-chip packet buffer, address table and system interfaces make it faster and cheaper. It has a set of APIs, a set of data structures and libraries. These

APIs, data structures and libraries are accessed through PCI V2.2 compliant interface. The network setup to use these APIs, libraries and data structures are shown in Figure 4.1.

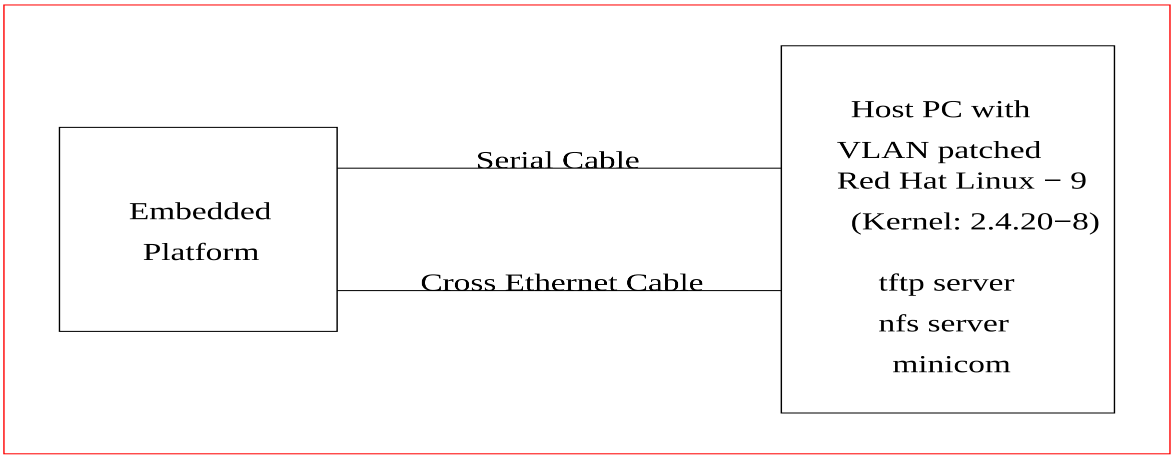

Figure 4.1: Network Setup for the Embedded Platform

4.1.1 Architecture of the Embedded Platform

The switch of the embedded platform is capable of segregating the incoming packets based on the destination MAC address. It accepts singly-tagged, doubly-tagged and un-tagged packets at any port. When the incoming packet enters through the ingress port, the L-2 and L-3 information are extracted. Then the normal MAC table lookup and learning functions (source and destination MAC addresses etc.) are performed. Based on these L-2 and L-3 information and the MAC table entries, the packets are classified for different flows. The packet data is stored in the embedded DRAM (Figure 4.2), which can store 2k full size Ethernet packets. The packets are considered as critical packets, if so specified by the MAC table. Otherwise they are considered as normal packets. All critical packets are forwarded to CPU, irrespective of the port state and validity checks and are treated separately.

The final destination generation takes into account the VLAN membership of source and destination ports, port authentication settings, link aggregation and port bridge status check results etc. The switch can be programmed to transmit packets based on the rules relating to flood index. The flood indices are maintained in a flood table. In this method, we can multicast/broadcast certain packets through some specific desired ports of desire, grouped as a flood index. Based on flow control, all/some packets to/from a port may be allowed to go/drop. This depends upon the buffer size and queue size of the switch. Then the packets are delivered to the egress port. Once a packet is delivered to the egress port, VLAN tagging and un-tagging functions are performed before the packet is sent to the MAC. The outgoing packets can be of VLAN tagged or un-tagged. Figure 4.2 shows the switching flow in the Embedded Platform in our architecture.

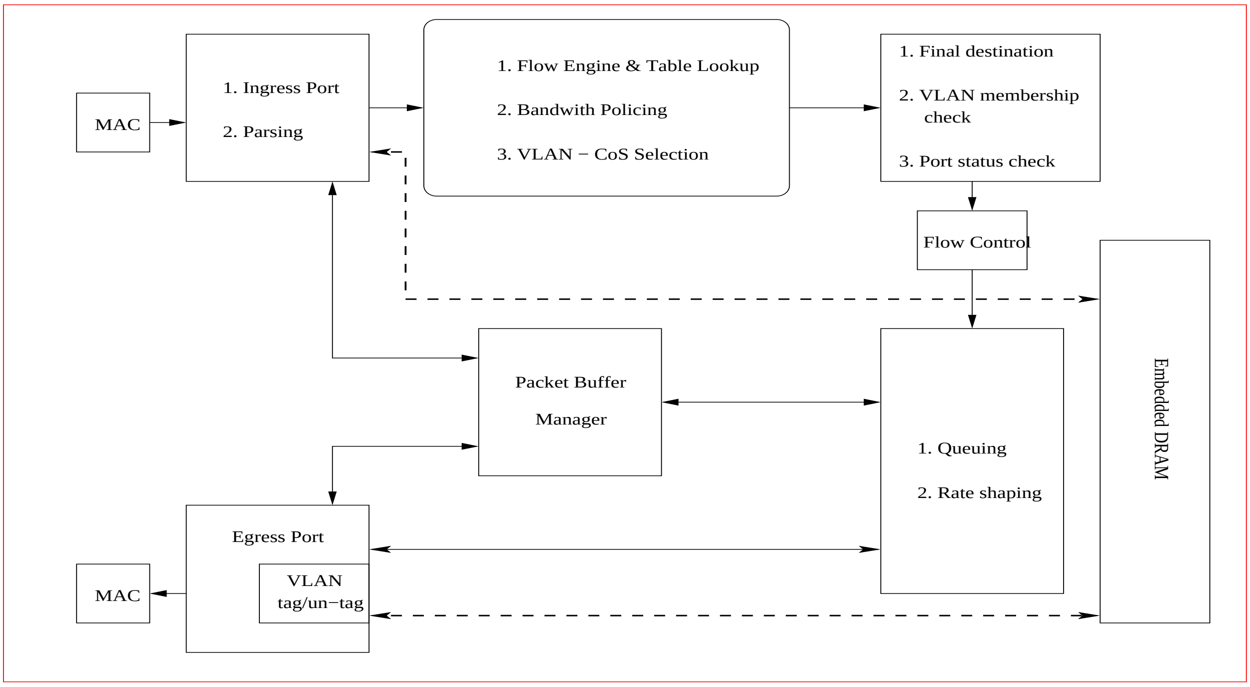

Figure 4.2: Switching Flow in the Embedded Platform

4.2 GARP

The GARP provides a generic attribute propagation capability that is used by GARP application participants to register and de-register attribute values with other GARP application participants within a bridged LAN. GMRP (GARP Multicast Registration Protocol) and GVRP (GARP VLAN Registration Protocol) [5] are such kind of GARP applications. In a given GARP application, the participants make/withdraw declarations with attribute values. These declarations/withdrawls are recorded by means of state variables in an Applicant and Registrar state machine. Registration/de-registration occurs only with respect to those ports which receive the GARP PDUs containing declaration or withdrawal information and not on the transmitting ports.

In Figure 4.4 we have described the operation of GARP in a two port bridge and one end station. Each port of the bridge is associated with a GARP application and a GARP Information Declaration (GID) component. The propagation of information between GARP participants for the same application in a bridge is carried out by the GARP Information Propagation (GIP) component. Protocol exchanges take place between GARP participants by means of LLC type 1 services (connection less) using the group MAC address and PDU format specific to the particular application concerned.

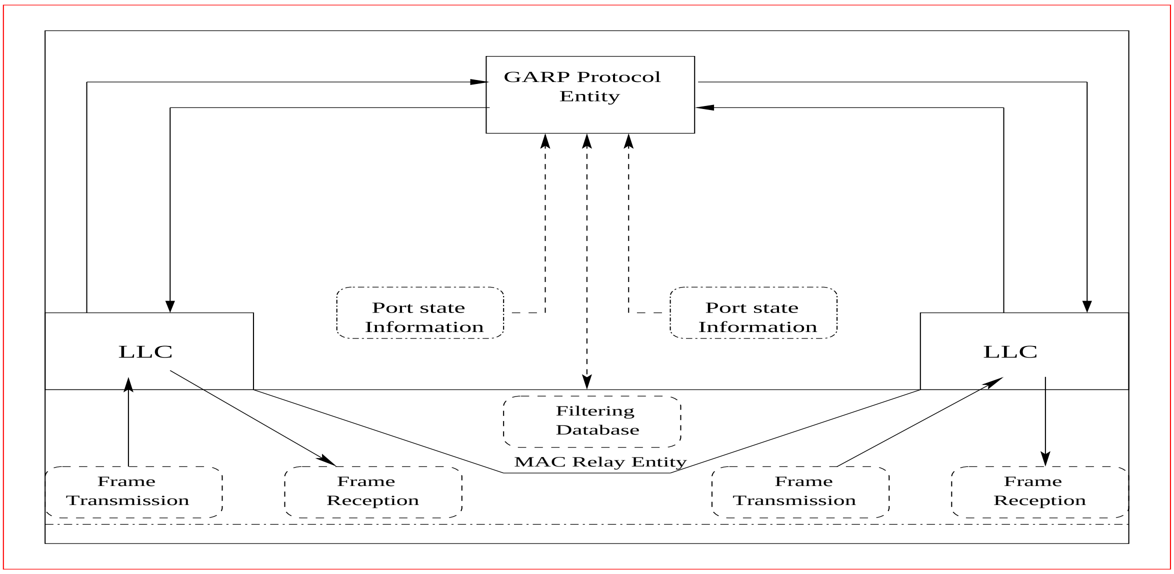

Figure 4.3: GARP Operation

Each GARP application must agree to certain parameters. They are (a) *attribute types*, (b) *attribute values*, (c) *attribute events*, (d) *group MAC address*, (e) *encoding structure*, (f) *state machines* and (g) *timers*. These parameters are application specific. We are dealing with one particular GARP application called GVRP in our project. This is discussed in the next section.

4.3 GVRP

The GARP VLAN Registration Protocol (GVRP) is a GARP application [3]. It provides dynamic VLAN registration/de-registration services by propagating the information to other bridges. It uses GID and GIP for the common state machine descriptions and the common information propagation mechanisms defined for use in GARP based applications. The GVRP-aware bridges update their active member set for each VLAN to which they are members of, after getting the messages containing the information for each VLAN. In GVRP the attribute values are VLANids and are propagated among the bridges with some attribute events resulting dynamic registration/de-registration of VLANs. Only GVRP-aware devices updates the VLAN information dynamically. The non GVRP-aware devices need not update this information. Since, GVRP PDUs are un-tagged ones, they can be propagated among all the bridges and end stations including GVRP un-aware devices.

In GVRP, both end stations and bridges in a bridged LAN can issue and/or revoke declarations relating to VLAN membership. The GVRP participant, which receives such kind of declarations needs to update the VLAN membership information by dynamic registration/de-registration for the received port. The VLAN updating is applicable only for the received ports and not for the transmitted ports. When all the members of a particular VLAN revoke declarations, then that VLAN is un-registered completely by each participant in that segment. The operation of GVRP is shown in Figure 4.4. Each GVRP participant has three entities, viz., (a) *GVRP application*, (b) *GID* and (c) *GIP*.

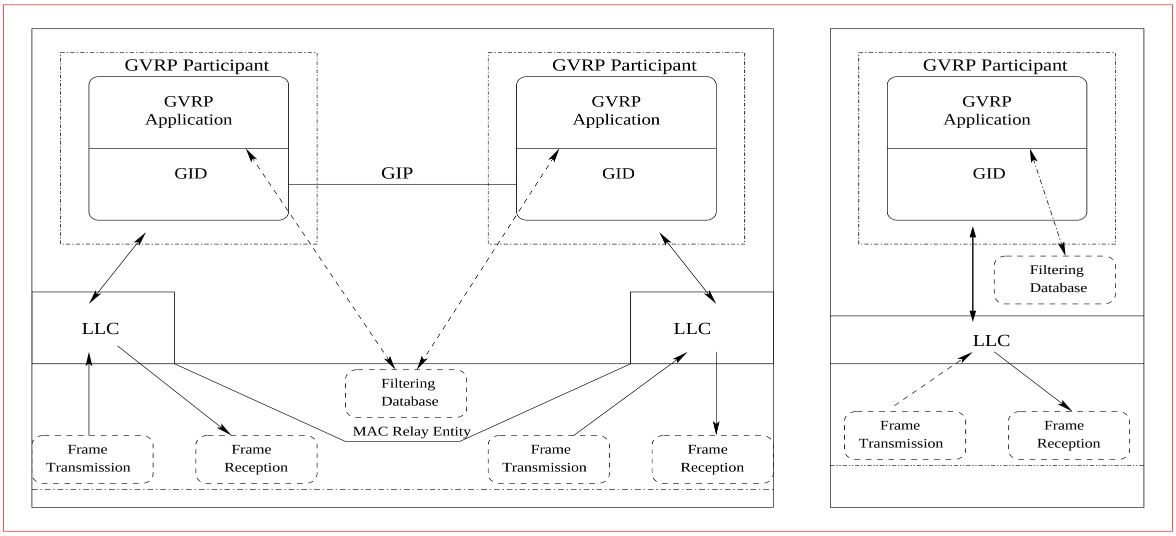

Figure 4.4: GVRP Operation

4.3.1 Declaration and Registration

The set of state machines, that defines the current registration and declaration state of all the attribute values associated with the GVRP participant constitute an instance of GID. Declaration primitives [5, 3] are defined in order to allow the users of GID to make/revoke (join/leave) attribute declarations through a given port using *requests*. They are:

1. *GID_Join.request (attribute_type, attribute_value)*
2. *GID_Leave.request (attribute_type, attribute_value)*

These requests are generated by GVRP application and by the GIP function.

Similarly, registration primitives are defined in order to allow the GID services to indicate to the user that a given attribute value is registered/de-registered (join/leave) on a given port. This is being indicated by the following two primitives:

1. *GID_Join.indication (attribute_type, attribute_value)*

2. *GID_Leave.indication (attribute_type, attribute_value)*

On receipt of such primitives, GID determines the action to be taken for these primitives. This depends upon the attribute type, attribute value and state machine associated with that participant.

4.3.2 GIP

GARP Information Propagation (GIP) function enables the attribute information registered on bridge ports to be propagated across the bridged LAN to other GVRP participants. The GIP operations for GVRP for the ports that are in forwarding state are as follows:

1. Any *GID_Join.indication* received by GIP from a given port in the set is propagated as a *GID_Join.request* to the instance(s) of GID associated with any other port in the set.

2. Any *GID_Leave.indication* received by GIP from a given port in the set is propagated as a *GID_Leave.request* to the instance(s) of GID associated with any other port in the set, iff, no registration exists for that attribute on any other port in the set excluding the port.

The effect of this propagation is that, for the set of ports defined by the GIP context, registrations are propagated on a given port, if any other port has seen a registration for the attribute concerned, and de-registrations are propagated on a given port, if all other ports are now de-registered for the attribute concerned.

4.3.3 GVRP in bridges

VLAN-aware bridges behave differently from the end stations. After registering to a VLAN in a port, the GVRP-aware bridge propagates this information to other bridges and end stations connected to them through all other ports. When the other bridges receive GVRP PDUs through any of their ports, they first check the port status and state machine information related to that port and VLANid associated with the attributes of the received PDU. Then depending the type of attribute event associated with the VLANid, they need to register/de-register for that VLAN and update their membership information. Further they need to transmit this information to all other bridges through their other ports. This results in automatic updating of the filtering databases and hence dynamic updation in the VLAN membership information.

4.4 GVRP Messages

GARP/GVRP has six attribute specific message types. They are:

1. *LeaveAll*: All registrations will shortly be de-registered; if any participants have a continuing interest in any of the registrations they need to rejoin in order to maintain the registration. The Hex value for it: 0x00.

2. *JoinEmpty*: The participant is not registered but wishes to declare for this attribute value. If there are any other participants wish to declare, then it cares for that attribute value. The Hex value for it: 0x01.

3. *JoinIn*: The participant wants to declare for this attribute value. It has either registered or does not care if there are any other participants wish to declare for this attribute value. The Hex value for it: 0x02.

4. *LeaveIn*: The participant had registered for this attribute value, but in the process of de-registering it. The Hex value for it: 0x03.

5. *LeaveEmpty*: The participant had registered but does not wish to de-register for this attribute value. It cares, if there are any other participants wish to declare for the attribute value. The Hex value for it: 0x04.

6. *Empty*: The participant is neither trying to declare nor is registered for this attribute value. But, it cares, if there are any other participants wish to declare it. The Hex value for it: 0x05.

When the above messages are sent by any of the participants and the other participants see those, they will not send the same message for that declaration. This is because both the recipient(s) and the transmitter for that message believe that there are participants that have made the declaration. In this kind of environment there is no acknowledgment being sent by any recipient for any message (use LLC type-1 service).

4.5 Applicant and Registrar

Each GARP/GVRP participant maintains two protocol components per attribute, viz., *Applicant* and *Registrar*. The Registrar just records attribute registration declared by other participants on the segment, without sending any protocol messages, whereas the Applicant ensures that the participant's declarations are registered by other participant's registrars. The applicants also ensure that the other participants have a chance to re-declare, after any one withdraws a declaration.

4.5.1 Registrar Behavior

The Registrar has three states. They are:

1. *LV*: The participant was registered for that attribute value, but is timing out the registration by using the LeaveTimer. After the expiry of the LeaveTimer the Registrar state goes to MT state. The Hex value for this: 0x0.

2. *IN*: The participant is registered for that attribute value. The Hex value for this: 0x1.

3. *MT*: All declarations for that attribute value are withdrawn. The Hex value for this: 0x2.

4.5.2 Applicant Behavior

Applicant maintains a simple counter (*send_count*) for any loss of messages (Join or Leave) between the participants. This counter value is either 0 or 1 or 2. That means, at maximum, the participant can send two JoinIn messages or receive two JoinIn messages or send one JoinIn message and receive one JoinIn message of a particular attribute value. This counter is incremented for every send/receive of JoinIn message for a particular attribute value. The counter resets to 0, for every receive of JoinEmpty, Empty, Leave, or LeaveAll message. Depending upon the *send_count* value the Applicants either go to *Very Anxious (V)* or *Anxious (A)* or *Quiet (Q)* state. If the *send_count* value is 2, then there is no need to send further messages.

The existence of Applicant comes into picture only when the participant tries to make a declaration. The bridge ports and end stations make active use of registered attribute values and maintain their GARP state machines [5]. When a participant attempts to make or maintain a declaration for a given attribute value, it is called a member in the Applicant state machine context. Any member can be an *Active Member (A)* or *Passive Member (P)* or *Observer (O)*.

4.6 State Machines in GVRP

The state machine description given here is the modified state machine implemented for GVRP for the Embedded Platform. Table 4.1 lists out the state machine parameters and their Hex values. These parameters are the real events in GVRP. Table 4.2 lists out the Registrar and Applicant states along with the Hex values.

Each participant maintains the Applicant and Registrar state tables for which they are members of Table 4.3 lists out the state table for the Applicant and Table 4.4 lists out the state table for the Registrar.

Event	Definition	Hex Value
rLeaveAll	Receive a LeaveAll message	0x0
rJoinEmpty	Receive a JoinEmpty message	0x1
rJoinIn	Receive a JoinIn message	0x2
rLeaveEmpty	Receive a LeaveEmpty message	0x3
rLeaveIn	Receive a LeaveIn message	0x4
rEmpty	Receive an Empty message	0x5
transmitPDU!	Opportunity to transmit a GVRP PDU JoinTimer starts	0x6
leavetimer!	LeaveTimer has expired	0x7
ReqJoin	GID service request to declare an attribute value	0x8
ReqLeave	GID service request to withdraw an attribute value	0x9

Table 4.1: State Machine Parameters of GVRP

4.7 GVRP Timers

GARP/GVRP defines three timers [5, 3] for safe and smooth operation. They are (a) *JoinTimer*, (b) *LeaveTimer*, (c) *LeaveAllTimer*. The JoinTimer controls the interval between the transmitPDU! (Table 4.1) events for the Applicant state machine. The LeaveTimer controls the period of time that the Registrar state machine will wait in the LV state before transmitting to the MT state. Similarly, the LeaveAllTimer controls the frequency with which the state machine generates LeaveAll PDUs. The timing relationship between these three timers in GVRP is shown in Figure 4.5. These timers are defined for per-port-per-attribute value. The timer resolution is 10ms. Also, there is another time parameter used called *Offset*. This offset is used for calculating the real time of expiry and wait in our implementation.

The updating of timers depends upon the Registrar state and Applicant state and the Events for that participant. The JoinTimer starts only if the participant wants to transmit a PDU for the attribute value. If JoinTimer expires and the *send_count* counter is less than 2, then the participant issues the same PDU and start the JoinTimer again.

The LeaveTimer starts only when the Registrar for a particular participant goes to LV state. If it expires and does not receive any JoinIn or JoinEmpty message for that attribute value, then the Registrar goes to MT state.

Registrar State			Applicant State		
Event	Definition	Hex Value	Event	Definition	Hex Value
IN	IN	0x0	VA	Very Anxious, Active	0x0
LV	Leaving	0x1	AA	Anxious, Active	0x1
MT	Empty	0x2	QA	Quiet, Active	0x2
-	-	-	LA	Leaving, Active	0x3
-	-	-	VP	Very anxious, Passive	0x4
-	-	-	AP	Anxious, Passive	0x5
-	-	-	QP	Quiet, Passive	0x6
-	-	-	VO	Very anxious, Observer	0x7
-	-	-	AO	Anxious, Observer	0x8
-	-	-	QO	Quiet, Observer	0x9
-	-	-	LO	Leaving, Observer	0xA

Table 4.2: Registrar and Applicant States

4.8 GVRP PDU

Each GVRP packet contains one or more GVRP messages. Each GVRP message is associated with one event and one attribute value (except for LeaveAll). The encoding of GVRP PDUs [5, 3] are as follows:

1. *Protocol Identifier*: This is the first two bytes of the PDU. For GVRP, the Hex value is 0x0001.

2. *Message*: It follows the Protocol Identifier. The Message ends with the *End of Mark*. The Hex value of the End of Mark is 0x00.

3. *Attribute*: Each message consists of an *Attribute Type* and an *Attribute List*, in order. Each attribute in the attributes list consists of an *Attribute Length*, an *Attribute Event*, and an optional *Attribute Value*, in order (if the event is LeaveAll, then the attribute vale field is not there, else, it is present). The Hex value of the attribute length is 0x04 if the attribute event is any thing other than LeaveAll and 0x02, if LeaveAll. The attribute value is a two byte VLANid in our case. Depending upon the state machine and events, either there will be a registration or a de-registration for that VLANid by the participant.

	State										
Event	VA	AA	QA	LA	VP	AP	QP	VO	AO	QO	LO
transmitPDU!	AA	QA	-	VO	AA	QA	-	-	-	-	VO
rJoinIn	AA	QA	QA	LA	AP	QP	QP	AO	QO	QO	AO
rJoinEmpty	VA	VA	VA	VO	VP	VP	VP	VO	VO	VO	VO
rEmpty	VA	VA	VA	LA	VP	VP	VP	VO	VO	VO	VO
rLeaveIn	VA	VA	VA	LA	VP	VP	VP	LO	LO	LO	VO
rLeaveEmpty	VP	VP	VP	VO	VP	VP	VP	LO	LO	LO	VO
LeaveAll	VP	VP	VP	VO	VP	VP	VP	LO	LO	LO	VO
ReqJoin	-	-	-	VA	-	-	-	VP	AP	QP	VP
ReqLeave	LA	LA	LA	-	VO	AO	QO	-	-	-	-

Table 4.3: Applicant State Table

The GVRP PDU format is shown in Figure 4.6.

4.8.1 Packet Format

GVRP PDUs are sent over Ethernet as IEEE 802.3 packets. So, they follow the IEEE 802.3 standards [19]. The first six bytes of these packets corresponds to the destination MAC address. In this case it is 01-80-C2-00-00-21, which is a multicast address in nature. The next six bytes corresponds to the source MAC address (MAC address of the packet generating device). The next two bytes corresponds to the length of the packet. After the length field, the packet contains the IEEE 802.2 LLC (Logical Link Control) header. Adding IEEE 802.2 LLC information to an 802.3 physical packet format requires three additional fields at the beginning of the Data field: a one byte Destination Service Access Point (DSAP) field, a one byte Source Service Access Point (SSAP) field, and a one byte Control field. Both DSAP and SSAP in this case are 0x42 and the Control field is set to 0x03. Since the control field is 0x03, it is similar to datagram kind of services and hence is un-acknowledged in nature. The GVRP packet structure is shown in Figure 4.7.

	State		
Event	IN	LV	MT
rJoinIn	IN	Stop LeaveTimer IndJoin IN	IndJoin IN
rJoinEmpty	IN	IN	IN
rEmpty	IN	LV	MT
rLeaveIn	Start LeaveTimer LV	LV	MT
rLeaveEmpty	Start LeaveTimer LV	LV	MT
rLeaveAll	Start LeaveTimer LV	LV	MT
LeaveTimer!	-	IndLeave MT	-

Table 4.4: Registrar State Table

4.9 GARP/GVRP Implementation on the Embedded Platform

We have implemented GARP/GVRP in the Embedded Platform under *uClinux* Environment. We have used raw sockets for receiving and sending the GVRP packets. The maximum number of VLANs that can be handled in our implementation is 4094. VLANid 4092 is the default VLANid and all the ports of the switch are members of it. So, at the start-up, the switch can send GVRP packets with VLANid (attribute value) as 4092 to all other connecting devices through all ports. By using the switch APIs and by manipulating the registers of the Embedded Platform, the switch is programmed to accept packets through all ports whose destination MAC address is 01-80-C2-00-00-21 (GVRP group address) and forwards them directly to the CPU as critical packets. The switch updates the MAC Table and insert the source MAC address of the packet into it. Using APIs and the updated MAC tables, we found the received port address for that packet. Then the packet is handed over to the *gvrp_pdu_recv()* subroutine, which separates the GVRP PDU part from the packet and updates the *packet_rx* pointer, which is global in nature. This pointer keeps the entire PDU (all attributes). This *packet_rx* pointer is passed to the *update_gvrp()* subroutine. This subroutine reads all the attributes till getting the *End Mark*. For each attribute, it finds out the the *Event*

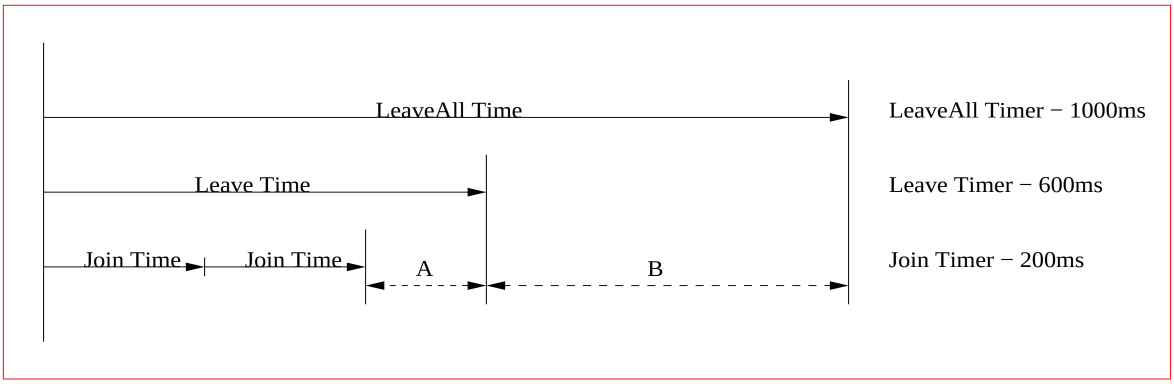

Figure 4.5: GVRP Timing Relationship

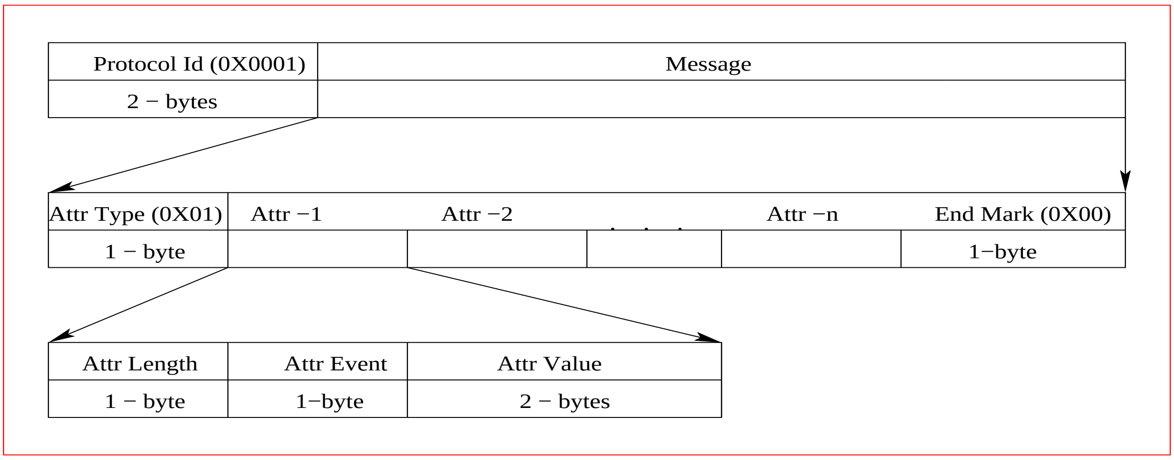

Figure 4.6: GVRP PDU Format

Type and the *Attribute Value*, which is present in all events other than LeaveAll event. If the event is not LeaveAll, then the switch may update the VLAN table and may transmit a GVRP packet in all other ports (except the received port).

If the switch is transmitting a GVRP packet, then it starts the JoinTimer associated with the VLANid and port and updates the *send_count* counter. This counter ensures the switch does not transmit more than two duplicate packets. The state tables are updated by the *check_port_status()* subroutine. Using *signal handler* we run one more thread in this program. This signal handler continuously monitor the timers (for all VLANids and all ports). Since we have 48 ports and 4094 VLANids, we have maximum of 48×4094 timers running at a particular instant. This is because, only one timer per port per VLANid can be running (out of the three GVRP timers) at any instant of time. So, the signal handler can find out the latest expired timer out of the set of 48×4094 timers. Here we add the offset time as a mater of safety for finding out the expiry time. After finding out the expired timer, we pass on this information either to *join_action()*

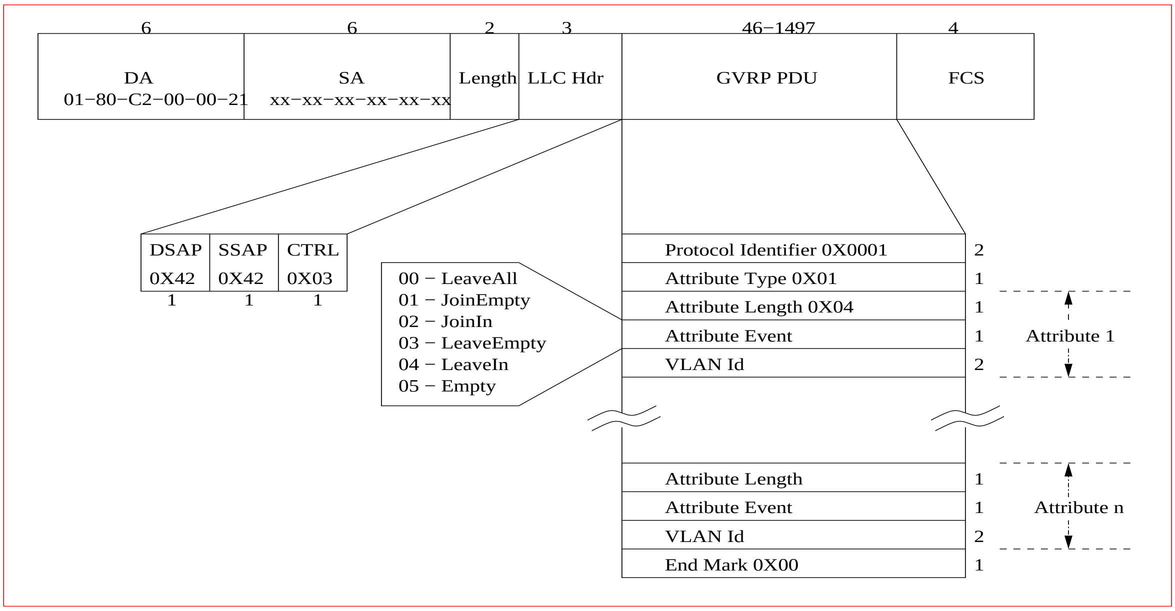

Figure 4.7: GVRP Packet Format

subroutine or to *leave_action()* or to *laveall_action()* subroutine for action. As a matter of safety, we use *pthread_mutex*, *pthread_mutex_lock* and *pthread_mutex_unlock* for locking the global variables during the updation by the signal handler. The pseudo code along with the subroutines are described in Appendix-A.

The schematic representation of the implementation of GARP/GVRP in the Embedded Platform is described in the subsections below.

4.9.1 Initialisation

Before starting the GARP/GVRP in the switch, we need to initialise the parameters of the Embedded Platform. It deals with the following things.

1. Setting the switch into VLAN-aware mode.
2. Setting the default VLANid 4092 for all ports.
3. Setting the up-linking port of the switch in forwarding mode.
4. Enabling the ingress filtering in all ports.
5. Disabling the egress tagging in up-link (port number 48 and 49) ports.
6. Inserting the GVRP group MAC address in the MAC table entry and enabling the switch to accept the GVRP packets as critical packets.
7. Creating the flood tables (48 flood tables are to be created) needed for transmitting the GVRP packets.

8. Initialising the *port_detail* structure.

The port_detail structure is shown below.

```
structport_detail {
   struct vid_port_pair PORT_VLAN[4094];
};

struct vid_port_pair {
   unsigned short int info;
   struct itimerval Port_Vid_Timer;    //can be Join/Leave/LeaveAll timer
   uints16 send_count;                 //can be 0,1,2
};
```

In the initialisation part, we have taken an array of (size 48) port_detail structures and initialised the parameters relating to the port and VLAN pair. It initialises *port_state, porttimer, timer_type* and *timer_state* parameters through bit manipulation of the *info*.

```
struct port_detail PORT_DETAIL[48];               //defined globally
.
.
PORT_DETAIL[i].PORT_VLAN[j].info=0x0000;          //initialization
.
.
PORT_DETAIL[i].PORT_VLAN[j].info ^=port_state;    //the first 2-bits (LSB)
PORT_DETAIL[i].PORT_VLAN[j].info ^=porttimer;     //the next 10-bits
PORT_DETAIL[i].PORT_VLAN[j].info ^=timer_type;    //the next 2-bits
PORT_DETAIL[i].PORT_VLAN[j].info ^=timer_state;   //the next 2-bits (MSB)
```

4.9.2 Raw Socket Implementation

The data structure of the header, PDU and the packet format of the GVRP packets are shown below.

```
struct garp_ethhdr                //corresponds to the packet header
{
 uint8 dest_hw_addr[6];           //Destination ether addr
 uint8 src_hw_addr[6];            //Source ether addr
 uint8 length[2];
 uint8 dsap[1];                   //0x42
```

```
 uint8 ssap[1];                  //0x42
 uint8 control[1];               //0x03 (Connection Less)
 };

struct gvrp_pdu {                //corresponds to the GVRP PDU
  uint8 proto_id[2];             //the protocol id - 0x0001
  uint8 attribute_type[1];       //the attribute type - 0x01
  uint8 attribute_length[1];     //the attribute length (either 0x02/04)
  uint8 attribute_event[1];      //the attribute event (0x00/01/02/03/04/05)
  uint8 attribute_value[2];      //the vlan id (optional)
  uint8 end_mark[1];             //the end mark  - 0x00
};

struct gvrp_packet {             //the GVRP packet
  uint8 packet_header[17];       //ethernet header (includes LLC)
  uint8 packet_message[1497];    //data
  uint8 packet_fcs[4];           //4-byte FCS
};
```

The use of RAW socket helps us to bypass the TCP/IP stack and to create our own header needed for GVRP. Also, we are not using IP header at all in these packets. The packet header contains the source and destination MAC address and the LLC header information. For the FCS part, we use our own algorithm to check and correct the packets.

```
struct sockaddr_ll socket_address;
s = socket(AF_PACKET, SOCK_RAW, htons(ETH_P_ALL));  //RAW socket
.
.
socket_address.sll_family   = PF_PACKET;
socket_address.sll_protocol = htons(ETH_P_IP);
socket_address.sll_ifindex  = ifindex;
socket_address.sll_hatype   = ARPHRD_ETHER;
socket_address.sll_pkttype  = PACKET_OTHERHOST;
socket_address.sll_halen    = ETH_ALEN;
```

4.9.3 Timers

As discussed earlier, in GVRP, we have maximum of 48×4094 timers running at a particular instant of time. Each timer can either be in running or in stopped or in expired mode. The timer values depends upon the type of timer (JoinIn, Leave or LeaveAll).

Instead of a fixed time, the timers run for a random period. The random period depends upon the maximum timer value of the timer. The random wait is generated by the use of the *random_wait()* subroutine. We use *start_timer()* and *stop_timer()* subroutines for starting and stopping the various timers. The prototypes of these subroutines are given below.

```
uints16 random_wait(uints16 max_wait) {
  .
  wait_time =1+(uints16) (max_wait*rand()/(RAND_MAX+max_wait));
  .
}

void start_timer(uints16 max_wait, uint32 portId, uint32 vid) {
  .
  wait_time=random_wait(max_wait)*1000;
  .
  PORT_DETAIL[portId].PORT_VLAN[vid].Port_Vid_Timer.it_value.tv_usec
                                                          =wait_time;
  setitimer(ITIMER_REAL,&PORT_DETAIL[portId].PORT_VLAN[vid].
                                               Port_Vid_Timer,NULL);
}

void stop_timer(uint32 portId, uint32 vid) {
  .
  PORT_DETAIL[portId].PORT_VLAN[vid].Port_Vid_Timer.it_value.tv_usec=0;
  setitimer(ITIMER_REAL,&PORT_DETAIL[portId].PORT_VLAN[vid].
                                                  Port_Vid_Timer,NULL);
}
```

4.9.4 Scheduling the Next Event

In the GVRP implementation, we create two threads; one as a normal thread for each GVRP packet it receives and acts, and the other for finding out the expired timer out of the 48×4094 timers and detecting the type of timer. This thread also, acts as signal handler for our implementation. The latest expired timer detected by the signal handler helps us to schedule the next event that the switch needs to look after with high priority. This is implemented by the use of the *pthread*, *pthread_mutex_lock*, *pthread_mutex_unlock* in our program. When the second thread is active, it just locks the global parameters from updating during its run time and calls the *timer_action()* subroutine to initiate an action for the specific participant of the expired timer.

4.9.5 VLANid Manipulation in the Embedded Platform

We use the switch APIs to insert and delete a VLANid for a port during the registration/ de-registration process of the GVRP. Also, we use *getMember()*, *displayMember()*, *deleteMember()*, subroutines for this purpose. The VLAN table updating depends upon the state table of both the registrar and the applicant, the *send_count* counter and the timer types related to a VLANid and port pair.

Chapter 5

Testing

For the testing of GARP/GVRP in the Embedded Platform we need at-least one end station (normal PC with VLAN-aware NIC) and two testing stations along with the Embedded Platform. The testing station can be any GARP/GVRP enabled switch or a GARP/GVRP enabled PC with multiple NICs. We have identified various LAN scenarios (Figure 5.1), where we can place these devices for our testing purpose [5, 20].

5.1 Various LAN Scenarios for GVRP Testing

5.1.1 Leaf LAN Scenario

In the Leaf LAN Scenario, the end station is connected to the GARP/GVRP enabled L-2 switch (Embedded Platform in our case). In turn the L-2 switch is connected to the Backbone LAN. In this scenario, the following tests can be carried out. (The Registrar of the switch is in MT state before the start of the testing in the first three cases)

1. *Initial Join and no Packet Loss*: In this case, the Registrar of the switch goes to IN state after receiving the first JoinIn message from the end station. There will not be any effect on the Registrar state even for the second JoinEmpty message (if, any) from the end station.

2. *Initial Join and First Join PDU Loss*: In this case, since the first PDU (JoinIn) sent by the end station is lost, the Registrar will still be in MT state till getting the second PDU (JoinEmpty) from the end station. After that it will go to IN state.

3. *Initial Join and Second Join PDU Loss*: The Registrar will be in IN state after receiving the first PDU (JoinIn) from the end station. The second PDU has no effect and hence the loss of it will not matter for the Registrar state.

4. *Last to Leave and no Packet Loss*: Before this test, we assume that the Registrar of the switch in IN state. After receiving the LeaveEmty message from the end station, the switch goes to LV state and will start the LeaveTimer. The second packet does not have any effect on the registrar state (unless a JoinIn/JoinEmpty message). After the expiry of the LeaveTimer, the switch goes to MT state.

5. *Last to Leave and First Leave PDU Loss*: In this case, since the first PDU is lost, the switch goes to LV state and start the LeaveTimer only after receiving the second LeaveEmpty PDU.

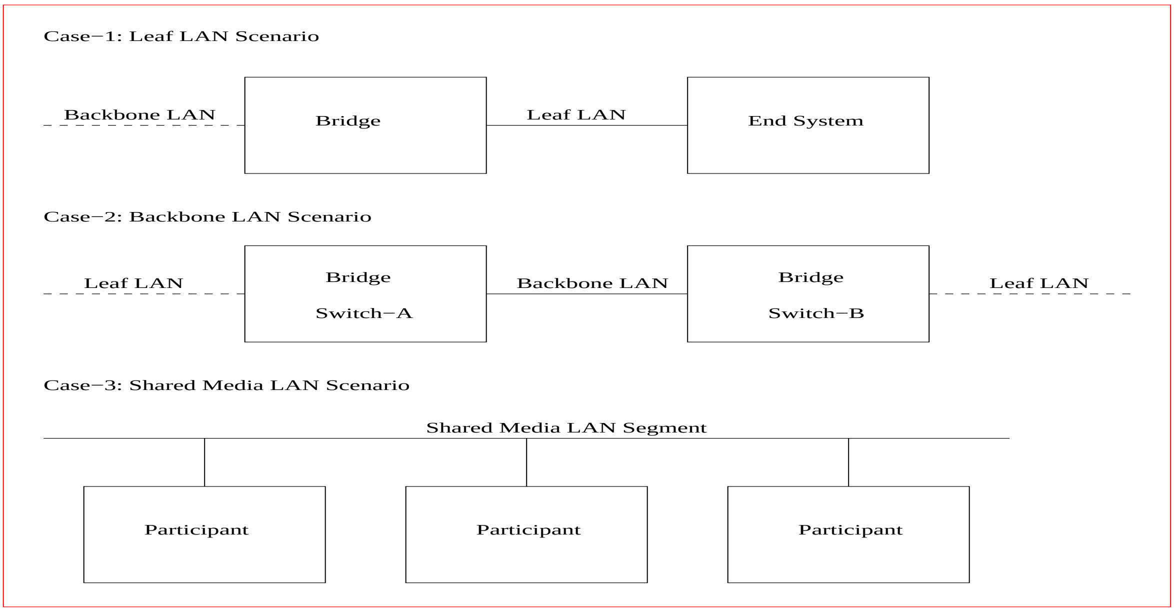

Figure 5.1: LAN Scenarios for GVRP Enabled Switch

5.1.2 Backbone LAN Scenario

In the Backbone Scenarios, two GVRP enabled switches are attached to the backbone LAN. Here the behaviour of bridge ports for the receive of GVRP PDU is tested. In this scenario, the Applicant of the Switch-A (Embedded Platform) receives a Join request from the end stations. This is the initial condition. The following tests can be carried out for this.

1. *Initial Join and no Packet Loss*: In this case, the Registrar of the Switch-A (Testing Station) goes to IN state, and it sends a GVRP PDU to Switch-B (Embedded Platform). It starts the JoinTimer. Switch-B after receiving the JoinIn request, goes to IN state and issues a JoinIn. This suppress the second JoinIn of Switch-A. Because of the JoinTimer of Switch-B, Switch-B will transmit another JoinIn message and will not have any impact on the Registrar value of A.

2. *Initial Join and First Join PDU Loss*: In this case, since the first PDU (JoinIn) sent by the Switch-A is lost and hence after the expiry of the JoinTimer, Switch-A will transmit the JoinIn message again. On receiving this message, Registrar of Switch-B will go to IN state and will issue a JoinIn message. On receiving this JoinIn message, Registrar of Switch-A will go to IN state. The second JoinIn message issued by Switch-B has no effect on Switch-A.

5.1.3 Shared Media LAN Scenario

In the Shared Media LAN Scenario, the switches and end stations are connected as shown in Figure 5.1. In this scenario, we can have the following test.

1. *Third Party Join*: In this case, any third party can receive the the GVRP PDUs sent by any switches. So, any third party can register/un-register for a particular VLAN after receiving the PDUs.

5.2 Flow of GVRP PDU

To test the dynamic flow the GVRP PDU, we have designed a test setup involving 3 GVRP enabled L-2 switch (including the Embedded Platform) and one end station. The end station, which is connected to Port-1 of Switch-C is a member of VLAN 100 (static). Similarly, Port-1 of Switch-A is a member of VLANs 1, 5, 6 (static).

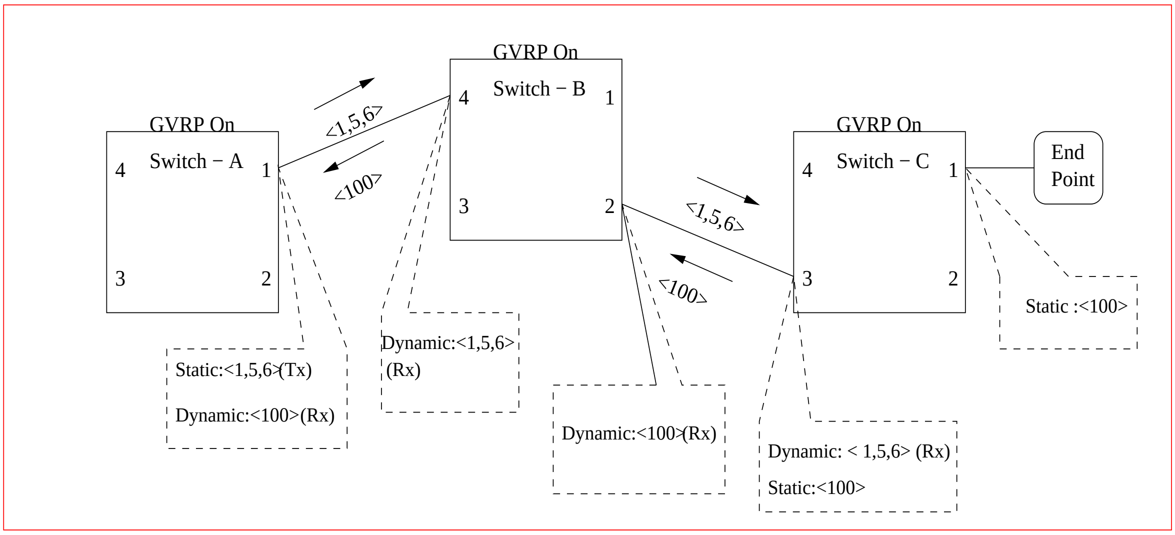

Figure 5.2: Example: Flow of GVRP PDU

If we start GVRP in all the three switches, the flow of GVRP PDUs and their directions will look like the flow and directions as shown in Figure 5.2. Since the ports which receive the PDUs only update their VLAN membership information dynamically and not the ports which forwards the received PDU, Port-2 of Switch-B should not be a member of VLANs 1,5,6 and Port-4 of Switch-B should not be a member of VLAN 100. The order of transmission of these PDUs between various switches and the timers can be viewed by using *Ethereal* [21].

5.3 Signaling between PE and CE

In this test, we can check the ACid to PWid translation and vice-versa, the encapsulation and de-encapsulation of ACids and PWids at PE for AC-bound and PW-bound traffic. Also, the complete path (Figure 1.1) connecting the two end stations forming the VPN can be checked. Using *Ethereal* [21] we can check the flow of data as well as signaling packets between the CEs and PEs. This will give a better understanding of the packet flow, packet loss and latency.

Chapter 6

Implementation Issues and Future Work

Since, we have considered access network part for our implementation, we designed the CE device which is a *uClinux* based Embedded Platform. This CE device is capable of dynamically assigning VLANs to customers using GARP/GVRP. Further, we have discussed various L2VPN services and their implementation issues such as: signaling techniques, end-point detection, forwarding schemes, AC provisioning etc., in our report. We have implemented signaling techniques between PE and CE, encapsulation and translation of AC-bound and PW-bound traffic at PE and encapsulation for CE-bound [1] and AC-bound traffic at CE.

In this chapter, we discussed the various implementation issues related to GARP/ GVRP in the Embedded Platform, scaling issues of VPWS and VPLS and the future scope of this report.

6.1 Issues related to implementation of GARP/GVRP in the Embedded Platform

There are some issues relating to the implementation of GARP/GVRP in the Embedded Platform using *uClinux*. They are categorised as follows:

6.1.1 Timers and Memory Management

In this implementation we have 48×4094 sets of VLAN-port-pair of participants (maximum) that can run simultaneously. For each participant we have one timer with one of the three states, one timer value, one Registrar state, one Applicant state and one send_count counter. So, the amount of memory it needs for implementing the timer

[1] customer traffic

and state information for all participant is large. To avoid this memory management problem, we use one *short integer* variable to store the port state, timer value, timer state, timer type with bit operations. This reduces the memory requirement drastically.

Since the implementation of GARP/GVRP is in user space (including the timers), there might be some problem relating to the timer values and the timers may not run for the exact time. It is because handling of traffic in the Embedded Platform has a higher priority than the user space timer. This is real-time constraints in *uClinux*.

6.1.2 Physical Limitations

There are a few physical limitations we faced during the implementation of GARP/GVRP in the Embedded Platform. They are:

1. In the Embedded Platform, the total number of VIDs is 1024, instead of 4096. The 4096 VLANs are mapped internally to 1024 VIDs.

2. There are only two up-link ports available in the Embedded Platform. So, we can connect only to two PEs for VPLS and VPWS.

3. The switch-on-chip can auto-learn/store 8K MAC addresses (maximum). This may not be enough for VPWS.

6.2 Issues related to VPWS and VPLS

6.2.1 Scaling Issues in VPWS

A VPWS is a VPN service that provides a L-2 point-to-point service [7]. That means we need to form $N \times (N-1)$ number of LSPs and $\frac{N \times (N-1)}{2}$ number of PWs. For each new addition of CE, we need to form N new virtual circuits. So we need to form N new PWs between the PE to which the new CE is connected and the other $(N-1)$ PEs to which the CEs of the same VPN are connected. Also, we need to provide individual ACs for these virtual circuits. In that case, between the new CE and the PE to which it is connected requires $(N-1)$ number of new ACs and hence ACids. It may lead to scaling problem, and the number of end-points that can be supported on a particular PE can be limited. For each virtual circuit in VPWS, CE should be able to map the destination MAC address to PWid and TAid. If the number of end point increases, then the database at CE will increase and may result to scaling problem.

6.2.2 Scaling Issues in VPLS

In VPLS, we have point-to-multipoint type of services and each ACid can be mapped to a set of PWids. Since VPLS is an L-2 service and it emulates LAN service across a [7] Wide Area Network (WAN), it will have all the scaling characteristics of a LAN. Other scaling issues might arise from the number of end-points that can be supported on a particular PE. Also, PE should be able to map the destination MAC address to PWids and hence need to keep the information of all the remote PEs as well as the MAC addresses of all the end points in its database. This may lead to scaling problem.

6.3 Future work

We have studied VLAN, VPWS and VPLS concepts and implemented the GARP/GVRP in CE. But, GARP/GVRP is not being implemented at PEs. So, the dynamic assignment of ACids and PWids are not possible in this implementation. The assignment of ACids and PWids have not been taken care of separately for VPLS and VPWS. All these implementations can done in future. For VPLS, there is a need to implement the Ethernet bridging and VSI at PE with full forwarding capabilities. Presently, the PW selection is independent of the Traffic Engineering Server. But for providing better services there is a need of Traffic Engineering server. Though we assumed the TE Server is in operational in our implementation, there is a need of co-ordination of TE Server and NSP of PE for translating the ACid to PWid (i.e., to select a PW dynamically). As we know, QoS is not in-built [1, 22] in the traditional Ethernet services, we need to provide the QoS [23] to the customers based on the SLA.

These issues need to be addressed in future implementation.

Appendix A

The Pseudo Code

```
while (RXD) {
 if (dst_MAC==GVRP_MAC) {
  packet_rx=gvrp_pdu_recv();
  portId=check_port_addr(src_MAC);

  i=0;
  while (!end_of_packet) {

  attri_type=pdu(i+2);
  attri_length=pdu(i+3);
  attri_event=pdu(i+4);

  vid_present=check_port_vid(VID,portId);

  if ((vid_present==False) && (attri_event==Leave)) {  //Leave/LeaveEmpty
    if ((port_lock==False) && (port_state==Forward)) {  //Learn/Forward
       leave_action(portId,attri_event,vid);
    }
  }

  else if ((vid_present==True) && (attri_event==Join)) {  //JoinIn/JoinEmpty

    if (port_lock==False) && (port_state==Forward))  {  //Learn/Forward
      if ((TimerType==JoinTimer) && (send_action <=2)) {
          join_action(portId,attri_event,vid);
      }
      else if (TimerType!=JoinTimer)
         join_action(portId,attri_event,vid);
```

```
     }
  }

 else
  leave_all_action(chipId,portId);
 j=attri_length+4;
 i+=j;
 }
}

//-----------------------1---------------------------
join_action(attri_event, portId, VID) {

 if (attri_event==Join) || (attri_event==JoinEmpty) {

  if (insert_vid(VID, port_id)!=-1)  {
    if (Registrar==LV)  {
       Registrar=IN;
       stop_leave_timer(leave_timer);
    }

    if (Registrar==MT)  {
       Registrar=IN;
       update_vid_table();
       txmit_pdu(pdu,floodIndex);
    }
  }
 }

 if (attri_event==Empty) {
   if (Registrar==LV)
    if (timer_check(leave_timer)==Expired)
      Registrar=MT;
 }
}

//-----------------------2---------------------------
leave_action(attri_event,portId,VID) {
```

```
 if (Registrar==IN) {
   Registarar=LV;
   start_timer(leave_timer);
 }

 if (Registrar==LV) {
   stop_timer(leave_timer);
   Registrar=MT;
   update_vid_table();
  }
}

//----------------------3---------------------------
leave_all_action(attri_event,portId,VID) {
  create_port_detail();
  reset_gvrp();
}

//---------------------4----------------------------
timer_handler() {

check_timer[i][j]=check_timer_status();

 if (timer_flag==expired) {
  if ((attri_event==Join) && (Registrar==IN))  { //JoinIn/JoinEmpty/Empty
    if (send_count<=2) {
     pthread_mutex_lock(message);
     transmit_pdu();
     update_gvrp();
     pthread_mutex_unlock(message);
    }
  }
  if ((attri_event==Leave) && (Registrar==LV)) {
    pthread_mutex_lock(message);
    update_gvrp();
    Registrar=MT;
    pthread_mutex_unlock(message);
  }
  if ((attri_event==LeaveAll) && (Registrar==LV)) {
```

```
    pthread_mutex_lock(message);
    leave_all_action();
    Registrar=MT;
    pthread_mutex_unlock(message);
 }
}
```

Bibliography

[1] R. Santitoro, *Metro Ethernet Services - A Technical Overview Forum.* Metro Ethernet Forum, December 2003. http://www.metroethernetforum.org/metro-ethernet-services.pdf.

[2] Ethernet Services Model, Phase 1, *Metro Ethernet Standards - Technical Specification.* Metro Ethernet Forum, November 2003. http://www.metroethernetforum.org/metro-ethernet-services.pdf.

[3] LAN/MAN Standards Committee, *IEEE Standards for Local and Metropolitan Area Network: Virtual Bridges Local Area Networks,* IEEE Std. 802.1Q-1998. IEEE Computer Society, December 1998. http://standards.ieee.org/getieee802/download/802.1Q-2003.pdf.

[4] L. Andersson and T. Madsen, "*PPVPN Terminology*," September 2003. Work in Progress, Internet Draft, http://www.ietf.org/internet-drafts/draft-andersson-ppvpn-terminology-04.txt.

[5] LAN/MAN Standards Committee, *IEEE Standard for Information Technology, Telecommunications and Information exchange between Systems, Local and Metropolitan Area Networks, Common specifications: Virtual Bridges Local Area Networks,* IEEE Std. 802.1D-1998. IEEE Computer Society, September 1998. http://standards.ieee.org/getieee802/download/802.1D-2003.pdf.

[6] S. Bryant and P. Pate, "*PWE3 Architecture*," October 2003. Work in Progress, Internet Draft, http://www.ietf.org/internet-drafts/draft-ietf-pwe3-arch-06.txt.

[7] L. Andersson and E. C. Rosen, "*L2VPN Framework*," October 2003. Work in Progress, Internet Draft, http://www.ietf.org/internet-drafts/draft-ietf-l2vpn-l2-framework-03.txt.

[8] E. C. Rosen and V. Radoaca, "*Provisioning Models and Endpoint Identifiers in L2VPN Signaling*," September 2003. Work in Progress, Internet Draft, http://www.ietf.org/internet-drafts/draft-ietf-l2vpn-signaling-00.txt.

[9] L. Martini, E. C. Rosen, and T. Smith, "*Pseudowire Setup and Maintenance using LDP*," December 2003. Work in Progress, Internet Draft, http://www.ietf.org/internet-drafts/draft-ietf-pwe3-control-protocol-05.txt.

[10] H. Shah, E. Rosen, W. Augustyn, G. Heron, S. Khandekar, V. Kompella, T. Smith, A. Vishwanathan, A. Moranganti, A. G. Malis, S. Wright, and V. Radoaca, "*ARP Mediation for IP Interworking of Layer 2 VPN*," October 2003. Work in Progress, Internet Draft, http://www.ietf.org/internet-drafts/draft-shah-l2vpn-arp-mediation-03.txt.

[11] L. Martini, E. C. Rosen, and G. Heron, "*Encapsulation Methods for Transport of Ethernet Frames Over IP/MPLS Networks*," December 2003. Work in Progress, Internet Draft, http://www.ietf.org/internet-drafts/draft-ietf-pwe3-ethernet-encap-05.txt.

[12] W. Augustyn and Y. Serbest, "*Service Requirements for Layer 2 Provider Provisioned Virtual Private Networks*," August 2003. Work in Progress, Internet Draft, http://www.ietf.org/internet-drafts/draft-augustyn-ppvpn-l2vpn-requirements-02.txt.

[13] X. Xiao, D. McPherson, and P. Pate, "*Requirements for Pseudo-Wire Emulation Edge-to-Edge (PWE3)*," November 2003. Work in Progress, Internet Draft, http://www.ietf.org/internet-drafts/draft-ietf-pwe3-requirements-07.txt.

[14] C. Lee and M. Higashiyama, "*CE-based Virtual Private LAN*," March 2003. Work in Progress, Internet Draft, http://www.ietf.org/internet-drafts/draft-lee-ce-based-vpl-02.txt.

[15] R. Aggarwal, R. Raszuk, F. L. Faucheur, C. Geoffrey, and J. D. Clercq, "*Signaling Tunnel Encapsulation/Deencapsulation Capabilities*," April 2003. Work in Progress, Internet Draft, http://www.ietf.org/internet-drafts/draft-raggarwa-ppvpn-tunnel-encap-sig-01.txt.

[16] "*uClinux*: Embedded Linux/Microcontroller Project," http://www.uclinux.org/.

[17] B. Davie and Y. Reckhter, *MPLS Technology and Applications*. Morgan Kaufmann Publishers, 2001.

[18] L. Martini, N. El-Aawar, S. Vogelsang, D. Tappan, J. Shirron, E. C. Rosen, T. Smith, A. Hamilton, J. Jayakumar, V. Radoaca, D. S. Vlachos, A. G. Malis, C. Liljenstolpe, V. Sirkay, G. Heron, D. Cooper, and K. Kompella, "*Encapsulation Methods for Transport of Layer 2 Frames Over IP and MPLS Networks*," April 2003. Work in Progress, Internet Draft, http://www.ietf.org/internet-drafts/draft-martini-l2circuit-encap-mpls-05.txt.

[19] J. Kadambi, I. Crayford, and M. Kalkunte, *Gigabit Ethernet: Migrating to High-Bandwidth LANs.* Prentice Hall PTR, 1998.

[20] University of New Hampshire, *Bridge Functions Consortium GARP/GVRP Operation Test Suite*. InterOperability Laboratory, July 2002.

[21] "Ethereal: A Protocol Network Analyzer," http://www.ethereal.com/.

[22] LAN/MAN Standards Committee, *Ethernet in the First Mile,* IEEE Draft P802.3ah/ D1.414. IEEE Computer Society, May 2003. http://www.ieee802.org/3/efm/public.

[23] Z. Wang, *Internet: QoS Architectures and Mechanisms for Quality of Service.* Morgan Kaufmann Publishers, 2001.