HELSINKI UNIVERSITY OF TECHNOLOGY
Faculty of Electronics, Communications and Automation
Department of Communications and Networking

Timo Karttunen

Implementing Soak Testing for an Access Network Solution

Master's thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in Technology.

Espoo, February 12, 2009

Supervisor: Professor Raimo Kantola
Instructor: M.Sc. Sami Larkimo

HELSINKI UNIVERSITY OF TECHNOLOGY Abstract of the master's thesis

Author: Timo Karttunen

Name of the Thesis: Implementing Soak Testing for an Access Network Solution

Date: February 12, 2009 **Language:** English **Number of pages**: 9 + 87

Faculty: Faculty of Electronics, Communications and Automation

Professorship: Networking Technology **Code:** S-38

Supervisor: Professor Raimo Kantola

Instructor: M.Sc. Sami Larkimo

The quality requirements are extremely demanding for telecommunications software. Operators usually have SLA agreements with their customers, and violations to that contract may lead to serious compensations. Furthermore, every moment that equipment or some service is not operating correctly means lost income for the operator. For these reasons, it is extremely important for a telecommunications equipment to continue functioning properly without service affecting breaks.

The purpose of this thesis was to design and implement automated soak testing for the IP/MPLS-based Tellabs 8600 router series. The system under test is composed of several network elements and a graphical Tellabs 8000 Network Management System. The purpose of this testing environment is to reveal defects that do not show up immediately in functional or regression testing but may manifest when the system is used for longer periods or operations are executed many times. A framework for automatically operating the test network and detecting problems programmatically was implemented in this thesis.

The testing environment was successfully implemented and satisfies the objectives initially set for it. Testing environment has been taken into use in system testing at Tellabs and after deployment has turned out to be useful and effective. Another identical environment was also implemented for the system testing group.

Keywords: IP, MPLS, VPN, access network, network management, testing, test automation, soak testing

TEKNILLINEN KORKEAKOULU Diplomityön tiivistelmä

Tekijä: Timo Karttunen

Työn nimi: Pitkän ajan testausjärjestelmän toteutus liityntäverkkoratkaisulle

Päivämäärä: 12.02.2009 **Kieli:** Englanti **Sivumäärä:** 9 + 87

Tiedekunta: Elektroniikan, tietoliikenteen ja automaation tiedekunta

Professuuri: Tietoverkkotekniikka **Koodi:** S-38

Työn valvoja: Prof. Raimo Kantola

Työn ohjaaja: DI Sami Larkimo

Tietoliikennelaitteiden ohjelmistojen toiminnalle asetetaan erittäin kovat laatuvaatimukset. Operaattoreilla on yleensä asiakkaiden kanssa SLA sopimukset, joiden rikkomisesta operaattorit saattavat joutua maksamaan suuriakin korvauksia. Lisäksi jokainen hetki, jolloin laite ei ole toimintavalmis, tuottaa operaattorille kustannuksia menetettyjen tulojen muodossa. Tämän vuoksi on erittäin tärkeää, että laitteet ovat jatkuvasti toimintakunnossa eikä palvelukatkoksia tule.

Tämän diplomityön tavoitteena oli kehittää automatisoitu pitkän ajan testausjärjestelmä IP/MPLS pohjaiselle Tellabs 8600 reititinperheelle. Testattava järjestelmä koostuu useista verkkoelementeistä sekä graafisesta Tellabs 8000 verkonhallintajärjestelmästä. Tämän testausympäristön tavoitteena on paljastaa ongelmia, jotka eivät tule esiin normaalissa toiminnallisessa tai regressiotestauksessa vaan vaativat ilmaantuakseen pidempää ajoaikaa tai useita toistoja. Työssä kehitettiin kehys sille, kuinka testausympäristössä voidaan suorittaa automaattisesti erilaisia operaatioita sekä voidaan ohjelmallisesti havaita mahdollisia ongelmatilanteita.

Testausjärjestelmä toteutettiin onnistuneesti ja täyttää sille asetetut tavoitteet. Testausjärjestelmä on otettu käyttöön Tellabsin systeemitestauksessa ja on käyttöönoton jälkeen osoittautunut hyödylliseksi ja tehokkaaksi järjestelmäksi. Systeemitestauksen käyttöön toteutettiin myös toinen täysin identtinen ympäristö.

Avainsanat: IP, MPLS, VPN, liityntäverkko, verkonhallinta, testaus, testauksen automatisointi, pitkän ajan testaus

Preface

This Master's Thesis was conducted for Tellabs Oy at their Espoo R&D department in 2008. First of all, I would like to thank Tellabs, and especially my manager Juha Mailisto, for giving me the opportunity to carry out this thesis. I found the subject extremely interesting and challenging, and thus very educational.

I wish to express my gratitude to my supervisor Professor Raimo Kantola, for all his valuable advice and guidance especially with the structure of this thesis.
I am also highly grateful to my instructor, M.Sc. Sami Larkimo, for finding the time to guide me at all stages of the thesis. His guidance and great ideas had a considerable contribution to this work. Furthermore, I would like to thank all my colleagues in the system testing group for all their valuable support during the writing of this thesis.

I would also like to thank my family and friends for their great support during my years of studies. Finally, my warmest thanks go to my dear girlfriend Satu for enduring me and for her encouragement throughout my studies.

Espoo, February 12, 2009

Timo Karttunen

Table of Contents

List of Abbreviations

2G	Second Generation Mobile Phone Standard
3G	Third Generation Mobile Phone Standard
API	Application Programming Interface
APS	Automatic Protection Switching
APSG	Automatic Protection Switching Group
AS	Autonomous System
ATM	Asynchronous Transfer Mode
BGP	Border Gateway Protocol
BMI	Buffered Message Interface
BMP	Broadband Management Protocol
BSC	Base Station Controller
CDC	Control and DC Power Card
CE	Customer Edge
CES	Circuit Emulation Service
CIDR	Classless Inter-Domain Routing
CLI	Command Line Interface
CoS	Class of Service
CPU	Central Processing Unit
CSPF	Constrained Shortest Path First
DiffServ	Differentiated Services
DSCP	Differentiated Services Code Point
E1	E-carrier level 1
eBGP	exterior Border Gateway Protocol
EGP	Exterior Gateway Protocol
ERO	Explicit Route Object
FDDI	Fiber Distributed Data Interface
FEC	Forwarding Equivalence Class
FTP	File Transfer Protocol
FR	Frame Relay
GSM	Global System for Mobile Communications
GUI	Graphical User Interface
HDLC	High-Level Data Link Control
HTTP	Hypertext Transfer Protocol
iBGP	interior Border Gateway Protocol
IETF	Internet Engineering Task Force
IGP	Interior Gateway Protocol
IMA	Inverse Multiplexing for ATM
IP	Internet Protocol
IPv4	Internet Protocol version 4
IS-IS	Intermediate System-to-Intermediate System
ISP	Internet Service Provider
ITU-T	International Telecommunication Union, Telecommunication Standardization Sector
JAR	Java Archive

L1	Layer 1
L2	Layer 2
L3	Layer 3
LAN	Local Area Network
LDP	Label Distribution Protocol
LER	Label Edge Router
LSA	Link State Advertisement
LSP	Label Switched Path
LSR	Label Switched Router
ML-PPP	PPP Multilink Protocol
MP-BGP	Multi-Protocol BGP
MPLS	Multiprotocol Label Switching
MSP	Multiplex Section Trail Protection
MSPG	Multiplex Section Protection Group
MTU	Maximum Transmission Unit
N2X	Agilent Test Equipment
N-PE	Network-facing Provider Edge
NMS	Network Management System
OAM	Operations and Maintenance
OSI	Open Systems Interconnection
OSPF	Open Shortest Path First
P-a	Provider router in access network
PDU	Protocol Data Unit
PE	Provider Edge
PHB	Per Hop Behavior
PLT	Packet Loop Test
PPP	Point-to-Point Protocol
PSN	Packet Switched Network
PW	Pseudo Wire
PWE3	Pseudo Wire Emulation Edge to Edge
QoS	Quality of Service
QTP	QuickTest Pro
RAN	Radio Access Network
RED	Random Early Detection
RFC	Requests For Comments
RIP	Routing Information Protocol
RNC	Radio Network Controller
RSVP	Resource ReSerVation Protocol
RSVP-TE	Resource Reservation Protocol - Traffic Engineering
SAToP	Structure-Agnostic TDM over Packet
SDH	Synchronous Digital Hierarchy
SLA	Service Level Agreement
SNMP	Simple Network Management Protocol
SONET	Synchronous Optical Network
SSH	Secure Shell
STM	Synchronous Transport Module
Tcl	Tool Command Language

TCP	Transmission Control Protocol
TDM	Time Division Multiplexing
TE	Traffic Engineering
TTL	Time To Live
U-PE	User-facing Provider Edge
UDP	User Datagram Protocol
VC	Virtual Circuit
VLAN	Virtual LAN
VP	Virtual Path
VPLS	Virtual Private LAN Service
VPN	Virtual Private Network
VPWS	Virtual Private Wire Service
VRF	Virtual Routing and Forwarding
WAN	Wide Area Network
WFQ	Weighted Fair Queuing
WRED	Weighted Random Early Detection

1 Introduction

1.1 Background

Access networks have traditionally been based on various circuit switching technologies. However, at the moment these distinct networking technologies are evolving towards converged IP networks. The change is happening in both wireline and mobile access networks. In mobile access networks the old technologies with limited capacity do not provide the long term scalability and flexibility that mobile network evolution demands. Thus, the underlying transport technologies of Radio Access Networks (RANs) are expected to undergo a transformation from Time Division Multiplexing (TDM) and Frame Relay (FR) to ATM and eventually to IP. This evolution will take a long time and means that operators need to support all legacy technologies simultaneously with IP.

In wireline networks the most popular business service is Local Area Network (LAN) interconnection, which is used to build corporate intranets and share company data and applications across remote sites. Traditionally, the technology for providing LAN interconnection has been TDM, FR and ATM. However, these are all limited in their capacity and are not very cost-efficient when used to transport data traffic in high volumes. This means that the current network infrastructure is becoming inefficient and expensive to maintain. Hence, it is no surprise that the trend today is towards Ethernet- and IP based services. An IP infrastructure is the ideal choice for quickly and cost effectively delivering different types of business and residential services. However, it is critical that this new infrastructure provides service levels comparable to traditional circuit switching technologies.

IP/MPLS has emerged as one of the strongest candidates for the above mentioned access network evolution. MPLS can support various other transport technologies, such as ATM, TDM, Frame Relay, and Ethernet, by tunnelling them over IP/MPLS network. Furthermore, MPLS allows service providers to make use of traffic engineering capabilities and it brings guaranteed QoS into best effort IP networks. The combination

of IP and MPLS provides the predictable properties of circuit switching but at the lower cost of Ethernet based devices, creating a robust, scalable and manageable platform for delivering next generation voice and data services.

1.2 Research problem and the objective of the thesis

Tellabs Oy, an enterprise operating in Finland, has designed Tellabs 8600 system to address the abovementioned challenges faced by the telecom service providers. It is an Internet Protocol/Multi-Protocol Label Switching (IP/MPLS) based access network solution primarily targeted for mobile networks, but also applicable in fixed networks. The sophisticated Quality of Service and Traffic Engineering features of the system allow the creation of services with strict service level requirements. By supporting also legacy networking technologies the system supports smooth transition towards all-IP networks. Tellabs 8600 network elements are managed with the graphical user interface (GUI) of the Tellabs 8000 Network Manager.

Today's Internet service providers (ISP) offer specific service guarantees to their customers according to agreed service-level agreements (SLAs). SLA is a contract between a network service provider and a customer that specifies what level of service the ISP will guarantee. To be able to keep the strict SLA's the ability of networking equipment to continuously function correctly is extremely important to service providers. Therefore the phase of testing becomes crucial in telecommunications. In this thesis, the word testing is used to specifically mean software testing.

The objective of this thesis is to design and implement an automated testing environment for soak testing of Tellabs 8600 series routers. The purpose of this soak testing environment is to find problems that do not show up immediately in functional and regression testing but could manifest themselves in the service providers operating network. Soak testing, also known as endurance testing, is a type of testing where the system will be run over a prolonged period of time in order to check the system's stability under sustained use. The purpose is to reveal resource usage problems such as memory leaks or buffer overflows. For example, networking equipment may work

normally when testing is executed for a few hours or some functionality is tested only a few times but may fail if the testing period is extended to several days. The implemented environment will be used in system testing of the Tellabs 8600 system and will continue to expand after this thesis.

At first the physical network environment will be built and measuring equipment configured so that data can be run through the network. Some methods to automatically operate the network are examined. Test tool called QuickTest Professional is already used in system testing of the 8600 system to run scripts that automatically operate the GUI of the Tellabs 8000 Network Management System. QuickTest Professional software is a tool for automating testing and for building functional and regression test cases. It is first checked whether this software is applicable also for soak testing purposes. In case it is not, some other ways to implement operations are examined, such as NMS macros or CLI configuration through telnet sessions. Automatic operation of the environment requires that also problems are automatically recognized. Therefore, various methods to automatically detect problems in the environment will be examined. The environment needs also some method to log everything that is done, in order to be able to afterwards trace where problems have occurred.

1.3 Structure of the thesis

First half of this thesis presents the various technologies involved. Since the Tellabs 8600 system is based on the IP/MPLS, this thesis starts by examining these technologies in Chapter 2. Chapter 3 contains an overview of the Tellabs 8600 Managed Edge System itself. The chapter presents the essential system technologies and main applications of the system. The chapter ends by explaining how Quality of Service is implemented in the routers. Chapter 4 presents the Tellabs 8000 Network Manager, which is used to manage the Tellabs 8600 network elements. Before explaining the Tellabs implementation of the network manager a short introduction to objectives of network management is given. Chapter 5 starts by examining some technologies that will be present in the implemented environment before explaining the services at the

latter part of the chapter. These services are layer 2 and layer 3 MPLS Virtual Private Networks (VPNs).

Sixth chapter provides an overview on the objectives of software testing. Testing levels and various techniques are presented before giving a brief introduction to soak testing and test automation. In Chapter 7 the soak testing for an access network solution is implemented. The network, including the measurement device, is built and configured. Different methods to automatically operate the network and detect failures are examined. Finally, in Chapter 8 a summary of the thesis and some suggestions for further developing the testing environment are given.

2 IP and MPLS

In this chapter, an overview of IP and MPLS technologies is given. The Tellabs 8600 Managed Edge System is an IP/MPLS based system and thus knowledge of these underlying technologies is essential. In the first part of this chapter, a brief overview of the IP technology is provided. The latter part of the chapter discusses MPLS technology. The services created with the Tellabs 8600 system are L2 and L3 MPLS based VPNs. Furthermore, this chapter shows how MPLS brings traffic engineering and Quality-of-Service support to best-effort IP networks.

2.1 IP

2.1.1 Interconnecting local networks

Local Area Networks (LANs) enable computers attached to it to communicate with each other. LAN represents the simplest possible network, since the hosts (computers) are all directly connected by some physical medium. However, LANs suffer from two main limitations. The number of hosts that can be attached to one LAN is limited and LANs are restricted to a relatively small geographical area. These limitations can be compensated to some extent with bridges and switches, which allow several LANs to be connected together. Nevertheless, LANs build with bridges and switches are limited in their ability to scale and are not very well suited for handling different types of LAN technologies. [29]

To be able to connect two or more networks, which can possibly be geographically dispersed and based on different LAN technologies, we need the concept of internetworking. Internetworking means connecting networks together to form an internetwork, using devices which operate at layer 3 of the OSI Basic Reference Model. Internetworks are large networks that are very heterogeneous and have fairly efficient routing; the most notable example being the Internet. Figure 2.1 shows an example of internetwork where various LANs are connected by a Wide Area Network (WAN).

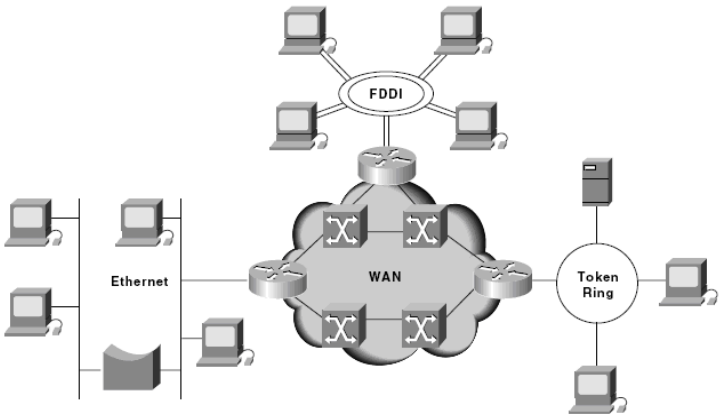


Figure 2.1: An example of internetwork composed of several networks [6]

Internetworks are often based on the TCP/IP architecture, which is a four layer model depicting how to implement the internetworking concept. The four layers of the model are from bottom to up: Network access, Internet, Transport, and Application. As in the case of the OSI model, in the TCP/IP model layers offer services to upper layers through interfaces and use services of lower layers. [9, 29]

The hourglass shape of the architecture is best described with the famous catchphrase by Vinton Cerf, the first president of the Internet Society, “IP over everything, and everything over IP”. At the bottom of the model are a wide variety of technologies, ranging from Ethernet to FDDI to ATM to single point-to-point links. In fact, this layer can be, and often is, a composition of several different protocols. Hence, IP runs over everything. The second layer consists of only one protocol, namely the Internet protocol. IP is the main protocol of the TCP/IP model and creates a single logical internetwork by connecting separate networks together. It hides the low-level technology details and enables hosts to communicate. While IP connects remote hosts together, the layer above it connects processes running in these hosts. Two main transport level protocols are TCP and UDP. At the top of the model are the application layer protocols, which enable the interoperability of different applications through networks. Typical examples are File Transfer Protocol (FTP) and HyperText Transfer Protocol (HTTP). [29]

Routers, also known as IP gateways, are special networking equipments that connect separate networks together at the OSI layer 3. As mentioned, these networks can be based on different technologies. Routers forward packets based on their routing tables that are populated by routing protocols. Routing and forwarding are discussed in section 2.1.3.

2.1.2 Internet Protocol

IP is the focal point of the TCP/IP model and is documented in the IETF RFC791. Internet protocol offers connectionless, best-effort delivery of packets through an interconnected set of networks. IP has been kept deliberately very simple. It does not provide any end-to-end data reliability, flow control, or sequencing but these are addressed by other protocols. The best effort service model of IP does not give any guarantees for packet delivery. It tries to deliver the packets to destinations, but if packets get lost, for any reason, the network is incapable of correcting the situation. IP packets are conveyed in a connectionless manner, meaning that no connections or circuits are created between source and destination. These datagrams contain enough information so that the network knows how to route them to destinations. [29, 30]

Each interface in the network is identified with the worldwide unique IP address; a 32 bit long word consisting of two parts; the network part and the host part. Initially there were three main classes of network numbers. Network part was 7-bits for class A addresses, 14 bits for class B and 21-bits for class C. Later, as network managers wanted to further structure their networks, third part was added to this two-level hierarchy. Subnetting divides organization's network into smaller parts called subnets.
The addresses of all nodes in a subnet start with the same binary sequence, identified with the 32-bit subnet mask. [15]

To address the backbone routing table explosion and the exhaustion of the 32-bit IP address space the concept of Classless Inter-Domain Routing (CIDR) was developed [29]. With CIDR the prefixes are not constrained to 8, 16 and 24 bits, but can be composed of any number of contiguous bits. CIDR uses variable length subnet masks to represent the network part of the address. Thus the division between network and host

part can occur at any bit position in the address. In the CIDR notation the number of contiguous bits representing the network part is shown at the end of the address. For example 10.121.104.4/30 indicates that the network part of the address is 30 bits long. [15]

2.1.3 Routing and forwarding

As mentioned, routers connect separate networks together. Routers have two main responsibilities; routing and forwarding. Routing is implemented by the routing protocols and involves obtaining reachability information to other networks. Based on the received routing information, the routing algorithm calculates the best routes to every destination network. This information is shown in the routing tables in the form destination/next hop association. Routing tables contain information on all the best paths to all the destinations that they know how to reach. Internet protocols operate in a distributed fashion, since there are no central point calculating the routes and distributing those to all routers. Routing directs the process of forwarding. Forwarding is the process of taking a packet from input interface for transmission and deciding the interface through which it should be transmitted. This is done by checking the destination address and looking the next hop from the routing table. After decision the packet is transmitted through the selected interface. [6, 13]

Various link-state protocols are nowadays the most widely used Internet routing protocols. Link-state protocols flood the information to all the nodes in their network. Routers send information about their directly connected links with link state advertisements (LSA). This LSA contains information about the destination network and the cost of this link. Routers receive these LSAs from all the other nodes in the network and store them in a link state database. This link-state database represents the topology of the whole network. Djikstra's algorithm is then used to calculate the best paths to destination networks and routing table is populated by the results. OSPF and IS-IS are examples of link-state algorithms. [15]

Routing protocols are commonly divided to interior gateway protocols (IGPs) and exterior gateway protocols (EGPs). Interior gateway protocols are used to distribute

routing information inside one autonomous system (AS) and exterior protocols are used to exchange routing information between ASs. An AS is a collection of IP networks and routers under the control of one administrative authority, typically single ISP. OSPF, IS-IS and RIP are all IGPs, the most common EGP is Border Gateway Protocol (BGP) [31].

2.2 Multiprotocol Label Switching

2.2.1 Introduction

Initial motivation for the development of MPLS was to achieve the speed of L2 switching also in IP based L3 forwarding. L2 forwarding is based on address lookup on full address where the first match in database is always unique and thus the exact answer. L3 lookup, in contrast, is based on the longest match principle, where several routes to the same destination can be found. So, the whole database needs to be checked as the first match is not always the best one. [51]

In the mid-1990s several companies had their own ongoing efforts to combine IP and ATM technologies. Of these technologies, the most significant ones where developed by Ipsilon (IP switching), Cisco Systems (Tag Switching), IBM (aggregate route-based IP switching), and Cascade (IP Navigator). The approach used in all these proprietary technologies was very much the same, since those all tried to utilize ATM in a way or another. In 1997 the Internet Engineering Task Force established the MPLS working group, in order to develop a standardized and vendor-independent protocol. The goal was to use existing IP routing protocols to discover the routes between endpoints and to allow the separation of control and forwarding components. [37]

At the same time the networking equipments improved and got much faster. These new routers were as competent as ATM switches and looking up longest best match was no longer an issue. This caused the original motivation to MPLS development fade away. Instead of making forwarding faster, new applications rose. The label switching principle of MPLS brings the following capabilities [37]:

- *QoS Support* - IP-based networks cannot provide the quality-of-service features available in circuit-based networks, such as Frame Relay and ATM. MPLS enables the capabilities of these connection-oriented protocols in the connectionless IP world.
- *Traffic Engineering* - The ability to choose the path that traffic will flow through the network is called Traffic Engineering. This can be based on the network utilization and demand, thus increasing the utilization of the network. [21]
- *VPN Support* - When the service provider uses its IP/MPLS backbone for VPNs, MPLS is used to separate different customers' traffic in the backbone. [43]
- *Multiprotocol Support* - In addition to all the above advantages, one of the most important advantages of MPLS is that it is independent of the layer 2 and layer 3 technologies and hence allows integration of networks with different layer 2 and layer 3 protocols.

2.2.2 MPLS Concept

MPLS based packet forwarding is done according to labels attached to packets. Each packet is assigned a label value and forwarding is done based on that. Figure 2.2 shows the basic components of an MPLS network. Routers participating in forwarding in the MPLS enabled network are called Label Switching Routers (LSRs). For each link the router that is in the direction of the data flow, is called downstream LSR, and the other router is naturally the upstream LSR.

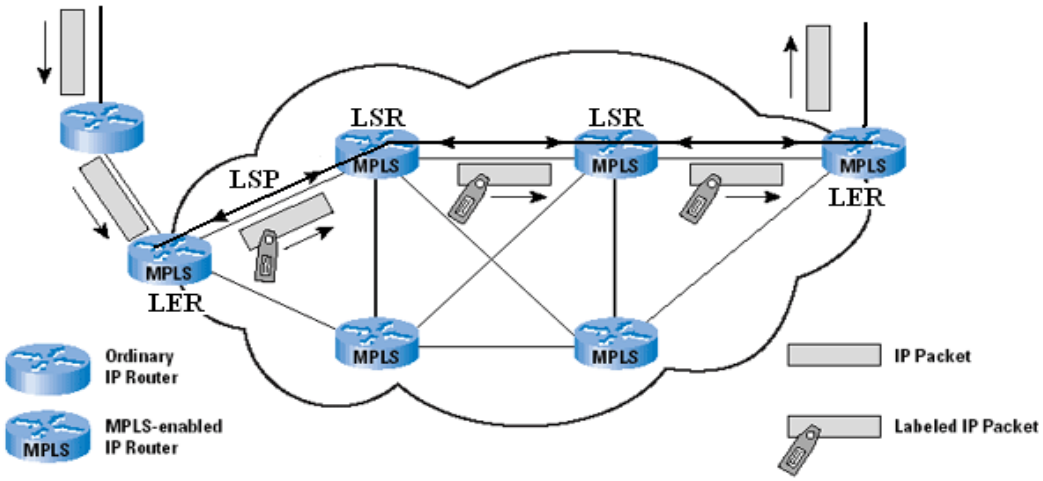


Figure 2.2: Components of MPLS network [37]

The routers at the entry and exit points of an MPLS network are special kind of LSRs called Label Edge Routers (LERs). The entry point of network occurs at ingress LER, which does the packet classification and assigns a label value to it. Whereas the exit point of the network, which is egress LER, is responsible for removing the MPLS header. The unidirectional path that a particular packet travels through the MPLS domain is called Label Switched Path (LSP). It is a sequence of LSRs that do forwarding decisions based on the label value of the packet. This path is set up before data transmission with the assistance of Label Distribution Protocols (LDP). Thus, MPLS operates in a connection oriented way. [37, 51]

The label operations that a MPLS enabled router can make are push, pop and swap [34]. When a packet first enters a MPLS network the LER *pushes* a label to the packet before forwarding it. As this packet travels through the network, the intermediate routers check this label and based on the label value *swap* it to a new label. The egress LER *pops* the label and forwards the packet without MPLS header. [10]

2.2.3 MPLS Label Stack Header

MPLS label stack header is often referred to as being Layer 2+ header, because it is located between L2 (data link layer) and L3 (network layer) headers of the OSI model. The MPLS label stack header, also known as MPLS shim header [51], is 32-bits long and consists of four parts [33]:

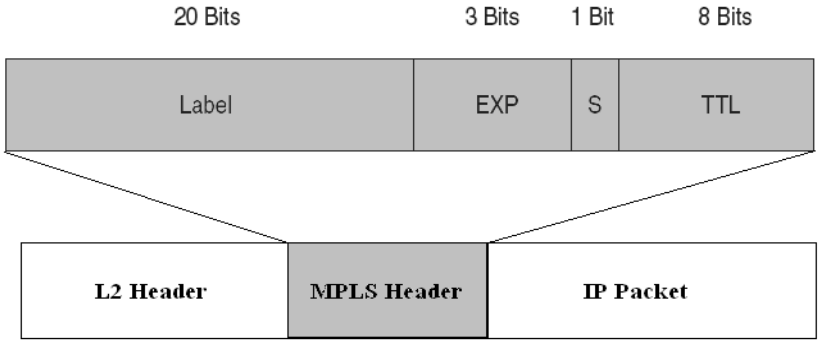


Figure 2.3: MPLS header

Label – Label field carries the actual label value of this MPLS packet.

EXP – These three bits are reserved for experimental use. These are nowadays also called as Class of Service (CoS) bits and are used to carry the packet priority information. For example, the queuing and scheduling algorithms applied to the packet can be derived from these bits.
S – Stack bit tells whether this is the last header in the stack.
TTL – An 8 bit Time-to-live value is usually decremented by one in every hop. When a MPLS packet with a TTL value of 1 is received, the receiving LSR discards the packet.

MPLS label can be encapsulated inside another MPLS label, thus creating a label stack. Label stacking allows the creation of hierarchical networks where multiple LSPs are aggregated inside one trunk LSP through the backbone network. The most significant benefit of this is the ability to create scalable L2 and L3 VPNs.
The processing of MPLS packet is always based on the label at the top of the stack, hence the label that was added last [34]. As the packet flows through the tunnel only the outer label is swapped, while lower labels remain the same. Thanks to this label stacking, the LSRs in the core network do not have to know the label binding for every LSP that goes through them. Rather it has to only know the bindings for the aggregating tunnels between edge routers, hereby enhancing the performance and the scalability of the network. [51]

With MPLS it is possible to use a mode called penultimate hop popping. In penultimate hop popping the LSR prior to egress router pops the outer label, instead of swapping, and forwards the packet with the inner label to the egress router. This enhances the performance of the egress router, since without penultimate hop popping the edge router would have to make two lookups. First it would have to derive that it is the egress router and then forward the packet based on the inner label. [51]

2.2.4 Forwarding Equivalence Class (FEC)

A forwarding Equivalence Class (FEC) is a term used to describe a group of packets which are treated in a same way at the intermediate nodes. FEC correspond to a certain label switched path (LSP), whereas the reverse is not true since LSP is usually used for multiple FECs. The mapping of packets to a certain FEC can be made with several

different characteristics. Some commonly used characteristics are the destination IP subnet, egress router of the MPLS cloud, and the value of the Type of Service field of IP packet. The packet classification and mapping to a particular FEC is done by LER as the packet first enters the MPLS network. Packets belonging to the same FEC will be assigned the same label value. [34, 51]

As can be seen, FEC relates to the level of aggregation used when forwarding packets, called forwarding granularity. For example, the forwarding granularity in current IP networks is based on the destination IP address. MPLS allows multiple levels of granularity to coexist in the same network, thus giving more control over the forwarding of packets. For example, the ingress LER can classify packets so that higher priority packets get better treatment over the same path to the same destination. [51]

2.2.5 Label Distribution Protocols

Before packets can be sent through LSPs, paths have to be established with Label Distribution Protocol (LDP). With LDP one LSR informs other LSRs about the label/FEC bindings that it has made. For each link the downstream LSR is responsible for the mapping and informing this to the upstream LSR. Routers that share this label/FEC binding information with each other are called label distribution peers. Label distribution can be done in one of two modes: [34]

- *Downstream-unsolicitated* is a mode, in which an LSR announces a label binding for all of its label distribution peers without any request from them.
- *Downstream-on-demand* is a mode, in which the upstream LSP explicitly requests the label from downstream LSR.

When an LSR receives FEC/label bindings from a downstream router, it may be the case that this downstream router is not a valid next hop for that FEC. In this case there are two options for label retention [34]:

- *Conservative* – Only labels that come from valid next hop are kept. This saves the label space in case there are a lot of peers but the adaptation to routing changes is slower than in the liberal mode.
- *Liberal* – In this mode all received labels are kept regardless of the fact whether the downstream LSR is a valid next hop for this FEC. The benefit from this is that labels are ready to be used if the downstream LSR becomes the next hop for this FEC due to routing changes.

LDP can actually refer to the label distribution protocols in general or it can also refer to a particular protocol designed for the same purpose. In fact, there are several protocols designed for this purpose, LDP and RSVP-TE are the two protocols supported by the Tellabs 8600 routers and presented here.

Original LDP specification is at the RFC3036, LDP Specification, but it is now obsoleted by RFC 5036. LDP is used mainly in situations where traffic engineering capabilities are not required. LDP is designed to support hop-by-hop label distribution. With LDP the LSPs are established according to the IP forwarding tables of LSRs. So packets travel the same path as normal IP packets would. Thus, LDP is not suitable for traffic engineering purposes. If, for any reason, an LSP is broken, LDP relies on IGP in path restoration. First the IGP routes have to converge and then the new path is established based on these new IP forwarding tables on LSRs. The main benefit of LDP comes from the ease of setting up a full mesh of LSPs between all of the routers on the network. It supports both modes of label distribution, while the most common is unsolicitated mode. [1, 51]

When LDP is enabled on some interface of an LSR it starts sending Hello packets in order to discover adjacent LSRs. After neighbour discovery, an LDP session is established between the corresponding routers. The hop-by-hop LSP creation from ingress router to egress router occurs in the following way if downstream-unsolicitated mode is used. The egress LER broadcasts label mappings with advertisement messages to all of its neighbours. These broadcasts spread over the entire network until they reach

the ingress routers. On each hop, the messages inform the upstream router of the label to use for each LSP. This way LDP floods the whole network and establishes LSPs between all of the LDP enabled routers. [51]

Resource Reservation Protocol - Traffic Extensions (RSVP-TE) is defined in RFC 3209 and is a traffic engineering extension to a well known resource reservation protocol RSVP, defined in RFC 2205. Some of the new features added to RSVP by RSVP-TE are label distribution, explicit routing, bandwidth reservation, rerouting of LSPs, and tracking of the route that an LSP takes. RSVP-TE is used to establish MPLS LSPs, called LSP tunnels, when there are traffic engineering or Quality of Service requirements. It works in an end-to-end fashion and allows the reservation of resources, such as bandwidth, for the entire path. Conventional IP routing can be bypassed with the use of explicit routing. Explicit routes can be injected manually or determined with the help of the CSPF (Constrained Shortest Path First) algorithm. This CSPF algorithm can take into account many characteristics, such as unreserved link bandwidth. [2, 51]

Setting up an RSVP tunnel requires the exchange of Path and RESV messages. When an ingress LSR recognises the need to establish an LSP to an egress LSR, it generates the Path message. This Path message might include the explicit route object describing the detailed path to the destination and the traffic parameters associated with the route. When this message is received by the egress LSR, it allocates a label for the LSP and determines from the traffic parameters the resource reservations needed for the tunnel. These are included in the RESV message, which is sent back to the ingress LSR. As the message travels back towards the ingress router, each intermediate router allocates a label for the LSP, sets up the forwarding table, and reserves the needed resources. The tunnel is established when this message reaches the ingress router. [2, 51]

3 Tellabs 8600 Managed Edge System

In this chapter, a brief overview of the Tellabs 8600 system is given. Firstly, this chapter presents the main applications of the Tellabs 8600 system. Hereafter, the essential system technologies and the routers, including their roles in the network, are presented. The chapter ends with a discussion of how the system implements different Quality-of-Service levels. Understanding the applications and capabilities of the system is important in order to design a test environment for the system.

3.1 Main Applications of the System

Tellabs 8600 Managed Edge system is a scalable IP/MPLS based access network solution for evolving networks. The system is a suitable transport solution for a service provider in environments where QoS and service management are important features. It is mainly targeted to be responsible for the transport part of the mobile access networks from the base station sites to the RNC/BSC sites. [44]

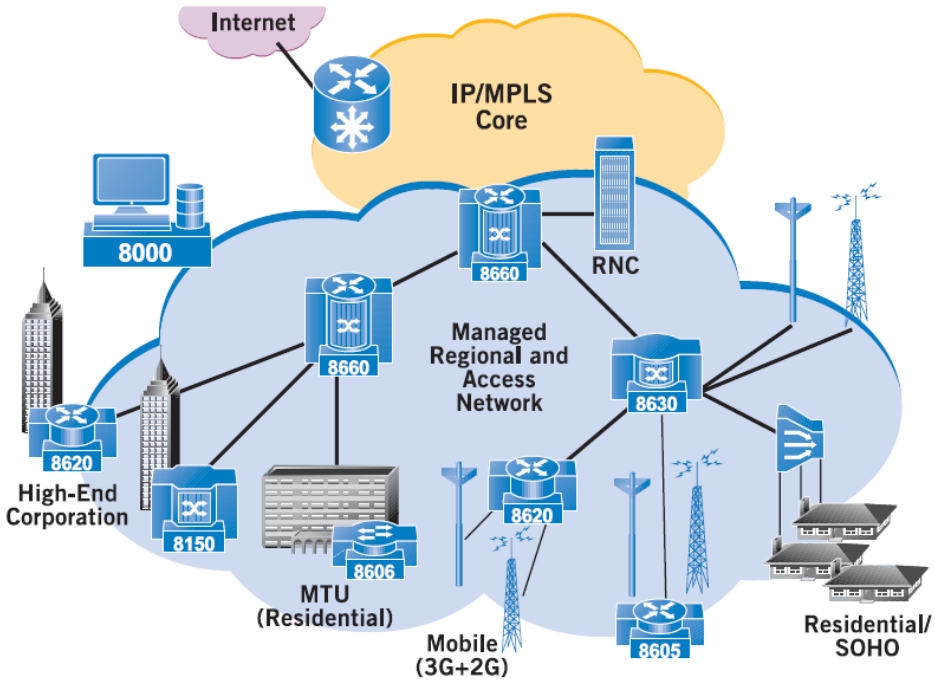


Figure 3.1: The positioning of Tellabs 8600 System in the service provider's network [44]

The Tellabs 8600 system supports multiple technologies, such as TDM, SDH, ATM, FR, and Ethernet, which are needed in the evolving networks. It is a single platform that can support TDM based GSM networks, 3G R99 ATM networks and 3G R5 MPLS/IP networks. On the wireline side, a service provider can construct an access network with the Tellabs 8600 system for providing LAN interconnections, Internet access and corporate voice services to business customers, for instance. These services are implemented with scalable and QoS capable Layer 2 and Layer 3 MPLS VPNs. Figure 3.1 shows the positioning of the system in the service provider access network. [44]

The main applications of the Tellabs 8600 Managed Edge System are [40]:

• *Mobile transport in 2G and 3G RAN* – 2G transport with TDM pseudo wires, 3G R99 transport with ATM pseudo wires, and 3G R5 transport over IP/MPLS network.
• *Managed voice and data leased line business services* – Various services can receive desired QoS treatment.
• *Managed LAN interconnection services* – Replacing leased lines as a way to offer corporate intranet services with lower-cost Ethernet tunneling over MPLS network.
• *Managed IP VPNs* – Provides many-to-many IP-level connectivity through operators network
• *Broadband service aggregation* – Helps network convergence by providing a single platform for aggregation of several services such as voice, data, and video [44].

The Tellabs 8600 system includes several different sized network elements and the service-oriented network management system (NMS). Together with the graphical user interface based Tellabs 8000 network manager, the network and end-to-end service management are easy and efficient to execute, thus enabling significant savings on operational and maintenance costs. The Tellabs 8000 network manager is presented in the next chapter. [44]

3.2 Essential System technologies

Some essential technologies of the system are listed below [40]:

- **MPLS –** for emulating different technologies by tunneling these via label switched paths.
- **RSVP –** to establish traffic engineered MPLS LSPs with bandwidth reservations and explicit routing.
- **DiffServ** - for categorizing traffic into different service classes.
- **Layer 3 (IP) VPNs -** RFC2547bis based BGP/MPLS VPNs for implementing multipoint-to-multipoint layer 3 connectivity between geographically dispersed customer sites through the operator network.
- **Layer 2 VPNs -** realizing layer 2 (TDM, ATM, Ethernet, Frame Relay and HDLC) connectivity between sites through the operator network with pseudo wires.
- **BGP -** for distributing customer routes through the operator network between IP VPN service endpoints.
- **OSPF** and **IS-IS -** for IP routing inside a single core or access network.
- **Ethernet**, **ATM**, **Frame Relay** and **TDM -** as customer access and aggregation technology, as an alternative to MPLS in the access network.
- **Traffic Engineering (TE)** – TE extensions for RSVP and OSPF protocols to reserve capacity for LSPs and to select the used path based on the Constrained Shortest Path First (CSPF) algorithm.
- **LDP** – used to establish MPLS LSPs when traffic engineering is not required.

The trend in access networks is towards Ethernet, since it has a significant cost advantage over other networking technologies, including ATM, FR, and SDH. As the system supports both legacy connection-oriented networks and packet switching technologies, it provides seamless path for service providers to shift from circuit switching to packet switching. MPLS together with IP can provide the quality-of-service features available in ATM networks but at the lower cost of Ethernet networks. The Tellabs 8600 system allows the operator to increase the utilization of the network

by using advanced QoS and traffic engineering features. These tools can guarantee superior treatment for premium services while best effort data traffic can utilize the excess bandwidth. [44]

The 8600 router family consists of elements for all parts of the mobile operators RAN from cell sites to RNC/BSC sites. The modular design of the Tellabs 8600 system elements allows the service provider to equip each element with desired interfaces and required capacities. These requirements are highly dependent on the position of the element in the network. Modules can be added or upgraded based on the changing demands of voice and data services. All network elements are based on the architecture where switching is distributed to all line cards, meaning that there is no separate switch card. [44]

The largest and highest-capacity network element is the *Tellabs 8660 Edge Switch* [48]. The position of the element is typically at large hub sites or close to Radio Network Controller/Base Station Controller (RNC/BSC) [48]. The rack of Tellabs 8660 Edge Switch has 14 slots, of which slots number 1 and 14 are reserved for integrated Control and DC Power Feed Cards (CDCs). One CDC is always active and the other one is in the passive state protecting the active one. If, for any reason, the active CDC is not operable, the passive one becomes the new active CDC. The remaining 12 slots are available for line cards, which again can be equipped with two separate interface modules. The *Tellabs 8630 Access Switch* is a more compact version of the 8660 and is aimed for medium and small hub sites or traffic aggregation sites in the mobile RAN [47]. It can be equipped with four line cards and 2 CDCs. The *Tellabs 8620 Access Switch* is targeted to be used at base stations, small hubs sites or as provider managed customer premises equipment. *Tellabs 8605 switch* is a cell site node specially designed for cost-efficiently switching or backhauling 2G and 3G traffic from a cell site towards RNC or BSC through TDM, ATM or Ethernet pseudo wires. As well it can be used as customer-premises equipment for providing business services. It is equipped with 16 E1 interfaces, two gigabit Ethernet interfaces, and two fast Ethernet interfaces. Latest addition to the 8600 router family is the upcoming *Tellabs 8607* router, which is more

or less designed for the same purposes as 8605 but with swappable modules. All of the network elements have QoS support, allowing the creation of predictable services. [44]

3.3 Quality of Service in the System

IP/MPLS is a suitable platform for supporting services with varying QoS requirements in terms of delay, jitter, bandwidth, and packet loss, thus eliminating the need to build separate networks for different purposes. Tellabs 8600 routers support DiffServ-based traffic management to classify IP or MPLS traffic. Based on the requirements, packets are classified to a limited number of QoS classes, as shown in the Figure 3.2. Packets in the same QoS class get the same forwarding treatment in the intermediate routers. Packets are classified and marked based on L2, L3 and L4 information at the edge of a Differentiated Service domain. QoS has been implemented in the 8600 router so that an incoming packet is mapped to a certain PHB, which defines the packets forwarding treatment inside the router. Every router contains default settings about how mapping is done from different networking technologies to Per Hop Behavior (PHB). In Figure 3.2 the mapping is done from Ethernet 802.1p bits and from VLAN, but it can also be made from DSCP bits in an IP packet or MPLS EXP bits, for example. Naturally, for an outgoing packet the mapping is reverse. [38]

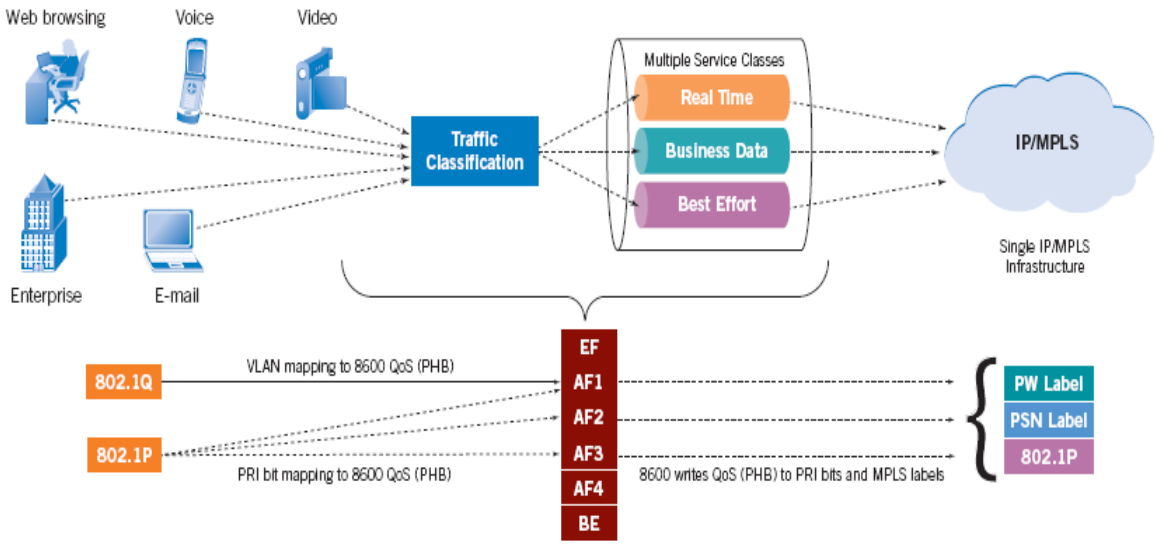


Figure 3.2: QoS mapping in Tellabs 8600 system. [46]

Sophisticated queue management and scheduling algorithms, based on the PHB of the packet, are used to ensure the required QoS for the packets. Each QoS class gets its own queue, where RED/WRED algorithms can be used for queue management. A combination of strict priority and WFQ (weighted fair queuing) is used for packet scheduling. Priority packets in the strict priority queues are always served first and the remaining capacity is allocated to other queues using WFQ. [38]

4 Tellabs 8000 Network Management System

In this chapter, the Tellabs 8000 Network Management System is presented briefly. First, the chapter discusses the objectives of network management in general. After that, we will introduce the Tellabs implementation of the network management system. The benefits and components of the management system are briefly discussed. The capabilities of the system are presented by introducing various packages that compose the system. A special emphasis is given to the macro manager package, which is used in the implemented soak testing environment to operate the network.

4.1 Network Management

Without proper management the networks are of not much value in the long run. The management of network resources is not an easy task and becomes more and more laborious as the number of network elements, users and services increases. Therefore, there is a need for efficient network management. Network management includes all the activities and tools that involve the planning, organizing, monitoring, accounting, and controlling of networked systems. [3, 7]

The major functions of network management systems are categorized into five separate areas by ITU-T in recommendation M.3400. The model is also known as FCAPS model [7, 19]:

- Fault management – The objective of fault management is to identify, log, report and possibly fix problems in the network.
- Configuration management – Configuration management is concerned with setting configurations and retracing any changes that take place on configuration information of the network elements.
- Accounting management – The goal is to measure statistics of the usage of network resources by individual users in order to be able to charge them correctly and to regulate them appropriately.
- Performance management – The goal of performance management is to gather performance related information from the network in order to determine the

efficiency and utilization of the network. This helps to ascertain a sufficient performance level and prepare the network for the future.

- Security management – The security management is concerned with preventing the access to the network resources by those without proper authorization.

4.2 Tellabs 8000 Network Manager

An essential part of the Tellabs solution is the GUI-based network management system which enables element, network and service level configuration. The Tellabs 8000 network manager offers tools for network building, service provisioning, traffic and network monitoring, and testing. It is designed to be easy to use and scalable, thus giving ability to manage massive networks without thorough knowledge of the underlying networking technologies. It enables quick end-to-end service provisioning, where only necessary parameters are given and the system automatically configures all the elements that take part in the service realization. In addition to the 8600 system, it can also manage other Tellabs product families, such as Tellabs 8100 Managed Access System, Tellabs 6300 Managed Transport System, Tellabs 7100 Optical Transport System, and Tellabs 8800 Multiservice Router Series. [41, 44]

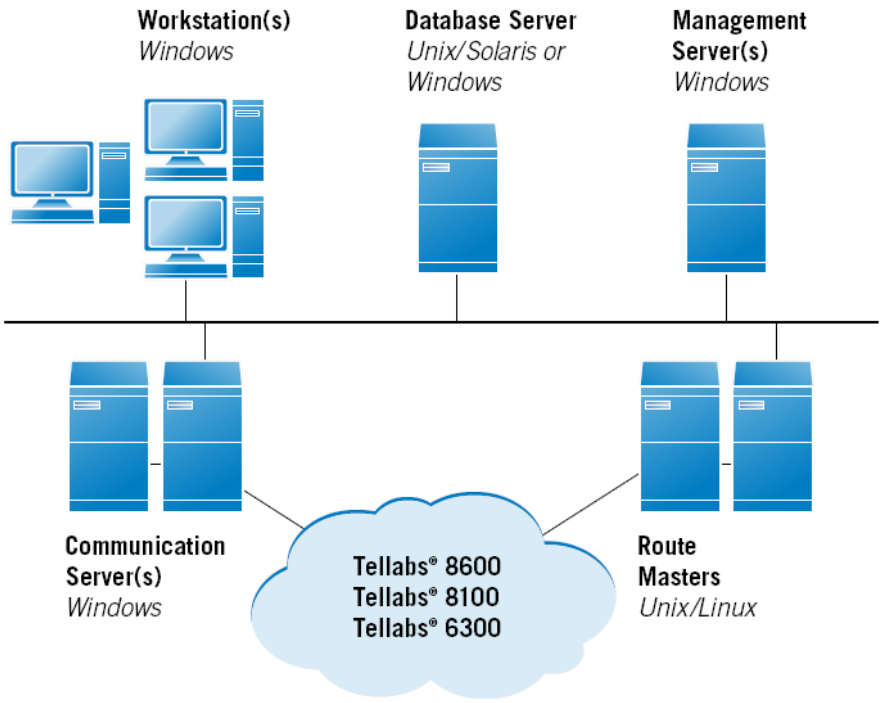


Figure 4.1: Tellabs 8000 Network Manager configuration. [41]

Figure 4.1 above shows the components of the management network of the Tellabs 8600 system. The management network consists of several workstations and servers, each with their own function. Communication servers connect the management network to the Tellabs 8600 network elements. All communication between the elements and the management system goes through these servers. Communication servers send configuration data to the nodes and poll them for performance and fault information. For reliability and scalability reasons there can be several communication servers connected to the same Tellabs 8600 network. [40]

All data related to the network and services is stored on the central database server. Thus, the whole network configuration is found from this server. Network and services can also be pre-planned in the Tellabs 8000 Manager database from where these configurations can later be downloaded to each network element. The centralized database helps to keep all workstations and integrated applications up to date, allowing all operators to see changes in network state in real-time. Management servers contain the business logic components of the Tellabs 8000 Manager. Simple operations done with the network manager may require complex operations to the network elements. Management servers convert the operations and ensure that these are done in the correct order. The network is monitored and configured from the workstations, which contain the GUI of the Tellabs 8000 Manager. Route masters can act as route reflectors, which improves the scalability of the network by removing the need for the full mesh of BGP sessions between all of the routers. [44]

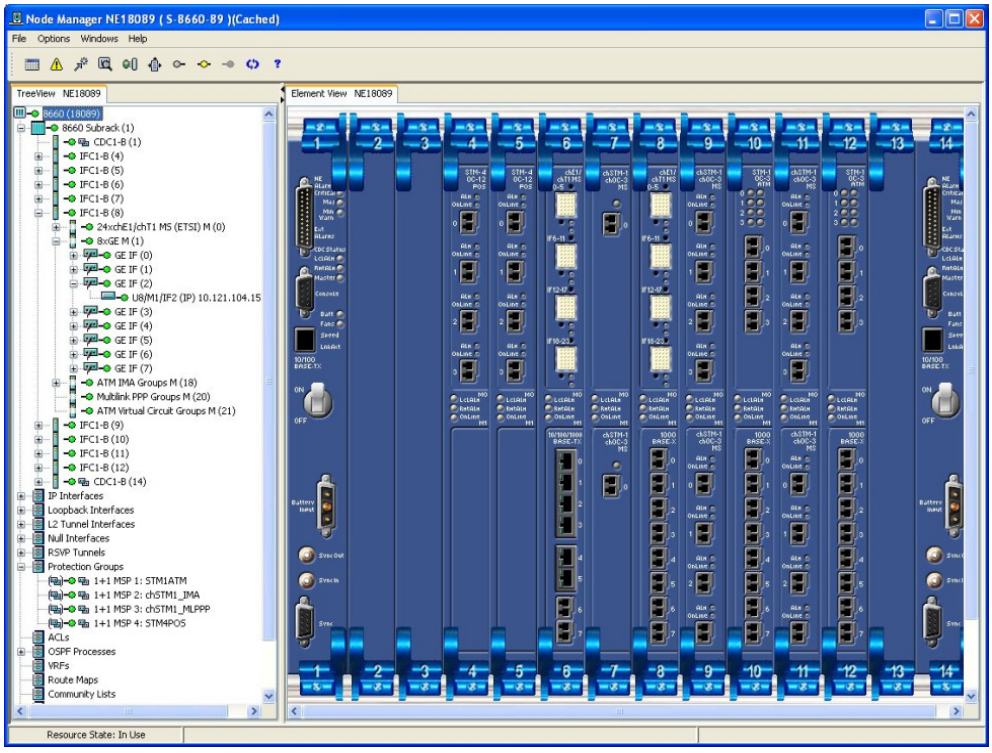
As mentioned, all communication between network elements and the Tellabs 8000 Manager goes through communication servers. Tellabs proprietary Broadband Management Protocol (BMP) and SNMP (Simple Network Management Protocol) are used as communication protocols between the management system and network elements. BMP is an object-oriented protocol designed particularly for managing data network elements. SNMP is used for retrieving performance statistics. This control traffic has high priority inside network element queues, thus ensuring successful communication eventhough the network is congested. In addition to using the network

manager, network elements can also be managed with a Command Line Interface (CLI) through telnet or SSH sessions. [44]

Management traffic can be carried in two disparate ways. Inband management means that the management traffic goes, with other traffic, through the managed network itself. There is no need for an external communication network, but the communication is dependent on the state of the managed network. The other option is to use outband management, where the management traffic is carried in a network that is totally distinct from the actual network that is to be managed. This external management network is required to provide IP connectivity between the Communication servers and all network elements that are managed by the system. [40, 45]

4.3 Network Manager Packages

The system is divided into several separately licensed packages and tools. The basic package includes various tools for building and configuring the network, monitoring faults, and for user administration. The Node Manager tool enables the element-level configuration and performance management.



4.2: Node Manager dialog of the NMS.

A screenshot of Node Manager is presented in Figure 4.2. Node manager itself includes several tools for configuring and monitoring the network elements. With it the user can set parameters, monitor faults and check consistency of the network element with the database. It also shows the current cards and modules installed in the network element. As shown in the figure, on the right side of the tool there is a picture of the network element and by right clicking an interface one can configure that particular interface. On the left side is also a tree view which enables the configuration of interfaces and other element-level configurations such as routing and creation of protection groups. [40]

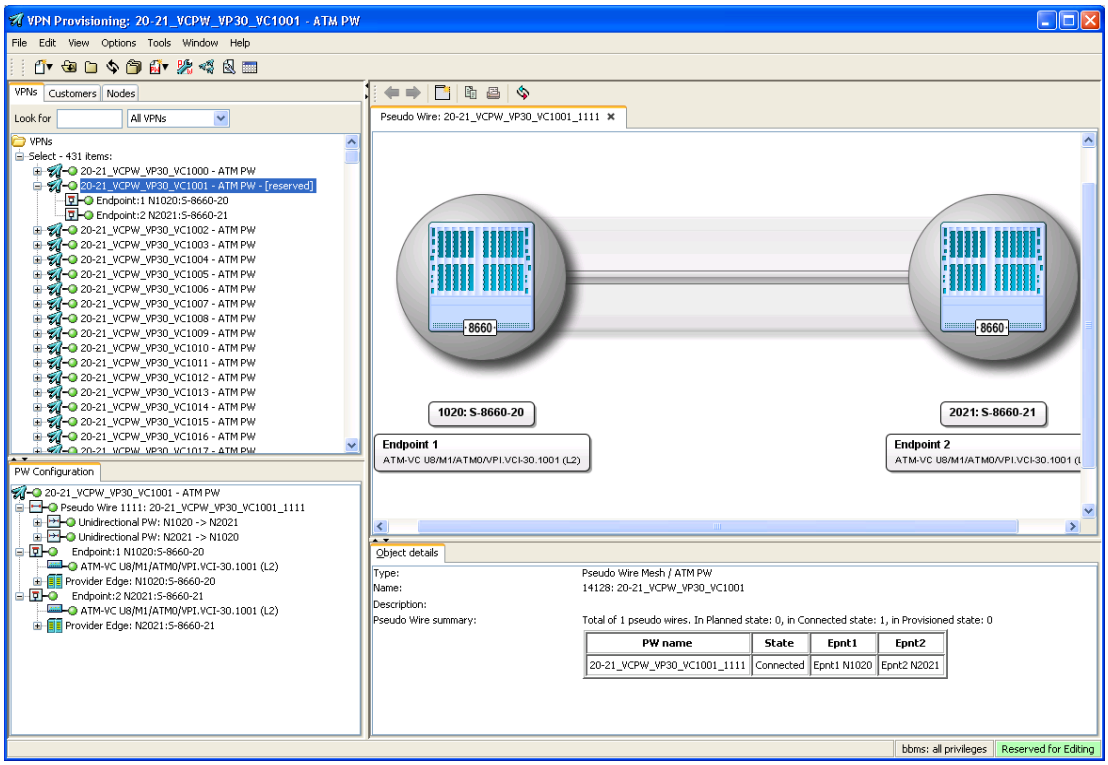


Figure 4.3: VPN Provisioning dialog.

Provisioning packages allow the creation of different kinds of services. With the VPN Provisioning tool, shown in figure 4.3, the user can create, modify, and delete various VPN services. These VPNs can be Ethernet pseudo wires (PWs), ATM PWs, TDM PWs, Frame Relay PWs, HDLC PWs, and IP VPNs. Packets belonging to a certain VPN can be mapped to a desired service class, allowing them to get differentiated treatment, with the provisioning tool. These VPN services are explained in more detail

in the Chapter 5. Tunnel engineering tool enables the manual creation of RSVP based traffic engineered MPLS LSPs. The operator can define explicitly the path that the tunnel takes, define some parts of the path, or let the system use the CSPF algorithm for automatically calculating the path. The Tunnel engineering tool also enables the reservation of capacities for LSPs and the mapping of VPN service classes to LSPs. [39, 40]

The Performance Management Package includes tools for network performance monitoring, reporting and troubleshooting. It allows the viewing and analysis of traffic flowing through various pseudo wires, tunnels, trunks, and physical and virtual interfaces. Communication Servers poll Tellabs 8600 network elements continuously in the background and save this performance data to the database. Packet and cell statistics can be viewed graphically either in real-time or in history mode. If real-time mode is used, the statistics are read directly from the network element according to user defined polling interval. In history mode the data is read from the database and shown from a desired period. The user can configure which performance indicators are shown. For packets and cells these can be link utilization, sent/received octets/packets, errored packets, or discarded packets, for example. [40]

Other optional application packages include testing package, service fault monitoring package, unit software management package, web reporter, planning package and macro package. [40]

4.3.1 Macro Manager Package

The Tellabs 8000 Network Manager offers a macro package which can be used for network management purposes. The purpose of this tool is to accelerate the network management in large networks and when large amount of network objects have to be configured. The macro manager replaces the graphical interface of NMS with macro commands. Macro command, also called simply macro, describes a single NMS operation. The command can be, for example, changing the interface parameter of one interface. Tellabs proprietary macro language supports also high-level control structures so that by combining macro commands with program statements the user can create

more complex macro programs. The user can execute with the macro manager almost all the same operations that are possible with the GUI of Tellabs 8000. Most of the macro commands that configure network elements also update the database so that it is consistent with the current configuration and the same settings can be also seen through the graphical user interface. [40]

Macro programs can be created by reusing sample macros which come with the tool, writing them manually with a text editor, or by recording own macros. Macro manager is capable to record operations done with the graphical interface to a macro file. Macros can be compiled and run with the graphical user interface of the Tellabs 8000 Network Manager. Additionally, those can also be run with the Macro Manager Command Line Interface so that no GUI is needed. [40]

5 Access Network Technologies and Services

In this chapter, a description of various access network technologies and services is given. The first part of this chapter presents technologies that enable the creation of services introduced in the latter part of the chapter. These technologies and services will be present in the implemented soak testing environment and thus it is important to understand these concepts in order to be able to test them.

5.1 Protections

High network uptime is essential for the service providers business. SLA contracts have usually very strict bounds to service availability, which is the probability that a service is available at any point in the future [16]. If the agreed service level is not met, the service provider may face penalties. Various protection mechanisms play an important role in achieving survivable networks and thus in fulfilling SLAs. Protection in networking means that backup resources are allocated for primary resources before a fault occurs [16]. If the primary fails, the traffic is automatically switched to the backup path. There are several different protection modes and the cost of achieving resilience depends on the chosen mechanism. 1+1 mode means that separate secondary resource is reserved for each primary resource. Traffic is sent on both resources and the receiving end selects one copy to be transmitted further. In 1:1 mode traffic is sent only to the primary resource, but if it fails, traffic switches to the secondary resource. When traffic flows through the primary resource, the secondary resource can carry other extra traffic. In general, 1+1 types of protection schemes are usually fastest, but also consume a double amount of capacity from the network. [26]

5.1.1 Multiplex Section Protection (MSP)

Tellabs 8600 routers support SDH Multiplex Section Trail Protection (MSP), defined in ITU-T recommendation G.841, for SDH networks and Automatic Protection Switching (APS), defined in T1.105.01, for networks operating in SONET mode. Tellabs 8600 system extends these standards by considering also hardware based faults on line cards and interface modules. MSP/APS is a 1+1 type of protection mechanism protecting the

multiplex section layer of SDH/SONET networks. MSP/APS provides sub-50-ms protection switch times for point-to-point links [44, 45]

A Multiplex Section Protection Group (MSPG)/ Automatic Protection Switching Group (APSG) includes two identical interfaces, the working (primary) interface is protected by the protecting (backup) interface. Traffic is constantly sent to both interfaces and the receiver selects the better signal for further processing. The selected interface is referred to as the active interface while the non-active interface is called passive. Switching the active side of transmission may be initiated by signal failure, signal degradation, or equipment failure. [18, 45]

Tellabs 8600 network elements support both unidirectional and bidirectional modes of MSP 1+1 protection. Uni-directional mode means that both ends of the link make switching decision independent of each other on the basis of the quality of the received signal. Thus, it is possible that a different line is selected on different directions. In bi-directional mode, on the other hand, the switchover is coordinated in such a way that transmission directions always use either working or protecting side of the group, hence the same link. In the event of switchover, this is communicated to the other side resulting in a switchover also at the far-end. Tellabs 8600 system supports both revertive and non-revertive modes of MSP, described in ITU-T recommendation G.841. [18, 45]

5.1.2 RSVP Path Protection

Tellabs 8600 network elements support also MPLS layer resiliency with the help of pre-signalled RSVP-TE paths. The system supports RSVP-TE based 1:1 LSP protection and MPLS OAM based 1+1 protection switching defined in ITU-T recommendation Y.1720. RSVP-TE based 1:1 protection mechanism uses two explicitly routed RSVP-TE signaled paths to provide less than 200 ms switching times. Protections work in an end-to-end manner where primary and secondary tunnels are pre-signalled between ingress and egress LSRs. These tunnels use Explicit Route Object (ERO) in PATH and Resv messages to build the paths and should travel along disjoint paths to eliminate a single point of failure. [20, 42]

Pre-signalled secondary MPLS paths offer superior performance compared to normal routing protocol based traffic rerouting [27]. In case of fault in a primary path, traffic is switched immediately to an already existing secondary path and thus switching time is highly dependent on the error detecting mechanism. Protection switching may be initiated by an L1 fault, an RSVP path error message, or routing protocol Hello timer expiration [45].

5.2 Inverse Multiplexing for ATM

Inverse Multiplexing for ATM (IMA) is a method of combining several physical links into a single higher-bandwidth logical link, and transporting ATM traffic over this virtual link. This bundle of links, also known as an IMA group, can contain 32 members at maximum. The capacity of the aggregate of links is roughly the sum of the bandwidths of the links that make up the IMA group. After IMA group creation, one can configure this IMA interface as if it was a normal ATM interface. The Tellabs 8600 system supports IMA versions 1.0 and 1.1 defined by ATM Forum. The system supports IMA on E1, T1, and STM-1/VC-12 ports. [50]

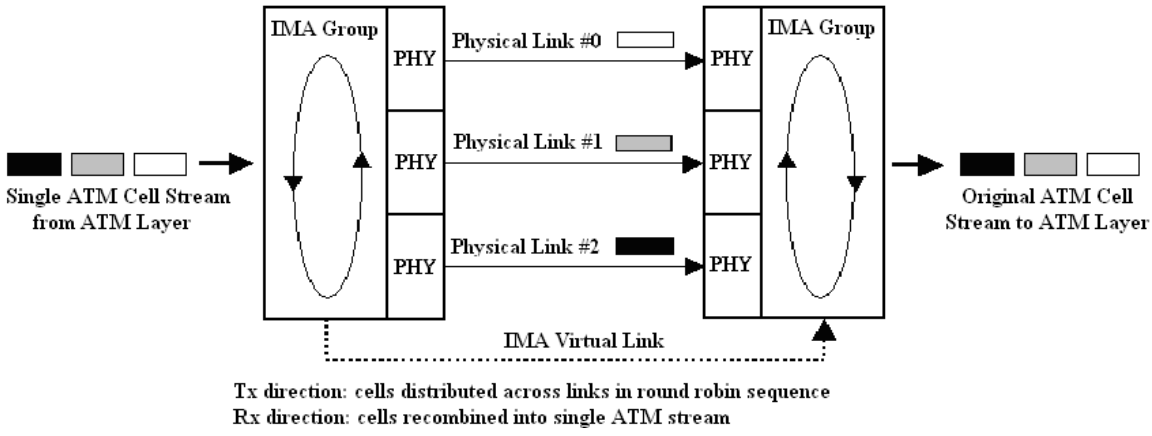


Figure 5.1: IMA operation over three separate physical links. [50]

The operation of IMA is described in Figure 5.1 where two ATM capable network elements are connected by an IMA virtual link consisting of three physical links. At the transmitting side, IMA receives ATM cells from the ATM layer and distributes the cells

in a round-robin manner to multiple links comprising the IMA group. The receiving side reassembles the cells from each link into a single stream and passes the stream to ATM layer for further processing. Naturally, the same applies also in the reverse direction. [50]

The activity of the IMA group depends on the number of active links and on the configurable parameter which defines the minimum number of transmit links required to be active for the IMA group to be in the operational state. If the number of active links is lower than the minimum number, the group is not able to forward traffic. If there are more active links than is required, some of those active links can go down and the group remains still operable. [50]

5.3 The PPP Multilink Protocol (MP)

The Point-to-Point Multilink Protocol is an extension to the point-to-point protocol (PPP), used to set up a direct connection between two endpoints. The initial application of PPP was to use it as an encapsulating protocol when transporting IP packets over point-to-point links [6]. Multilink Point-to-Point Protocol (ML-PPP) aggregates several physical PPP links together in a way similar to IMA. ML-PPP is specified by the IETF in the RFC1990. A single data stream is distributed to multiple PPP links and recombined at the far-end, thus creating a logical point-to-point connection. The bandwidth of the ML-PPP virtual link is approximately the sum of the individual links. This ML-PPP group can be given normal IP interface configurations, such as IP address and routing. [36]

5.4 Services

This section covers MPLS based L3 and L2 VPNs supported by the Tellabs 8600 series routers. A VPN is a private network that uses a public telecommunication infrastructure, such as the Internet, to provide secure connections. VPNs are used to connect companies' remote offices and to support remote access to company resources. Service providers offer VPN solutions as a cheaper alternative to leased lines.

5.4.1 MPLS Layer 3 Virtual Private Networks

The Tellabs 8600 Managed Edge System supports IP VPNs based on the IETF specification RFC 2547bis. RFC 2547bis describes an approach for a service provider to offer any-to-any type IP VPN services by using their IP backbone [32]. It is also called BGP/MPLS VPN since BGP is used for customer route distribution over the public backbone network and MPLS is used for customer traffic forwarding. This VPN service model allows for multiple different VPN topologies, such as full mesh and hub-and-spoke. Figure 5.2 shows the RFC 2547bis IP VPN model. The Customer edge (CE) router connects the customer's local area network to the service provider's network by forming an adjacency with the Provider Edge (PE) router. CE is typically a normal IP router, which can be owned either by the service provider or the end customer. PE routers are responsible for the service intelligence at the edge of the IP/MPLS core network. Provider (P) routers are core network routers performing ordinary MPLS forwarding based on the outermost MPLS label. Since P routers do not need to have any knowledge about the VPNs, the scalability of the model is greatly enhanced. [22, 44]

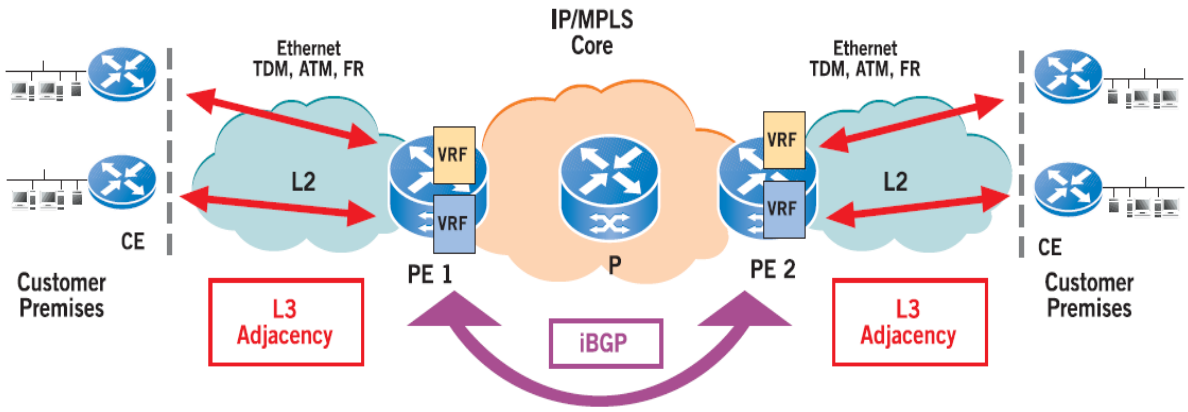


Figure 5.2: RFC2547bis based IP VPN model. [44]

PE routers are routing peers with connected CE devices and with other PE routers in the service provider's network. CE device can advertise site's routes to PE router with eBGP, OSPF, RIP, or alternatively with static routing. PEs use iBGP to distribute customer routes of the connected sites to other PE routers across the backbone. In order to deliver this VPN routing information, PE routers may establish a full mesh of iBGP

sessions. Since full mesh of BGP sessions would cause some scalability problems, the operator can ease the burden of edge devices with route reflectors. Every PE router can have a BGP session with the route reflector and advertise the routes to it, which again distributes the routes to other PEs. In this case every PE needs only one iBGP session. [22, 32]

PE routers maintain a separate VPN Routing and Forwarding (VRF) table for each customer. VRF tables are used for isolating the customer VPN routes from the global IP routing table. Each customer interface of a PE router is associated with the VRF table of that customer. VRFs can control the import and export of routes with the BGP extended community route target attribute. Routes are installed only if the received route's export route target attribute is the same as VRF's import target. By choosing the import and export target attributes correctly, one can create different VPN topologies. [22]

The customer VPN traffic is tunnelled through the service provider network using LSPs established between the PE routers. Several LSPs might exist between the same pair of PE routers, for example, in cases where the service provider wants to offer different levels of QoS to various customers. These LSPs are established with LDP or RSVP-TE, like normal MPLS paths. A label stack of two labels is needed for BGP/MPLS IP VPNs. The intermediate P routers use the outer label for traffic forwarding from the ingress PE router to the egress PE router. The inner label, which is also called the BGP label, identifies the correct VRF and thus the VPN, at the PE routers. BGP label is assigned and distributed by the MP-BGP (Multiprotocol BGP). P routers are not aware of this label. Since inner label identifies the customer, several customers' traffic can be tunneled through the same tunnels across backbone. [22, 45]

Different customers might use the same IP addresses from the private IP address space in their intranets. This causes problems to the exchange of routing information between PE routers, since with conventional BGP every IPv4 address needs to be globally unique. To solve this problem, RFC2547bis defines VPN-IPv4 address family to distinguish these possibly overlapping addresses from each other. A VPN-IPv4 address

is formed by adding an 8 byte route distinguisher to the basic IPv4 address. This route distinguisher points the VRF where the routes should be inserted. Since normal BGP can carry routing information only for IPv4, IETF defined Multiprotocol Extensions for BGP in RFC 2858. This extension enables BGP to carry routing information for many network layer protocols, including VPN- IPv4 routes. [22, 32]

In addition to the traditional IP VPN described above, the Tellabs 8600 system offers a hierarchical model for L3 VPN services. It takes the same approach that was used in the traditional model in the core network and extends it also to the access domain. The basis for the architecture has been the IETF hierarchical model for VPLS service, which has been enhanced to cover IP VPN services. With the hierarchical model, an MPLS VPN does not have to end at the edge of the core network, but this can be spanned through the access network. This has the benefit of bringing traffic engineering and QoS capabilities to the access network thus improving the management of the services. [44]

The components of the distributed IP VPN model are shown in the figure 5.3. In the model the functionalities of PE routers are divided between two separate routers. These new router types replacing PE routers are U-PE (user-facing PE) and N-PE (network-facing PE) routers. The U-PE is at the edge of the operator's access network and has an IP layer connection with the CE router. N-PE routers are at the edge of the service provider core network and communicate with other N-PE routers using iBGP. U-PE distributes routes across the IP/MPLS access network to the N-PE router with eBGP since these devices reside in different ASs. Routers between U-PE and N-PE performing normal MPLS forwarding are called P-a (P in access) routers to distinguish them from P routers performing similar operations in the core network. This model requires three separate LSPs; one LSP across each of the access domains and one over the core network. In addition to improving the manageability of the services, the model greatly enhances the scalability by moving some of the service intelligence closer to the customer demarcation point. [44]

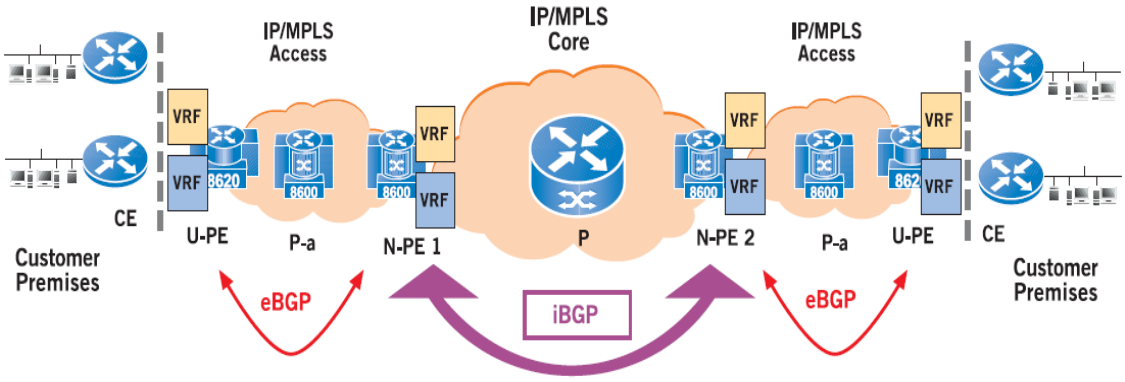


Figure 5.3: Hierarchical IP VPN model. [44]

5.4.2 MPLS Layer 2 Virtual Private Networks

Layer 2 VPNs can be either point-to-point Virtual Private Wire Service (VPWS) or multipoint-to-multipoint Virtual Private LAN Service (VPLS). VPWS is a service that emulates traditional leased lines whereas VPLS is used to emulate LAN over WAN. In both services the CE device is connected to the PE router via layer 2 attachment circuit, such as Ethernet port, VLAN ID or ATM VPI/VCI. The PE devices are connected with pseudo wires, which carry customers layer 2 packets over the MPLS/IP network. [43]

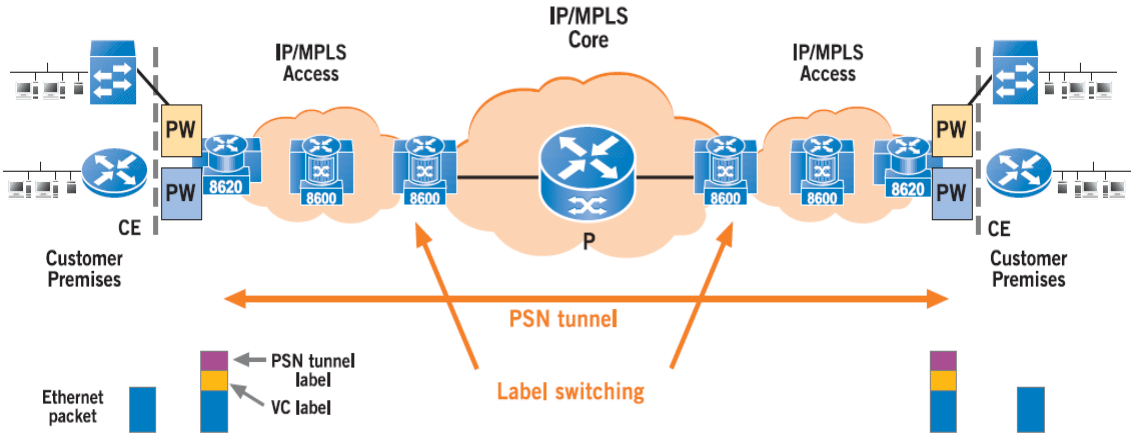


Figure 5.4: MPLS-based pseudo wires used to offer L2 VPNs. [44]

The VPWS implementation supported by the Tellabs 8600 system is the Martini draft, also known as PWE3 (Pseudo Wire Emulation Edge-to-Edge). PWE3 architecture is defined in RFC 3985 and is shown in the figure 5.4. Pseudo wires emulate native layer 2 services over MPLS- enabled Packet Switched Networks (PSN). This emulated

service can be ATM, Frame Relay, Ethernet, or TDM (SAToP, CES), for example. Pseudo wires are established by creating two unidirectional MPLS LSPs. The customer traffic is sent from a CE device through the attachment circuit to the PE router, which is the pseudo wire edge node. This PE router maps the traffic from attachment circuits to correct pseudo wires. At the other end of the pseudo wire the traffic is mapped from the MPLS label to the appropriate customer interface. Pseudo wires are tunneled across the backbone inside trunk LSPs established between PE routers with either LDP or RSVP-TE. [4, 44]

As in the case of IP VPN, again a stack of two labels is needed when forwarding pseudo wire traffic. The outer label, known as PSN label, is used to forward the data through the network from ingress PE router to the egress PE router. The inner label identifies the correct customer interface at the ends of the pseudo wire. There are two slightly different VPWS implementations defined. Draft Martini, supported by Tellabs 8600, uses LDP as the inner label signalling protocol, whereas draft Kompella uses BGP. In addition to LDP signalling, Tellabs 8600 network elements support also static label assignment. This means that the inner label is generated by Tellabs 8000 Network Manager during the service provisioning process. [44, 45]

Pseudo wires are advantageous for backhauling mobile traffic from cell sites through RAN to the RNC/BSC sites with Ethernet, ATM and TDM pseudo wires. Pseudo wires offer a more cost-efficient way of delivering legacy technologies by separating the transport protocols from the transmission media. Furthermore, they allow the benefits of statistical gain and provide smooth transition towards all-IP RAN. [49]

VPLS service is a service provider offering which makes it possible to provide Ethernet based multipoint to multipoint connectivity over IP/MPLS networks. Currently, there are two options for VPLS implementations. RFC 4761 based VPLS, co-authored by Kireeti Kompella, uses BGP for PE label signalling and RFC 4762, co-authored by Kireeti's brother Vach Kompella, uses LDP protocol for the VPLS signalling setup. Tellabs 8600 system supports the latter draft. [44]

6 Software Testing Overview

In this chapter, a description of the software testing process and methods are given. After the general discussion of software testing, we will examine the objectives of soak testing in more detail. As soak testing requires automating the tests, we discuss this in the end of the chapter.

6.1 Software Testing

Telecommunications operator's income is composed of a set of communications services offered to customers. The quality of these services and their perception by users is essential for the service provider's business. These services are composed of several different types of network elements, different communication protocols and interfaces, which all make them extremely complex types of services. Due to the high complexity of these services and the importance of continuous operation to the service providers, the phase of testing becomes crucial in telecommunications.

Testing can be understood as a "process used for revealing defects in software, and for establishing that the software has attained a specified degree of quality with respect to selected attributes" [5]. Where quality is defined as "The degree to which a system, component or process meets specified requirements" and "The degree to which a system, component, or process meets customer or user needs or expectations" [17]. After the testing phase, if some defects have been discovered, starts the debugging or fault localization phase, where the idea is to find the root cause, repair the code, and to retest the software [5].

Misconceptions or mistakes made by humans cause *errors* in the system. This error can, for example, be a typing mistake made by the programmer in the coding phase. These errors in the system cause *faults*, also known as *defects*, in the system. Faults are abnormal conditions that may cause the system to perform certain functions against their specifications. On many occasion these are also called bugs. A fault may or may not lead to a *failure* in the operation, which is a deviation of a program from its

expected behaviour. Failure is what the user of the program sees when executing the program. [5]

Testing is sometimes misunderstood as a process which reveals all the defects in the software. Nevertheless, it is impossible to show that software is free of defects, but testing should take the approach of showing the presence of defects. Even if testing does not find any more defects it is not a proof of defect-free software since all parts of the software cannot be tested with all the possible inputs. Instead of trying to test everything, testing should use risk and priorities to focus efforts on those parts of the software that are the most important. Every somewhat larger piece of software contains errors, but good testing can help to considerably reduce the risk that failures manifest themselves in real operational environments. Thus, testing has a vital role in contributing to the quality of the system. It is generally recognized as a fact that the quicker problems are found the less it costs to fix those. Especially when defects are discovered at the customers operational environment, the cost can be substantial. [24]

Common perception of testing is that it only consists of running tests, i.e. executing the software. However, software testing includes also other activities than just test execution. First the testing activities have to be planned. After planning and before actual test execution the test-cases have to be created and the test environment developed. Everything done during the test execution should be documented so that another person is able to replicate the results. After tests have been executed their results have to be evaluated in order to determine whether or not the test has been successful. Immediately after finding problems, the tester should write a problem report which is used later as a basis for debugging and fixing the problem. Even though the test has passed successfully, it has to be documented into a test log. [24]

6.2 Classification of Testing Techniques

Figure 6.1 shows one kind of classifications of different testing techniques.
Testing techniques can be divided into dynamic and static techniques. Dynamic testing is testing in more traditional sense where programs are executed and the outcomes are

verified against expected results. Static testing does not comprise execution of the program but includes examination of the written code. Examples of static testing are code inspections, code reviews, walkthroughs, and desk checking. [24]

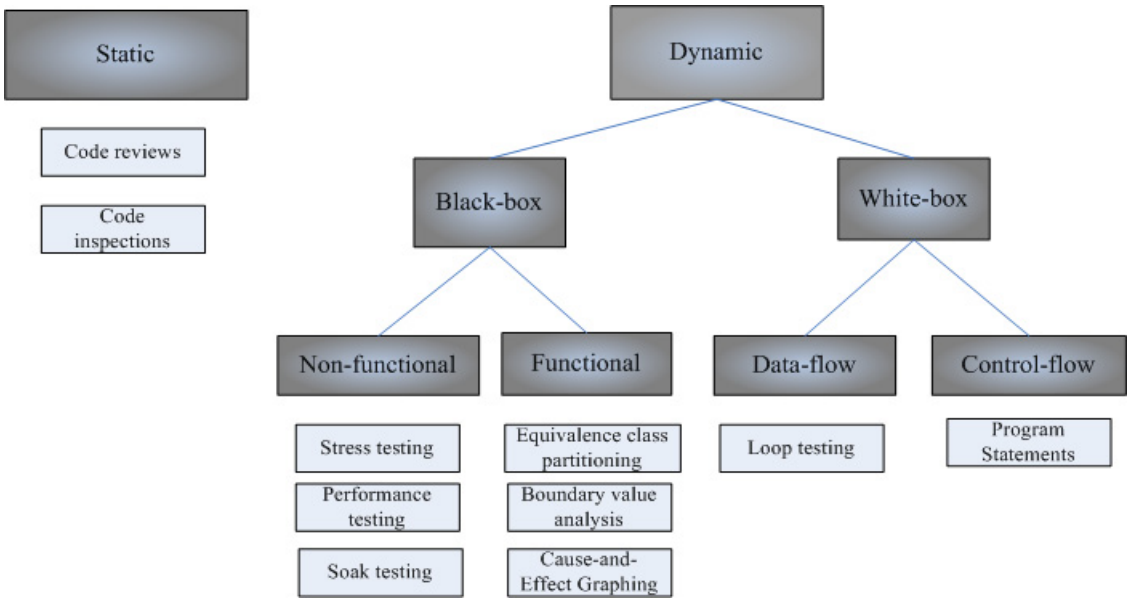


Figure 6.1: Classification of testing techniques

Dynamic testing techniques can be further divided to black-box and white-box methods [5]. Black-box testing, also called functional or specification-based testing, considers the system to be tested as a black-box without knowledge of its internal structure. The tester takes an external viewpoint on the system, knowing only the inputs and the expected outputs and not how those outputs are attained. White-box techniques, also called structural or glassbox methods, are based on knowledge of the internal structure of the system or module to be tested. [24]

In order to design and execute test cases using white-box strategy the tester needs to see the innermost of the system, i.e. the actual code. The object of white-box methods is to verify that the internal components of the program are functioning correctly. White-box testing is typically used for small software components. It usually involves inspecting the code and following the execution of the program to determine problems in the program structure. These methods can be further divided to data-flow based and

control-flow based methods. The first one is based on the idea of following the values of the variables in the code. The latter one means following the progression of control in the program. [5, 24]

White-box testing methods alone are not enough to test software thoroughly since it might miss certain types of defects, such as missing functionality. Those techniques are also incapable of measuring qualities such as performance and usability of the product. As mentioned, black-box testing abstracts the innermost of the system and considers it as a black-box. Black-box can be either functional or non-functional testing. Non-functional testing tries to find out the quality characteristics of the system. Examples of non-functional testing are load testing, reliability testing, and soak testing.
Functional tests, on the other hand, try to determine whether the functionality of the software is working correctly. Functional testing is not concerned with how processing occurs, but rather, with the results of this processing and that the behaviour of the system complies with the requirements specification. [5, 28]

6.3 Levels of Testing

Testing in large software projects is usually divided into different levels, each of which has their own testing goals. The V-model, which is shown in figure 6.2, is an easy to understand software development model originally developed from the waterfall model. It describes the relationship between various phases of the software development process and its corresponding phase of testing. The model is shown in figure 6.2. Left side of the model shows how development phases follow each other starting from the initial requirements from the user's point of view and ending at the final coding phase. At each step the specification gets more detailed. When one development phase is completed the design of the corresponding testing phase can begin and the development specification is used as basis for test designs. Right side of the model shows then how testing phases follow each other. When travelling upwards the model, in each testing phase bigger parts of the system are covered by single tests. [5, 23]

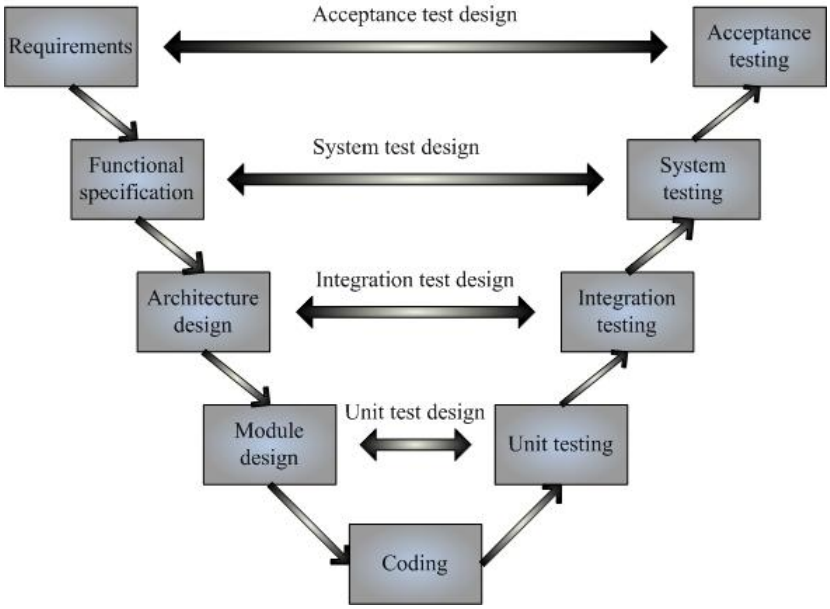


Figure 6.2: The V model.

Testing in the V-model is implemented in four separate levels. These are unit testing, integration testing, system testing, and acceptance testing as shown in the figure. Each testing level concentrates on different aspects of the system implementation as the testing is done against the specifications of the corresponding development phase. When moving upwards in the V-model, testing methods usually shift from white-box methods more and more towards black-box methods. [12]

The first stage of testing, according to the V-model, is the unit testing, where individual components are tested separately. Unit refers to the smallest software component that can be tested. Defects found at the unit testing level are easier to fix since the root cause of the fault is easier to pinpoint and thus costs less to fix. Unit testing is often done by programmers and involves inspecting the source code. The goal is to find functional and structural defects of the unit and to verify that the interface of the unit functions as specified. The importance of good unit testing is emphasized by Boris Beizer, who has stated that "The most notorious bugs in the history of software development were all unit bugs - bugs that would have been found by proper unit testing." [5]

Units that have passed unit testing may introduce new defects when connected to other individually working units. Integration testing involves testing to make sure that these individually working units work properly together. Integration testing has two major goals: “to detect defects that occur on the interfaces of units” and “to assemble the individual units into working subsystems and finally a complete system that is ready for system test” [5]. Integration testing can be carried out on many levels, such as component integration or system integration. Component integration means testing individual components together, whereas system integration involves testing fully functional systems together. Integration testing should be done iteratively, which means integrating new units into the tested subsystem one at a time. [5, 12]

System testing means testing the system as a whole. It includes usually both functional and non-functional requirements of the software. The upmost object of system testing is to verify that the system performs according to its requirements. System testing is usually carried out by an independent testing group and not by the developers. Due to the complexity and large workload, system testing is often automated using testing tools. System testing is a black-box type of testing since it abstracts the inner parts of the system and is mainly concerned with inputs and corresponding outputs. [5]

After system testing the software is ready for acceptance testing where the purpose is to make sure that the software satisfies customer’s needs and requirements. The clients and staff from the development organization together create test cases which will be run in acceptance testing. This is the last possibility for the customer to make sure that the software meets their requirements specified at the start of the development process. Clients should be allowed to participate in acceptance testing since they are the ones that ultimately decide whether software is accepted or not. The software is run in actual usage environments with customers own input data.
Variants of acceptance testing are alpha and beta (field) testing. Both of these testing types are conducted by potential customers and not by the people of the development organization. The difference between the two is that alpha testing is performed at the

facilities of the developing organization, whereas beta testing is conducted at the customers own locations. [5, 12]

6.3.1 Regression testing

When modifications are made to the software, either by adding new functionality or changing old modules, testing is needed to make sure that the capabilities of the old version still exist and everything that worked previously is still functioning correctly. This kind of retesting that tries to ensure that the new version has not caused any unintentional effects is called regression testing. Regression testing can be performed at any test level. Regression testing can be highly time-consuming and expensive to execute if automated testing tools are not used. A distinction should be made between regression tests and re-tests, which mean re-running the same tests that found defects after those faults have been fixed. [5, 12]

6.4 Soak Testing

Soak testing means that a system under test is run at high level of load over a prolonged time in order to detect slow to appear faults. When system is tested for a long time without resetting it, such problems as memory leaks, stack corruption, wild pointers, and buffer overflows may cause serious troubles. For example, in software testing, a system may function properly when tested for a few hours. However, if test cases are executed continuously for over 2 weeks, problems may cause it to eventually fail or behave unexpectedly. Executing this kind of long tests requires automation since it would be impossible for humans to continuously operate and detect failures in the system. In addition to the above mentioned problems, soak testing tries to find following types of defects; failures to close connections of various tiers of a multi-tiered system, problems in closing database cursors and thus possibly leading to a entire systems stalling, and degradation in the response time of some components due to internal data structures getting more inefficient. [23, 35]

Soak testing tries to simulate normal operation of the system, but in an accelerated way. In soak testing events are introduced to the system more often than would normally happen in a real operating situation. The speed up of events allows the tester to detect

problems quicker than in a normal operating environment. For example, some faults may manifest themselves after certain operations are executed sufficiently many times. Other names for this kind of testing are endurance testing, duration testing, long sequence testing, and burn-in testing. Soak testing is sometimes associated with stress testing. In a way, soak testing can be considered as specialized type of stress testing. Stress testing is a type of testing where one tries to deliberately break the system. This can happen by either overloading the system or by reducing some of its resources. When the system is gradually overloaded, the effects of this activity to the system are examined. [8]

It is important to follow the memory usage of a system under test, since this may reveal some memory leaks. If the memory usage goes upwards during tests, new memory is reserved somewhere. This is often normal, especially when complicated configurations are added, but if the amount of allocated memory does not return to lower values after the configuration has been removed, it is possible that there is a memory leak. Another interesting measure is the CPU usage, and how it changes during the test. In addition to measuring the resource utilization of the system under test, also such facilities as Java Virtual Machine should be monitored. [8, 35]

The preferred duration of soak testing is naturally dependent on the tested application and what is the required amount of time for the system to run without problems. However, the duration is often determined by the available time in the test lab. Systems usually have a regular maintenance window, which is a good starting point when determining the duration of soak testing. Due to the long run time, it is virtually impossible to implement soak testing without some sort of automation. Therefore, test automation is presented next. [8]

6.5 Test Automation

Testing software is hard and time consuming work, thus it is a natural step to use tools to make the job easier and more efficient. Test automation means using software to test other software. It is an attractive addition to manual testing, which in some case might

be extremely helpful but when wrongly used can distract testers or squander resources. When properly used, automation can save time, speed development, extend the reach of testing, and make it more effective. A common perception among some people may be that the more tests are automated the better. However, automation is not suitable for everything but should be used only when it advances the mission of testing. [23, 25]

Automation is especially useful in situations where the same tests have to be run many times to different software versions. Smoke tests are run after the new build is ready and is done to verify that it is worth further testing. It tests basic functionality of the software that is expected to always work. Automated smoke tests can be run by anyone during build process. These tests help to give fast feedback to developers immediately after the new build is ready and thus accelerate the software development. Also automating regression testing has many benefits, since regression tests are usually run for every software version and stay relatively the same from one version to another. [23, 24]

The quality of automated testing depends on the ability to automatically detect problems. When a human performs tests manually, he or she can detect all kinds of anomalies in the behavior of the software. Whereas, computer detects only problems that it is programmed explicitly to identify. Every time the test is run, it does the same things in the same order and verifies the correctness in the same way. Thus, automated test might miss some failures that a human would easily detect. A human can sometimes do things differently and change the execution based on the behavior of the program. Automation is not a replacement of manual testing, but rather an extension to manual test which allows different types of tests. Some tests just cannot be executed without automation and other tests can be further expanded. Various performance tests are good examples of where automation can be particularly helpful. Here are some examples [23]:

- *Load test* – Measuring a system's performance and resource utilization under heavy load

- *Performance benchmarks* – Collecting measurements to determine whether the system performance is getting better or worse.
- *Configuration test* – Running same tests on different platforms and in different configurations.
- *Endurance test* – Also called soak testing. Running system for a long time.
- *Combination errors* – Using automation to run tests that use different features of the software in different combinations.

Capture-replay tools are popular and easy to use test automation tools. These tools can record user actions while executing the program under test. It stores the captured user events into scripts which can be replayed later. These scripts examine user interface elements, locate the correct objects, and use them as specified in the script. While running, it compares the current screen of information to information that was stored at the recording phase and decides whether the test is passed or failed. However, capture-replay tools have major limitations. The test input and output are hard-coded to the scripts and they require that the user interface does not change much. When the GUI changes, the test tool might not find correct elements anymore. To avoid changing the tests every time the GUI changes, one needs to abstract the GUI from the tests. Few techniques for accomplishing this abstraction are provided below [23, 40]:

- *Window maps*- Window maps tell to the scripts how to identify certain GUI components. In this way there is no need to embed the identification method explicitly to every reference to this component. When the GUI changes only these window maps have to be changed.
- *API-based automation* – Avoids the use of GUI by using programming interfaces found nowadays in many software products.
- *Task libraries* – Divide the program execution into smaller tasks which are distinct from each others. After this, scripts can be created by combining these tasks conveniently. In case the user interface changes, only the task that uses the part of GUI that changed have to be updated. This avoids changing the whole test case.

An improvement to capture-replay methods is the data-driven approach where inputs and outputs are not hard-coded to the scripts but reside in separate file. Since these are now separated from the script, the same script can execute several similar test cases with different input and output combinations. Keyword-driven methods use input files that in addition to data contain also keywords. These keywords define which functions or tasks will be run from the task library. By combining these tasks conveniently, also non-programmers can easily and rapidly create new automated tests. [23, 25]

7 Implementation and Verification of the Soak Testing

In this chapter, the implemented soak testing environment will be presented. The soak testing environment was implemented by first building a dedicated network for only soak testing purposes and then creating software for operating the network. First the objectives for the soak testing environment are presented in section 7.1. Hereafter, the physical environment itself is presented in section 7.2. This includes both the physical structure of the soak testing network and the various configured services. The network elements need to be operated somehow and various mechanisms for this are discussed in section 7.3. Failure detection mechanisms are examined in section 7.4. Integration of these various parts into one functioning program is explained in section 7.5 before discussing the deployment of soak testing in section 7.6.

7.1 Objectives

At the beginning of this thesis, it was decided that in addition to building the actual network the requirements for the soak testing environment are:

- *Automatic operation of the network* – We need to find some methods to be able to automatically create various operations to the network.
- *Automatic failure detection* – Ways need to be able to automatically verify whether everything is working correctly or not.
- *Logging* - We want to keep some kind of log file about the operations done in the network so that we are able to check afterwards that what has been done.

Logging is especially important in situations where some kinds of errors are encountered so that one can easily see what was going on in the network at the moment when these errors appeared. Operations are created to the network for mainly two reasons. The first one is to reveal problems in resource usage, which show up after the operation is done sufficiently many times. The other one is to reveal defects that show up rarely, but when operations are done many times the probability of those defects showing up will be increased. The purpose is to emulate a real operating network in a

highly accelerated way. In the first stage a few operations will be implemented so that the environment is operational. Afterwards, the environment will be gradually enlarged so that it will encompass more operations and functionality. The first step was to build the network and the presentation of implemented soak testing starts by presenting the environment including the network elements and services.

7.2 The Soak Testing Environment

Figure 7.1 below shows the physical network of the implemented soak testing environment. There are five Tellabs 8600 network elements; one of each type. Nodes are numbered so that 8660 is node 20, 8630 is node 21, 8605 is node 22, 8620 is node 23 and 8607 is node 27.

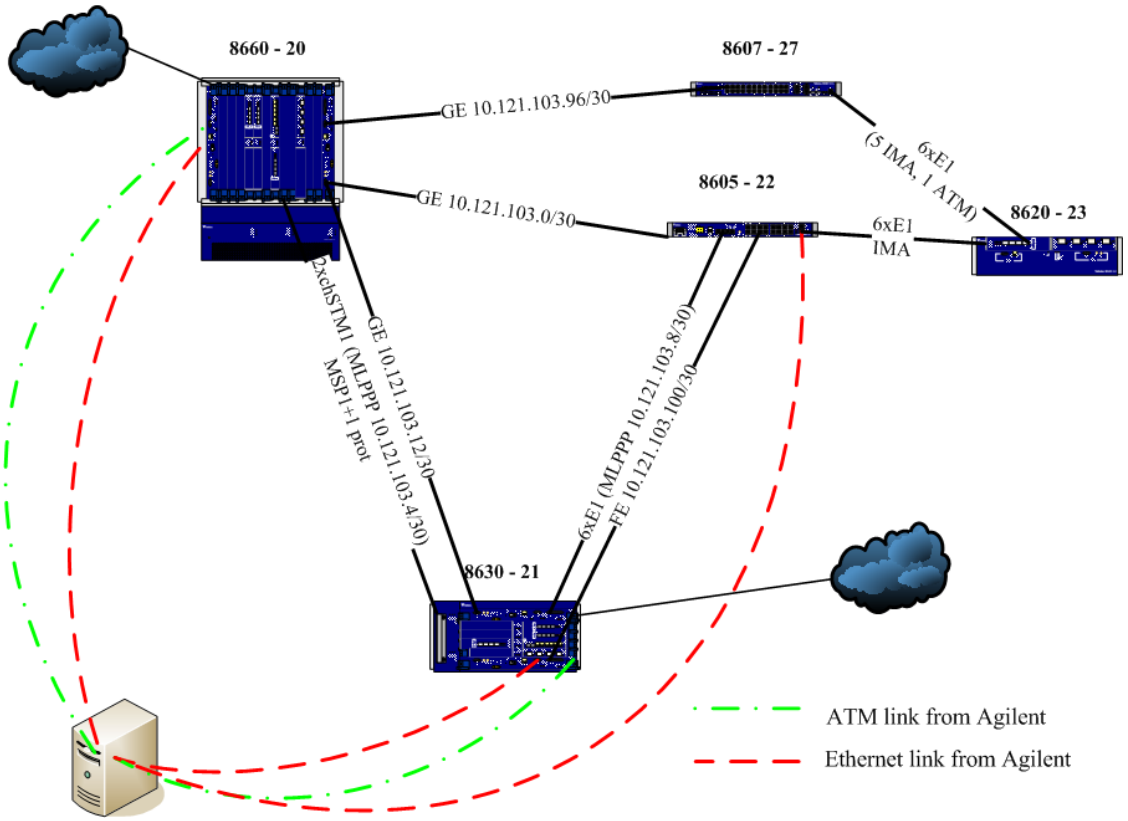


Figure 7.1: The soak testing environment

The 12 member ML-PPP group between nodes 20 and 21 is configured on the channelized STM1 interfaces and is MSP 1+1 protected. As shown in the figure 7.1, the

measurement equipment is connected to the network with ATM and Ethernet links. There is a STM1ATM loop in the node 23 so that the transmitting side is connected to the receiving side of the same port. This way ATM cells transmitted to that port will be received in the same port. The reason for looping traffic back in this way is that there is no STM1ATM interface in 8605 and 8607 nodes which could be connected directly to the measurement equipment. With this arrangement, the ATM traffic entering the network from nodes 20 or 21 will propagate through the network and will be eventually received by the same measurement equipment port that transmitted those cells. The biggest node, which is node 20, emulates a Tellabs node at the BSC/RNC site while the smallest nodes (8605 and 8607) emulate nodes at a base station site. The network is currently rather small, but it will be enlarged in the future as more network elements and links will be added.

In order to introduce more IP routes to the network, we configured the measurement equipment to emulate a number of OSPF routers. These emulated routers advertise IP routes, represented by the clouds in the figure 7.1, to the network. These OSPF emulation settings can be changed rather easily from the measurement equipment and are an easy way to increase the load on the routers. For example, the number of routes advertised or the number of OSPF neighbours can be changed with ease. Furthermore, by disconnecting correct links in the network we can force new route calculation in the whole network.

7.2.1 Services

The network includes an IP VPN service and various pseudo wires between the nodes. There are both LDP and RSVP-TE signalled tunnels in the network to signal the outer label of the VPNs. The RSVP tunnel between the nodes 20 and 22 is RSVP path protected so that the primary path goes straight from node 20 to 22 via the straight Gigabit Ethernet link, while the secondary path goes via node 21. There are two separate RSVP sessions established between the nodes 20 and 21; one utilizing the MSP1+1 protected MLPPP link and the other travelling via the GE link. Traffic is divided between these two tunnels by mapping service classes to these tunnels so that the higher priority packets go via the MLPPP link and lower priority packets via the GE

link. In addition to the MPLS tunnels mentioned above, the measurement equipment is used to emulate a large number of LDP and RSVP sessions. The measurement equipment can be used to emulate also various other protocols to the network. Emulations are a rather easy way to increase the load on the routers.

Table 7.1: Services of the environment

VPN type	Nodes	Endpoints 1	Endpoints 2	Service Class
ATM PWs	20 and 21	VP 100-199	VP 200-299	CBR
ATM PWs	20 and 21	VP 30 VC 1000-1099	VP 30 VC 1000-1099	nrt-VBR
ATM PWs	20 and 22	VP 600-624	VP 300-324	CBR
ATM PWs	20 and 22	VP 50 VC 2000-2024	VP 50 VC 2000-2024	CBR
ATM PWs	21 and 22	VP 400-424	VP 400-424	CBR
ATM PWs	21 and 22	VP 60 VC 2000-2024	VP 60 VC 2000-2024	CBR
ATM PWs	20 and 27	VP 85 VC 300-349	VP 75 VC 300-349	CBR
ATM PWs	20 and 27	VP 89	VP 79	CBR
Ethernet PWs	20 and 21	VLAN 1101-1110	VLAN 2101-2110	rt-VBR
Ethernet PWs	20 and 21	VLAN 1150-1159	VLAN 2150-2159	UBR
Ethernet PWs	20 and 22	VLAN 1301-1310	VLAN 3301-3310	rt-VBR
Ethernet PWs	21 and 22	VLAN 2201-2205	VLAN 3201-3205	rt-VBR
IP VPN	20 and 21	VLAN 1001-1007	VLAN 2001-2007	

Table 7.1 list the pseudo wire meshes configured to the soak testing environment, excluding the TDM PWs. A Pseudo wire mesh is a VPN which is composed of several point-to-point pseudo wires. For most PWs, the PW endpoint is directly connected to the measurement equipment, but the ATM PWs having another endpoint at 8605 and 8607 nodes are an exception. From these nodes the ATM cells coming from PW travel over the IMA link towards the 8620 node from where these are looped back as explained in the previous subsection. The figure 7.2 shows the ATM and Ethernet PWs and the outer label signalling protocols. The L3 VPN is omitted from the figure but is configured between nodes 20 and 21. There are seven L3 VPN endpoints in the nodes 20 and 21. These endpoints are in the same physical Ethernet interface, but virtually separated by VLANs. The measurement equipment is configured to send ATM cells to each individual ATM PW and Ethernet frames to Ethernet PWs and to IP VPN so that the state of every VPN can be monitored by looking at the stream from measurement equipment.

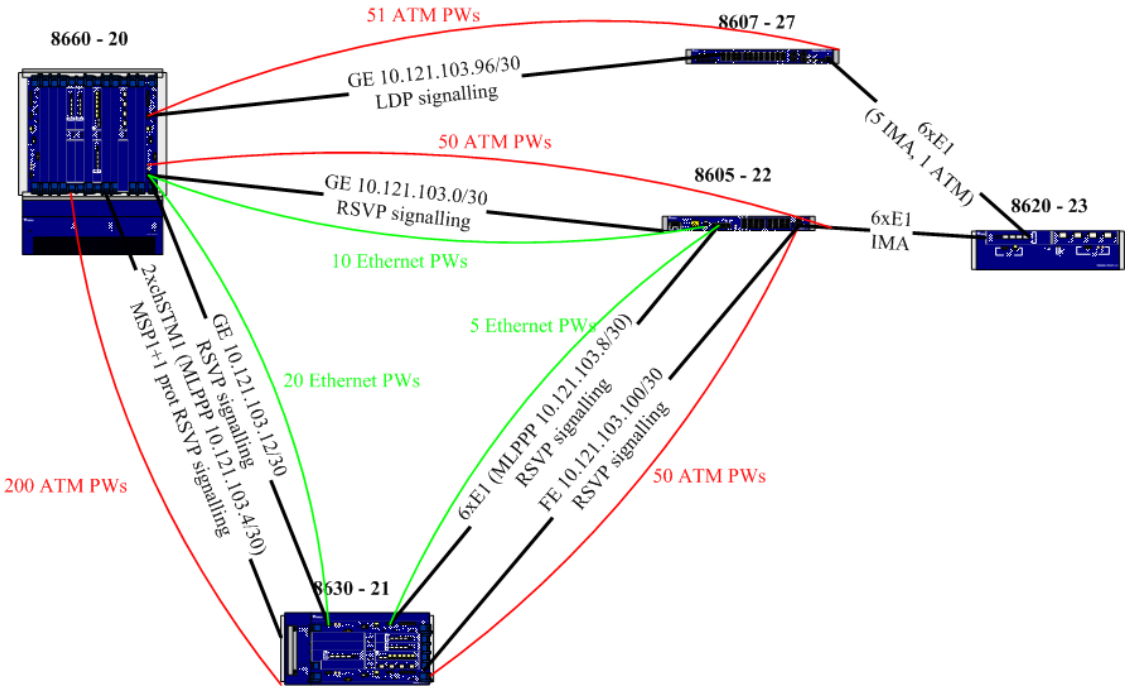


Figure 7.2: Pseudo wires configured in the network.

There are also several CES and SaTOP TDM PWs in the environment. These are configured between the nodes 20 and 22 and between nodes 20 and 21 so that the full capacity of one channelized STM1 interface is utilized in the node 20. No data is sent to these TDM pseudo wires from the measurement equipment, but the TDM PWs operate so that even though there is no data coming towards the pseudo wire it will still send TDM frames. This way we could introduce more traffic to the network without needing to configure and use ports on the measurement equipment. Furthermore, by configuring also TDM pseudo wires we get more versatile configuration and higher burden on the network elements.

7.3 Automating the operations

Operations are meant to simulate the real operating networks at a highly accelerated way. The operations that we implemented in this first phase, and the ones that will be implemented later, are selected based on two factors. Firstly, the experience gained from system testing was taken into account. Based on the experience, we can select such features where we are most likely to find defects also in the soak testing. The second reason was the ease of implementation.

7.3.1 Operating network with QTP

First we examined whether the Quick Test Professional automation tool could be used in the soak testing environment. QTP is an automated Graphical User Interface (GUI) testing tool primarily targeted to regression testing. It identifies application GUI objects and performs desired operations, such as mouse click, on these objects. It allows the automation of user actions by using a scripting language built on top of VBScript to control the application. This tool was already used by the regression team in the system testing of 8600 system and thus the test libraries would have been already ready to be utilized also in soak testing. [14]

In order to be able to check the applicability of the tool to soak testing, we created some simple sample scripts. These scripts tried to do some basic operations from NMS GUI for an extended period of time. An example script tried to change the protection side of the MSP 1+1 protection group. The operation was first manually executed and then the captured script was modified so that it repeated the same operations over and over again. The dialog for changing the protection side of the group with Tellabs 8000 network management system is shown in Figure 7.3. The script opened the window from the tree view of the node manager every 45 seconds and changed the side to protecting or working depending on which side it was currently. After each GUI event, such as mouse click, the script waited for 4-7 seconds for the GUI to settle down before making any further operations.

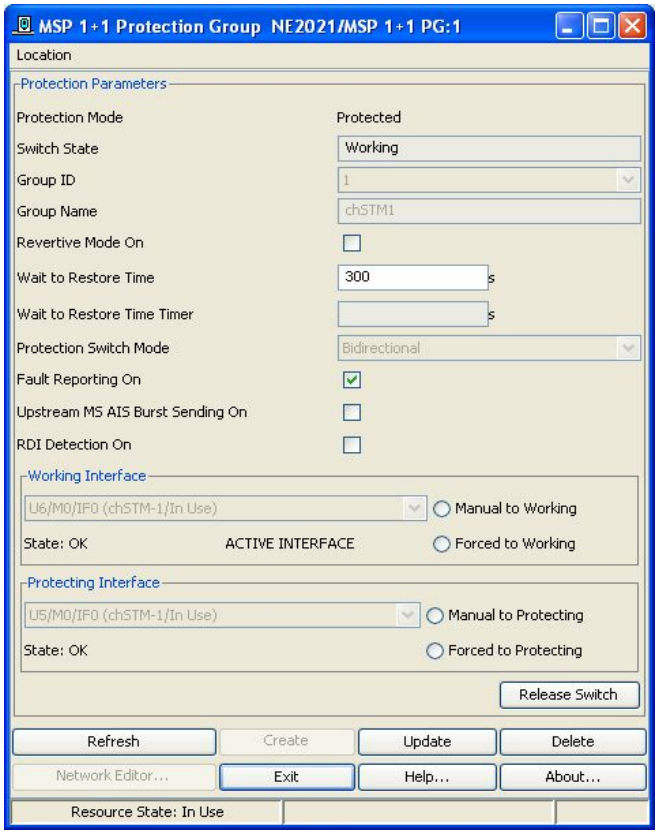


Figure 7.3: NMS dialog for MSP 1+1 protection

This script was executed several times with the result that at some point the testing tool or network management system was in such a condition that operations were not possible before restarting the applications. In the NMS integration testing this same procedure was tried manually with no problems. From this it was derived that there were some problems in NMS and QTP co-operation in extended use. Similar results of jammed windows were found with other scripts which tried to execute the same operations several times. From the NMS point of view, it is not so important to test that whether some GUI dialog can be used for very long periods or repeat the same procedure many times. Rather it is more important to see that the NMS servers are stable and continuously ready for operations. For these reasons, it was decided that the GUI-based QTP would not be suitable for this soak testing environment and some other methods would be needed.

7.3.2 Operations with NMS macros

Since the QTP based approach was not successful, we quickly decided that we will try to create operations with the macro functionality of the network management system. Macro commands and scripts can be run from the command line shell with the Macro Manager - Command Line Interface, called mmcc. Macros can be easily created by recording the operations that user does through the GUI of NMS. This recorded macro can be later edited with some text editor. Macro scripts have the benefit that they are independent of the GUI of NMS and thus do not require changes even though NMS GUI changes. This is highly advantageous from the maintainability point of view. Furthermore, the execution of operations is much faster with macros than would be possible if GUI was used. This enables us to do more operations per period.

The figure 7.4 below shows the structure of the macro based operations. The *MacroInterpreter* at the highest level in the figure is a Java program that launches the macro execution. Macros itself have a two level hierarchy. At the lowest level there are common macros, which include macros that carry out elementary operations needed in many places. These are macros that are repeatedly needed and with parameterization can be used from many higher level macros. In the figure there are two example common macros, called *DisconnectVPN* and *RemoveConfigurationMplsvpn*. As the name implies, the first one disconnects one pseudo wire and the second removes the configuration from the database. The name of the pseudo wire that is operated is given as a parameter. By compiling these common macros appropriately and calling with suitable parameters one can accomplish one higher level operation to the network. The macro package in the figure represents this kind of collection of calls to common macros.

For every operation there is a separate higher level macro script file, which will be called when the operation is wanted to be performed. In the figure 7.4, there is a simple example loop that first disconnects and then removes the configuration from database for nine TDM pseudo wires. Common macros reside in a separate file which is imported into the higher level macro package. Currently all common macros are in the same file,

but as the system gets larger those can be divided to several files for maintainability purposes.

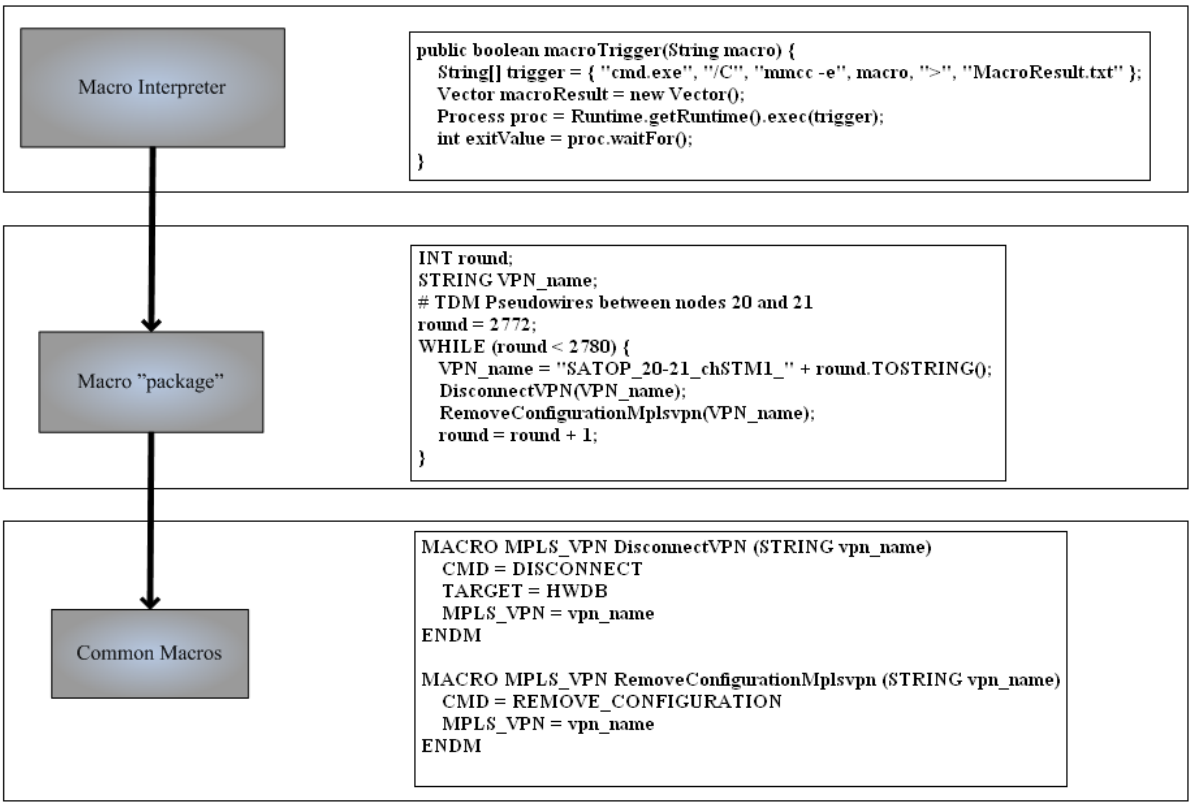


Figure 7.4: Structure of the macro execution.

As mentioned, the high level macro is called by *MacroInterpreter* Java class. This Java code simply launches macros by calling macro manager from CLI. In the figure there is only a small example piece of code from the macro trigger method. The part that is shown executes an external application, macro manager in this case, via command line shell. The name of the macro file that will be executed is given as parameter when macro manager is started. During the execution of the macro script the Macro Manager prints information about the progress of the execution. This output of macro execution is redirected to a file, the name of which is also specified when macro manager is called. In the figure 7.4 shown above, this output file is named *MacroResult.txt*. After starting the macro manager, the Java program waits while the macro manager has finished executing the program before checking the exit value of macro manager. If the

exit value is something else than zero, meaning that the macro manager terminated normally, the file where the macro manager output was redirected will be checked next. If the execution of this macro program was successful, the last line of the result file should be “Execution completed successfully”. On the other hand, in case of some problems the last line is “Execution completed”. Boolean variable describing the success of macro execution is returned by the *MacroInterpreter* object’s *macroTrigger* method to the calling object. The structure and the used files for creating an example operation with macros are presented in Appendix A.

The described macro procedure enables easy addition of new events to the network. Only thing that is needed is to compile new events by combining common macros in an appropriate way so that this event is accomplished and just invoke this higher level macro script from the main program.

Some of the operations that are implemented with macros at this first stage are disconnecting and then connecting some of the ATM, Ethernet, and TDM pseudo wires, and reducing the number of members in the MLPPP and IMA groups and then putting them back. We are also breaking links by putting those in the shutdown state with the macros. In this way we can repeatedly test RSVP path protection and force OSPF route calculation in all nodes, for example. All of the operations are not, at least not yet, supported by the macros. Among these operations are switching of the protection side of the MSP 1+1 protection group and the CDC protection switching. To be able to perform also such operations that do not have macro support some new methods are needed.

7.3.3 Operations with telnet connection and CLI

Since there are no macro support for every operation some other ways for implementing these was investigated. Furthermore, we realized that we will need some methods for reading information directly from the network element. For example, the change in dynamic memory consumption during tests is one measure that we are interested in. After a quick study, it was decided that these would be implemented by establishing a Telnet connection to the node and sending CLI commands through this connection. As

in the case of macros, Java was used as the programming language. There was no need to implement our own Telnet client since free clients were available from various websites. We decided to use Apache commons project's *Net* package, which implements the client side of various basic Internet protocols, including Telnet. This JAR file can be freely downloaded from the Apache website (http://commons.apache.org/net/). The Commons is an Apache project intended on creating all kinds of reusable Java components.

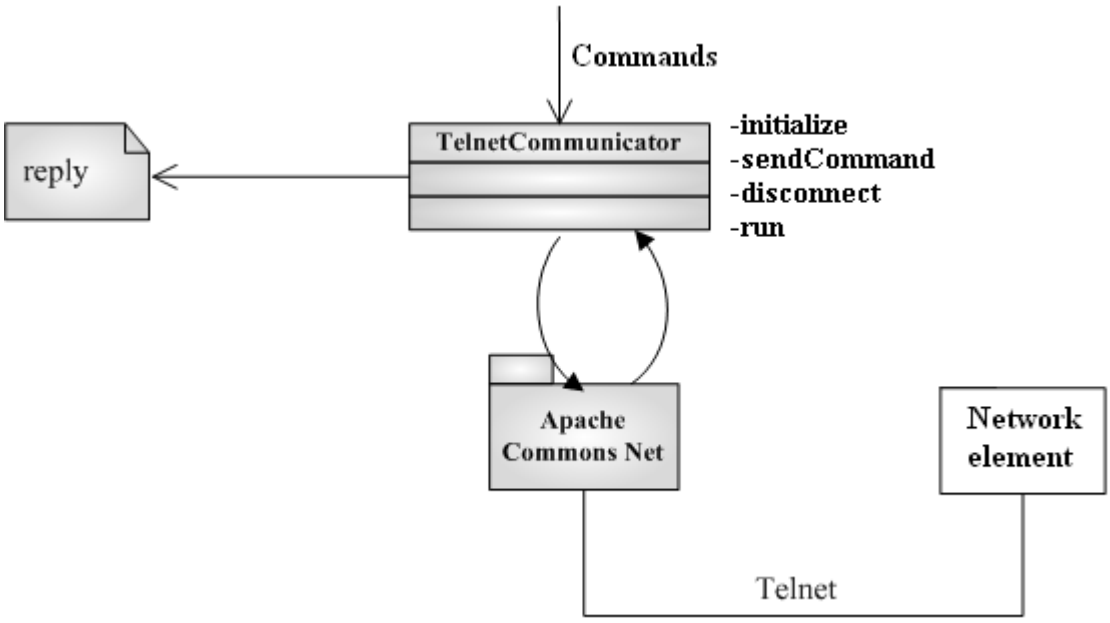


Figure 7.5: Structure of the telnet based operations

Figure 7.5 above shows the structure of the telnet based operations, where the *TelnetCommunicator* class has a central role. There are four different methods in the *TelnetCommunicator*; *initialize*, *sendCommand*, *disconnect*, and *run*. *Initialize* simply calls the *connect* method of *Net* package and gives the address of the network element and the TCP port as a parameter. *Net* package then takes care of establishing a telnet connection with the Tellabs 8600 network element. *Initialize* method also opens the streams for sending commands to the node and for writing the reply of the element to a text file. Finally, the *initialize* method starts a reader thread. *TelnetCommunicator* implements the *Runnable* interface and thus contains the *run* method. This separate thread is used to just read the data that comes in from the network element and prints it

to a text file, which is named reply in the figure 7.5. As long as there is still data coming from telnet connection, the thread loops to read 1024 bytes at a time and writes it to the reply file. This text file can be later used to check what was the response of the node to some commands or queries. The name of this text file, in addition to the node address, is given as a parameter when the connection is established. Thus, we can print the replies from separate connections to individual files. *SendCommand* method sends the command to the network element by simply printing and flushing the command given as a parameter to the output *PrintStream*. After the commands have been sent and we do not want to read anymore replies of the network element, the connection is ended with a call to the *disconnect* method.

The operations that are implemented with telnet in this first stage are reloading the active CDC and the line cards participating in the MSP 1+1 protection group. The CDCs are realoaded alternately so that the one that is currenlty active is reloaded. This causes the activity to change to the other CDC. The reloading is done similarly with line cards participating in the MSP 1+1 protection group. The card that contains the currently active interface is reloaded. The activity thus changes to the other interface and data should flow through that interface without interrupts. Additionally, the telnet connection allows us to read various information from the network elements, such as the dynamic memory consumption of line cards.

7.3.4 NMS operations

In addition to the above mentioned macro and CLI based operations also some NMS operations were configured to the network. One PC was configured to be a monitoring workstation where several monitoring applications are open and monitoring the state of the network. The SNMP agent was enabled on every node allowing them to be polled by NMS for performance data with SNMP queries. Real-time performance management windows are open on the monitoring workstation and are continuously polling the nodes for performance data. History data is also collected on 15 minute intervals to the database.

Tellabs 8000 Manager Packet loop test (PLT) application can be used to schedule various tests to the network on regular intervals. These tests can measure throughput, delay, and packet loss on various links, PWs, IP VPNs, and TE tunnels, for example. We configured the packet loop application to run throughput tests, and one-way delay tests through various links on 15 and 60 minute intervals. The result of all of these tests can be later verified from the PLT. From NMS point of view, we are more interested in whether these tests are actually executed continuously rather than the results of those tests. Furthermore, in this way we can easily introduce more traffic load to the network as the tests generate even 1Gbit/s on various links. Also the fault management application is open and is manually checked that it collects faults from operations that are done in the network. Tellabs 8000 Manager is also configured to take configuration snapshots from all of the network elements on one hour intervals. These snapshots can be used to restore the settings to the network element in case of some problematic situation. The Manager requests the node to create the snapshot and uploads this to the database automatically. The audit log application logs every configuration given to the node, whether it is via CLI or through BMI interface with BMP protocol. It polls the commands from the network element and saves the results to the database. The monitoring workstation and all other NMS operations are manually checked for failures.

7.4 Detecting failures

Soak testing requires automated methods to detect various failures in the environment. After a quick study, we decided that at first we would check after every operation whether data is still running properly through the network or not. There are lots of data streams in the network and a wide variety of problems will be noticed by seeing that data is not going through some streams. We realized that the easiest way of accomplishing this goal was to read various statistics related to the stream from the measurement equipment that is used to generate traffic to the network. It was decided that after every operation the packet loss of the streams affected by the operation would be verified. We also wanted to follow the change in average latency of the streams before and after every operation. The latency can change due to changed paths or

congestion situations, but should come back to the original level after the operation is finished.

The measurement equipment used in the system testing at the moment is Agilent N2X, which is also used in the soak testing environment. Agilent N2X is a multiservice test solution for testing network equipments for voice, video and data services. The systems traffic generation and analysis capabilities include technologies such as IP, MPLS, FR, ATM, Ethernet and SONET/SDH. Agilent N2X has a Tcl API with predefined Tcl commands to gain access to all of the tester's capabilities. Commands are entered via standard input or file and are interpreted by the Tcl or Wish shell, also called API client. This API client can be located in the same machine as the tester or in a remote computer. The client communicates with the tester through TCP connection.

The Tcl interface provided a suitable interface for automating the statistics reading from the Agilent N2X. Agilent organizes the traffic into streams and stream groups. Streams represent PDUs (Protocol Data Units) having the same stream ID, which is used to measure per-stream statistics. One stream sends traffic to one pseudo wire configured in the environment. Streams are aggregated into stream groups which all have a common PDU template and PDU length distribution. Thus one stream group can send data to many, for example ATM PWs, if the ATM VPs/VCs are selected appropriately.

To automate the data stream checking, we first created a Tcl script which reads statistic remotely from the Agilent N2X and saves these statistics into a file in the local computer. This script fetches statistics for one stream group at a time. The Tcl script fetching the statistics takes the name of the stream group and a save file name as parameters when it is called. The statistics retrieved from measurement equipment are cumulative statistics from the start of the measurement. Thus, to get the instantaneous statistics the streams have to be sampled twice. The instantaneous value is then obtained by dividing the difference of these two samples by the time between the samples. The Tcl script simply writes both samples to the text file and *StatisticsReader* Java class is used to handle all the post processing of the retrieved information. The structure of

automated statistics retrieving and checking is shown in Figure 7.6 below. Appendix B presents the Tcl script for fetching the stream statistics.

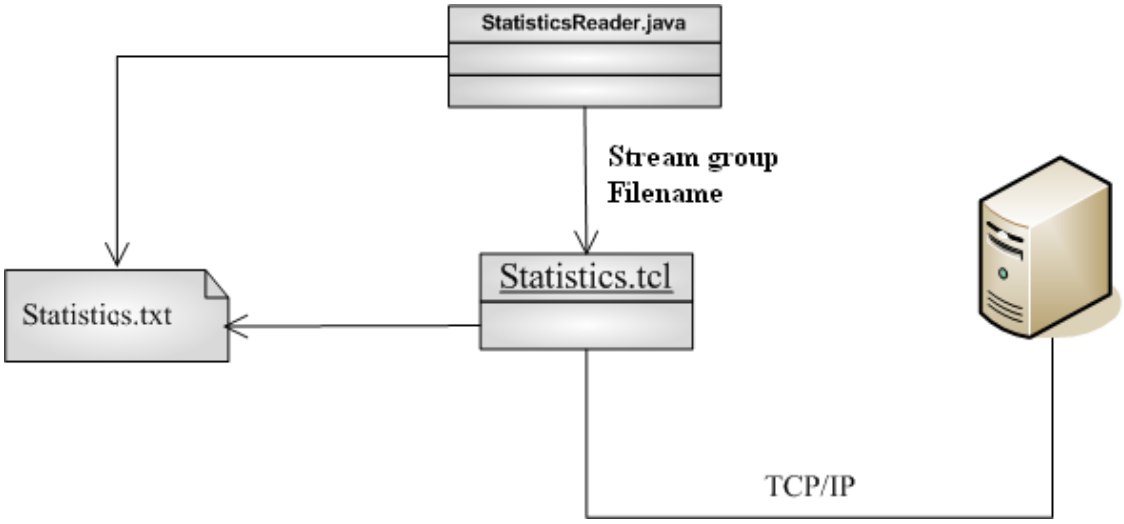


Figure 7.6: Detecting failures by fetching stream statistics from Agilent N2X

The *StatisticsReader* Java class runs the Tcl script through Tcl shell and also takes care of the further processing of the retrieved statistics data. As mentioned, the Tcl script saves the statistics data into a text file on the local computer. *StatisticsReader* opens the text file containing the stream group data and calculates packet loss or average latency to every stream in the stream group. Packet loss can be directly calculated by dividing the difference of packet loss of two samples by the sampling period. If the value is greater than a certain tolerance value, it is derived that there are problems with the stream. This tolerance is needed since the packets might still be in the buffer of Agilent N2X and this is seen as a packet loss. This same approach is not applicable to average latency. Average latency of streams is calculated with the following formula.

$$Latency = \frac{LatencySumB - LatencySumA}{\mathrm{Re}\,ceivedPacketsB - \mathrm{Re}\,ceivedPacketsA} \qquad (1)$$

Latency sum is the sum of latencies of all packets on a particular stream since the start of the test. Difference of these latencies is divided by the sum of received packets between sampling intervals. So we need to get two separate statistics per sampling

moment to calculate the average latency. There are methods in the *StatisticsReader* Java class for verifying the packet loss and latency of the streams.

The described method checks for problems by verifying that data is still running properly through the network. Additionally, the same method can be used to configure the measurement equipment automatically during test. This is accomplished simply by creating a new Tcl script and calling this script instead of the statistics script. From the operator's point of view, the most important thing is that data is still running correctly and there are no revenue affecting problems. However, this method does not necessarily mean that the network is free of problems since something that disturbs the traffic can happen between the verification instants. For example, a line card or whole network element can boot between sampling moments and this can go unnoticed or there may be some deadlock in the node preventing further configuration. These kinds of spontaneous resets and deadlocks are detected by manual checks at the moment, but should also be automated in the future.

It is also important to detect failures in various NMS server processes, which are running in the management and communications servers. Automatic error detection is currently based on Dr. Watson, which is a program error debugger provided by Windows. When an error occurs in any one of those processes, Dr. Watson creates a text and binary log files that contains information about the computer at the moment the error appeared. A manual check is made every day to see if those log files have been created on certain folders. The detection of defects in various NMS features that are running in the background is not automated yet. Performance management application, packet loop tester, configurations audit log, and fault management application are all manually checked from the monitoring workstation whether they are functioning correctly.

7.5 Implementing logging

One important part of this kind of automated testing is the logging of various operations done in the network. We need to be able to see back what was going on in the network

at a certain time. All operations that are done, including the time of operation, are simply written to a text file. Both the starting time and if the execution was successful also the end time are logged to this file. Actions related to statistics are logged into a separate file. This file also includes the time and date alongside with the name of the stream group that the operation is applied to. In case there are problems with some of the streams, the name of the stream group and the id of the stream are logged to the file. If some problems are detected, for example by night, the log files help to determine which operations were done at the time the problems appeared. Comparing the time when problems first appeared in the statistics files with the operation done in the operation file helps to determine the cause of the problems. Additionally, we keep counters which record the current number of operations done since the start of the test. There is a separate counter for every operation type and these are updated after every operation.

The dynamic memory consumption of network elements is probed on a regular basis and is also logged to a separate file. At the moment, this reading is done by one hour intervals to determine if the reservation of dynamic memory grows by time. Later it could be enhanced so that the consumption of dynamic memory is read always before and after operations.

7.6 Integrating the pieces together

7.6.1 Overall architecture

Operation execution and failure detection were tied together to create a program, structure of which is shown in the figure 7.7. All other parts, except the Tcl scripts used to access measurement equipment and macros used to implement some of the operations, are made with Java. The operations are described with text files that are put into a certain folder designated for these operation files only. The *Agent*, which interprets the files and takes care of executing the operations, is a central component in the overall architecture.

The test execution proceeds as follows:

1. *Scheduler* creates a text file describing the operation into a folder.
2. *Agent* picks the file from the folder and interprets it.
3. *Agent* executes the operation either through telnet connection or with macro manager.
4. *Agent* checks the stream statistics to verify whether data is still running through the network. In addition to verifying streams also other methods can be used to detect failures.
5. The executed operation is written into a log file before picking the next operation.

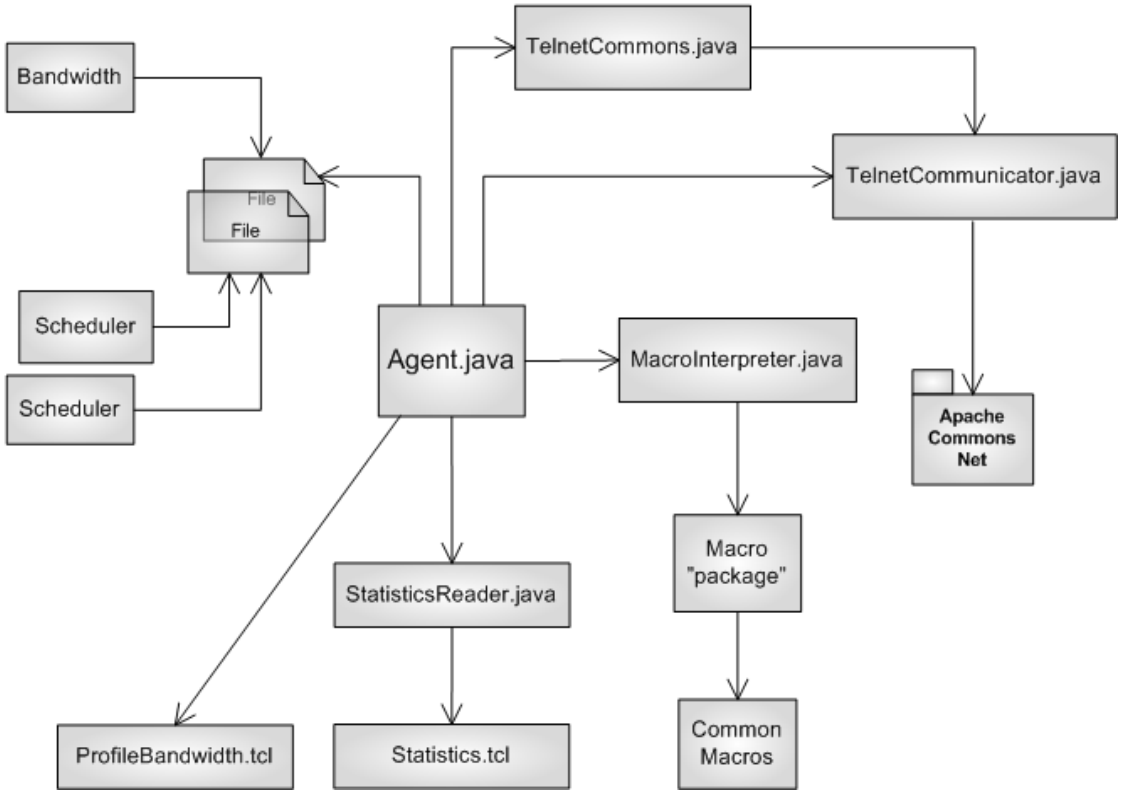


Figure 7.7: Overall architecture of the system

As mentioned, the main component in tying various parts together is the *Agent* class. The *Agent* checks every 5 seconds whether there are files in the folder and performs them when one exists. From the name of the file, it knows whether it is a telnet based

operation or macro based operation and thus how to interpret the information in the file. If it finds a file from the folder it executes the operation via telnet connection or with macro manager, as described in subsections 7.3.2 and 7.3.3. After the operation is completed the *Agent* can instruct the *StatisticsReader* class to start fetching the statistics for one stream group at a time. The text file contains the names of the stream groups that need to be checked. As mentioned in section 7.5 these statistics are retrieved by the Tcl script and are saved into a file in the local computer. Hereafter, the *Agent* calls *StatisticsReader* to check packet loss or the change in average latency of the stream. The name of the file containing these statistics is given as a parameter. If there are no problems with the streams, the *Agent* removes the file from the folder and starts searching for a new file. On the other hand, if there are problems with the streams, the *Agent* stops executing the operations. It prints to the CLI that problems are detected and checks the erroneous streams every 2 minutes. If at some point these streams are functioning correctly, *Agent* continues the execution as described in the file. In addition to verifying the statistics, the *Agent* can use a telnet connection to verify the state of the element after the operation. By sending a command to the element and checking the answer it can determine whether it is allowed to continue or not. This and other frequently used common methods related to telnet connection are implemented in the *TelnetCommons* class shown in Figure 7.7.

7.6.2 Operation description

Whenever some operation needs to be performed in the network, a text file with a specific format and name is put into the folder. A separate text file is created for every operation. All operations that are accomplished through telnet connection are described with a text file of the same structure. Similarly, all macro operations have the same structure which, however, is a bit different from telnet operations. As the figure shows, there are several *schedulers* which create files to a certain folder on regular intervals. These files describe operations to the *Agent*, which then performs them. *Schedulers* are implemented with timers and the interval how often these files are put is dependent on the operation. One *scheduler* is currently creating the files for the telnet based operations, one creates files for VPN operations, and one creates IMA and MLPPP operations. Naturally, one option would have been to combine everything into one big

scheduler that creates all the files for all the operations. The reason for dividing operations to several *schedulers* is that now we can stop some operations while others are still being generated and executed. Another option would have been to use a separate file, which would give instructions to the big *scheduler* about operations that are being executed.

The figure 7.8 below shows the format of the file for telnet based operations. All of the fields are on separate line. When *Agent* interprets this file, it can read one line at a time. Most of the fields are self-evident, but some require further explaining. The statistics files are the names of the files where Tcl script will save the statistics retrieved from the Agilent N2X. The reason for two statistics files is that it allows us to do comparison of situation before and after operations, as is the case when we want to monitor the change in average latency. It is also possible to execute operations without retrieving any statistics. The verification operation means what kind of method we will use to detect failures. Currently there are three alternatives. The packet loss and average latency of streams can be monitored, as explained in section 7.5. The third alternative is to send some command to the node and wait for a specific reply to verify that we can proceed. This can be used to verify that a unit comes up after reloading it or making sure that connections are working by pinging from one node to various destinations. There are no restrictions on the amount of verification operations or on their order of execution. Files for macro based operations are quite similar. The most notable difference is that instead of commands send to the node, the file includes the names of the macro files that will be executed.

Field	Purpose of the field	Example
Description	Description about the operation for logging purposes	Reloading CDC 14
Address of the node	IP address of the node where the commands will be sent	172.19.169.20
Statistics file 1	Name of the file where statistics will be retrieved after the operation	AfterCDC.txt
Statistics file 2	Name of the file where statistics will be retrieved before the operation	No_stats
Nbr commands	Integer value telling how many commands will be sent to the node	2
Commands	Actual commands that will be sent to the node one by one	reload-sw slot 14 y
Wait	Amount of time in milliseconds before proceeding	300000
Verification operation	The operation that will be used to detect potential problems	Packet loss
Stream group	The name of the stream group that will be checked	VP 100-199

Figure 7.8: Structure of the operation file for telnet based operations

7.6.3 Additional considerations

The structure described above executes the operations in a sequential order, meaning that two operations cannot be executing simultaneously. When execution and verification of one operation is ready, the *Agent* searches for the next operation. This is important since there are operations that are not allowed to be performed at the same time. For example, when the CDC is switched it is not allowed to configure the element at the same time. Another benefit of the structure shown above is that operations are not executed in the same order. Operations are created on different intervals and thus the interleaving of operations changes constantly. Actually, the described approach is not totally true, since there are two separate threads that pick files and make operations based on them. The main thread picks and executes most of the operations and is stopped when faults are detected. The second thread executes only such operations that we want to perform always, even though main operations are stopped due to some sort

of failure. Examples of such operations are updating the transmit throughput of Agilent N2X and reading of dynamic memory consumption of network elements.

We also wanted to create different bandwidth profiles to the links. For this reason, we created a Tcl script that updates the transmitted throughput of a stream group on the Agilent N2X. In the figure 7.7 this Tcl script is named *ProfileBandwidth.tcl*. The special scheduler called *Bandwidth* calculates the new transmitted throughput every 2 minutes for all stream groups that we want to update. Again the text file is used to convey this information to the *Agent*. This text file has a different format than telnet or macro based operations. The text file simply includes the names of the stream groups followed by the new transmitted throughputs. This Tcl script updates the bandwidths of certain data streams so that we achieve traffic profiles where the bandwidth utilization follows a daily cycle. On nights the transmitted amount is much lower than on days. A sine function was used as basis for the traffic profile and certain noise component was added to this. Additionally, the bandwidth can slowly increase or decrease. This emulates the common traffic profile found in the real operating environments. Below is the algorithm which calculates the transmitted throughput.

$$bandwidth = A + B\sin(\frac{\pi}{2} * (360 - (\frac{i * MINUTES}{360}))) + j * \frac{D * MINUTES}{1440} + noise \quad (2)$$

A is a base bandwidth to which a cyclic part, calculated with the help of sine function, is added. B determines the min and max values of the cyclic part, which makes full cycle in a day. Constant MINUTES is the interval how often this new transmitted throughput is calculated. The middle part of the formula allows us to set a small increasing or decreasing component to the calculation. Constant D is the daily increase or decrease of the bandwidth and thus determines how fast this change happens. The noise component is created by taking a random number between values 0 and 1 with the random number generator and then scaling this to appropriate value range.

7.7 Deployment of Soak Testing

When the system was taken into use, we realized that the system occasionally stopped execution of operations due to false alarms. We quickly noticed that the reason for these stoppages was false alarms introduced by the delay measurement. It was rather difficult to set suitable tolerance to the variation of latency before and after operations. For these reason, we decided that the latency check would be removed from tests until it is enhanced.

One difficulty in this kind of testing is that when defects are discovered and corrected, it takes a long time before we can evaluate the fix. It might be the case that this fix comes in a newer software version but we do not want to stop testing with the previous version and change to another one. It is not preferable to change to different software in the middle of tests, since this requires that the node is booted and essentially starts the test again from the start. Additionally, when this fixed version is taken into use, it can take quite a long time before we can decide that the bug is corrected. To ease these problems, we have built another identical soak testing environment. We can now run the same tests simultaneously with two different software versions or we can run tests in one environment and use the other environment for creating and testing new operations that will be taken into use in soak testing.

8 Summary and Conclusion

8.1 Summary of the Thesis

The purpose of this thesis was to design and implement an automated soak testing environment for the Tellabs 8600 Managed Edge system. Tellabs 8600 Managed Edge system is an IP/MPLS based access network solution designed for evolving networks. Soak testing, also known as endurance testing, is a type of testing where the system will be run over a prolonged period of time in order to check the system's stability under sustained use.

This thesis started by reviewing the basics of the underlying IP and MPLS technologies. After this introduction of the basic technologies, an overview of the Tellabs 8600 Managed Edge System itself was given. The essential system technologies and main applications of the system were discussed. From here we proceeded to presenting the Tellabs 8000 Manager, which is used to manage 8600 network elements. Additionally, some access network technologies and services were presented. Understanding of these technologies is important, since all of those are also present in the implemented soak testing environment. The objectives of testing in general and soak testing were also examined before introducing the implemented soak testing environment.

The implementation started by examining the objectives for the soak testing environment. These objectives were, in addition to building the actual network, to be able to automatically produce operations to the network, automatically detect failures, and implement logging so that everything could be traced back. After the network was built, services configured, and the measurement equipment configured to send data, it was time to implement operations to the network. Initially, the QTP test tool was considered as a method to produce these operations. This tool was already in use in regression testing and thus the test libraries would have already been ready to be utilized also in soak testing. However, it was quickly discovered that there were some problems in cooperation of QTP and NMS when used for longer periods. Therefore,

macro based operations were examined next. A hierarchical structure for macro operation was implemented so that common macros, which include repeatedly used parts, are used by higher level macros. Since there was no macro support for all operations, some of the operations were implemented with CLI commands through telnet connection. There was no need to implement an own telnet client since one was found from Apaches common project. Operations are implemented by creating a file containing the description of the operation to a certain folder. This file is then interpreted by the *Agent*. Additionally, some NMS operations are continuously running in the background. Important part of any automated testing is the ability to automatically detect failures. It was decided that initially data streams would be checked to ensure the state of the network. For this reason, the automation of measurement equipment was examined. The Tcl interface offered a convenient way to automatically read data and configure the Agilent N2X. Logging of operations was simply implemented by writing all operations including the time into various text files. Software was coded with Java programming language to integrate these various parts together.

8.2 Conclusions

The implemented soak testing environment has been successfully taken into use in the system testing of Tellabs 8600 system. It has been turned out to be an effective environment and seems to satisfy the initial requirements. In the beginning of this thesis in section 1.2, we mentioned that the purpose of this soak testing environment is to find defects that do not show up immediately in functional or regression testing but could manifest themselves when the system is used for a long time. All defects found by soak testing are from such features that are also tested by functional or regression testing but have not showed up in these tests where the testing period is shorter or tests are executed only a few times. Many of the defects that are found by soak testing environment have serious effects on the services. For example, these defects can cause data to drop from several pseudo wires or cause the whole network element to crash. Thus, it can be determined that there is certainly need for this kind of testing and the implemented soak testing adds value to the current testing of Tellabs 8600 system.

It is typical for this environment, that when one defect is found, this same defect keeps showing up until it is fixed or the operation that causes it is removed. If nothing changes, the test usually stops at some point due to the same problem. Thus, new defects are usually found by gradual improvements of the system, which allows us to execute longer test runs.

The defects that are found by soak testing can be roughly divided into three separate categories. The first category includes defects that do not necessarily require that operations are executed many times but manifest themselves with some, possibly small, probability. These defects happen usually due to the fact that the execution of two independent operations occurs simultaneously. This might lead to mutex deadlocks, for example. The probability that these operations occur simultaneously in normal environment is rather small. When operations are executed continuously for a long time, the probability of detecting these defects is increased. An example of this kind of simultaneous activity was the occurrence of node configuration snapshot, which is initiated by NMS, at the same time with emergency snapshot, which is initiated by network element. The second category includes resource usage problems that manifest themselves when a certain operation is executed a sufficient number of times. When we reloaded the active CDC alternately, at some point a certain log file containing information about the switchover was filled up. In the worst case, this could cause the network element to reboot. In addition to these two types of defects, the environment has also been able to find various functional defects. These are defects that could have also been found by functional testing.

We have roughly categorized the 15 defects that were found during past few months. About seven of them are caused by some simultaneous operations, five are due to resource usage problems, and three are functional defects. The number of these found defects would probably be much higher, if we could fix the already known bugs faster. As these figures show, the system is quite good in revealing defects that happen because of simultaneous operations. It can be expected that when more functionality is added to the system, more defects of this type will be revealed.

An effective framework for soak testing was created in this thesis, against which it is easy to start developing the environment further. Adding new operations is rather easy since the minimum that is needed is to add a new *scheduler* for writing the operation file. On the other hand, the fact that we made the testing software by ourselves and did not use any ready commercial software causes extra maintenance. To be able to make changes to the system and debug in case of problems one needs to have a good understanding of the overall idea and architecture of the whole testing system. There are many separate components in the overall testing system, which can complicate ones comprehension of the system. For these reasons, a good documentation is needed.

The fact that GUI-based operations are not done automatically has both benefits and drawbacks. The drawback is naturally that the GUI part is not tested in soak testing. However, this has not been seen as a serious omission since, as already mentioned in subsection 7.3.1, it is more important that the NMS servers are continuously up and running. The benefit from omitting GUI-based operations is that soak testing is not dependent on any changes made in the NMS GUI. From maintainability point of view, macro scripts are more advantageous since GUI changes do not affect macros and macro commands are modified rather seldom.

8.3 Further Research

One important aspect in the future development of the environment is the expanding of the coverage of soak testing. In the future, more operations should be added to the soak testing environment so that more features will be tested. It should be investigated which operations are most worthwhile to take into use in soak testing. Experience gained from functional and regression testing and the easiness of implementation help to decide which operations will be implemented. Also more different types of services and more network elements should be added. It has been discussed that adding more 8605 network elements to the environment is needed. This requires that we circulate traffic streams conveniently in the network.

Another way to expand the coverage would be to use several separate executing objects. This means that there would be more than one *Agent*, which picks and performs the operations. At the moment, only one *Agent* is executing operations one by one in sequential order. Using multiple execution objects might help to find problems quicker since it enables performing more operations per time. The problem with multiple *Agents* is the need to coordinate the execution so that two instances are not making conflicting configurations to the same network element at the same time. Implementing several executing objects requires also that we investigate how to carry out failure detection as one object can cause problems to data streams verified by another object at the same time.

Second major objective of the future research would be the enhancement of reporting. This objective can be further divided into two separate aspects. The first one is to improve reporting so that developers can find and correct defects easier and quicker. This means that we need to give more exact information about what was going on in the network at the moment when problems manifested themselves at the first time. Second aspect of reporting is the ability to report the results of soak testing. This would give managers and other personnel an overview of the general state of the software so that resources could be directed to places where most needed. One way to improve this aspect would be to create web pages where the duration of soak testing is automatically updated and could be seen graphically. Additionally, these web pages could show the amount of certain operations performed. Furthermore, these pages would make it easy to compare results of various test runs with separate software versions. This kind of approach is already in use in integration testing of Tellabs 8000 Network Manager, where the results of various tests are automatically updated to the web pages. To build this kind of web pages only for soak testing might be too laborious, but it should be examined if already existing methods from integration testing could be re-used.

Third place to look for improvements is the automatic verification part. In a way this can be also seen as part of expanding the coverage by allowing us to detect new kinds of defects, but is nevertheless considered here as a separate question. Firstly, we are not

currently sending data to TDM PWs and thus we are not able to monitor the state of those PWs with current mechanisms. To be able to detect failures also in TDM PWs, the environment should be able to send data to these PWs also. Part of the verifications is currently done manually, and thus more mechanisms to automatically detect failures in the system should be examined. At the moment, the checking is done only after some operation is executed and thus it is possible that the network element resets itself in between the operations and this goes unnoticed. This kind of spontaneous resets of the whole system or line cards are currently noticed by manual checks. There can also be some smaller problems that cause temporary interrupts in the data flow. If these interrupts occur at some point between the sampling moments, we are not able to detect those with the current system. We have already started to utilize the logging of data streams in the Agilent N2X and it should be further investigated if this enables us to detect also this kind of temporary data cuts. Additionally, the automation of failure detection in NMS functionality should be investigated. When adding more failure detection mechanisms, it should be kept in mind that false alarms do not interrupt the test execution.

References

[1] Andersson, L., Minei, I. and Thomas, B., *LDP Specification*, IETF RFC 5036, 2007, http://www.ietf.org/rfc/rfc5036.txt.

[2] Awduche, D., Berger, L., Gan, D., Li, T., Srinivasan, V., Swallow, G., *RSVP-TE: Extensions to RSVP for LSP Tunnels*, IETF RFC 3209, 2001, http://www.ietf.org/rfc/rfc3209.txt.

[3] Black, U., *Network Management Standards*, 2nd ed., McGraw-Hill, New York, 1994.

[4] Bryant, S. and Pate, P., *Pseudo Wire Emulation Edge-to-Edge (PWE3) Architecture*, IETF RFC 3985, 2005, http://www.ietf.org/rfc/rfc3985.txt.

[5] Burnstein, I., *Practical Software Testing*, Springer, New York, 2003.

[6] Cisco Systems, Inc., *Internetworking Technologies Handbook*, 4th ed. Cisco Press, Indianapolis, 2000.

[7] Clemm, A. *Network Management Fundamentals*, Cisco Press, Indianapolis, 2006.

[8] Collard, R., *SYSTEM PERFORMANCE TESTING, Case Study*, New York, 2005.

[9] Comer, D. E., *Internetworking with TCP/IP – Volume I: Principles Protocols, and Architecture*, 5th ed., Prentice Hall, New Jersey, 2005.

[10] De Ghein, L., *MPLS Fundamentals*, Cisco Press, Indianapolis, 2007.

[11] Fewster, M., *Common Mistakes in Test Automation*, Presentation at Software Test Automation Conference, March 5-8, San Jose, 2001.

[12] Haikala, I. and Märijärvi, J., *Ohjelmistotuotanto*. Suomen Atk-kustannus Oy, Helsinki, 1998.

[13] Halabi, S., *Internet Routing Architectures,* 2nd ed., Cisco Press, Indianapolis, 2001.

[14] Hewlett-Packard Development Company, *HP QuickTest Professional software*, Data Sheet, 2008.

[15] Huitema, C., *Routing in the Internet*. 2nd ed., Prentice Hall, New Jersey, 1999.

[16] Hung-Yi Chang, Pi-Chung Wang; Chia-Tai Chan; Chun-Liang Lee, *A New Service Level Agreement Model for Best-Effort Traffics in IP over WDM,* AINAW Conference, March 25-28, Okinawa, 2008.

[17] IEEE Standards Board, *IEEE Standard Glossary of Software Engineering Terminology*, IEEE Std 610.12-1990, New York, 1990.

[18] ITU-T G.841, *Types and characteristics of SDH network protection architectures*, Telecommunication Union - Telecommunication Standardization Sector Recommendation, 1998.

[19] ITU-T M.3400, *TMN management functions,* International Telecommunication Union - Telecommunication Standardization Sector Recommendation, 2000.

[20] ITU-T Y.1720, *Protection switching for MPLS networks,* International Telecommunication Union - Telecommunication Standardization Sector Recommendation, 2006.

[21] Juniper Networks, Inc., *Multiprotocol Label Switching - Enhancing Routing in the New Public Network*, White Paper, 2000, http://www.juniper.net/solutions/literature/white_papers/200001.pdf.

[22] Juniper Networks, Inc., *RFC 2547bis: BGP/MPLS VPN Fundamentals*, White Paper, 2001.

[23] Kaner, C., Bach, J. and Pettichord, B., *Lessons Learned in Software Testing*, John Wiley & Sons Inc., New York, 2002.

[24] Kaner, C., Falk, J. and Nguyen, H. Q., *Testing Computer Software*. 2nd ed., John Wiley & Sons Inc., New York, 1999.

[25] Logigear Corporation, *Achieving the Full Potential of Test Automation*, White Paper, 2006, http://www.bleum.com/pdf/Achieving_the_full_potentia_of_test_automation.pdf.

[26] Martin, D. W., 1:1 Protection Switching Overview, Presentation at IEEE 802.1, July 16-19, San Francisco, 2007, http://www.ieee802.org/1/files/public/docs2007/ay-martin-protection-0707-v01.pdf.

[27] Pasqualini, S., Iselt, A., Kirstädter, A. and Frot, A., *MPLS Protection Switching vs. OSPF Rerouting – A Simulative Comparison,* Barcelona, 2004.

[28] Perry, W., *Effective Methods for Software Testing,* 2nd ed., John Wiley & Sons Inc., New York, 2000.

[29] Peterson, L. and Davie, B., *Computer Networks – A Systems Approach*, 3rd ed., Morgan Kaufmann Publishers, San Francisco, 2003.

[30] Postel, J., *Internet Protocol*, IETF RFC 791, 1981, http://www.ietf.org/rfc/rfc0791.txt.

[31] Rekhter, Y., Li, T. and Hares, S., *A Border Gateway Protocol 4 (BGP-4)*, IETF RFC 4271, 2006, http://www.ietf.org/rfc/rfc4271.txt.

[32] Rosen, E. and Rekhter, Y., *BGP/MPLS IP VPNs*, IETF RFC 2547bis, 2004, http://tools.ietf.org/html/draft-ietf-l3vpn-rfc2547bis-03.

[33] Rosen, E., Tappan, D., Fedorkow, G., Rekhter, Y., Farinacci, D., Li, T. and Conta, A., *MPLS Label Stack Encoding*, IETF RFC 3032, 2001, http://tools.ietf.org/html/rfc3032.

[34] Rosen, E., Viswanathan, A. and Callon, R., *Multiprotocol Label Switching Architecture,* IETF RFC 3031, 2001, http://www.ietf.org/rfc/rfc3031.txt.

[35] RPM Solutions Pty Ltd, *Soak Tests,* 2004, http://www.loadtest.com.au/types_of_tests/soak_tests.htm.

[36] Sklower, K., Lloyd, B., McGregor, G., Carr, D. and Coradetti, T., *The PPP Multilink Protocol (MP)*, IETF RFC 1990, 1996, http://tools.ietf.org/html/rfc1990.

[37] Stallings, W., *MPLS*, The Internet Protocol Journal – Volume 4, Number 3, 2001, http://www.cisco.com/web/about/ac123/ac147/archived_issues/ipj_4-3/ipj_4-3.pdf.

[38] Tellabs Oy, *Quality of Service in the Tellabs 8600 Managed Edge System,* White Paper*,* 2006, http://www.tellabs.com/papers/tlab8600qos.pdf.

[39] Tellabs Oy, *Tellabs 8000 Network Manager Provisioning Packages*, Data Sheet, 2006, http://www.tellabs.com/products/8000/tlab8000nmprovpkg.pdf.

[40] Tellabs Oy, *Tellabs 8000 Network Manager R16A System Description,* Internal Tellabs Oy document, 2008.

[41] Tellabs Oy, *Tellabs 8000 Network Manager – A Powerful Network and Service Management Tool,* Data Sheet, 2007, http://www.tellabs.com/products/8000/tlab8000nm.pdf.

[42] Tellabs Oy, *Tellabs 8600 Managed Edge System FP1.1 and FP2.9 MPLS Applications Configuration Guide,* Internal Tellabs Oy document, 2008.

[43] Tellabs Oy, *Tellabs 8600 Managed Edge System FP2.7 VPNs Configuration Guide*, Internal Tellabs Oy document, 2007.

[44] Tellabs Oy, *Tellabs 8600 Managed Edge System Overview*, Tellabs Oy document, 2006, http://www.tellabs.com/products/8000/tlab8600sysoverview.pdf.

[45] Tellabs Oy, *Tellabs 8600 Managed Edge System training material,* Internal Tellabs Oy document, 2007.

[46] Tellabs Oy, *Tellabs 8600 Managed Edge System – Reliability, Protection and QoS in Mesh IP/MPLS Networks,* Application note, 2007, http://www.tellabs.com/products/8000/tlab8600-mesh_an.pdf.

[47] Tellabs Oy, *Tellabs 8630 Access Switch*, Data Sheet, 2007, http://www.tellabs.com/products/8000/tlab8630as.pdf.

[48] Tellabs Oy, *Tellabs 8660 Edge Switch,* Data Sheet, 2007, http://www.tellabs.com/products/8000/tlab8660es.pdf.

[49] Tellabs Oy, *The Business Case for Pseudowires in the RAN,* White Paper, 2007, http://www.tellabs.com/papers/tlabpseudowireran.pdf.

[50] The ATM Forum Technical Committee, *Inverse Multiplexing for ATM (IMA) Specification Version 1.1*, AF-PHY-0086.001, 1999.

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

Appendix A: Structure of Operation Execution with NMS Macros

Java method that executes the macros

```
public boolean macroTrigger(String macro) {
    String[] trigger = { "cmd.exe", "/C", "mmcc -e", macro, ">", "temp\\MacroResult.txt" };
    Vector<String> macroResult = new Vector<String>();

     try {
        Process proc = Runtime.getRuntime().exec(trigger);
        int exitValue = proc.waitFor();
        if ( exitValue != 0 )
        {
            System.out.println("Exit value " + exitValue);
            return false;
        }
        String line;
        BufferedReader bR = new BufferedReader(new FileReader("temp\\MacroResult.txt"));
        line = bR.readLine();
        while(line != null) {
            macroResult.addElement(line);
            line = bR.readLine();
        }
    }  catch ( Exception e ) {
        e.printStackTrace();
        return false;
    }

    for ( int i = macroResult.size() - 1; i >= 0; --i ) {
        if ((macroResult.elementAt(i)).equals("Execution completed successfully.") )
            return true;
        if((macroResult.elementAt(i)).equals("Execution completed."))
            return false;                   // There were some problems in the execution
    }

    return true;
}
```

Example higher level macro that connects several TDM pseudo wires

```
IMPORT "common.mac"

MACRO PROGRAM Connect_TDM_VPNs
                    INT round;
                    STRING VPN_name;

                    # TDM Pseudowires between nodes 20 and 21
                    round = 2772;

                    WHILE (round < 2780)
                    {
                                        VPN_name = "SATOP_20-21_chSTM1_" + round.TOSTRING();
```

```
                        CreateConfigurationMplsvpn(VPN_name);
                        ConnectMplsvpn(VPN_name);
                        round = round + 1;
        }

        round = 2809;
        WHILE (round < 2825)
        {
                        VPN_name = "CES_20-21_chSTM1_" + round.TOSTRING();
                        CreateConfigurationMplsvpn(VPN_name);
                        ConnectMplsvpn(VPN_name);
                        round = round + 2;
        }


        ###################################################################
        # TDM Pseudowires between nodes 20 and 22
        round = 1;
        WHILE (round < 3)
        {
                        VPN_name = "SATOP_20-22_" + round.TOSTRING();
                        CreateConfigurationMplsvpn(VPN_name);
                        ConnectMplsvpn(VPN_name);
                        round = round + 1;
        }

        round = 1;
        WHILE (round < 5)
        {
                        VPN_name = "CES_20-22_8605_" + round.TOSTRING();
                        CreateConfigurationMplsvpn(VPN_name);
                        ConnectMplsvpn(VPN_name);
                        round = round + 1;
        }

ENDM
```

Examples of common macros

```
MACRO MPLS_VPN CreateConfigurationMplsvpn (STRING vpn_name)
        CMD = CREATE_CONFIGURATION
        MPLS_VPN = vpn_name
ENDM

MACRO MPLS_VPN ConnectMplsvpn (STRING vpn_name)
        CMD = CONNECT
        TARGET = HWDB
        MPLS_VPN = vpn_name
ENDM
```

Appendix B: Tcl Script Fetching the Statistics from the Agilent N2X

```
# This script retrieves the packet loss and statistics for calculating average latency for all streams in the
# streamgroup
# The name of the stream group needs to be given as a parameter

# The path of the AgtClient.tcl package
cd "C:/soaktest/FINAL/TCL"
source AgtClient.tcl
package require AgtClient
# Set the server hostname here
AgtSetServerHostname fism3000
# The session ID
AgtConnect 24

# Selecting the statistics
set hStatistics [AgtInvoke AgtStatisticsList Add AGT_STATISTICS]
AgtInvoke AgtStatistics SelectStatistics $hStatistics {AGT_STREAM_PACKET_LOSS
AGT_STREAM_LATENCY_SUM AGT_STREAM_PACKETS_RECEIVED}

# Selecting the Streamgroups
set hStreamGrp1 [AgtInvoke AgtStreamGroupList GetHandle [lindex $argv 0]]

AgtInvoke AgtStatistics SelectStreamGroups $hStatistics $hStreamGrp1

# Some wait and then retrieve the statistics. Then wait 4 seconds and retrieve statistics again
after 3000
set hResults1 [AgtInvoke AgtStatistics GetStreamGroupStatistics $hStatistics $hStreamGrp1]
after 4000
set hResults2 [AgtInvoke AgtStatistics GetStreamGroupStatistics $hStatistics $hStreamGrp1]

# the sampling interval
set hInterval1 [lindex $hResults1 0]
# the number of streams
set hNbrStreams1 [lindex $hResults1 1]

# the sampling interval
```

```
set hInterval2 [lindex $hResults2 0]
# the number of streams
set hNbrStreams2 [lindex $hResults2 1]

set hStreams1 [lindex $hResults1 2]
set hStreams2 [lindex $hResults2 2]

# The path of the statistics file
cd "c:/soaktest/FINAL/StatisticsFiles"
if {$argc == 2} {
                    set filename [lindex $argv 1]
} else {
                    set filename "statistics.txt"
}
set fileId [open $filename "w"]

puts $fileId [AgtInvoke AgtStreamGroupList GetName $hStreamGrp1]
puts -nonewline $fileId "$hInterval1 \t\t\t\t$hInterval2\n"
puts -nonewline $fileId "$hNbrStreams1 \t\t\t\t$hNbrStreams2\n"

set i 0
set j 0
# Writing the results into a file: packet loss A, average latency A, packet loss B, average latency B
while {$i < $hNbrStreams1} {
                    puts -nonewline $fileId "[lindex $hStreams1 $j] "
                    set j [expr $j + 1]
                    puts -nonewline $fileId "[lindex $hStreams1 $j] "
                    set j [expr $j + 1]
                    puts -nonewline $fileId "[lindex $hStreams1 $j] \t"
                    puts -nonewline $fileId "[lindex $hStreams2 [expr $j - 2]] "
                    puts -nonewline $fileId "[lindex $hStreams2 [expr $j - 1]] "
                    puts -nonewline $fileId "[lindex $hStreams2 $j] \n"
                    set i [expr $i + 1]
                    set j [expr $j + 1]
}
close $fileId

# Remove the handle. Otherwise you cannot get a handle after the script has been run enough many
times.
AgtInvoke AgtStatisticsList Remove $hStatistics

AgtDisconnect
```