Επέκταση του εργαλείου ανάλυσης κίνησης δικτύου Wireshark για την υποστήριξη του πρωτοκόλλου GMPLS, με βάση το RFC 6004

Πατσιούρας Μιλτιάδης

AEM 652

Διπλωματική Εργασία

Πανεπηστημιο Θεσσαλίας
Τμήμα Μηχανικών Η/Υ Τηλεπικοινωνιών και Δικτύων

Wireshark network traffic analyzer extensions for GMPLS protocol support, based on RFC 6004

Patsiouras Miltiadis

Abstract

Network analysis (also known as traffic analysis, protocol analysis, sniffing, packet analysis, eavesdropping, and so on) is the process of capturing network traffic and inspecting it closely to determine what is happening on the network.

Wireshark is a network analyzer that decodes the data packets of common protocols and displays the network traffic in readable format. Protocols have headers and use specific values to be distinguishable. Wireshark and other similar software need to have knowledge of these values in order to display the network traffic in readable format.

The present thesis deals with the addition of code to a pre-existing dissector for GMPLS support for Metro Ethernet Forum and G.8011. As long as there are new protocols being used or new values for extra functions are added, network analysis software has to be properly updated. To do this the developer has to find the additions needed to be done, read and understand the source code and finally apply the changes.

Table of Contents

Part A: Wireshark protocol analyzer

1. Introduction to Network management

In computer networks, network management term refers to the operation, administration, maintenance, and provisioning of networked systems [1]. Network management is essential to command and control practices and is generally carried out of a network operations center.

Operation field deals with keeping the network (including the services the network provides) up and running smoothly. It includes monitoring the network, using tools like Wireshark, to spot the problems as soon as possible ideally before users are affected.

Network administration involves a wide array of operational tasks that help network to run smoothly and efficiently. The main tasks associated with network administration include:

- Design, installation and evaluation of the network
- Execution and administration of regular backups
- Creation of precise technical documentation, such as network diagrams, network cabling documents, etc.
- Provision for precise authentication to access network resources
- Provision for troubleshooting assistance
- Administration of network security, including intrusion detection

Network maintenance is concerned with performing repairs and upgrades – for example, when equipment must be replaced, when a router needs a patch for an operating system image, when a new switch is added to the network. Maintenance also involves corrective and preventive measures to make the management network run "better", such as adjusting device configuration parameters. Network maintenance includes five major elements.

- **Fault tolerance management** focus on the network devices and the physical and virtual connections that operate at the three lower levels of the OSI reference model.
- **Configuration management** develops a configuration map of software and hardware on the network. Also adds new devices to the configuration map, ensures that only one version of software is operating on the network, changes the operational characteristics of managed objects, and record any changes in the state of them.
- **Performance management** measures the workload, the throughput, the resource waiting time, the response time, the propagation delay and any quality of service (Qos) change.
- **Security management** deals with the sensitivity of information and defines the levels of access needed by each user. Furthermore services related to data integrity, and delivery of information identified by OSI documents on security management, but not yet defined.
- **Accounting management** has not been completely specified by OSI. Rules under consideration include the use of accounting meters, which are triggers for updating data and for reporting usage.

Network provisioning refers to the provisioning of the customer's services to the network elements. It requires the existence of network equipment and depends on network planning and design. In the context of computer networking, provisioning is divided into the following subjects:

- **Internet access provisioning** which includes multiple client system configuration steps that vary according to connection technology and involve modem configuration, driver installation, wireless local area network (LAN) setup, Internet browser configuration, email system configuration and additional user-requested software installation.
- **Server provisioning** which prepares a server for network operation via the installation and connection of data, software and systems.
- **Storage Area Network (SAN) provisioning** which optimizes performance by efficiently appropriating storage to all users.

Data for network management is collected through several mechanisms, including agents installed on infrastructure, synthetic monitoring that simulates transactions, logs of activity, sniffers and real user monitoring. In the past network management mainly consisted of monitoring whether devices were up or down; today performance management has become a crucial part of the IT team's role which brings about a host of challenges—especially for global organizations.

2. Network Analysis/Packet sniffing and Wireshark

Network analysis (also known as packet sniffing, traffic analysis, protocol analysis, packet analysis, eavesdropping and so on) is the process of capturing network traffic and inspecting it closely to determine what is happening to the network. The software or device used to do this is called packet sniffer. A network analyzer decodes the data packets of common protocols and displays the network traffic in readable format. A sniffer is a program that monitors data travelling across the network. Packet sniffing has legitimate uses to monitor network performance or troubleshoot problems with network communications. However, it is also widely used by hackers and crackers to gather information illegally about networks they intend to break into. Using a packet sniffers it is possible to capture data like passwords, IP addresses, protocols being used, and other information that will help the attacker infiltrate the network.

Protocol analyzers also known as packet sniffers are commonly used by network technicians to diagnose network-related problems. Protocol analyzers work by intercepting and logging network traffic that they can 'see' via the wired or wireless network interface that the packet sniffing software has access to on its host computer. Depending on the type of the network and network structure, the capabilities of protocol analyzers may differ. Once the information is captured the packet sniffing software decodes it from raw digital form to a human readable format that permits its users to easily review the exchanged information. Hackers can use sniffers to eavesdrop on unencrypted data in the packets to see what information is being

exchanged between two parties. They can also capture information such as passwords and authentication tokens (if they are sent in the clear). Hackers can also capture packets for later playback in replay, man-in-the-middle, and packet injection attacks that some systems may be vulnerable to. Protocol analyzers vary in their abilities to display data in multiple views, automatically detect errors, determine the root causes of errors, generate timing diagrams, reconstruct TCP and UDP data streams, etc.

There is a wide variety of packet analyzer software on the internet for different users' demands. Open source and freeware software applications are the applications that are used mostly by network engineers and hackers, as well.

Widely used packet analyzers are the following:

1. **Tcpdump** is a popular command line, free network sniffer that was used by almost everyone before Wireshark showed up. Tcpdump works primarily on Unix-like operating systems but there is a port of it that works on Windows too. It is great for tracking down network problems or monitoring activity. (For more information: tcpdump.org)
2. **Wireshark** also known as Ethereal (till summer 2006) is a free and open source packet analyzer. It is one of the most popular packet analyzers in the world and works on both Unix, as well as Windows. Wireshark also includes a tcpdump-like console version Tshark. (For more information: wireshark.org)
3. **Kismet** is a free console based wireless network detector and intrusion detection system for 802.11 wireless links (more specifically 802.11a, 802.11.b, 802.11g, and 802.11n). Works on Windows and Unix-like operating systems and log traffic in Wireshark/ Tcpdump compatible format. (For more information: kismetwireless.net)

A network analyzer can be a standalone hardware device with specialized software, or software that is installed on a desktop or laptop computer. The differences between network analyzers depend on features such as the number of supported protocols it can decode, the user interface, and its graphing and statistical capabilities. Other differences include inference capabilities (e.g., expert analysis features) and the quality of packet decodes. Although several network analyzers decode the same protocols, some will work better than others for your environment.

Wireshark is the world's foremost network protocol analyzer and is the focus of this thesis. It is an open source network analyzer for Unix and Windows that allows user to examine data from a live network or from a capture file on disk and interactively browse the captured data, delving down into just the level of packet detail you need. Wireshark has several powerful features, including a rich display filter language and the ability to view the reconstructed stream of a TCP session. It also supports hundreds of protocols and media types. A tcpdump-like console version named ethereal is included. One word of caution is that Ethereal has suffered from dozens of remotely exploitable security holes, so stay up-to-date and be wary of running it on untrusted or hostile networks (such as security conferences).

A typical network analyzer (in figure 1 there is an example of Wireshark display window) displays captured traffic in three panes:

- **Summary:** This pane displays a one-line summary of the capture. Fields include the date, time, source address, destination address, and the name and information about the highest-layer protocol.
- **Detail:** This pane provides all of the details (in a tree-like structure) for each of the layers contained inside the captured packet.
- **Data:** This pane displays the raw captured data in both hexadecimal and text format.

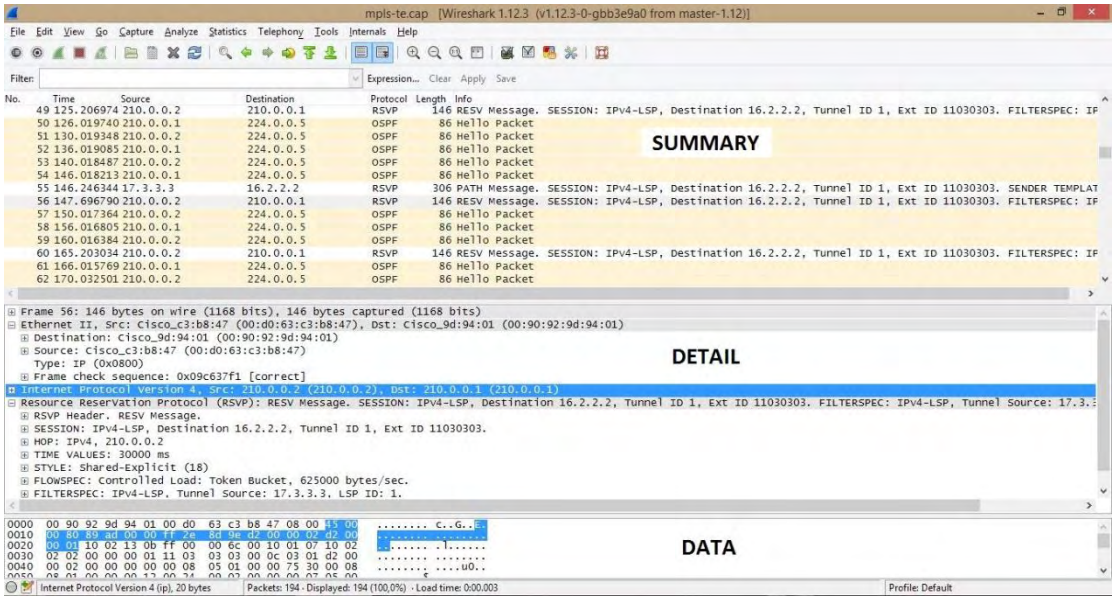

Figure 1: Wireshark Analyzer Display Window

3. What is a Wireshark Dissector

As it was mentioned before Wireshark is a packet analyzer that will try to capture network packets and tries to display that packet data as detailed as possible. It has to be clear that Wireshark is not an intrusion detection system. Thus, warning you when someone does strange things on the network isn't provided but, if strange things happen, Wireshark might help you finding out what is really going on. Furthermore, Wireshark will only "measure" things from the network and not interact with it at all.

Wireshark is divided in individual code parts for each protocol. These code parts are called dissectors. So, when Wireshark captures the information each dissector decodes its part of the protocol, and then hands off decoding to subsequent dissectors for an encapsulated protocol. Every dissection starts with the Frame dissector which dissects the packet details of the capture file itself (e.g. timestamps). From there it passes the data on to the lowest-level data dissector, e.g. the Ethernet dissector for the Ethernet header. The payload is then passed on to the next dissector (e.g. IP) and so on. At each stage, details of the packet will be decoded and displayed.

Dissection can be implemented in two possible ways. One is to have a dissector module compiled into the main program, which means it's always available. Another way is to make a plugin (a shared library or DLL) that registers itself to handle dissection. There is little difference in having your dissector as either a plugin or built-in. On the Windows platform you have limited function access through the ABI exposed in libwireshark.def, but that is mostly complete. The big plus is that your rebuild cycle for a plugin is much shorter than for a built-in one. So starting with a plugin makes initial development simpler, while the finished code may make more sense as a built-in dissector.

3.1 Permanent Dissector

Wireshark requires certain things when setting up a protocol dissector. Basic skeleton code for a dissector is provided so it can be copied to a new file and fill in. It is recommended that where possible you keep to the IANA abbreviated name for the protocol, if there is one, or a commonly-used abbreviation for the protocol, if any (skeleton code lives in the file "packet-PROTOABREV.c" in directory Wireshark/doc/). New dissector should be placed in directory Wireshark/epan/dissectors where the others packet-*.c files exist.

Dissectors that use the dissector registration API to register with a lower level protocol (this is the vast majority) don't need to define a prototype in their .h file. For other dissectors the main dissector routine should have a prototype in a header file whose name is "packet-", followed by the abbreviated name for the protocol, followed by ".h"; any dissector file that calls your dissector should be changed to include that file.

3.2 Plugin Dissector

Writing a plugin dissector is not very different from writing a standard one. All functions described in README.developer file can be used in plugin dissectors just like they are used in a standard one. Plugin dissectors are easier to start with. Writing a plugin dissector provides two options to programmer, 1) to add dissector as a custom extension, 2) build a permanent addition. Using a custom extension makes the dissector easier to configure but dissector won't be used for inclusion in next Wireshark's distribution. However, not all OSes on which Wireshark run can support plugins.

Plugin should be placed in Wireshark/plugins/protoabbrev directory with at least the following files:

- AUTHORS
- COPYING
- ChangeLog
- CMakeLists.txt
- Makefile.am
- Makefile.common
- Makefile.nmake
- moduleinfo.h

- moduleinfo.nmake
- plugin.rc.in
- And of course the source and header files for your dissector.

AUTHORS, COPYING, ChangeLog, Makefile.nmake, and plugin.rc.in files don't need any changes at all, you can copy them from any other plugin dissector folder. CMakeLists.txt, Makefile.am, Makefile.common, moduleinfo.h, and moduleinfo.nmake need minor changes which can be made according to corresponding files from any other plugin dissector folder.

(These files and their use are described in detail in Wireshark/doc/README.plugin file.)

Part B: Implementation

The purpose of this thesis is to extend Wireshark protocol analyzer in order to recognize and decode signaling based on the additions to Resource ReserVation Protocol (RSVP) that are introduced in RFC 6004 view to control two specific types of Ethernet switching via Generalized Multi-Protocol Label Switch (GMPLS). Before continuing with the final step of implementation and code presentation we should get familiar with GMPLS protocol suite and RSVP.

1. Generalized Multiprotocol Label Switching (GMPLS)

1.1 From Multiprotocol Label Switching (MPLS) to GMPLS

MPLS is a mechanism in high-performance telecommunications networks that directs data from one network node to the next based on short path labels rather than long network addresses, avoiding complex lookups in a routing table. Each packet gets labeled on entry into the service provider's network by the ingress router. All subsequent routing switches perform packet forwarding based only on those labels. Finally, the egress router removes the label(s) and forwards the original IP packet towards its final destination. MPLS can be used to carry many different kinds of traffic, including IP packets, as well as native Asynchronous Transfer Mode (ATM), Synchronous Optical Network (SONET), and Ethernet frames.

GMPLS is a protocol suite extending MPLS to manage further classes of interface and switching technologies other than packet interfaces and switching, such as time division multiplexing, layer-2 switching, wavelength switching and fiber switching. The support of additional types of switching has driven GMPLS to extend certain base functions of traditional MPLS and, in some cases, to add functionality. Unlike MPLS label, GMPLS label is not arbitrary set and it can represent (a) a single fiber in a bundle, (b) a single waveband within fiber, (c) a single wavelength within a wave band (or fiber), or (d) a set of time-slots within a wavelength (or fiber). The Generalized Label can also carry a label that represents a generic MPLS label, a Frame Relay label, or an ATM label.

The three main protocols that compose GMPLS are:

- Resource Reservation Protocol with Traffic Engineering extensions (RSVP-TE) signaling protocol.
- Open Shortest Path First with Traffic Engineering extensions (OSPF-TE) routing protocol.
- Link Management Protocol (LMP).

3. GMPLS basics

Before someone try to study and understand how GMPLS protocols developed out of MPLS it is necessary to examine some of the requirements of a transport network and what is the difference between a transport network and a MPLS-TE packet network.

3.1 GMPLS Label

The label in MPLS architecture is an arbitrary tag for a data packet that is used as an index into the Label Forwarding Information Base (LFIB). The resources are not tightly coupled with the labels. The resource management in MPLS is purely statistical, so the labels based on resources will only indicate the amount of resources that are statistically reserved. For example, the available bandwidth on an interface is only divided by the number of LSPs that use the interface and the total resource reservation may actually be allowed to exceed the available bandwidth because all flows will be at their maximum capacity.

In transport networks the physical resources are exactly the switchable quantities. That is, in a WDM network the lambdas are switched, in a TDM network the timeslots are switched, and so forth. Thus, a label that identifies a switchable data stream in GMPLS also precisely identifies a physical resource. So in a lambda switching network a label identifies a specific wavelength, in a TDM network a label identifies a specific timeslot, and in a fiber switching network a label identifies a specific port or fiber. This fact brings challenges that are not found in packet switching environments. One implication, for example, is that labels come from a disjoint set (for example, identifying the frequencies of the lambdas) rather than being arbitrary integers. Similarly, the set of valid labels is likely to be much smaller in a transport switch. Further, the interpretation of a label must be carefully understood — no longer is this an arbitrary tag, but it identifies a specific resource, and both ends of a link must have the same understanding of which resource is in use.

In GMPLS the meaning of a label is private between two adjacent LSRs, but they must have the same understanding of that meaning. TDM labels are given a special encoding so that the referenced timeslot may be deduced, but for lambda and fiber switching the meaning of the label is left as a matter for configuration or negotiation through the Link Management Protocol.

3.2 Switching types

LSRs or more precisely interfaces on LSRs, included in GPMLS architecture can be subdivided into the following classes:

- Interfaces that recognize packet boundaries and can forward data based on the content of the packet header. Such interfaces are referred to as Packet-Switch Capable (PSC).
- Interfaces that forward data based on the data's time slot in a repeating cycle. Such interfaces are referred to as Time-Division Multiplex Capable (TDM).
- Interfaces that forward data based on wavelength on which the data is received. Such interfaces are referred to as Lambda Switch Capable (LSC).

- Interfaces that forward data based on position of the data in the real world physical spaces. Such interfaces are referred to as Fiber-Switch Capable (FSC).
- Interfaces that recognize frame/cell boundaries and can switch data based on the content of the frame/cell header. Such interfaces are referred to as Layer-2 Switch Capable (L2SC).

A circuit can be established only between, or through, interfaces of the same type. Depending on the particular technology being used for each interface, different circuit names can be used, e.g., SDH circuit, optical trail, light-path, etc. In the context of GMPLS, all these circuits are referenced by a common name: Label Switched Path (LSP).

3.3 Label Switched Path (LSP)

At each switch along the circuit, resources are cross connected. All switches take traffic from an incoming resource and send it to an outgoing resource. In GMPLS the resources are associated directly with labels so we are able to define LSP as a contiguous series of cross-connected resources capable of delivering traffic. In data plane this gives us a trail of {interface, label, cross-connect} triplets. In control plane LSP describe the state (that is control blocks, memory, and so forth) that is used to manage LSP in data plane.

3.4 Bandwidth in GMPLS

In MPLS transport networks, an LSP is directly related to physical switchable resource, so the bandwidth can only be divided up according the capabilities of the switching device –this typically forces the bandwidth division to be in large units of bytes per second. For example a bandwidth of 2.5, or 10, or even 40 Gbps will be allocated (depends on channel capacity) for a service over a wavelength switching network that requires only 10Kbps.

On the other hand, in a GMPLS there is no danger that a traffic flow will violate the user-network agreement and consume more than the allotted bandwidth that was allocated during service setup.

Furthermore there are various techniques such as the use of hierarchical LSPs and Forwarding Adjacencies that have been developed to help GMPLS to make optimal use of bandwidth where the service needs to use only a small proportion of the available resources.

3.5 Bi-directionality in GMPLS

Contrary to MPLS, in GMPLS networks the LSPs are bidirectional. That attribute gives GMPLS transport networks many advantages over MPLS. Although it is possible to construct bidirectional connectivity using a pair of unidirectional LSPs, using bidirectional LSPs is preferred. A bidirectional LSP can be set up more smoothly, quickly and with less processing as it needs only one set of control plane messages to be exchanged, also there is often a requirement for connectivity in each direction to share common links (such as fiber pairs) to provide fate sharing.

The separation of control and data plane may increases the speed of communication but it also brings a lot of complications and challenges for the GMPLS protocols. For example, with separate control and data planes new techniques are required to verify the connectivity and aliveness of data plane links, because the successful delivery of signaling messages can no longer be used. Further, mechanisms need to be added to the signaling protocols to allow the management of data plane failures. For example, if some controller is notified by a data plane hardware component about a failure, it should be able to send an appropriate notification to a node that is responsible for service recovery. It also should be possible to set up and shut down services in an alarm free manner, so that no false alarms are raised.

As a result of control and data plane separation, failures on the data plane don't necessarily affect control plane and vice versa. For example if a data plane LSR fails then there will be no problem in delivering data because the control plane entity will take care of it and will fix the problem or find an alternate route for the data. Similarly when an entity in the control plane fails there will be no problem in data plane. Although data services are partially controlled, they will continue to function properly indefinitely.

3.6 Hierarchical LSPs and Tunneling

GMPLS supports the concept of hierarchical LSPs which occurs when a new LSP is tunneled inside an existing higher-order LSP so that the preexisting LSP serves as a link along the path of a new LSP. This hierarchy of LSPs can occur on the same interface, or between different interfaces.

For example, a hierarchy can be built if an interface is capable of multiplexing several LSPs from the same technology (layer), e.g., a lower order SONET/SDH LSP (e.g., VT2/VC-12) nested in a higher order SONET/SDH LSP (e.g., STS-3c/VC-4). Several levels of signal (LSP) nesting are defined in the SONET/SDH multiplexing hierarchy.

The nesting can also occur between interface types. At the top of the hierarchy are FSC interfaces, followed by LSC interfaces, followed by TDM interfaces, followed by L2SC, and followed by PSC interfaces. This way, an LSP that starts and ends on a PSC interface can be nested (together with other LSPs) into an LSP that starts and ends on a L2SC interface. This LSP, in turn, can be nested (together with other LSPs) into an LSP that starts and ends on a TDM interface In turn, this LSP can be nested (together with other LSPs) into an LSP that starts and ends on a LSC interface, which in turn can be nested (together with other LSPs) into an LSP that starts and ends on a FSC interface.

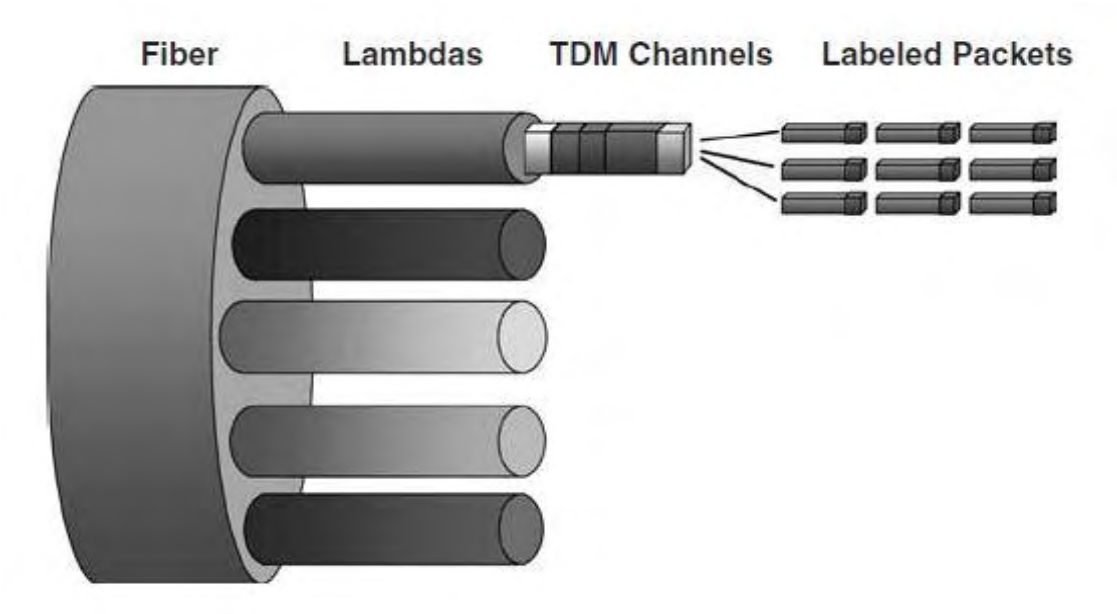

Figure 2: Example of LSPs Encapsulation

The separation of control and data plane may increases the speed of communication but it also brings a lot of complications and challenges for the GMPLS protocols. For example, with separate control and data planes new techniques are required to verify the connectivity and aliveness of data plane links, because the successful delivery of signaling messages can no longer be used. Further, mechanisms need to be added to the signaling protocols to allow the management of data plane failures. For example, if some controller is notified by a data plane hardware component about a failure, it should be able to send an appropriate notification to a node that is responsible for service recovery. It also should be possible to set up and shut down services in an alarm free manner, so that no false alarms are raised.

As a result of control and data plane separation, failures on the data plane don't necessarily affect control plane and vice versa. For example if a data plane LSR fails then there will be no problem in delivering data because the control plane entity will take care of it and will fix the problem or find an alternate route for the data. Similarly when an entity in the control plane fails there will be no problem in data plane. Although data services are partially controlled, they will continue to function properly indefinitely.

3.7 GMPLS signaling

Signaling is the process of exchanging messages within the control plane to set up, maintain, modify, and terminate data paths in the data plane. In the GMPLS context, these data paths are LSPs. In GMPLS data switches are called LSRs. In contrast to other protocol suites in GMPLS the data switch and the signaling controller may be physically diverse and communicate with a control/management protocol. Also a single signaling controller may manage more than one data switch. A possible configuration is shown in **Figure 3.** GMPLS signaling is using the Resource Reservation Protocol that is presented briefly later in this document.

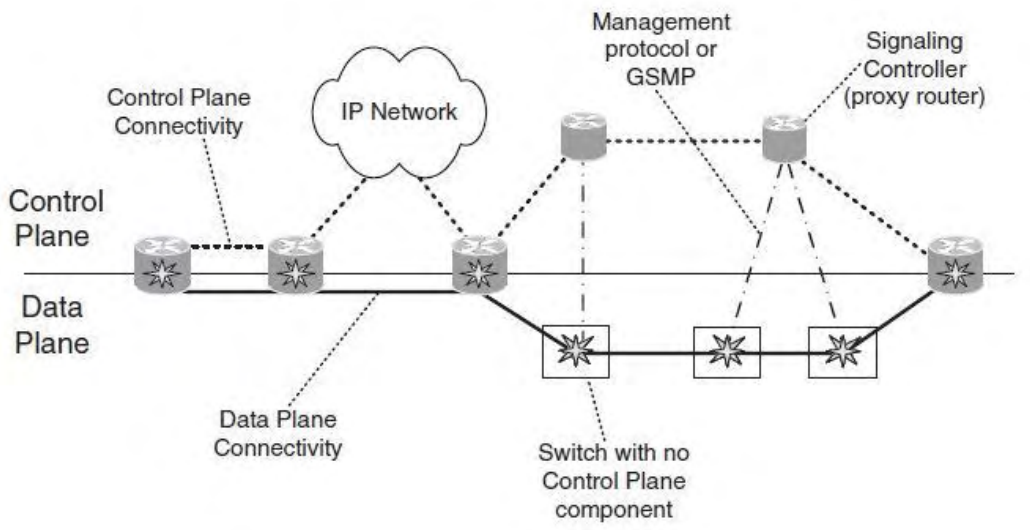

Figure 3: Possible configuration of signaling controllers and data switches

3.8 LSPs in GMPLS

In RSVP a session is the grouping of traffic flows to a particular destination. All traffic flows that share the session can share resources within the network. Resource Reservation Protocol - Traffic Engineering (RSVP-TE) - an extension of RSVP- introduced the concept of an MPLS tunnel. MPLS tunnel has explicit ingress/egress and insertion of data in the tunnel almost always guarantees the delivery to exit point of tunnel.

The focus in GMPLS is on delivering an end to end service, which could be considered as a tunnel. Each service in GMPLS may be supported by more than one LSP. Because LSPs have the properties of tunnels sometimes are referred to as LSPs tunnels.

An LSP is a path through the network formed of cross-connected labels (resources) on a series of data plane links. The route that the LSP uses can be selected in three ways depending on the requirements of the application that is establishing the LSP, and the capabilities of the network.

- The route can be left completely open for the network to select.
- The route of the LSP may be completely specified by the application or the operator.
- Alternatively, the operator or application may leave the selection of the route to the control

3.9 Generalized label

As it was mentioned before labels in GMPLS are not just arbitrary tags that identify the LSP in the data plane, but they are also associated with physical resources such as a timeslot, a wavelength, or a whole fiber. It is important, therefore, that the control plane should communicate what sort of LSP is required, what resources should be allocated, and therefore what type of label to use.

A Path message should contain as specific an LSP Encoding Type as possible to allow the maximum flexibility in switching by transit LSRs. In GMPLS, the label is requested using the Generalized Label Request object [2] (figure 4). This allows the ingress to specify three parameters to the LSP

- The LSP Encoding Type indicates the way data will be packaged within the LSP
- The Generalized PID (G-PID) identifies the use to which the LSP will be put - that is, the payload. This field is of use only to the egress LSR and allows it to know whether it can successfully terminate the LSP.
- The Switching Type governs the type of label that is allocated. This field indicates what type of switching should be performed on the LSP when the data is received by the downstream LSR.

Generalized Label object is entirely arbitrary and its content is a local matter between adjacent LSPs. This allows implementations to place any useful information within it as long as its interpretation is explained to their neighbors. It also allows for multiple labels to be negotiated and assigned to a single LSP.

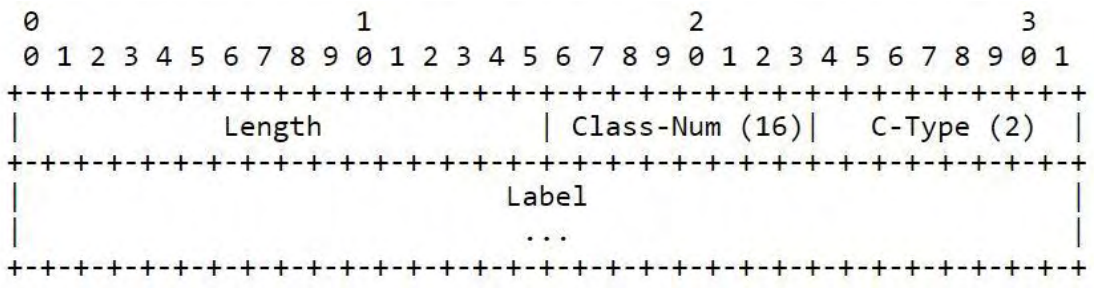

Figure 4: Generalized Label Format

4 Resource Reservation Protocol (RSVP)

RSVP is an internet signaling protocol [3] designed to reserve resources for simplex flows across a network and it is the means by which applications communicate their requirements to the network in an efficient and robust manner. RSVP does not provide any network service; it simply communicates any end-system requirement to the network. RSVP operates over an IPv4/IPv6 Internet Layer and provides receiver-initiated setup of reservations for multicast (one source - many receivers) or unicast (one source - one receiver) data flows. It is designed to operate with other routing protocols and provides them with several reservation styles which can be increased in future protocol revisions.

4.1 RSVP Overview

RSVP is the signaling protocol that installs and maintains reservation state information at each router along the path of a stream. RSVP transfers reservation data as opaque data; it can also transport policy control and traffic control messages. RSVP does not perform its own routing but it uses underlying routing protocols to determine where it should carry reservation requests. As routing changes paths to adapt to topology changes, RSVP adapts its reservation to the new paths wherever reservations are in place due to its “soft state” conservation.

A host uses RSVP to request a specific Quality of Service (QoS) from the network, on behalf of an application data stream. RSVP carries the request through the network, visiting each node the network uses to carry the stream. At each node, RSVP attempts to make a resource reservation for the stream.

RSVP created to fulfill demands that the growth of the internet and thus, the great variety of users’ demands and capabilities emerge. Hence, RSVP was built based on some design goals towards this direction. These design goals will be reported in summary.

First design goal for RSVP is to provide the ability for heterogeneous receivers to make reservations specially tailored to their own needs. For instance, a source may be sending a layer encoding of video signal; it is possible that certain receivers, which are doing the decoding in software, would only have sufficient power to decode the low-resolution signal or maybe the paths used to reach the receivers may not have the same capacity and so forth.

Another issue raised by the presence of multiple receivers; the membership in a multicast group can be dynamic. Reinitiation of the reservation protocol can be very demanding for large groups because as the group grows the changes in group membership become more frequent too. So the second design goal for RSVP is to deal gracefully with changes in multicast group membership.

Third design goal for RSVP is to allow end users specify their application needs so that the aggregate resources reserves for a multicast group can more accurately reflect the resources actually needed by the group. For example, in an audio conference with several people, there is usually one person, or at most a few people, talking at any one time because of the normal dynamics of human conversation. Thus, instead of reserving enough bandwidth for every potential speaker to speak simultaneously, in many circumstances it would be adequate to reserve only enough network resources to handle a few simultaneous audio channels.

Examining the example above there is another issue appeared. A receiver should be able to switch among sources without the risk of having the change request denied, as could occur if a new reservation request had to be submitted in order to switch channels. So, the fourth design goal for RSVP is to enable this channel changing feature.

RSVP is not a routing protocol, and should avoid replicating any routing functions. RSVP's task is to establish and maintain resource reservations over a path or a distribution tree, independent of how the path or tree was created. At the same time, however, RSVP should be able to cope with the resulting routing changes. That’s why RSVP’s fifth design goal is to

deal gracefully with such changes in routes, automatically reestablishing the source reservations along the new paths as long as adequate resources are available.

The sixth design goal for RSVP is to control protocol overhead; this means both avoiding the explosion in protocol overhead when group size gets large, and also incorporating tunable parameters so that the amount of protocol overhead can be adjusted.

Finally the last design goal for RSVP is not to specify the problem at hand but rather to be a general matter of modular design; so the design of RSVP is relatively independent of the other architectural components such as flow specification, routing, admission control, and packet scheduling.

4.2 RSVP Header Format

The RSVP messages are carried in RSVP Header. Header format is shown in **figure 5**.

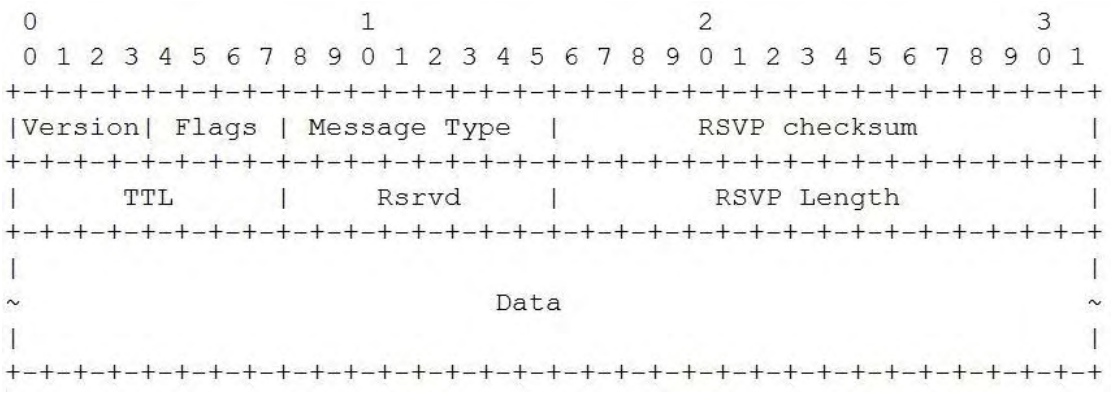

Figure 5: RSVP Header Format

Version: This 4-bit field defines the version of RSVP. (So far protocol version 1 is defined)

Flags: This 4-bit field is used to support for protocol extensions to neighboring RSVP routers. Last bit (R) is used to indicate Refresh reduction capability. From the 8 other values, 0x01 signals Refresh overhead reduction and 0x02-0x08 are reserved for later use.

Message Type: This 8-bit field shows the message type that is going to follow in data field. Most commonly used messages are these which described below (0x01 PATH, 0x02 RESV, 0x03 ResvConf, 0x04 PathErr, 0x05 ResvErr, 0x06 PathTear, and 0x07 ResvTear). There are a lot of values assigned so far.

RSVP Checksum: This 16-bit field is the one's complement of the one's complement sum of the message with the checksum field cleared to zero for the purpose of computing the checksum. If cleared to zero, no checksum was transmitted.

TTL: This 8-bit value shows the IP TTL value with which the message was sent.

Rsrvd: This 8-bit field is reserved for later use.

RSVP Length: This 16-bit field shows the total length of RSVP message including the common header and the variable length objects that follow.

Data: This is a variable length field and it contains one or more Objects.

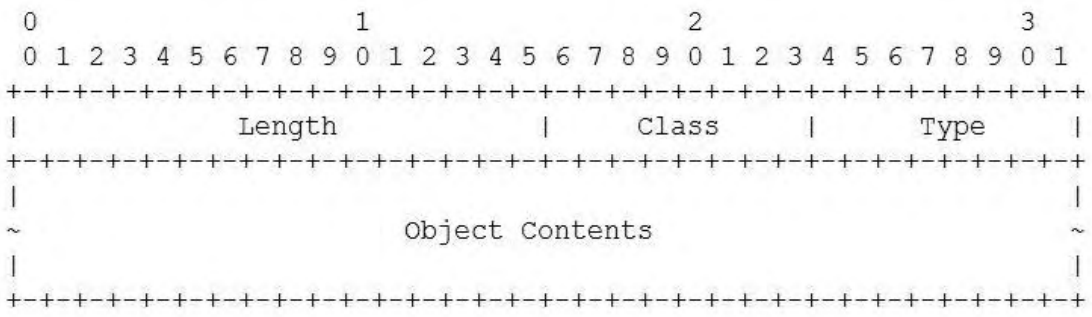

Figure 6: RSVP's Object Format

RSVP objects are carried in data field following the common RSVP Header. RSVP Objects have a specific but they have a variable length. Object's format is shown in **Figure 6**.

Length: This 16-bit field shows the total length of object (in bytes) including the length of this field.

Class: This 8-bit filed indicates the class of the object. Two new objects: LINK_CAPABILITY object (defined in RFC 6001 with Class-Number=133) and CALL_ATTRIBUTES object (defined in RFC 6001 with Class Number=202) were added to RSVP dissector for the purpose of this thesis.

Type: This 8-bit field specifies the object type. This is a sub value of the class.

Object Contents: This is a variable length field that contains one or more sub-objects, TLVs. Labels etc. depending on Class Number and Class Type. So this field does not have a specific format

4.3 RSVP messages

RSVP uses messages to accomplish the reservation of resources. An RSVP message consists of common header, followed by a body consisting of a variable number of variable length, typed objects. The main RSVP messages are PATH and RESV messages. RSVP receivers use reserve (RESV) message to periodically advertise to the network their interest in a flow, specifying the flow and filter specifications. RSVP senders on the other hand, use the PATH message to indicate that they are senders and give information such as the previous hop IP address, the multicast group IP address, templates for identifying traffic from that sender, and sender traffic specifications. The message is sent to all receivers in the multicast tree using the forwarding table maintained by the multicast grouping protocol. The RESV message is forwarded back to the sources by reversing the path state of PATH messages (using the previous hop IP address stored from PATH messages). In addition, there is a reservation confirmation message (ResvConf), two types of error messages (PathErr and ResvErr), and two

types of teardown messages (PathTear and ResvTear). These messages are explained briefly in the table below.

Message	Meaning	Purpose
PATH	Path establishment	Used by senders to specify their traffic characteristics.
RESV	Reservation request	Use by receivers to advertise to the network their interest in a flow.
ResvConf	Reservation confirmation	Indicates to the receiver successful installation of a reservation at an upstream node.
PathErr	Path error	Indicates to the sender an error in the path message.
ResvErr	Reservation error	Indicates to the receivers that reservation request has failed or an active reservation has been preempted.
PathTear	Path teardown	Deletes path state and dependent reservation state.
ResvTear	Reservation teardown	Deletes reservation state.

5. Implementation

The purpose of this thesis was to fix a Wireshark's bug and more precisely bug 7841 (New dissector for RFC 6004 - GMPLS Support for Metro Ethernet Forum and G.8011). After studying RFC 6004 and all the referenced RFCs it was clear that there was no need for a new dissector, instead of that, I had to extend the preexisting dissector for Resource Reservation Protocol to meet the needs of GMPLS signaling.

This thesis's objective is to follow the instructions given in RFC 6004 in order to extend RSVP's signaling recognition. For this purpose a number of RFCs and documents that are relevant with GMPLS and more specifically with the signaling of it was studied. Another big part of this thesis was the understanding of Wireshark's code and all the functions that are allowed to use.

5.1Request For Comments (RFC)

A Request for Comments (RFC) is a publication of the Internet Engineering Task Force (IETF) and the Internet Society, the principal technical development and standards-setting bodies for the Internet.

An RFC is authored by engineers and computer scientists in the form of a memorandum describing methods, behaviors, research, or innovations applicable to the working of the Internet and Internet-connected systems. It is submitted either for peer review or simply to convey new concepts, information, or (occasionally) engineering humor. The IETF adopts some of the proposals published as RFCs as Internet standards.

Request for Comments documents were invented by Steve Crocker in 1969 to help record unofficial notes on the development of ARPANET. RFCs have since become official documents of Internet specifications, communications protocols, procedures, and events

An Internet Document can be submitted to the IETF by anyone, but the IETF decides if the document becomes an RFC. Eventually, if it gains enough interest, it may evolve into an Internet standard.

Each RFC is designated by an RFC number. Once published, an RFC never changes. Modifications to an original RFC are assigned a new RFC number.

A list with relevant RFCs that were studied for the implementation of RFC 6004:

1. **RFC 3209** (RSVP-TE: Extensions to RSVP for LSP Tunnels)
2. **RFC 3471** (Generalized Multi-Protocol Label Switching (GMPLS) Signaling Functional Description)
3. **RFC 3473** (Generalized Multi-Protocol Label Switching (GMPLS) Signaling Resource ReserVation Protocol Traffic Engineering (RSVP-TE) Extensions)
4. **RFC 3477** (Signaling Unnumbered Links in Resource ReserVation Protocol – Traffic Engineering (RSVP-TE))
5. **RFC 4875** (Extensions to Resource Reservation Protocol – Traffic Engineering (RSVP-TE) for Point to Multipoint Label Switched Paths (LSPs))
6. **RFC 4974** (Generalized MPLS (GMPLS) RSVP-TE Signaling Extensions in Support of Calls)
7. **RFC 6001** (Generalized MPLS (GMPLS) Protocol Extensions for Multi-Layer and Multi-Region Networks(MLN/MRN))
8. **RFC 6002** (Generalized MPLS (GMPLS) Data Channel Switching Capable (DCSC) and Channel Label Extensions)
9. **RFC 6003** (Ethernet Traffic Parameters)
10. **RFC 6005** (Generalized MPLS (GMPLS) Support for Metro Ethernet Forum and G.8011 User Network Interface (UNI))

5.2 RFC 6004

RFC 6004 describes a method for controlling two specific types of Ethernet switching via Generalized Multiprotocol Label Switching. RFC 6004 supports the types of switching corresponding to the Ethernet services that have been defined in the context of Metro Ethernet Forum (MEF) and International Telecommunications Union (ITU) G.8011. Specifically, switching in support of Ethernet Private Line (EPL) and Ethernet Virtual Private Line (EVPL) services and support for MEF- and ITU-defined parameters are covered.

RFC 6004 uses a common approach to support the switching corresponding to the Ethernet services which builds on standard GMPLS mechanisms to deliver the required control capabilities. Within the context of GMPLS, support is defined for point to point unidirectional and bidirectional Traffic Engineering Label Switched Paths (TE-LSPs) which are described in

detail in RFC 3473, and unidirectional point to multipoint TE-LSPs which are described in RFC 4875.

A set of service attributes that are associated with EPL and EVPL connections are defined in (G.8011.1), (G.8011.2), (MEF11) and (MEF10.1). Some of these attributes are based on the provisioning of the local physical connection and are not modifiable or selectable per connection. Other are specific to a particular connection or must be consistent across the connection. These are the attributes that RFC 6004 focus on and describe them in detail. More specifically the attributes that will be supported in signaling include

- Endpoint Identifiers
- Connection Identifiers
- Traffic Parameters
- Bundling / VLAN IDs map (for EVPL only)
- VLAN ID preservation (for EVPL only)

In RFC 6004, common procedures that used to support Ethernet LSPs and procedures related in to the signaling of switching in support of EPL and EVPL services are described.

5.3 GMPLS Support for Metro Ethernet Forum and G.8011

The document is divided in three parts. The first and main part of document describes the common signaling support that is needed for implementing the GMPLS control plane over an Ethernet network. The other two parts describe EPL's and EVPL's parameters which needed to be added. Final additions to dissector were not only from RFC 6004 since there were additions in RSVP-TE described in other RFCs but there were not already implemented in Wireshark.

5.3.1 Signaling support

More specifically in section 2 of RFC 6004, the Ethernet Endpoint identification, the Connection Identification, traffic parameters, and Bundling and VLAN identification are described.

For Ethernet Endpoint Identification, Ethernet Endpoint ID TLV is used. The Ethernet Endpoint ID TLV follows the Attributes TLV format defined in FRC 6001. The Endpoint ID TLV has the following format:

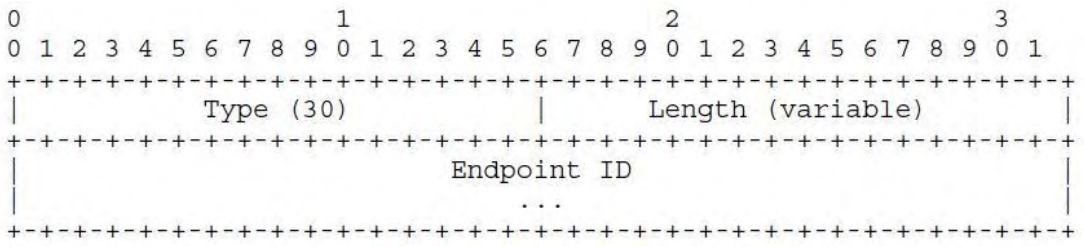

Figure 7: Endpoint ID TLV

Type: This 8-bit field defines the type of TLV and for Endpoint ID TLV this field must be set to thirty (30).

Length: This 8-bit field defines the length of the whole TLV (in bytes) including Type and Length field.

Endpoint ID: This is a variable size filed that carries an endpoint identifier. This endpoint identifier is described in more detail in MEF 10.1 and G.8011 and it must be null padded.\ as defined in RFC 6001.

Signaling for Ethernet connections follows the procedures defined in RFC 4974. In particular, the Call-related mechanisms are used to support endpoint identification. In the context of Ethernet connections, a Call is only established when one or more LSPs (connections in RFC 4974 terms) are needed. An LSP will always be established within the context of a Call and, typically, only one LSP will be used per Call.

Ethernet connections established according RFC 6004 must use the traffic parameters defined in RFC 6003 in the FLOWSPEC and TSPEC objects. Additionally the switching Granularity field of the Ethernet SENDER_TSPEC object must be set to zero (0). Service attributes are carried in the traffic parameters. These attributes support:

- Bandwidth profile
- VLAN Class of Service (CoS) Preservation
- Layer 2 Control Protocol (L2CP) Processing

Layer 2 Control Protocol TLV is used for L2CP processing and is defined in RFC 6004. Layer 2 Control Protocol TLV has the following format:

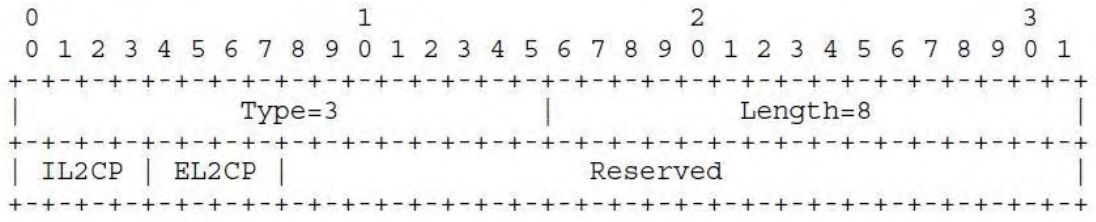

Figure 8: Layer 2 Control Protocol TLV

Type: this 8-bit field defines the type of TLV and for Endpoint ID TLV this field must be set to three (3).

Length: This 8-bit field defines the length of the whole TLV (in bytes) including Type and Length field. Length value must be set to eight (8).

Ingress Layer 2 Control Processing (IL2CP): This 4-bit filed controls processing of Layer 2 Control Protocols on a receiving Interface. Permitted values are shown in the following table:

Value	Description	Reference
0	Reserved	-
1	Discard / Block	MEF10.1 , G.8011.1 , G8011.2
2	Peer / Process	MEF10.1 , G.8011.1 , G8011.2
3	Pass to EVC / Pass	MEF10.1 , G.8011.1 , G8011.2
4	Peer and Pass to EVC	MEF10.1

Egress Layer 2 Control Processing (EL2CP): This 4-bit field controls processing of Layer 2 Control Protocols on a transmitting interface. Permitted values are shown in the following table:

Value	Description	Reference
0	Reserved	-
1	Based on IL2CP Value	MEF 10.1
2	Generate	G.8011.1 , G.8011.2
3	None	-
4	Reserved	-

Reserved: This 24-bit field is reserved. It must be set to zero on transmission and must be ignored on receipt. This field should be passed unmodified by transmit nodes.

Ethernet connections established according to RFC 6004 must include IL2CP TLV in the RFC 6003 traffic parameters carried in the FLOWSPEC and TSPEC objects.

Finally in signaling support, Bundling and VLAN Identification is supported only for EVPL services.

5.3.2 Ethernet Private Line Service (EPL)

Ethernet Private Line services carry "Ethernet characteristic information over dedicated bandwidth, point-to-point connections, provided by SDH, ATM, MPLS, PDH, ETY or OTH server layer networks". Signaling for the EPL service types only differ in the LSP Encoding Type used. The LSP Encoding Type used for each are:

EPL Service	LSP Encoding Type (Value)	Reference
Type 1/MEF	Ethernet (2)	RFC 3471
Type 2	Line (e.g., 8B/10B) (14)	RFC 6004

The other LSP parameters specific to EPL Service are:

Parameter	Name (Value)	Reference
Switching Type	DCSC (125)	RFC 6002
G-PID	Ethernet PHY (33)	RFC 3471, RFC 4328

5.3.3 Ethernet Virtual Private Line Service (EVPL)

EVPL service allows for multiple Ethernet connections per port, each of which supports a specific set of VLAN IDs. The service attributes identify different forms of EVPL services, e.g., bundled or unbundled. Independent of the different forms, LSPs supporting EVPL Ethernet type switching are signaled using the same mechanisms to communicate the one or more VLAN IDs associated with a particular LSP (Ethernet connection). The relevant parameter values that must be used for EVPL connections are:

Parameter	Name (Value)	Reference
Switching Type	EVPL (30)	RFC 6004
LSP Encoding Type	Ethernet (2)	RFC 3471
G-PID	Ethernet PHY (33)	RFC 3471, RFC 4328

LSPs that provide EVPL service type Ethernet switching MUST use the EVPL Generalized Label Format (figure 9), and the Generalized Channel_Set Label Objects per RFC 6002.

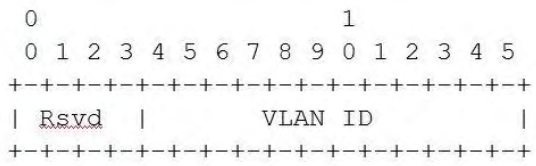

Figure 9: VLAN ID

6. Appendix

The code for this thesis is uploaded at https://code.wireshark.org/review/#/c/5323/. In this paragraph I' m going to add a summary of objects, specific types of preexisting objects, sub-objects, and TLVs that were needed for RSVP's extension.

Here is the list of objects that added or changed:

Object	Action	Reference
GENERALIZED_LABEL Object	Added	RFC 6004
IF_ID RSVP_HOP Object	+ case 4 (IPv6)	RFC 3473
IF_ID ERROR_SPEC Object	+ case 4 (IPv6)	RFC 3473
LINK_CAPABILITY Object	Added	RFC 4974
CALL_ATTRIBUTES Object	Added	RFC 6001
SENDER_TSPEC Object	+ case 2 (Ethernet Bandwidth Profile TLV) + case 3 (Layer 2 Control Protocol TLV)	RFC 6004
FLOWSPEC Object	+ case 2 (Ethernet Bandwidth Profile TLV) + case 3 (Layer 2 Control Protocol TLV)	RFC 6004

Additions also made in the following TLVs.

TLV	Action	Reference
Ethernet_Tspec	+ case 3 (Layer 2 Control Protocol TLV)	RFC 6004
Call_attributes	+ case 2 (Ethernet Endpoint ID TLV)	RFC 6004

Here is an example with an LINK_CAPABILTY object addition step by step.

1. Find the class number of the object searching IANA's Resource Reservation Protocol Parameters.(LINK_CAPABILITY C-Number = 133)
2. Register object

Assign value to a variable

```
enum rsvp_classes[]{

        RSVP_CLASS_NULL                 = 0,

        RSVP_CLASS_SESSION,

        …

        RSVP_CLASS_RESTART_CAP,

        RSVP_CLASS_LINK_CAP  = 133,

        …

}
```

Assign variable to a string.

```
static const value_string rsvp_class_vals[] = {
        …
        { RSVP_CLASS_RESTART_CAP,          "RESTART-CAPABILITY object"},
        { RSVP_CLASS_LINK_CAP,             "LINK-CAPABILITY object"},
        { RSVP_CLASS_VENDOR_PRIVATE_5,     "VENDOR PRIVATE
        object(10bbbbbb: ""ignore if unknown)"},
        ...
}
```

Assign variable to a filter name

```
static inline int rsvp_class_to_filter_num (int classnum) {
        switch (classnum) {
                ...
                case RSVP_CLASS_LINK_CAP :
                   return RSVPF_LINK_CAP;
                case RSVP_CLASS_DIFFSERV :
                   return RSVPF_DIFFSERV;
                …
}
```

Assign variable to a tree type

```
static inline int rsvp_class_to_tree_type (int classnum) {
        ….
        case RSVP_CLASS_LINK_CAP :
            return TT_LINK_CAP;
        ….
}
```

3. Create the dissection function.

```
static void dissect_rsvp_link_cap(proto_item *ti, packet_info* pinfo, proto_tree
*rsvp_object_tree, tvbuff_t *tvb, int offset, int obj_length,  int rsvp_class, int type) {

        proto_item_set_text(ti, "LINK CAPABILITY: ");

        switch(type) {

        case 1:

                proto_tree_add_uint(rsvp_object_tree, hf_rsvp_ctype, tvb, offset+3, 1,
                type);

                dissect_rsvp_ero_rro_subobjects(ti, pinfo, rsvp_object_tree, tvb, + 4,
                obj_length, rsvp_class);

        break;

        default:

                proto_tree_add_uint_format_value(rsvp_object_tree, hf_rsvp_ctype, tvb,
                offset+3, 1, type, "Unknown (%u)", type);

                proto_tree_add_item(rsvp_object_tree, hf_rsvp_record_route_data, tvb,
                offset+4, obj_length - 4, ENC_NA);

        break;

}  }
```

4. Add the dissection function in dissect_rsvp_msg_tree()

```
static void dissect_rsvp_msg_tree(tvbuff_t *tvb, packet_info *pinfo, proto_tree *tree, int
tree_mode, rsvp_conversation_info *rsvph, gboolean e2ei) {

        ….

        switch(rsvp_class){

                …

                case RSVP_CLASS_LINK_CAP:

                dissect_rsvp_link_cap(ti, pinfo, rsvp_object_tree, tvb, offset, obj_length,
                rsvp_class, type);

                break;

                ….

        }…}
```

References

[1] A. Clemm, Network Management Fundamentals. CiscoPress, 2006

[2] L. Berger, “Generalized Multi-Protocol Label Switching (GMPLS) Signaling Functional Description” (RFC 3471), January 2003

[3] L. Berger, Ed. “Generalized Multi-Protocol Label Switching (GMPLS) Signaling Resource ReserVation Protocol-Traffic Engineering (RSVP-TE) Extensions”, January 2003.

[4] A. Farrel, I.Bryskin, “GMPLS Architecture and Applications”, December 2005

[5] D. Awduche, L. Berger, D. Gan, T. Li, V. Srinivasan, G. Swallow, **“**RFC 3209 RSVP-TE: Extensions to RSVP for LSP Tunnels”, December 2001

[6] L. Berger, “RFC 3471 Generalized Multi-Protocol Label Switching (GMPLS) Signaling Functional Description”, January 2003

[7] L. Berger, “RFC 3473 Generalized Multi-Protocol Label Switching (GMPLS) Signaling Resource ReserVation Protocol Traffic Engineering (RSVP-TE) Extensions”, January 2003

[8] K. Kompella, Y. Rekhter, “RFC 3477 Signaling Unnumbered Links in Resource ReserVation Protocol – Traffic Engineering (RSVP-TE)”, January 2003

[9] R. Aggarwal, D. Papadimitriou, S. Yasukawa, “RFC 4875 (Extensions to Resource Reservation Protocol – Traffic Engineering (RSVP-TE) for Point to Multipoint Label Switched Paths (LSPs)”, May 2007

[10] D. Papadimitriou, A. Farrel, “RFC 4974 Generalized MPLS (GMPLS) RSVP-TE Signaling Extensions in Support of Calls”, August 2007

[11] D. Papadimitriou, M. Vigoureux, K. Shiomoto, D. Brungard, JL. Le Roux, “RFC 6001 Generalized MPLS (GMPLS) Protocol Extensions for Multi-Layer and Multi-Region Networks (MLN/MRN)”, October 2010

[12] L. Berger, D. Fedyk, “RFC 6002 Generalized MPLS (GMPLS) Data Channel Switching Capable (DCSC) and Channel Label Extensions”, October 2010

[13] D. Papadimitriou, “RFC 6003 Ethernet Traffic Parameters”, October 2010

[14] L.Berger, D.Fedyk, “RFC 6004 Generalized MPLS (GMPLS) Support for Metro Ethernet Forum and G.8011 Ethernet Service Switching”, October 2010

[15] L. Berger, D. Fedyk, “RFC 6005 Generalized MPLS (GMPLS) Support for Metro Ethernet Forum and G.8011 User Network Interface (UNI)”, October 2010