University of Ottawa

A Bridging Methodology for Internet Protocols Standards Development

by

Igor Sales

Thesis

Submitted to the school of Graduate Studies and Research in Partial Fulfilment of the Requirements for the degree of

Master of Computer Science

School of Information Technology and Engineering – S.I.T.E.
Department of Computer Science
Faculty of Engineering
Ottawa, Ontario, Canada
K1N 6N5

September 30, 2001

Abstract

An Internet protocol must follow a certain standard set by the IETF (Internet Engineering Task Force). Internet Standards are plain text documents that are often hard for implementers to interpret. Hence, IETF requires at least two distinct, interoperable implementations before accepting a proposal as an Internet standard. To improve the quality of the IETF standards, we propose a *bridging methodology* which uses a semi-formal and a formal description technique to *bridge* from English to a more precise specification.

Our bridging methodology (denoted USHLTD) uses "UCMs (Use Case Maps) as the requirements source notation (a semi-formal description technique) and SDL (Specification and Description Language, an executable international standard formal description technique) as the design destination notation.

To validate USHLTD, we conduct a case study on a widely deployed Internet routing protocol of great current interest called OSPF (Open Shortest Path First) and a key function, LSA (Link State Advertisement) refreshment. We analyse the results of applying our approach to the case study, and make recommendations for further research and validation.

Acknowledgements

I would like to thank Dr. Robert Probert for his unconditional financial support and for his help.

I would like to give my special thanks to Dr. Ostap Monkewich whose ideas always encouraged me. Special thanks to Mrs. Louise Desrochers whose support and encouragement was a blessing to this work. Special thanks to Rossana Andrade, who once my professor at my home university always trusted in us more than we trusted in ourselves. Roz, um beijão. Agradeço de coração! Also thanks to the colleagues at the Nortel Networks ASERT Laboratories, Alan Williams and Daniel Amyot for their support, To the other friends, Vivi et al too, many thanks. Sem esquecer os colegas de faculdade, João Mendes Jr. e Marcelo "Magoo" Romcy que me deram aquelas palavras fatais: "Só falta a tese ? Então escreve, pô!". My special thanks to Dr. Creto Vidal for all his help in getting me here.

Very special thanks to my parents, Sr. Jerre e D. Lêda, meu irmãozão Ian, e meu irmãozinho Ítrio, que não têm a menor idéia que esta tese não estaria sendo escrita sem a colaboração, compreensão, carinho e atenção dados por eles. Eu amo muito vocês. Not forgetting my girlfriend Elisa, who more than anyone else, was able to put up with my annoyance, and always encouraged me to finish this work. Un beso enorme, mi amor! Vovó também que mesmo distante e não tendo a menor idéia do que eu estou escrevendo, sempre me deu seu apoio. Um beijão, vó!

Last, but not least, to all those who helped me get up to this point: Heather Leighton, the twins Rick and Jen, Leesa, John "B." Helis and Gabriela Henriquez.

Also thanks to Donald Trefusis, Katie Hanks, Xie Yinggong, Lennie Briscoe, Claire Kincaid, Jack McCoy, and Adam Schiff for sharing their adventures with me. And thanks to all the others I have not mentioned, who have no idea of how supportive and necessary they were.

Table of Contents

List of Tables

List of Figures

Glossary of Terms

ATM	Asynchronous Transfer Mode	8
CASE	Computer Aided Software Engineering	16
CG	Construction Guidelines	20
DD	Database Description	40
EFSM	Extended Finite State Machine	19
ETSI	European Telecommunications Standards Institute	7
FSM	Finite State Machine	19
IETF	Internet Engineering Task Force	8
ISO	International Organization for Standardization	7
ITU	International Telecommunication Union	7
LOTOS	Language Of Temporal Ordering Specifications	16
LS	Link State	40
LSA	Link State Advertisement	44
MOST	MOscow Synthesizer Tool	23
MSC	Message Sequence Charts	18
NBMA	Non-Broadcast Multi-Access	48
OSPF	Open Shortest Path First	40
PSP	Personal Software Process	37
RFC	Request For Comments	11
RT-TROOP	Real-Time TRaceable Object-Oriented Process	22
SDL	Specification and Description Language	14
SPEC-VALUE	Specification-Validation Approach with LOTOS and UCMs	21
TCP/IP	Transfer Control Protocol/Internet Protocol	11
TTCN	Tree and Tabular Combined Notation	8
UCM	Use Case Maps	13
UML	Unified Modelling Language	12
UML/RT	Unified Modelling Language for Real-Time	22
URN	User Requirements Notation	10

1. Introduction

1.1. Background

Protocol Engineering [54] is a very complex field stimulated by the ever-increasing demand for Internet-based services. The large number of vendors competing for market share are working to delivery schedules that stress product time to market sometimes at the expense of quality. Reduced quality in protocol specifications and standards leads to reduced product interoperability in a global multi-vendor market. The market place of today can no longer be dominated by a single vendor. To ensure interoperability of products of different manufacturers, it is necessary to release products that are compliant with a common set of specifications or standards to which vendor-specific product differentiation features can be added. To achieve this, it is necessary to develop *Internet protocol standards* that are error-free, precise and unambiguous.

Each vendor's product must comply with a set of standards (sometimes referred to as a *protocol stack*). It is, therefore, necessary to have a basic framework in place for enabling products to communicate properly with one another. In theory this appears very simple, but in practice there are significant difficulties to overcome. These difficulties are discussed in more detail in chapter 2.

Internet Protocol standards and specifications may contain errors and ambiguities that are subject to different interpretations. However, technology is currently available to permit the development of standards and specifications that are precise and unambiguous. Today, Internet drafts and RFC's (Request for Comments) are required to be written in plain text format, so that they can be read with even the most primitive word processor software [10]. Standard Documents published in Geneva, under the ITU (International Telecommunications Union) allow standards to be exchanged in Microsoft Word format [34]. This however, does not address the fundamental problem of ensuring that the Standard is correct. The process of analysing a protocol standard still relies on the human reader to review immense amounts of text with tables and complex "*if…then…else*" statements.

There exist languages, notations and tools that can handle such descriptions with greater accuracy than natural language, plain text, and even Finite State Machines (FSM)

[21]. These technologies may be categorised as *Semi-formal Description Techniques* (SDT or SFDT) and *Formal Description Techniques* (FDT).

Semi-Formal Description Techniques are highly graphical languages that allow the specifier to represent the requirements in a modular, easy to understand, and easy to maintain way. Examples of such languages include UCM's (Use Case Maps) [13], Use Cases [55], and others. They can be drawn with the aid of commercially available word processors or specialised tools such as the Use Case Maps Navigator [40] (for UCM's), or Rational Rose [52] (for Use Cases).

Formal Description Techniques describe the behaviour of protocols using the concepts of *states*, *events* and *transitions*. A *transition* is a set of *actions* that take place when an *event* occurs in a particular *state*. *Formal Description Techniques* are so named because of the formal semantics associated with them. Examples include SDL (Specification and Description Language) [16], and LOTOS [28]. We prefer SDL for its particularly low learning curve, ease of understanding, graphical description properties, level of industrial acceptance, and good industrial-strength tool support. It should be mentioned that SDL is an ITU-T and ISO/IEC standardized Formal Description Technique (FDT) that has formal syntax and semantics. Descriptions in SDL can be compiled using commercial compilers for execution on many platforms. SDL tools include Telelogic TAU [61] and Telelogic (former CS Verilog) ObjectGEODE [60]. LOTOS tools include LOTOS Ceasar [20], and Lotos Laboratory (LOLA) [51] among others.

We define a *bridging methodology* as a process to transform one *source* model into a *destination* model. In our UCM to SDL High-Level To Design (USHLTD) bridging methodology, the *source* model is written in a *semi-formal* description technique (Use Case Maps) and the *destination* model is written in a *formal description technique* (Specification and Description Language, SDL).

Following is the motivation for this thesis work.

1.2. Motivation

The goal of Internet standards development is to create a working framework to facilitate Internet protocols inter-operation. Using this framework, it is necessary to ensure that each protocol works. To achieve this with natural language and available manpower is

not practical or perhaps even possible. We believe that Description Techniques and Tools may assist developers in achieving this goal. However, a single language or formal description technique cannot be used alone. For example, to have an Internet Protocol described in a *Formal Description Technique* is too big a step from pure natural language. In addition, a semi-formal description technique alone would not be adequate, due to its lack of sufficient precision to allow implementation. We believe, therefore, that there is the need for a stage-wise *bridging methodology* to move from a higher-level *semi-formal* "source" notation to a less abstract level *formal* "destination" notation during the Internet standards development process. We believe Use Case Maps (UCMs) [13] and the Specification and Description Language (SDL) [16] are suitable candidates for this thesis work because they are either standardized, or in the process or standardization, they have tool support, and we have not yet encountered a bridging methodology that bridges from UCMs to SDL. We believe UCMs are relatively easy to understand and write, and we believe SDL is a mature enough language for standards protocol development.

The Open Shortest Path First (OSPF) routing protocol [44],[46] is a relatively new Internet routing protocol. This is a typical example of an Internet protocol described in an IETF RFC (Request For Comments). It has a certain deficiency in its routing table refreshment which may cause network congestion [65] because of the timing of periodic floods of Link State Advertisement (LSAs) in the network. We found this a perfect example to apply our USHLTD methodology to demonstrate in a short period of time that this deficiency existed, and to prove that the proposed solution in [65] is also valid.

The purpose of this work is to create a basis for greater synergy between requirements and design specification by bringing together Description Techniques such as Use Case Maps and SDL, Internet Standards and a Bridging Methodology. We show how to use the best features of each of two description techniques, Use Case Maps and SDL (and the associated tools) to:

1) improve the design of Internet protocol standards,
2) shorten standards development time, and

We also show how tedious development cycles can be avoided with the aid of fast development, CASE-tool aided techniques. Expected benefits are:

1) Major *bugs* will be removed from products much earlier than with traditional techniques.
2) The legacy these standards will leave behind will facilitate the understanding for newcomers to the field.
3) There should be a considerable reduction of the time necessary to learn a protocol and the standard under development.
4) Executable simulation models and applications of the Internet standard can be generated quickly and automatically.

We now present the thesis contributions.

1.3. Thesis Contributions

This thesis contributions are:

1) A bridging methodology (USHLTD) for bridging a semi-formal source model (in UCMs) to a formal destination model (in SDL) with respect to the development of Internet protocol standards.
2) A case study showing the use of our bridging methodology and the results of the improvement on an internet-draft based on a well established Internet routing protocol (OSPF).
3) An assessment of the bridging methodology compared to others, together with an assessment of the use of the bridging methodology, recommendations for the IETF standards process and recommendations for the improvement of the USHLTD (UCM to SDL High-Level to Design) bridging methodology.

We now present the outline of this thesis.

1.4. Outline of this Thesis

This thesis is organized as follows.

A discussion on Internet standards procedures and some references to relevant work done in the area of protocol standards and description techniques is given in chapter 2. Also in this chapter there is a discussion about current technologies for the development of protocols using semi-formal and formal description techniques along with their respective tool support.

Chapter 3 explains our USHLTD methodology in more detail. This discussion includes a review of some other bridging methodologies that transform models from

requirement models to semi-formal and formal notations. This chapter also presents our proposed approach in detail, highlighting points relevant to Internet standards development.

In chapter 4, we show the case study in which the methodology described in this work is applied to the specification and verification of the OSPF (Open Shortest Path First) [44] Internet routing protocol Link State Advertisement (LSA) refreshment function [65].

An assessment of the case study and comparison of the bridging methodologies is done in chapter 5. We include detailed results of our approach based on this case study and give recommendations for future enhancements.

We finish with some conclusions and recommendations concerning related future work in chapter 6.

2. Basic Definitions and Concepts

2.1. Introduction

This chapter discusses protocols and Internet protocols (section 2.2), the typical and Internet standards processes (section 2.3), description techniques that can be applied to Internet Standards (section 2.4), and tool support for these description techniques (section 2.5).

2.2. Protocols

This section briefly explains the main concepts of protocols. It also shows the difference between protocols and Internet Protocols.

2.2.1. Protocols

A *protocol* is a special set of rules for communicating which the end point entities in a telecommunications connection use to send signals back and forth. More precisely, A *protocol* is a set of rules governing the format and meaning of the frames, packets, or messages that are exchanged by the peer entities within a layer. Protocols exist at several levels or layers in a telecommunications connection. For example, there are physical layer protocols. There are network and transport layer protocols between the end points in communicating programs within the same computer or at different locations in a network. Both end points must recognize and conform to the same protocol. Protocols are often described in an industry standard or international standard. Entities use *protocols* in order to implement their *service* definitions. More precisely, a *service* is a set of primitives (operations) that a layer provides to the layer above it. The *service* defines what the layer is prepared to perform on behalf of its users, but it says nothing on how these operations are implemented. For more information on *protocols*, *services*, and *standardized protocols*, please refer to [7], [58], [59].

2.2.2. Internet Protocols

Internet protocols are like regular protocols, except that their peers are connected through the Internet. An Internet protocol may or may not use the concept of a *service*. The definition of an *Internet protocol* is particular to its RFC (Request For Comments), provided it complies with the IETF regulations. These regulations are given in RFC2026 [10]. We simply define an *Internet protocol* as a set of rules that enable the

implementation of some service over the Internet. More information on IETF and RFCs is given in the next section.

2.3. Introduction to Standards Development Process

Different minds think differently, and for the same reason, different vendors think differently. Standards try to accommodate each vendor's different views into one common view. Internet Protocol Standards are no different than any other standards in this sense.

Standards are written, checked and approved by a number of standardization bodies. However, it is very difficult to achieve high-quality standards. This makes it essential to apply special protocol engineering methods assisted by appropriate tools in the development of standards [48].

There are many standardization bodies in the world. Some address national or regional standardization while others are dedicated to international standards. In addition, there are forums that write technical specification agreements. Designated government agencies publish standards that are binding and require product compliance by government law. Other standards are non-binding and are developed by consensus for use in accordance with market demand. Below is a list of the main standardization bodies.

International Organization for Standardization - ISO

The International Standardization Organization, ISO [29], is an international organ that standardizes a very wide variety of standards, not just communication standards. One of the best known ISO contributions for data communications is the ISO/IEC JTC1 OSI Layered Model [7].

International Telecommunications Union - ITU

The International Telecommunications Union, ITU [33], is a technical committee of the United Nations and is one of the biggest standardization bodies in the communications field. It is divided into ITU-R, ITU-D, and ITU-T, respectively Radio, Development, and Telecommunications Standardization Sectors.

European Telecommunications Standards Institute - ETSI

The European Telecommunications Standardization Institute, ETSI [19], is a standardization body that develops European telecommunication standards. ETSI uses

and provides guidelines on how to apply formal methods in its standards [17]. ETSI is also responsible for the standardization of the testing language TTCN-3 (Tree and Tabular Combined Notation version 3) [18].

Asynchronous Transfer Mode (ATM) Forum

This is a body that develops technical agreements for the Asynchronous Transfer Mode [6] technology. The ATM Forum is worth noting for its effort to apply some formal methods in developing its technical specifications. For example, all ATM Forum conformance test suites [41] are specified in TTCN (Tree and Tabular Combined Notation) [37], a formal description language for specification of protocol test suites.

Internet Engineering Task Force - IETF

Responsible for Internet protocols, the Internet Engineering Task Force, IETF [36], takes a different approach in the process of standardizing computer protocols. In fact, specifications produced by IETF are not standards, but Requests For Comments (RFCs). The more important class of RFCs are approved within IETF as standards-track RFCs.

We are particularly interested in studying this standardization body because of its impact on the world's largest computer network, the Internet. Another major difference of the IETF from the other standardization organizations is its openness. Everyone is allowed to participate as an individual in IETF, without company affiliation, or even without submitting written contributions.

2.3.1. Generic Standards Process (non-IETF)

There are several different naming conventions for organizing standard bodies, however they all resemble the same model. Some bodies require a membership, some do not. Some bodies are more open to discussion, some others are more preoccupied with intellectual property and related issues. We now give the generic levels of progress for standards.

Proposals

Once the documentation of a particular protocol standard is ready to be delivered, a *proposal* is submitted to the respective standardization body. All the other parties interested in that proposal should review it and send back *comments*. These comments are sometimes very specific and need intensive collaboration until the issues and resolutions

are agreed upon. For this reason, the standardization bodies members meet on a regular basis. Sometimes these issues are solved with *voting*. When points of conflict arise, *ballots* are cast to decide which side should be favoured. Not all standardization bodies use *voting*.

Approvals

The approval process used is the main point of difference between standardization bodies and the nature of the standards they produce. In ITU-T, standards are approved by a well defined process based on consensus that involves government administrations and industry sector member organizations, each of which has one vote.

In addition to the procedural requirements, standards must meet some level of technical quality. Draft standards need to be verified. If the draft has been well accepted by the community and agreed on with respect to all the technical details, the standard is forwarded for approval.

Depending on the protocol and on the standardization body requirements, the draft standards may lack completeness and other qualities essential for implementation.

In ITU and ETSI, a standard may be approved prior to any implementation. In such cases it is essential that the standard has been proven to be accurate and free of major problems and gaps in its description prior to its approval [30]. Early implementations may be submitted as evidence that the standard proposal is complete and free of serious errors. IETF, as we shall see takes a different approach [10].

Publication – Standard Achieved

When the standard is *published*, it receives a unique identifier that allows implementers to claim conformance or interoperability to that particular standard. National regulators sometimes require that a certain vendor's implementation be *certified* as conformant before being eligible to be sold in that country. Regulated products may suffer from a bottleneck from the amount of implementers submitting their implementation to a *certification organization*. Voluntary conformance to non-binding standards is market-driven. *A standard* in non-IETF bodies *is achieved* when the proposal in English (possibly augmented by formal descriptions) is approved by all member bodies and published.

Study Groups

One important concept in ITU is the notion of *Study Groups*. A *Study Group* is established by a higher body to manage the development and approval of a group of standards related to an area of study. Each *Study Group* consists of one or more *Rapporteur* groups responsible for answering an approved *Question*. These *Rapporteur* groups work on a particular problem question in a very specific and detailed way. Their work may result in one or more new standards. Their objective is to ensure the highest level of clarity in the *standards document* for which they are responsible.

An example very related to this thesis work is the current effort to standardize Use Case Maps [13] as User Requirements Notations, URN [35].

Goals for Standards

As mentioned before, standards are important mainly for providing *interoperability* between different vendor's products. However, there are other aspects pertinent to standards that are not often mentioned and deserve some attention.

Interoperability: If an implementation does not need to interoperate with another product, it need not conform to a standard. In cases where standards have an impact on the environment (such as the level of radiation emitted by a radio device), then the standard should dictate operational constraints.

Conformance: When a particular implementation conforms to a standard, users and operators can be sure that if a protocol failure occurs during the use of that implementation, the fault will lie outside the implementation. Customers may obtain advantages by choosing from among several conforming implementations, possibly from different vendors, to perform the same function.

Suitability for Regulation: Standards are either used voluntarily by manufacturers or are referenced in legislation under government regulations mandating that manufacturers must meet specified requirements. In such cases products are monitored, tested and certified by government regulators. Regulation and certification agencies **test** implementations against the implementation's respective standard to ensure compliance.

2.3.2. Internet Standards (IETF) Process

An Internet Protocol Standard is a detailed definition given in plain text for that Internet protocol. There are no mathematical models specified for such. The text usually describes

protocols belonging to the TCP/IP stack, and is usually focused on the protocol's implementation and written in pure ASCII format.

In IETF, once a protocol is ready to be standardized, a *Working Group* is formed to head the effort under the direction of a *Working Group Chair*. Several working groups may be grouped into a common area of study, for which an *Area Director* is elected. Working with this key person, and the working group chair, are the working group members who provide all the necessary effort for delivering that protocol standard. Included in this effort are requirements capture, design, implementation, testing and documentation.

Request For Comments and Internet Drafts

IETF prefers one-to-one discussions or possible demonstrations of feasibility for conflict resolution. IETF proposals are better known as RFC's (*Request For Comments*) even if under the standards track. A new protocol proposal is started by a group of enthusiasts called *Birds Of a Feather*, or BOF. If approved to proceed with the development of the RFC, a charter is drafted for the formation and operation of a Working Group. In IETF, in addition to Request for Comments, there are *Internet-drafts* which are comments on RFCs or proposals for new protocols.

Standard Achieved

In IETF, RFC standards are approved by working group chairs, who, in principle are not affiliated with any organization and may have only a personal interest. IETF does not require the standard to be unambiguous or completely validated. However, in IETF, *a standard is achieved* when at least two distinct working implementations have been produced and they interoperate [10]. Note that considerable interpretation of an RFC is required when developing an implementation. Once an implementer makes his interpretation, other vendors may not be able to interoperate with his product. In such cases the quality of RFCs would have benefitted from reduced ambiguity through the use of formal methods, and the USHLTD bridging methodology proposed in this thesis.

IETF protocols do not require any certification. In fact, IETF does not recognize conformance testing (passing an approved set of test cases). There is no practical way for certification to guarantee that an implementation conforms to an IETF standard.

One particular point to be raised about the IETF is the restriction to ASCII text documents. The IETF does not make use of any kind of word processing tool: therefore its protocol specifications include poor drawings and confusing text. However, a recent step has been taken to accept PostScript files as normative or informative appendices to its documents. For more information on IETF policies and procedures, see [10].

We will comment in chapters 3, 4, and 5 on our personal experience and resulting recommendations related to improving the IETF process. The area for improvement which this thesis proposes involves the use of semi-formal and formal description techniques, discussed next. While ITU and other standards organizations encourage or allow the use of description techniques, IETF does not as yet.

2.4. Description Techniques

Description Techniques are special notations used to describe software *architecture* and *behaviour*. We classify description techniques as *Semi-Formal* and *Formal*. A *Semi-formal Description Techniques* (SDT or SFDT) is a specification language for which the syntax/or semantics is only partly fixed. We believe these techniques may prove very useful in the earlier stages of design. Examples of SFDTs include Use Case Maps (UCM) [13], and the Unified Modelling Language (UML) [1]. A *Formal Description Technique* (FDT) is a specification language with a well-defined syntax and semantics. Therefore models written with this type of notation may be formally verified [22]. Examples of FDTs are the Specification and Description Language (SDL) [16], the Language Of Temporal Ordering and Specification (LOTOS) [28], and the Tree and Tabular Combined Notation (TTCN) [37]. The interested reader may consult the IFIP series on Protocol Specification, Testing and Formal Methods [25], on Formal Description Techniques [26], and on Testing of Communicating Systems [24]. Moreover, there is the SDL-Forum series [57] for the FDT SDL. More on *Semi-formal description techniques* and *Formal Description Techniques* can be found in [62],[63].

Next we present the description techniques used in this thesis, and the rationale for the choice.

2.4.1. Semi-Formal Description Techniques (SFDT)

In this sub-section we briefly present the Use Case Maps notations and other semi-formal description techniques.

2.4.1.1. Use Case Maps (UCM)

A Use Case Maps system design consists of at least one *starting point*, one *thread of execution* (or *path of execution*), and one *end point*, as in Figure 2.1.

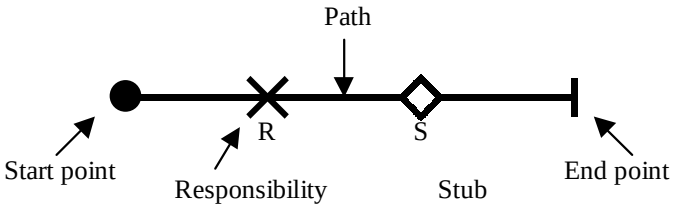


Figure 2.1 - Basic Use Case Map

The path works as a timeline from the *start point* until one *end point*. Use Case Maps, by definition describe the *causality* of events. Figure 2.1 also shows two other constructs, namely *responsibility* and *stub*. A *responsibility* is something that happens (an *event* or an *action*) during the course of the path, at that particular moment when the follower of the path reaches that *responsibility*. The path of execution dictates the order in which the responsibilities happen. For more discussion on UCM's, please refer to [13],[12],[64]. The interpretation and design of UCM's are relatively straightforward. To illustrate this, consider the UCM in Figure 2.2 which shows a more complete example of a Use Case Map. This UCM condenses several pages of text description in the IETF RFC for OSPF. This UCM is better explained in chapter 4, subsection 4.3.2.

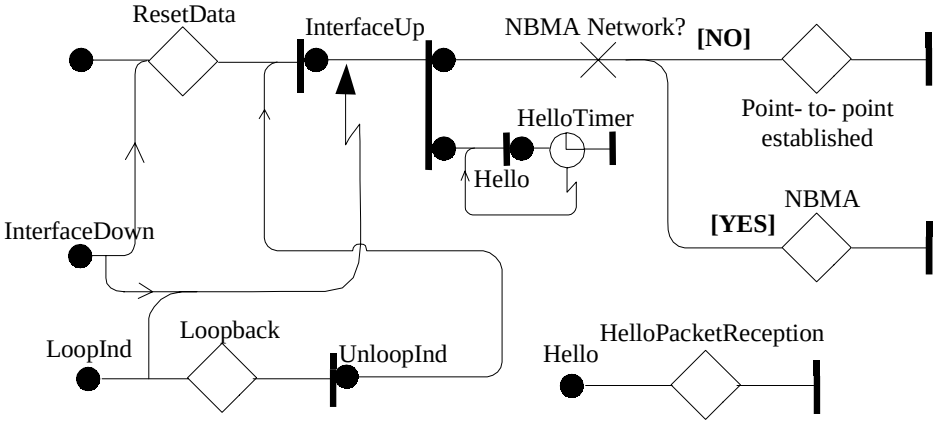


Figure 2.2 - OSPF Interface Use Case Map

The UCM notation is also going through the process of standardization, and receiving the name of User Requirements Notation, URN [35]. This is evidence of its perceived significance.

We chose Use Case Maps as our bridging methodology's Source notation (Semi-Formal Description Technique) because:

- It is undergoing the process of standardization, and may become very widely adopted.
- It was designed for real-time systems and protocol engineering.
- It is graphical
- It is easy to read and write
- It has tool support
- There is no current bridging methodology from UCMs to SDL.

In addition, it was our personal experience that using UCMs to help navigate the disjointed descriptions of functionality in the IETF writings kept us from becoming overwhelmed and lost in the details.

2.4.1.2. Other SFDTs

Among other semi-formal description techniques we list UML (Unified Modelling Language) [55]. We found that not many description techniques are recognized under the name of *semi-formal description techniques*.

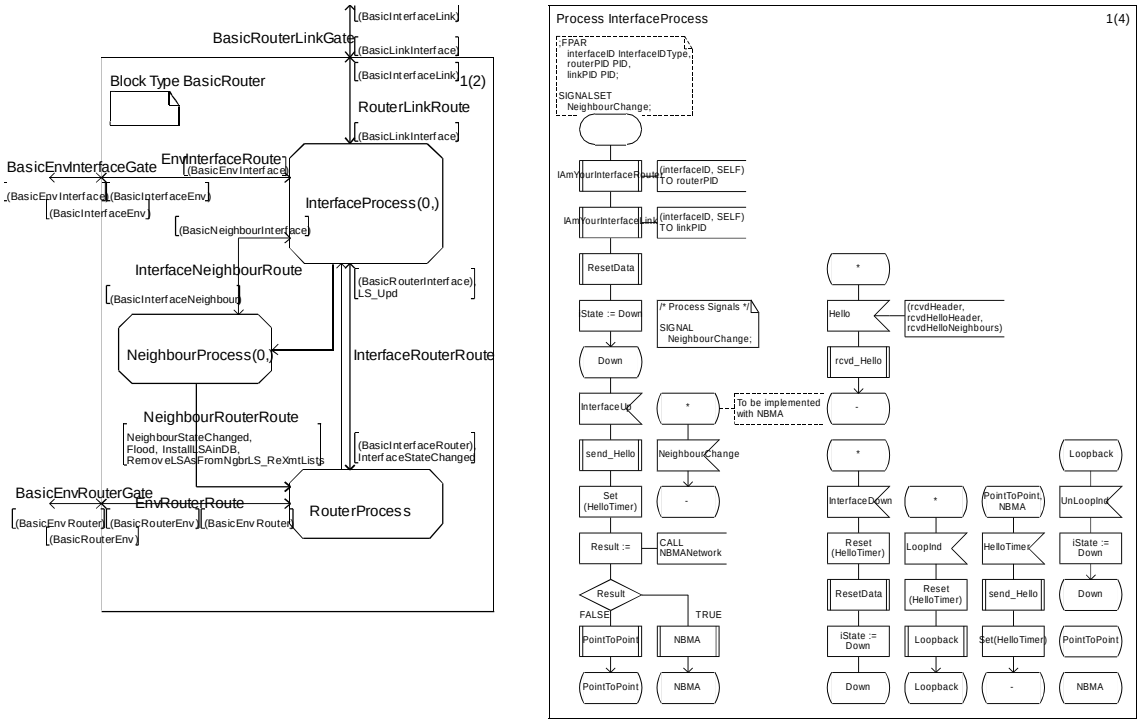
2.4.2. Formal Description Techniques (FDT)

In this sub-section we briefly present the Specification and Description Language (SDL) Notation and we present other formal description techniques.

2.4.2.1. Specification and Description Language (SDL)

SDL describes both *architecture* Figure 2.1(a) and *behaviour* Figure 2.1(b). For the *architecture* (or *structure*) there are *systems*, *blocks* and *processes*. These are always nested in this order. In the figure we can see three *processes*, **RouterProcess**, **NeighbourProcess** and **InterfaceProcess**. These processes communicate by sending and receiving *signals*. The definition of these signals is shown in the square brackets near the end of arrows connecting two processes. In the behaviour, *states* are denoted by a rectangular shape with rounded sides. Each *state* must have an *input signal* (rectangular shape with an arrow pointing to the inside) associated with it to describe a *transition*. Each *transition* describes the set of actions taken after the (*state, input signal*) pair. There are many more constructs in SDL. A further explanation of the language is not done here

because there is enough available literature on the subject (See [16],[31] for more details).



(a) SDL Architecture

(b) SDL Behaviour

Figure 2.1 - SDL Architecture and Behaviour Example

SDL is not as easy to read as UCM. The relationship between the SDL in Figure 2.1 and the UCM in Figure 2.2 is explained in section 4.3.3. We chose SDL our *destination* the Formal Description Technique in our bridging methodology because:

- It is an international standardized notation.
- It has been recognized by other standardization bodies as a de facto standard formal description technique.
- It has recognized excellent tool support.
- It has formal semantics.
- Its tools support automated Validation and Verification
- It is relatively easy to understand, since it is based on an extension of a well understood model, the Finite State Machine (FSM).
- It is constantly under improvement

- We could not find any related work on the bridging from UCMs to SDL (see chapter 3).

2.4.2.2. Other FDTs

Formal Description Techniques include, the Language Of Temporal Ordering Specifications (LOTOS) [28], and Estelle [62], among others.

2.5. The Technology: Support Tools for Protocol Development

In this section we show the main existing tools for the techniques we chose for our bridging methodology. These tools are also known as CASE (Computer-Aided Software Engineering) tools.

2.5.1. UCM Tools

We are only aware of one tool developed so far to support UCMs.

UCM Navigator (UCMNav)

Use Case Maps were developed at Carleton University, in Ottawa, Ontario, Canada, by Ray Buhr [12], and a group from the same university developed a tool that supports such notation. This tool is called the UCM Navigator, or UCMNav, and was developed by a Master's student named Andrew Miga [39]. Miga continues to support the tool.

The UCMNav tool is constantly under development and there are a few more contributors to it. Its capabilities include *full notation support*, *generation of diagrams in printable format*, *model documentation capabilities*, and *stub navigation capabilities*. It features an intuitive user interface. There are two features worth special attention: The generation of Message Sequence Charts from UCM's, programmed by Miga himself, and the generation of abstract test cases in LOTOS done by Leïla Charfi as her master's thesis [14]. Although this tool has been used by industry, it is not an *industrial-strength* (explained in the next section) tool yet because of its instability and some usability problems.

2.5.2. SDL Tools

Telelogic TAU

Telelogic TAU [61] consists of a *industrial-strength* toolset that supports many of the previously mentioned notations, namely UML, SDL, MSC, HMSC, and TTCN. It is denoted *industrial-strength* because it is commercially supported, and because it can

handle large-scale models. Its bigger strength is towards the SDL → MSCs → TTCN chain. The UML side was only recently integrated, therefore leaving a gap between a world chosen notation and the tool's powerful features. Starting with SDL, the user can design a model, check the notation syntax on the fly, and execute (simulate) the SDL model. This execution can be through navigation or simulation. This tool also allows an SDL specification to be verified against deadlocks, unspecified receptions and other things. This is done through state space exploration techniques [11].

TAU includes a TTCN toolset that allows us to quickly create TTCN test suites from scratch or based on an SDL model [49]. TAU's support of UML (since version 4.0) has a feature that translates an UML model to an SDL model, thus allowing the UML model to be validated, verified, and have its code generated, and even to create a test suite. One other very nice TAU feature is its ability to store models in a repository. This means that two or more users may work on the same UML model at the same time, avoiding having to exchange and merge model files.

Of great *industrial-strength* interest is the ability of TAU to directly generate complete executable code for stand-alone systems in a variety of real-time operating systems [61]. In the domain of this thesis, this means that a fully executable LSA refreshment function can be automatically generated from the destination SDL in our bridging methodology. This satisfies the IETF constraint of having a distinct implementation of an RFC in a very efficient manner.

For all the reasons, the choices of SDL and TAU appear to have been the best suited to the problem this thesis addresses.

Other SDL Tools

There are other tools for SDL such as Cinderella SDL [15], ObjectGEODE [60]. ObjectGEODE, once a competitor to TAU, has now been acquired by Telelogic, the developer of TAU. Generally, we found that other SDL tools do not support as many SDL features, and it does not support automatic code generation.

In the next chapter we discuss the bridging methodology we have developed.

3. USHLTD Bridging Methodology

3.1. Introduction

Here we describe a process for improving the development of Internet protocol standards with *semi-formal* and *formal description techniques*. We start by giving some definitions (section 3.2) that are used throughout this chapter. We define a *bridging methodology* as a process to derive a *destination* model from a *source* model. Then we present a literature review on the bridging methodologies in section 3.3. It is common to have more than one notation per project, as different notations aid different phases of standards development [2]. Some of the bridging methodologies (as we call them), more specifically *synthesis* techniques, also have tool support, which helps automating the bridging stage of the development process. In section 3.4 we describe in very fine detail the bridging methodology we are proposing, namely USHLTD (UCM to SDL High-Level to Design). Our proposed bridging methodology helps transform UCM (Use Case Maps) *source* models into SDL (Specification and Description Language) *destination* models. We validate our methodology in the case study presented in chapter 4.

3.2. Concepts and Definitions

A *scenario* is a sequence of *events* between *actors* of a *model* or *system*. These can be expressed in *natural language* or in *sequence diagrams* such as UML sequence diagrams, or MSCs [32]. An *event* or an *action* is an interaction between one or two *actors*. An *actor* is a component of the system that is capable of communicating (sending and receiving signals) through *queues* with other actors. Please refer to the figure below (Figure 3.1) for an example of a *scenario* in terms of a Message Sequence Chart, or MSC [32]. UML denotes Message Sequence Charts as Sequence Diagrams. Their semantics are similar with different graphics. Message Sequence Charts have well-defined semantics as discussed in [32].

The *architectural view* of a system, also called *structural*, describes the static relationship between the several components of a system. The other view of the system, *behavioural*, describes the system's dynamic functioning. This *behaviour* describes its reactions to *external inputs* and *internal events* like timers. Therefore, the *behaviour* captures the possible *scenarios* of that system.

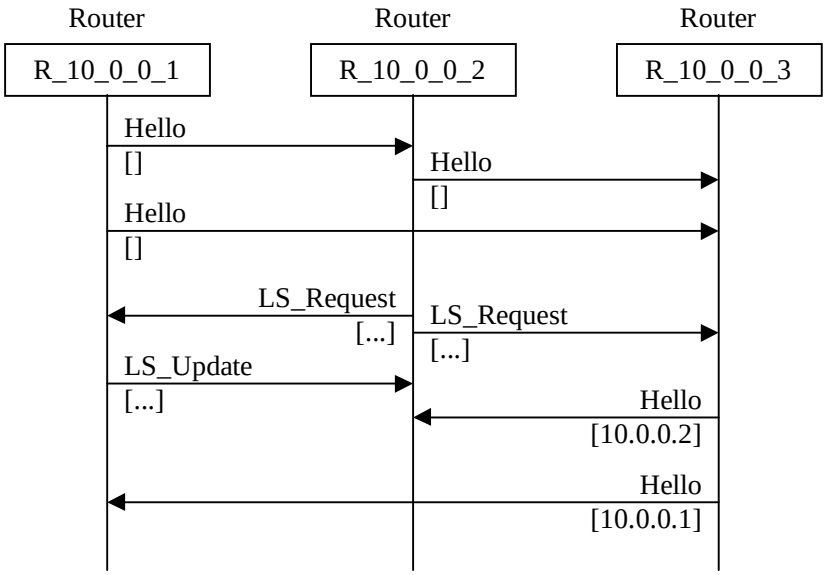


Figure 3.1 - Example of a Message Sequence Chart (MSC)

A *model* is a set of behaviours formally or semi-formally described using a certain notation. Along with the behaviour there may be the description of the *structure* of that model. The *structure* may describe the *channels* and *signals* between the *processes*. The *channels* are used to structure the message flow between components. It is common to see these channels as *queues* of messages. There are several ways of expressing a model's behaviour, and each description technique mentioned here has its own way of describing behaviour. Examples are Finite State Machines (FSM) [21], Extended Finite State Machines (EFSM) [21], and flow diagram-like notations [12]. An example of an FSM is shown in Figure 3.2.

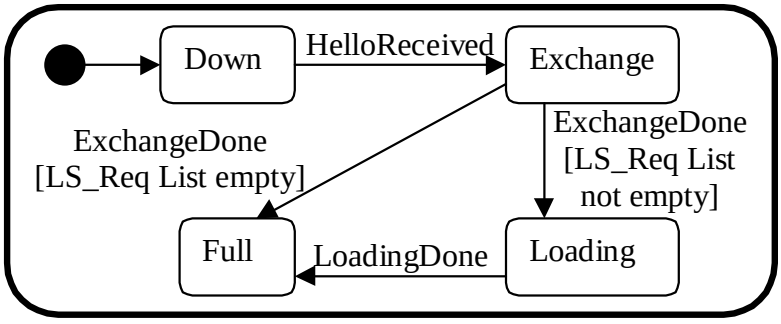


Figure 3.2 - Example of a Finite State Machine as an UML State Diagram

A *model* may contain several *processes*. We define a *process* as a particular instance of a running component type (or process type). Note that several processes may

communicate, and they do not need to be of the same type, or run under the same processor. In addition, *processes* provide the behaviour of a model.

We define that a bridging methodology may be *scenario-driven*, *responsibility-driven*, or *hybrid*. It is *scenario-driven* when the model construction is based on scenarios. It is *responsibility-driven* when the model construction is based on procedural descriptions (or function descriptions). It is *hybrid* when there is a blend of both *scenario-driven* and *responsibility-driven* construction methods.

A bridging methodology may be an *iterative development methodology*. It is an *iterative development methodology* if it describes the process that define the iterations for the development of the models. If a bridging methodology simply consists of a transformation step, it is not an *iterative development methodology*.

There are two types of bridging methodologies that can be applied to a group of notations: *synthetic* or *analytic*.

A *synthetic* approach takes a model *m* given in a certain notation *S* (source notation) and **transforms** it into another model *p* in a different notation *D* (destination notation). Let *f* be the set of rules that transforms a model, then, $f(m) = p$ where $f : S \rightarrow D$. Synthesis methods may make use of several input models (different notations) and generate one or more output models. Generalizing, $f(m_1,m_2,...,m_k) = \{p_1,p_2,...,p_l\}$ where $f : S_1 \times S_2 \times ... \times S_k \rightarrow D_1 \times D_2 \times ... \times D_l$.

An *analytic* approach depends heavily on the designer. This makes the approach more open, because it introduces *design decisions* in the models. Instead of describing an *analytic* approach as a function, we found they may be described as a set of rules, called *Construction Guidelines* [4].

We define the *source model* as being the original model from which we will apply the *synthesis rules* or the *construction guidelines*. The *destination model* is the model generated by applying such *synthesis rules* or *construction guidelines*.

Some bridging methodologies may be *automated*. This means that the bridging methodology process will be supported by a tool. If not *automated*, it will require the designers to manually proceed with the bridging methodology steps.

We define a description technique as *executable* if its models may be executed. SDL, for instance, is *executable*, while UCM is not. With *industrial-strength* tool support

for *executable* description techniques we may perform consistency checks (such as verification), guided simulation (and validation), and automatic code generation. We have already discussed this in section 2.5.

Now we present a brief overview of some other bridging methodologies.

3.3. Other Bridging Methodologies

In this section we review some bridging methodologies we found in the research literature.

3.3.1. SPEC-VALUE

Daniel Amyot developed this approach as his Ph.D. thesis [4]. It uses his Master's thesis [3] as starting point for the development of SPEC-VALUE (Specification-Validation Approach with LOTOS and UCMs) [5]. It makes use of two of the notations mentioned previously, namely Use Case Maps and LOTOS. Each step of this approach is rigorously defined. Figure 3.1 depicts the SPEC-VALUE process.

This is an *analytic* approach, and it is also *rigorous*. Its characteristics include *separation of functionalities from the underlying structure*, *fast prototyping*, *test cases generation*, and *design documentation*.

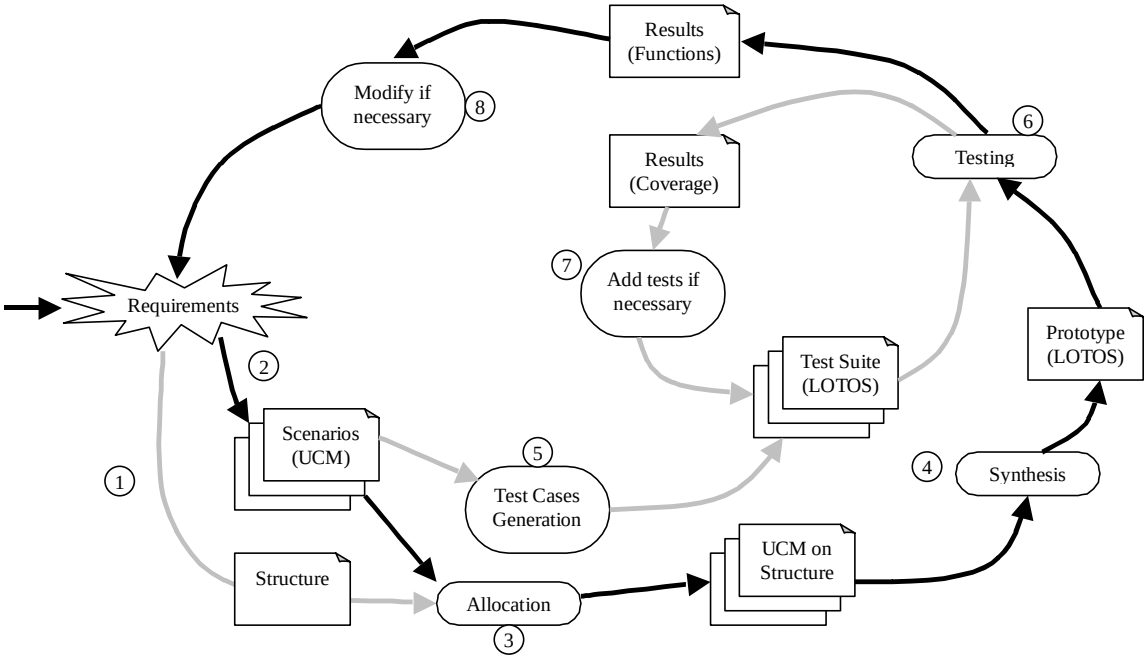


Figure 3.1 - SPEC-VALUE process description

We believe it is specially suitable for the development of new protocols. We found some drawbacks in this methodology regarding the Internet Protocol development:

- Its notations lack industrial-strength CASE tools for automatic code generation.
- LOTOS is known as a hard to learn language [62]. In addition its graphical counterpart is neither widely accepted nor straightforward.
- It focuses on the high-level specification and validation at early stages of design, while Internet standards need lower level details (see subsection 2.3.2).
- It is very rigorous, therefore may be complicated.

3.3.2. RT-TROOP

The Real-Time TRaceable Object-Oriented Process is a *scenario-driven* approach to construct models in UML/RT or ROOM [56]. It is a *rigorous analytic* approach that starts from the very early stages of development up to an executable implementation in UML for Real-Time. It uses the Scenario Textual Description (STD) notation which is based on UML use cases. Then it transforms this textual notation into a Use Case Maps model, and then Message Sequence Charts are generated. Then the MSCs are used to produce Hierarchical State Machines described in UML/RT. Figure 3.1 depicts the RT-TROOP process. This figure was taken from F. Bordeleau's thesis [8]. A detailed description of the transformation of Message Sequence Charts into Hierarchical State machines is given in [9].

This process is a systematic approach to the development of executable models from textual requirements. It is a rigorous approach for constructing finite state machines. In our opinion it is a rigorous way of describing how we naturally design state machines for real time systems. Even though this approach describes a systematic approach to development, it has some drawbacks.

- In our opinion, STDs are extremely tiresome and difficult to write, despite the fact that this approach claims it does not necessarily requires the use of STDs
- It is based on patterns, which are somewhat open to different interpretation by different individuals.
- It does not use a Formal Description Technique, therefore does not enable verification.

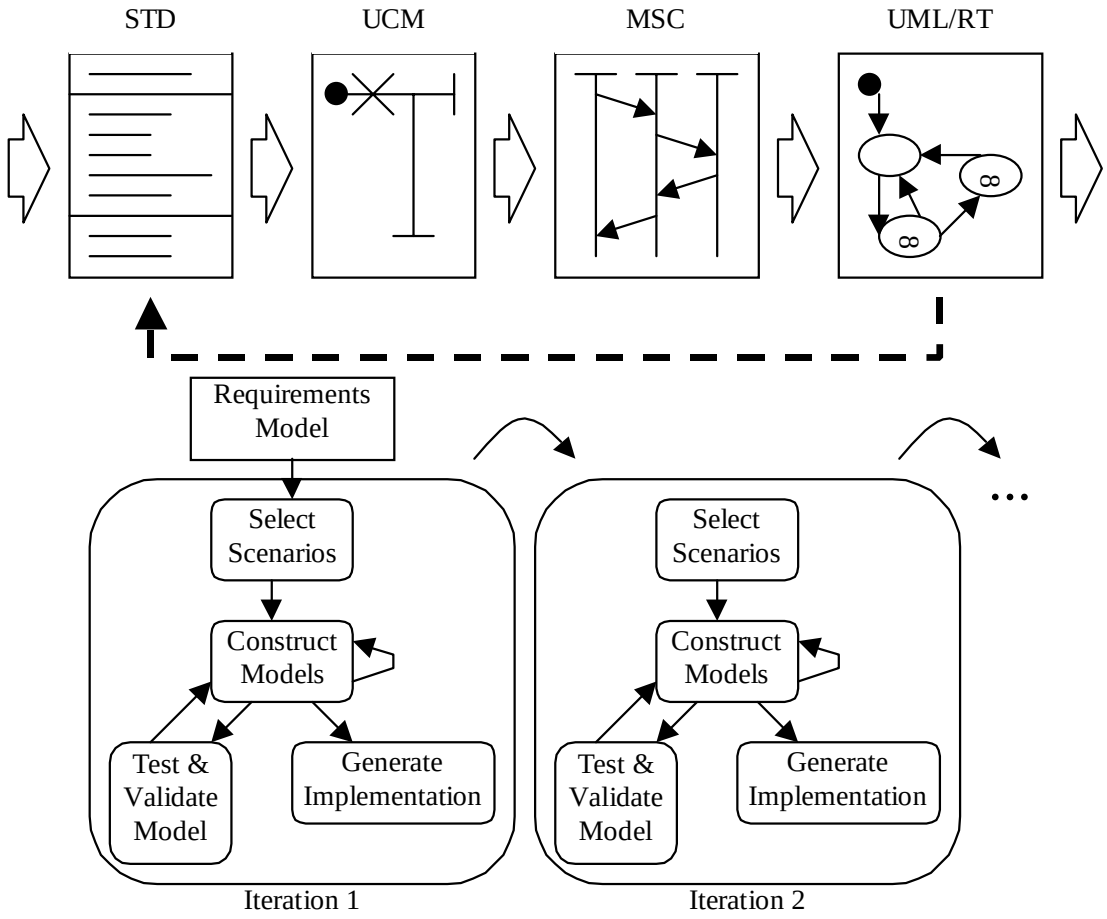


Figure 3.1 - RT-TROOP process description

3.3.3. MOST

The MOscow Synthesizer Tool, MOST [27] is an implementation of a *synthesis* technique. This *synthesis* technique has a set of Message Sequence Charts as its *source* model and delivers an executable SDL model as its *destination* model. This model reflects the behaviour of the source MSCs. It works based on the idea of MSC slices, which are portions of an MSC (an actor, its messages and actions) with respect to one and only one actor at a time. Each SDL process is constructed by synthesizing all the MSC slices for that process. Then a system is created by structuring all the synthesized processes together. And then an SDL model is generated. Figure 3.1 exemplifies how this synthesis technique works.

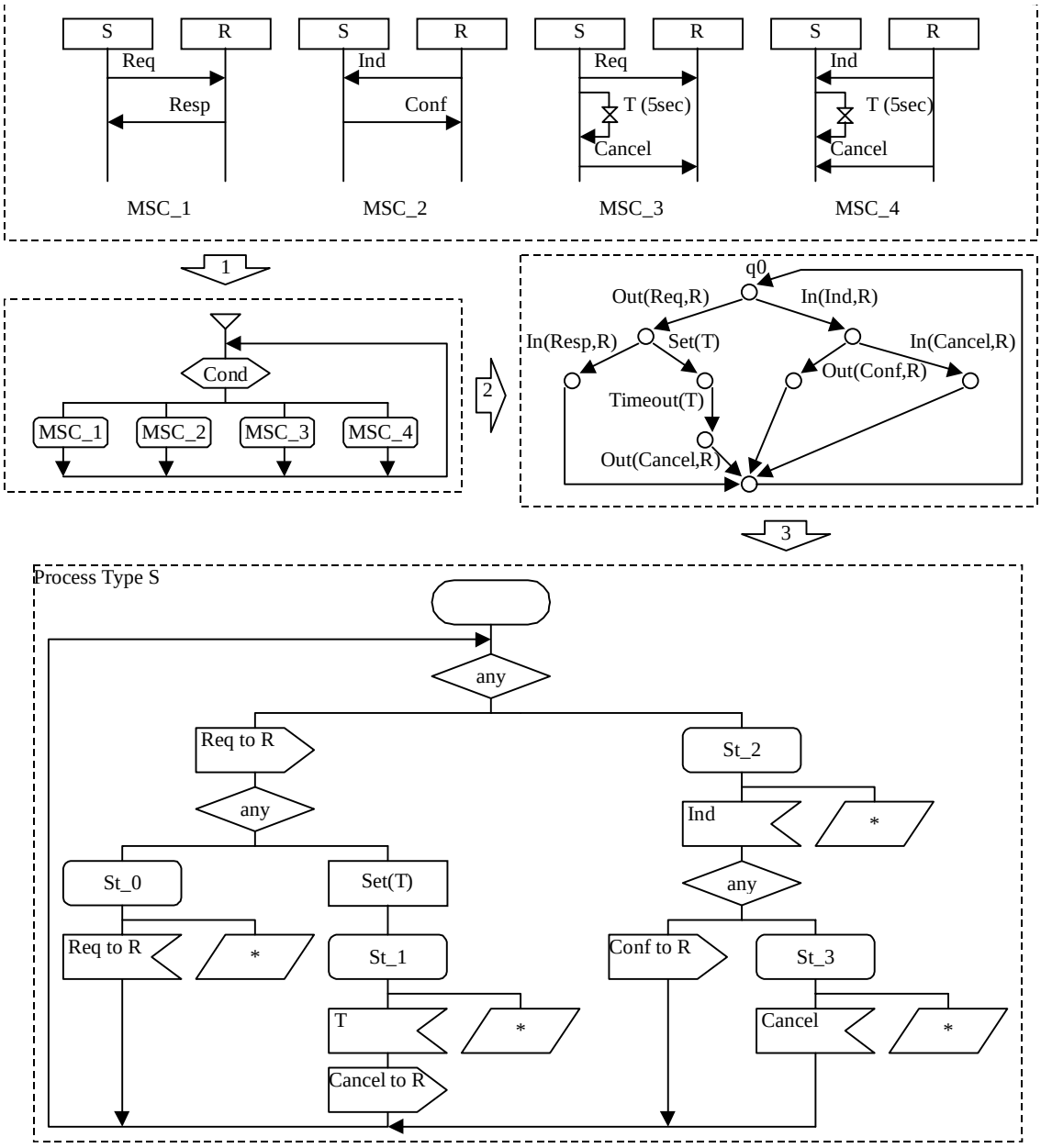


Figure 3.1 - MOST Synthesis process description

This technique was fully automated (MOST tool) and has achieved some success [38]. However, it may be only partially useful for Internet protocols. That is when standards developers have scenario descriptions available. In most cases we found Internet standards are developed in an incremental, *responsibility-driven* manner, and scenarios may change quite drastically before reaching a certain level of maturity. Moreover, we found other disadvantages to this approach with respect to Internet Protocol development:

- It is tied to two formal description techniques, MSC and SDL.
- It avoids deadlocks with an SDL *save* construct. This is not a very clean way of designing SDL models. If the SDL model has a deadlocks or an unspecified reception, *save* may mask it.
- The generated state machine is not straightforward to understand. The generation is not under the control of the user. For example, if the standard requires a specific state machine, MOST may not generate even a similar state machine.

We now present our bridging methodology.

3.4. USHLTD Bridging Methodology

In this thesis we are presenting a bridging methodology (UCM to SDL: High-Level To Design) USHLTD, that consists of three parts: A *Process Description*, a set of *Construction Guidelines*, and a set of *Parallel Activities*.

The initial experience of using UCMs in an industrial context was reported in [47]. Part of it consisted of a preliminary, non-systematic UCM to SDL approach. The intent of the project was to find design errors (defects and omissions) early in the development process by using a parallel *requirements capture & design modelling methodology* which operated in parallel with the SPEC-VALUE approach. A successive and cross-validation approach was used which captured requirements in MSCs, and HMSCs, built an SDL Requirements Model that was cross-verified on HMSCs, built an SDL Design Model from the designer-supplied UCMs, and validated the model by using coverage and verification tools within TAU. Then we finally compared the model behaviours to the LOTOS model secnarios according to scenarios generated from specifications. This methodology provided valuable experience toward developing the bridging methodology presented in this thesis.

"From Use Case Maps to Specification and Description Language: From High-Level Behaviour to High-Level Design" [53] was an article we published on the topic of transforming a partial requirements model into an executable model. It consists of adding some constraints in the UCM language so that we can easily and *manually* transform the UCM high-level behaviour model into an executable SDL model. This is an *analytic* approach because of the need for design decisions in the construction guidelines (CG).

3.4.1. Process Description

From experience we find our methodology suitable to different practices in the development of Internet protocol standards. This bridging methodology brings a systematic approach for describing, documenting, and validating original Internet protocol standards that basically consist of plain text. Designers can easily go from the SDL models to the implementation. Improvements in existing Internet protocol standards are achieved as a result of following the process in Figure 3.1, which we now describe.

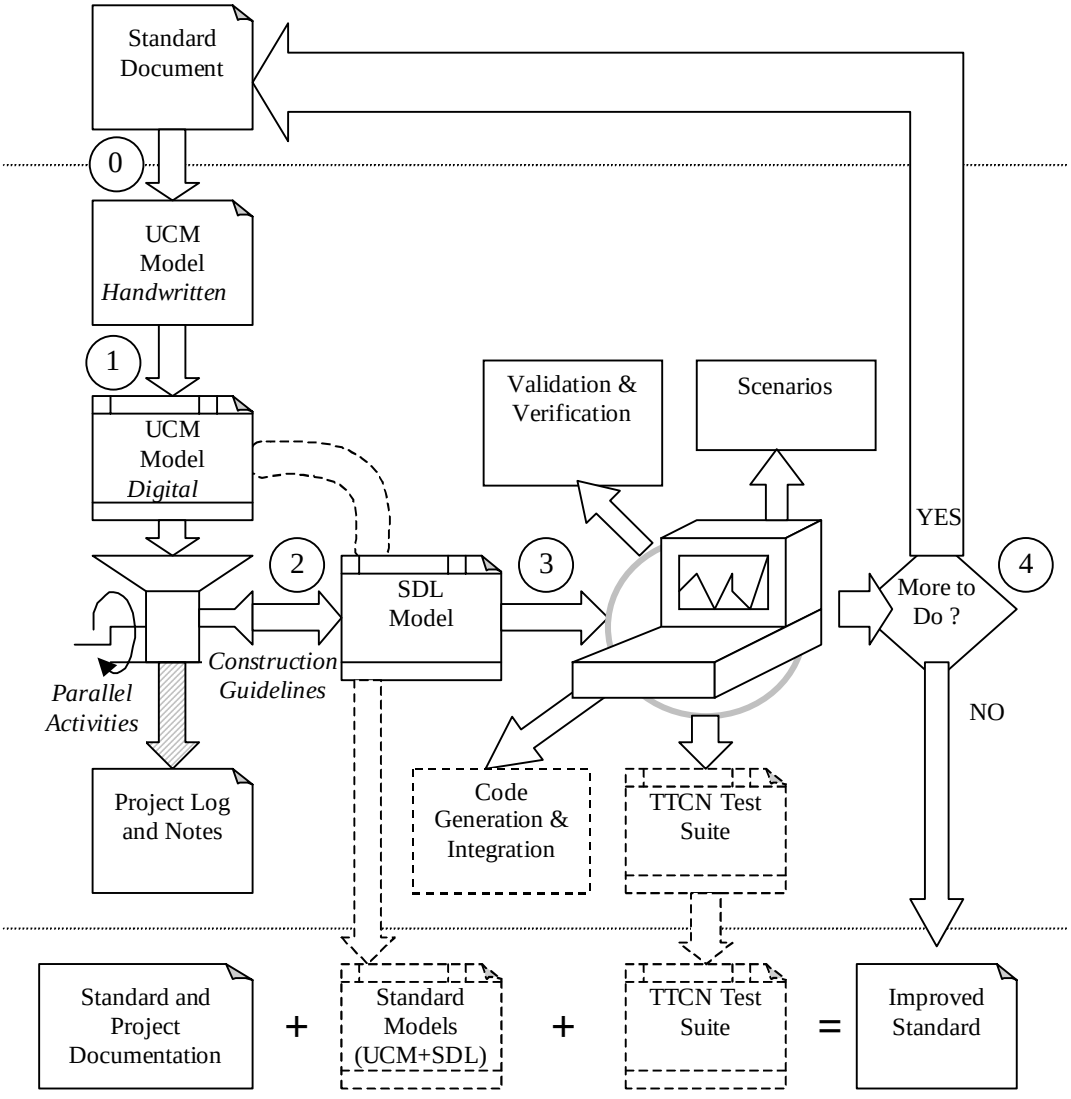


Figure 3.1 – The Proposed Bridging Methodology

First, the designer chooses the Internet protocol standard to work with. This can be an *RFC (Request For Comments)* or an *Internet-draft*. Then, the requirements are interpreted and represented using *Hand-written* Use Case Maps (Step 0) . These

requirements are manually re-checked and transformed in *Digital* UCMs (Step 1). Step 1' can be skipped if using a tool like the UCM Navigator [40]. However, we found this tool is not mature yet for industrial-strength needs.

With these UCMs and the *Construction Guidelines* from the following sub-section, we generate an SDL model (Step 2). This SDL model can be simulated/executed (Step 3) for validation, verification and scenario-generation. Also an implementation of the protocol may be automatically generated from the SDL model. This process is iterated until the artefacts are ready (Step 4). *Traceability, comments* and *activity logging* are also kept (as parallel activities) as directed in section 3.4.3 below. In the end, we have enhanced the Internet protocol standard with the semi-formal and formal model plus a possible implementation, and collected data about the Internet protocol development process. We leave room for the addition of TTCN in this methodology (dashed lines in the figure), discussed in 6.3. [49],[50] present a tested approach to the construction of test cases from SDL models.

This thesis focuses on Step 2 and on the related parallel activities because they are the strongest points of our work. Note that this process is iterative and the steps are overlapping. For example, as will be shown in the case study (chapter 4), the extraction of UCMs from the standard text, the construction of the SDL from the UCMs, and the Validation and Simulation of the SDL model overlap, and to some degree proceed in parallel.

3.4.2. Construction Guidelines

The generation of an SDL model from a UCM model follows the construction guidelines in this subsection.

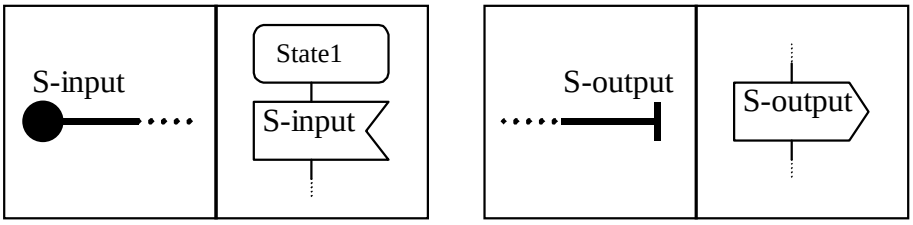
Each Use Case Map defines the behaviour for one SDL process or procedure avoiding the creation of *signals* between different components, *start points*, and *end points* in the models at the transformation stage.

CG1: In order to introduce signals to represent *stimuli* or *incoming signals* to the executable model we define that the name associated with a *start point* in a Use Case Map to be an *incoming signal*.

If the *start-point* does not contain an identifier, then the designer transform that *start-point* as the initial transition of the SDL diagram. The *spontaneous transition* construct in SDL may also be used if necessary in this case.

CG2: To introduce *outgoing* signals to the executable model, we define that the name associated with an *end-point* in a Use Case Map to be an outgoing *signal*.

Figure 3.1 shows how to transform *start points* and *end points* into SDL constructs. *S* is the *signal* name in this case. So this means that a *labelled* start point in a Use Case Map becomes an *input* in SDL. Similarly, an *output* is an UCM *end-point*. "State1" is a state named by the designer (and one of the design decisions), therefore it should be something meaningful. Moreover, an *end-point* may also transform into an SDL *to-state*, a *process end*, or a *procedure return*.



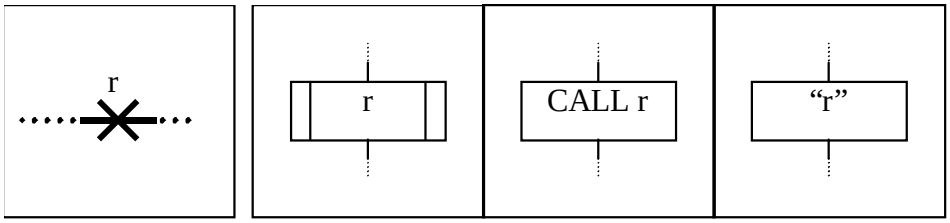
(a) CG1: UCM Start point to SDL input (b) CG2: UCM End point to SDL output

Figure 3.1 - UCM start and end points transformation into SDL input and output

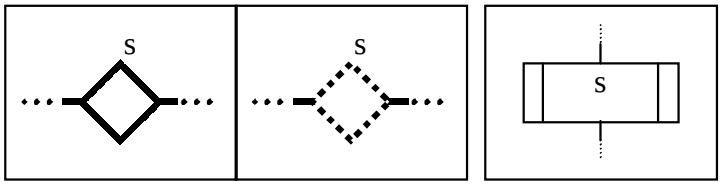
The next two guidelines are the Use Case Maps *responsibilities* and *stubs*. Figure 3.2 depicts this transformation.

CG3: A *responsibility* may be mapped to a *procedure call*, to a *procedure call* within a *task box* or to a *task* named after the responsibility (see Figure 3.2a).

As *responsibilities* mean *actions*, any of these constructs may be used. The decision on which to choose depends on the designer's reasoning. Sometimes, the *responsibility* simply increments a counter (for example), so a task with the counter increment will suffice.



(a) CG3: UCM Responsibility to SDL procedure call, procedure call in task, or task



(b) CG4: UCM static stub and dynamic stubs to SDL procedure call

Figure 3.2 - UCM Responsibilities and Stubs transformation into SDL Tasks and Procedure calls

CG4: In the case of stubs, both *static* and *dynamic stubs* must be transformed to an SDL procedure call (see Figure 3.2b).

In case of dynamic stubs, plug-ins are chosen according to selection policies. Pre-conditions and post-conditions are then added within each sub-map. In SDL, these conditions may be verified before and after the procedure call with *decision* boxes. Moreover, the decision of which *dynamic stub* to choose varies according to the designer.

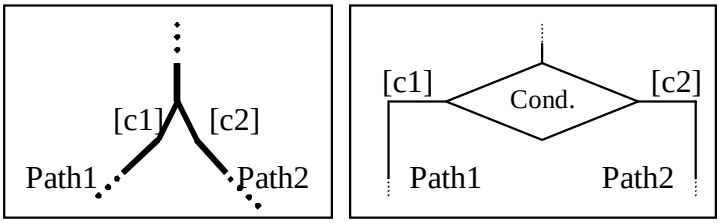
It is perfectly acceptable to have the choice of the dynamic stub inside the *procedure* "s" itself, having it make calls to *sub-procedures*. Suggestively each *sub-procedure* may be named "s-choice-1", "s-choice-2", etc.

As for every language, UCMs and SDL have constructs to denote choice. In UCMs these are called *or-forks* and in SDL *decision* points. CG5 explains how to map these constraints.

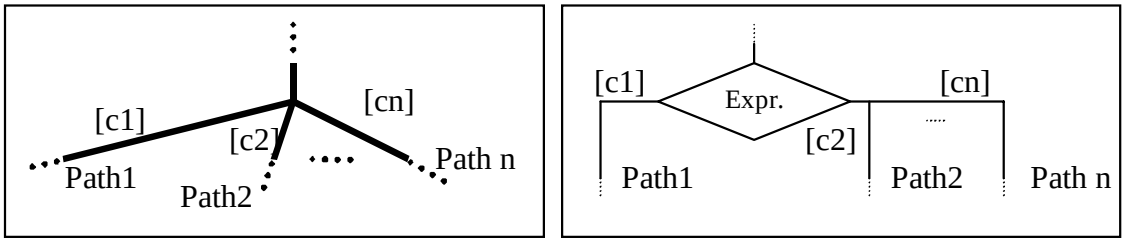
CG5: UCM or-forks should be mapped to SDL decision points.

It is straightforward to understand the conversion from UCMs *or-forks* to SDL *decision* points, like shown in Figure 3.3. It is easy to see from (a) the common case, an *or-fork* with only two exits. The condition is usually a logical expression that evaluates to TRUE or FALSE., or "c2" is ¬"c1". In the other case, (b), There are several exits, but it is still straightforward to see how the mapping goes. Basically each branch of the *Or-Fork* becomes a branch of the SDL *decision* symbol. The SDL *condition* or *expression*

may be extracted from the result of a previous *responsibility*, from *a process variable*, or from the *SDL non-deterministic condition* ANY.



(a) CG5: UCM Or-Fork to SDL Decision point

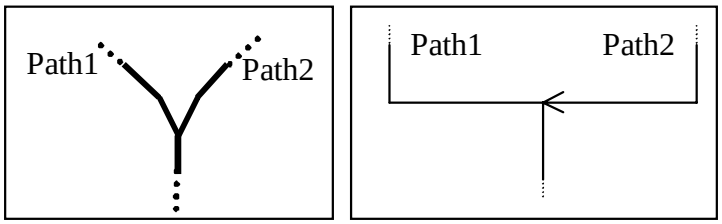


(b) CG5: UCM Or-Fork with several exits to SDL decision point with several exits

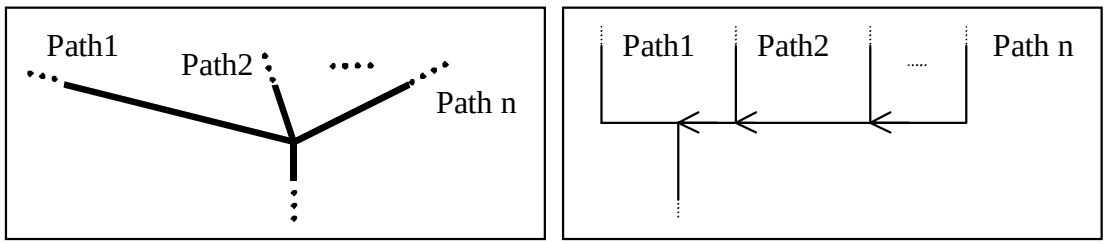
Figure 3.3 - UCM Or-Fork transformation into SDL decision point

Similarly to *or-fork*, UCMs have a construct called *or-join*. This is when two paths converge to one. It is exemplified in Figure 3.4 with its corresponding SDL.

CG6: Several different paths converging to another path are expressed with or-joins in UCMs, and in SDL the construction is preserved with the flow path concept.



(a) CG6: UCM Or-Join to SDL join



(b) CG6: UCM Or-Join with several entries to SDL join

Figure 3.4 - UCM Or-Join transformation into SDL Join

One very important construct of UCMs is the causality of events.

CG7: The causality of events in a UCM is preserved in SDL with a sequence of tasks, or a sequence of tasks and procedure calls.

Figure 3.5 demonstrates that the UCM causality rules are not broken since *r1* must necessarily happen before *r2* in both notations. We show two responsibilities here, but it could be any two UCM constructs. The order is always preserved.

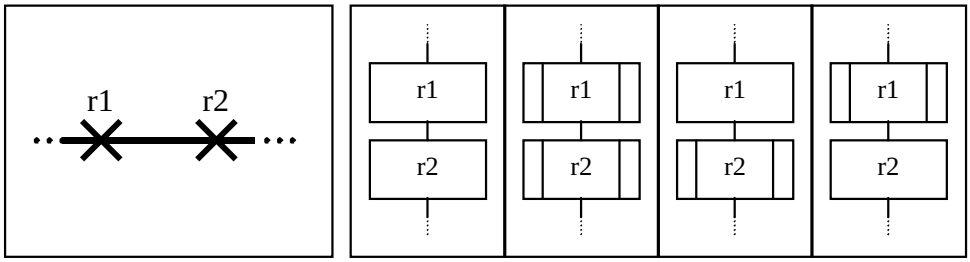
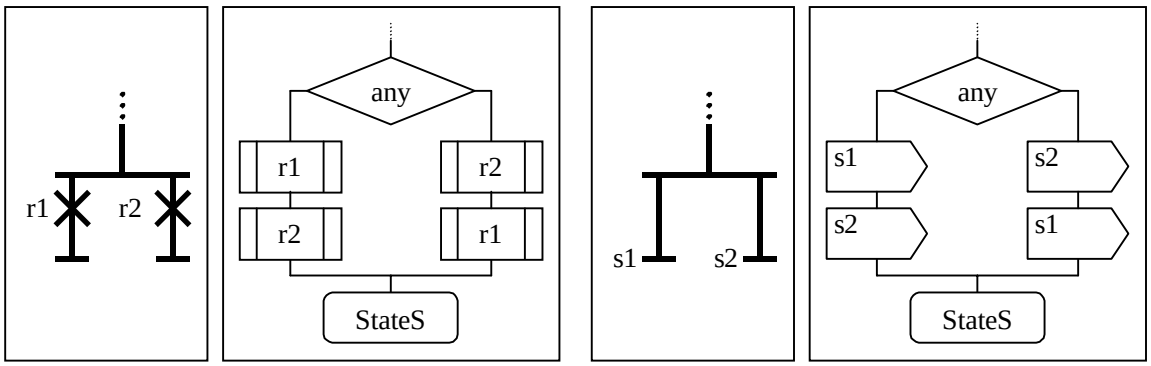


Figure 3.5 - UCM causality preservation in SDL

There are three different construction guidelines to transform UCM and-forks into SDL. These are due to concurrent paths that are not easily reflected in SDL. These three are presented in CG8 (the case with two paths with one responsibility of signal each), CG9 (the case with two paths with one responsibility in one path and more in the other path) and CG10 (the general case).

CG8: We have noticed that it is common to encounter *and-forks* like the one in Figure 3.6 with two simple actions. The most natural transition rule for such is also given in the same figure. Note that it is possible to combine the rules from (a) and (b).



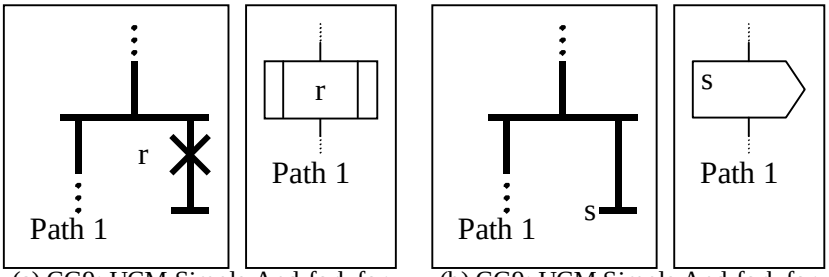
(a) CG8: UCM Simple And-fork for responsibilities to SDL (b) CG8: UCM Simple And-fork for signals to SDL

Figure 3.6 - UCM And-fork with two simple paths to SDL description

In CG8, we assume that r1 and r2 (or s1 and s2 respectively) will happen in one of two orders (r1, r2) or (r2, r1). That is why we use the *non-deterministic any* decision symbol in SDL.

CG9: As for CG8, we introduce another rule to simplify concurrency issues. It is illustrated in Figure 3.7. This rule forces *r* to happen, or *s* to be sent, before anything in *Path1*.

However, we have found that requirements are usually stated in such a way that the transformation of a parallel UCM construct to a sequential SDL construct is not problematic.



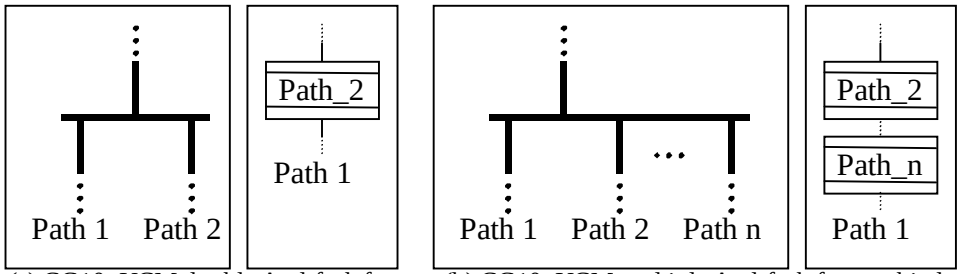
(a) CG9: UCM Simple And-fork for one responsibility into SDL

(b) CG9: UCM Simple And-fork for one signal output into SDL

Figure 3.7 - And-fork with one responsibility or one signal transformation into SDL

For the more general case of *and-forks* we show the transition rule in Figure 3.8.

CG10: This UCM construct permits several concurrent paths in SDL. Each path (except for Path 1) runs on a separate process. The creation of a new process should be customized by the designer. In our experience with Internet standards, this case is very unlikely to occur.



(a) CG10: UCM double And-fork for any kind of path into SDL create process

(b) CG10: UCM multiple And-fork for any kind of path into several SDL create process symbols

Figure 3.8 - UCM And-fork transformation into SDL create process symbols

The most complicated transition rule is the one that elaborates on *and-joins*.

CG11: The and-join concept in UCMs synchronizes several concurrent paths in one point. To accomplish this, we need to have a special *synchronization* procedure in SDL. Figure 3.10 shows a template procedure that makes the first process wait for *synchronization* signals from the others. After receiving all of them, it sends one

go signal to each of the other processes. There must be calls to that procedure in order to synchronize all the processes, as shown in Figure 3.9. The designer **must** customize this procedure to his needs. For example, set the number of parallel processes, decide which process will be the synchronizing one, and define an appropriate *signal* name.

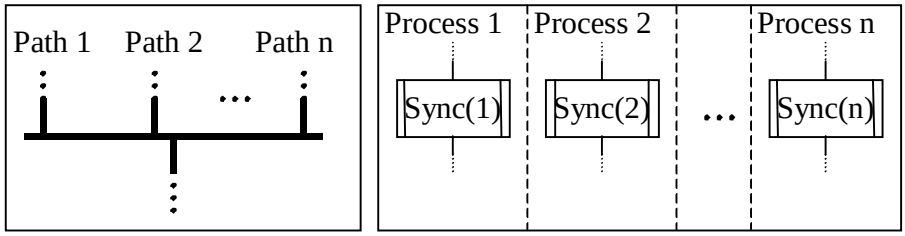


Figure 3.9 - UCM And-join transformation into SDL synchronization

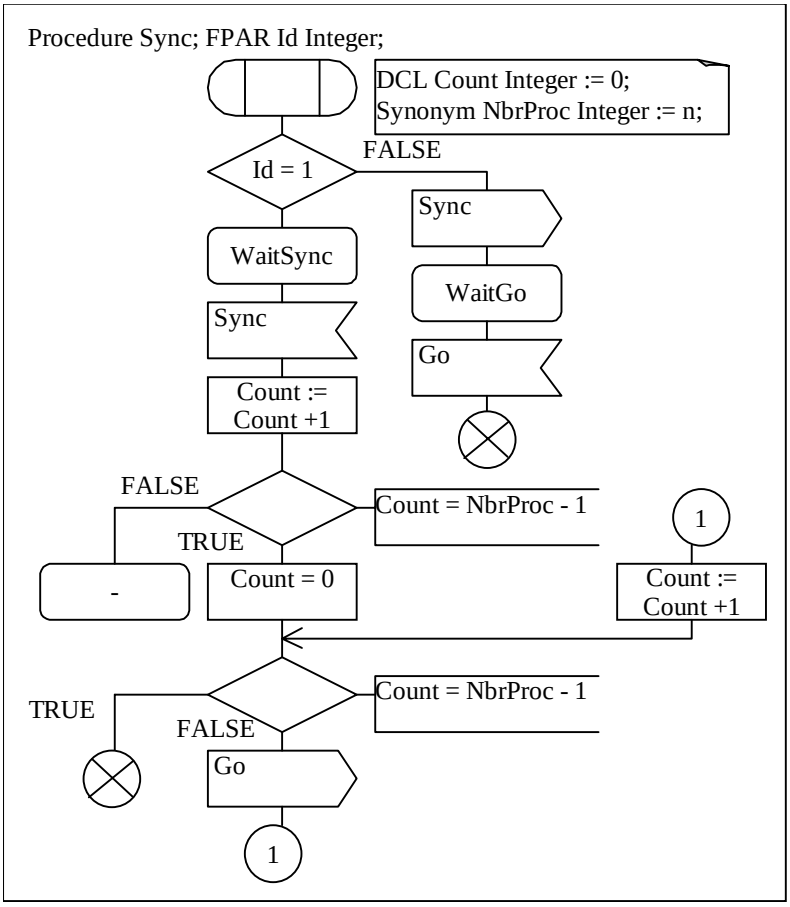


Figure 3.10 - SDL synchronization template procedure

CG12: The timer constructs in UCMs are mapped directly to timers in SDL, as shown in Figure 3.11. In (a) we see the timer being transformed to *Set* in SDL. In (b) there

is the timer expiration with an *input* symbol in SDL. Finally (c) shows the *reset* timer event.

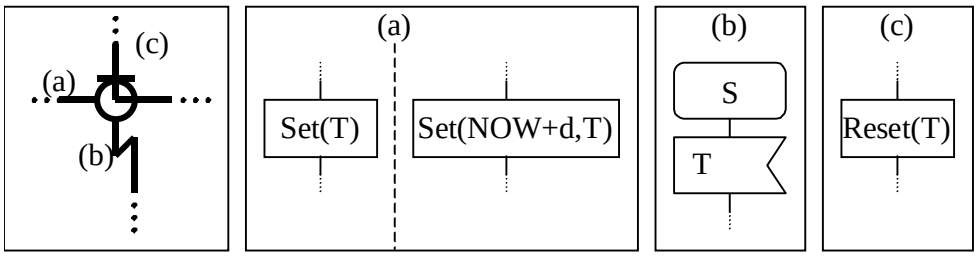


Figure 3.11 - UCM Timer transformation into SDL timer

The architecture of the system remains to be discussed. Unlike the synthesis technique used by the MOST tool, we do not bind any architectural constraints. This is one problem that must be solved by the designer. We believe this will not pose a problem for most designers, because structuring *processes*, *blocks*, and *systems* is straightforward in SDL. It is straightforward to design the SDL architecture because it can imitate the Use Case Maps component structure (known as *bound* UCMs) [53].

3.4.3. Parallel Activities

The parallel activities carried out during the development process are shown below. They are *Traceability*, *Commenting*, and *Logging*.

Traceability

This is the capability of quickly and consistently linking different models and a standard document together. This can be done *manually* or *semi-automatically*.

Manual traceability consists of the developer inserting comments (remarks or references) in the *destination* models that point to the corresponding points in the *source* models. As the models change, this traceability has to be consistent, and therefore must be maintained by the developer. This can be quite a burdensome process. Furthermore the standards document must be easily traceable too, as it is the normative component of a standard. Though burdensome, the process of tracing references between models and the document is very simple. It consists of adding some extra text in terms of comments that refer to the particular related section of the document as shown in Figure 3.1.

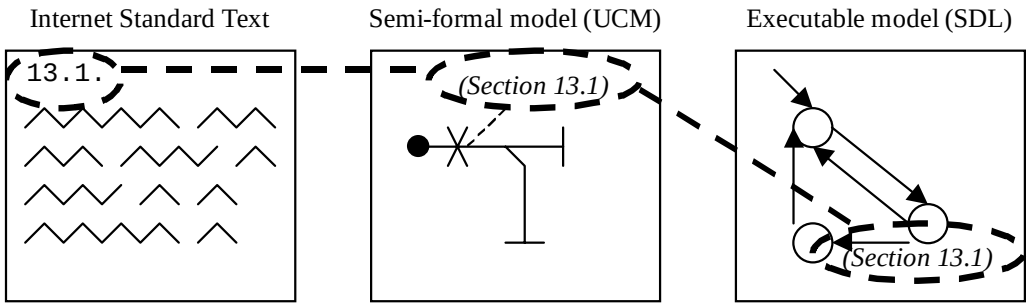


Figure 3.1 - Traceability in the bridging methodology

Semi-automatic traceability is the means of having the traceability described above with tool support for inserting, comments and references. This is basically the manual traceability process assisted by a machine. The major advantage is instant access of the target reference.

In our bridging methodology we recommend using *manual* traceability for the following reasons:

- It allows more flexibility in assigning cross-references.
- It is Tool-independent
- It is applicable to different notations and different tools. *Semi-automatic* traceability is bound to one tool (does not cross between the notations of different tools).
- It enables permits traceability even to a text document. *Semi-automatic* traceability does too, but requires extra effort with no corresponding gain.

The process of making the semi-formal and formal models traceable is quite simple as depicted in the figure above. This is based on the fact that Internet standards are well-structured and organized into meaningul sections and subsections, following best practices [10].

Commenting

Comments are necessary in most projects. Through comments we are able to note all points of disagreement during the development of the standards document. If this is useful for the document, it must also be useful for the models we are constructing, since these will be executed and validated by third parties. However, this is not as straightforward as it seems. The comments must be documented too, and all references

must be specified exactly. Moreover, documentation must be time-stamped because comments may vary in significance over time.

In this bridging methodology process, *comments* are an important artefact. Comments can be introduced into any of the three notations: natural language, UCMs, and SDL.

To keep track of key decisions and their rationales, we found that the best practice is to use a very efficient manual method. Design issues are often raised by questions. These should be documented by adding comments next to the text with clear references of *page number*, *paragraph number*, *text*, *question* and *possible answer*. An example is shown in Figure 3.2.

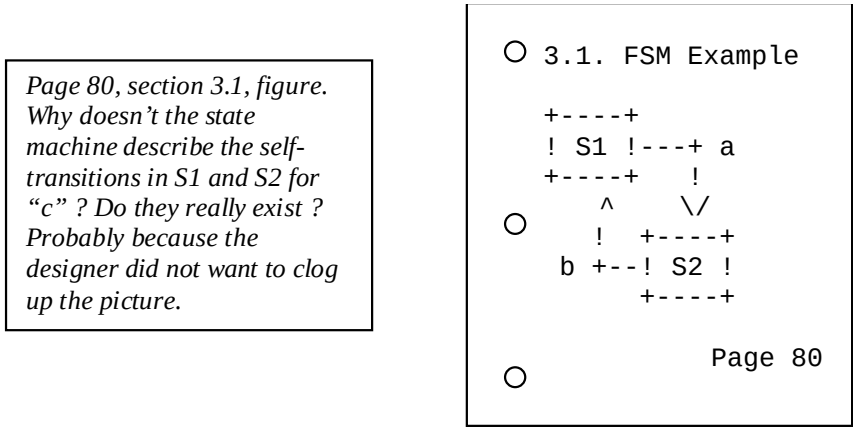


Figure 3.2 - Example of externally commenting a standard document

Activity Logging

Logging is a very inexpensive way of keeping track of the progress of a project. The Personal Software Process, PSP [23] is heavily based on *activity logging*, and our *logging* strategy was taken from the PSP. For standards development, *logging* is specially important because it shows in detail who worked on what portions of a standard project. As in any project, we must be able to identify the person responsible for the work. Logging the process will provide this information, and in addition it provides some other valuable information which may be overlooked, such as:

- Amount of effort put into the project
- Order of achievement of milestones

- Detailed description of tasks, including resource allocations and design decisions
- Rationale for Design decisions
- Reviewing of work on the fly
- Duration of Task
- Pitfalls avoided and workarounds taken

The value gained from *activity logging* is more accurately seen when used to keep track of a description technique model. Few tools support logging; even the ones that support it, restrict the logging unnecessarily, making it inappropriate for the purposes mentioned above. We recommend a way of logging the activities which is not bound to any automated tools. From experience, we have found it is enough to keep track of *Date*, *time*, and *description of the activity*. For example, the duration of an activity is the difference between its *time* and the *time* of the next activity.

We have also found that such a methodology helped us focus on the specific activities under development, as it constantly requires attention redirected to the log and potential review of the task or activity just finished.

We present in the next chapter a case study where we applied our USHLTD bridging methodology.

4. Case Study: Application to OSPF

4.1. Introduction

We new present the case study. Section 4.2 presents the purpose and motivation for the choice of case study as well as an overview of OSPF (Open Shortest Path First), the LSA (Link State Advertisement) refreshment function, and the assumptions and constraints required for the case study. Section 4.3 presents the experience details of conducting the case study.

4.2. Introduction to Case Study: LSA refreshment in OSPF

We use the bridging methodology described in this thesis to develop the *Link State Advertisement efficient refreshment function* for the Open Shortest Path First Internet routing protocol [46],[65].

4.2.1. Purpose of Case Study

We wanted to test our bridging methodology on a realistic example, and to gather sufficient data to evaluate the effectiveness and efficiency. As well, any aspects which needed improving could be identified and result in enhancements to the methodology.

4.2.2. Justification for selecting OSPF and the LSA refreshment function

These are the reasons for our choice. OSPF/LSA is:

- *Representative of IETF work*. We wanted to show that the methodology is applicable to Internet Protocol Standards development.
- *Challenging*. OSPF is a large IETF standardized protocol, tolerant of many exceptions and of different network configurations.
- *Timely*. OSPF is a fairly new routing protocol widely deployed over many networks on the Internet.
- *Manageable*. The LSA refreshment function seemed a tractable problem to tackle inside the OSPF standard (not too large nor complex).
- *Feasible*. It is a small problem within a larger problem that could be done in a reasonable time frame.
- *Opportunistic*. An opportunity to introduce SFDT and FDT into the Internet Standards development process because of expected benefits from the case study.

4.2.3. An overview of Open Shortest Path First

Figure 4.1 shows an overview of an OSPF router and its connections to the attached networks. It contains many functions, such as the *LSA refreshment* function and the *flooding* function. Inside an OSPF router, there are *interface* data structures and *neighbour* data structures. An *interface* data structure is associated with each *physical* interface of the router. Each *neighbour* data structure is associated with a *neighbouring* router found through an *interface*. The *links* connect router *physical interfaces*. Through a link, OSPF may send 5 types of packets: *Hello* packets, *Database Description* packets, *Link-State Request* packets, *Link-State Update* packets, and *Link-State Acknowledgement* packets.

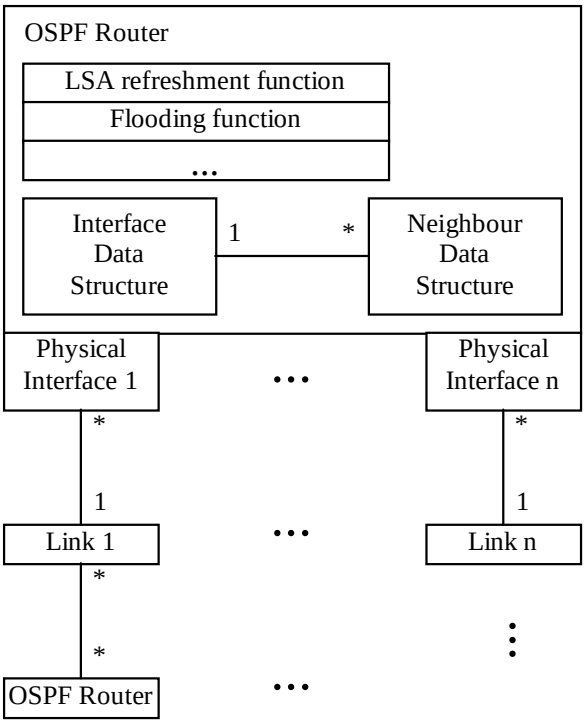


Figure 4.1 - Overview of an OSPF router

The Open Shortest Path First (OSPF) routing protocol is responsible for the dynamic routing of Internet Packets (IP). The description of OSPF can be found in [46], and a more readable format in [44]. OSPF is known as a *link-state* protocol. A *link-state* is what describes the status of a link (i.e. whether a link is *up* or *down*, its *cost* and so on). This protocol is based on a distributed database of router *link-states*. For routers to communicate, they must keep track of their links' status because each router is connected

to other routers through these links. A router can also be directly connected to a network. However, this case is not considered in our case study.

Assume a very simple network topology like the one in Figure 4.2. There are two paths for a packet to go from A to B, namely R1→R3 and R1→R2→R3. Suppose the cost of transmission in link L1 = 3, in L2 = 3, and L3 = 2. Then the cost of R1→R3 is 3, while the cost of R1→R2→R3 is 5. Suppose that L1 fails to carry packets, or its cost raises above 5, then the packets from R1 to R3 should start to be routed through R2 (fine dashed line).

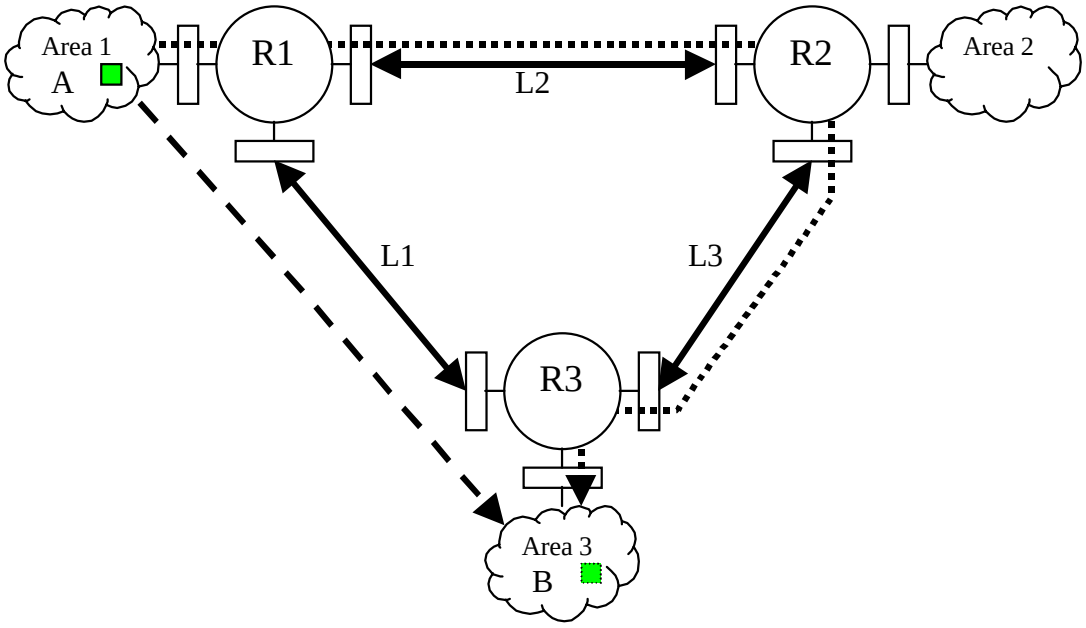


Figure 4.2 – A simplified OSPF Network Topology

We now describe the key behaviours including data structures of the OSPF protocol.

Generic Behaviour

The purpose of OSPF is to ensure that any packet travelling in the network is routed through the best path (shortest) possible. The OSPF shortest path calculation uses Dyjkstra's algorithm. We will not focus on routing table calculation since it is not a *functional requirement* and is well described in the OSPF texts. But, in order to have the data to calculate this routing table each router must have the data for all the links in the network.

In an OSPF router there are *neighbour* and *interface* data structures. An *interface* data structure is associated with each one of the router's physical interfaces to other routers or networks. A *neighbour* data structure is associated with every neighbouring router found through a particular *interface*. The OSPF standard describes a state machine for both of these data structures. However, these state machines only describe the triggering events (as shown in Figure 4.3 and Figure 4.4), not depicting any of the actions executed in a transition. In addition, these are the only two state machines in the RFC and have been extensively enhanced from the originals. It is hard to go directly from the RFC as written to SDL. The UCMs are a very useful intermediate stage.

Interface data structure state machine

The state machine depicted in Figure 4.3 is straightforward to understand. When the *interface* is brought up, it checks whether it is attached to a *Point-to-Point* or another type of link. If attached to a *Point-to-Point* it goes to state **Point-To-Point**. Otherwise it tries to become a *designated router*. More details about this state machine are not relevant for this thesis. Figure 4.3 is an enhanced graphical version of the ASCII diagram in the standard.

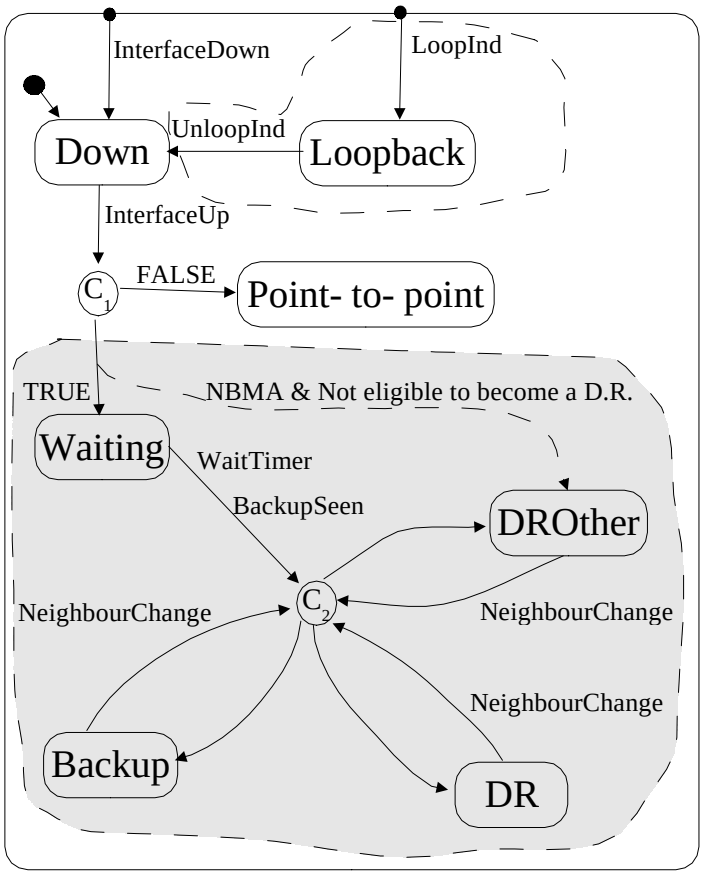


Figure 4.3 - OSPF Interface Data Structure State Machine

Neighbour data structure state machine

The *neighbour* data structure is responsible for *establishing adjacencies* with the neighbouring routers, *exchanging databases* and *updating link-state advertisements* on request. Figure 4.4 is a graphical representation of the *neighbour* state machine presented in the RFC.

When an *interface* identifies a *Hello Packet*, it sends a **HelloReceived** to the corresponding *neighbour* data structure. The *neighbour* then waits for a **2-WayReceived** from the interface. **2-WayReceived** means that this router knows that the neighbouring router sees it. This process gets more complicated and will not be discussed in further detail here. See [46] for more information.

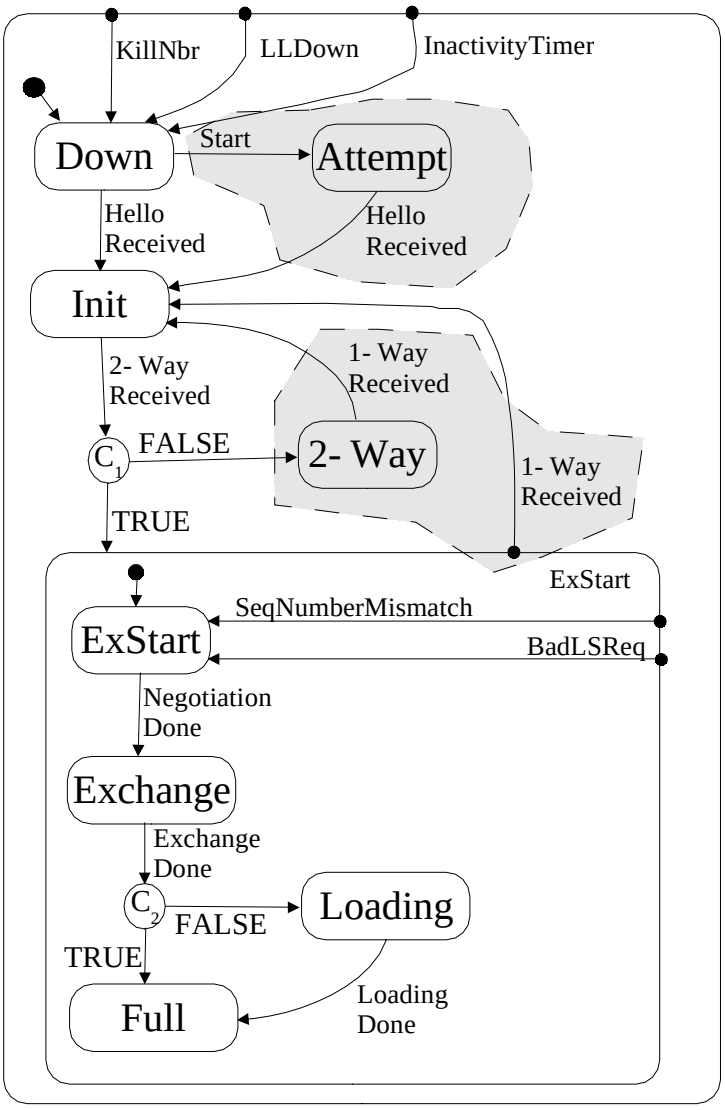


Figure 4.4 - OSPF Neighbour Data Structure State Machine

We now present a specific function studied in this case study. This function is responsible for periodically refreshing the router's database.

Link State Advertisement Refreshment Function

The Link-State information is passed within Link State Advertisements (LSAs). A router generates the LSAs for itself and for its attached networks only. These are called *originated LSAs*. This information is passed to the other routers by a process called *flooding*. *Flooding* is the act of sending an LSA or a group of LSAs to the neighbour routers. The neighbour routers then *flood* the received LSA or group of LSAs to the their neighbouring routers, and so on, until all routers in the network have a copy of the

flooded LSAs. To ensure all routers have the same database of LSAs, a periodic flood of LSAs is required. For example, R1, R2 and R3 (Figure 4.2) have to communicate their LSA databases to each other periodically. OSPF defines that the LSAs should be refreshed every 30 minutes. However, [65] has identified that Area-Routers tend to have enormous databases of LSAs describing their attached networks. The amount of LSAs flushed at once may overload the network causing congestion and degradation of performance. In fact, depending on the implementation of the OSPF router, this network overload may render the router unavailable for a few moments or even cause it to crash.

The main reason for choosing the LSA periodic refreshment in our case study is to show that our bridging methodology enables a designer to specify and validate a particular protocol behaviour without getting into implementation details. We also present the improvements of the proposed solution (LSA efficient refreshment function *internet-draft*) in comparison with the LSA periodic refreshment (RFC2328).

4.2.4. Case Study Plan: Assumptions and Constraints

Initially we planned to develop a complete specification of OSPF in UCM and SDL, and then specify the LSA efficient refreshment function on top of it. We did so until realizing we would not need all of OSPF for the LSA efficient refreshment function specification. We planned to use our bridging methodology to accomplish the goal of showing that the *internet-draft* proposal was feasible.

In this case study some assumptions had to be made to accomplish the project goals. Also not all of OSPF was specified since it would add more complications to understand (and build) the model while also adding unnecessary overhead to the model execution.

The following assumptions and constraints are considered in our models.

1) Only *Point-to-Point* and *Broadcast* types of links were implemented. These are the simplest OSPF link types. The other link types abstract more possible network topologies and layouts, but are of no relevance to the results sought.
2) The LSAs only contain the *Age*, *Sequence Number*, and *Advertising Router ID*. The other fields are also irrelevant to this case study and would only complicate readability and introduce unnecessary overhead.

3) With focus in the LSA refreshment we do not have the behaviour of most of OSPF since it is not needed. Therefore, we assume the routers are already started with all the LSAs in their databases.
4) The routers are numbered 10.0.0.*X*, where *X* is the router number. The router area LSAs contained in its database are numbered 10.0.*X*.*Y*, where $1 \leq Y \leq n$. *Y* is the number of the LSA and *n* is the number of LSAs. The reasoning is to avoid having to manually introduce each one of the LSAs manually. This can be easily modified.
5) The routers and the links never fail. This allows for the experiments to be focused on the network exhaustion possibility. This assumption also eliminates the possibility of unexpected behaviour caused by other OSPF functions.

4.3. Conduction of Case Study

This section details the execution of the case study. It shows in detail how each of the steps from our methodology were used. In this section we refer to two types of Internet standard text: *RFC* (Request For Comments) and *Internet-drafts*. (Refer to section 2.2 for the definitions of *RFCs* and *Internet-drafts*). We have classified our RFC models as *generic behaviour* and our Internet-draft models as *Link State Advertisement refreshment function*.

4.3.1. Preparation Phase

Before starting we were provided with the *Internet-draft* to be studied and specified, and provided with a book [44] on the OSPF protocol. We had at our disposal two machines, one Windows™ based and one Solaris™ based. Also we were provided with the Telelogic TAU toolset (versions 3.6, 4.0 and 4.1). We then started looking for the Standard text from the IETF website. These texts are found in [45], [46] and [65].

We proceed now to describe the OSPF/LSA specific stages in the USHLTD Bridging methodology illustrated in Figure 3.1.

4.3.2. Step 1: Extraction of Use Case Maps from the Standard

This step involves Step 1 and Step 1’ from our bridging methodology process (Figure 3.1). We describe this step for the *generic behaviour* and then we describe this step for the *Link State Advertisement refreshment function*.

Generic Behaviour (1)

There are several OSPF behaviours that are easily understandable with UCMs, as opposed to the textual description. Figure 4.1 presents the textual description of how to know if an adjacency between two routers should be established. This was taken from RFC2178 [45]. It was the first RFC we started the case study with, and is typical of descriptions of procedures in the standard.

```
10.4.  Whether to become adjacent

   Adjacencies are established with some subset of the router's
   neighbors.  Routers connected by point-to-point networks, Point-to-
   MultiPoint networks and virtual links always become adjacent.  On
   broadcast and NBMA networks, all routers become adjacent to both the
   Designated Router and the Backup Designated Router.



Moy                          Standards Track                    [Page 82]
_
RFC 2178                      OSPF Version 2                    July 1997


   The adjacency-forming decision occurs in two places in the neighbor
   state machine.  First, when bidirectional communication is initially
   established with the neighbor, and secondly, when the identity of the
   attached network's (Backup) Designated Router changes.  If the
   decision is made to not attempt an adjacency, the state of the
   neighbor communication stops at 2-Way.

   An adjacency should be established with a bidirectional neighbor when
   at least one of the following conditions holds:

   o   The underlying network type is point-to-point

   o   The underlying network type is Point-to-MultiPoint

   o   The underlying network type is virtual link

   o   The router itself is the Designated Router

   o   The router itself is the Backup Designated Router

   o   The neighboring router is the Designated Router

   o   The neighboring router is the Backup Designated Router
```

Figure 4.1 - RFC Text for "Whether to become adjacent"

Figure 4.2 illustrates the corresponding UCM extracted from Figure 4.1 (section 10.4 in the RFC2178). It is straightforward to understand if the reader is accustomed with the UCM notation. Basically it is a graphical representation of the choices depicted in the natural language text, augmented by explanatory comments sprinkled throughout the text.

The process of deriving UCMs from natural language text is highly subjective, and requires operational knowledge of UCMs.

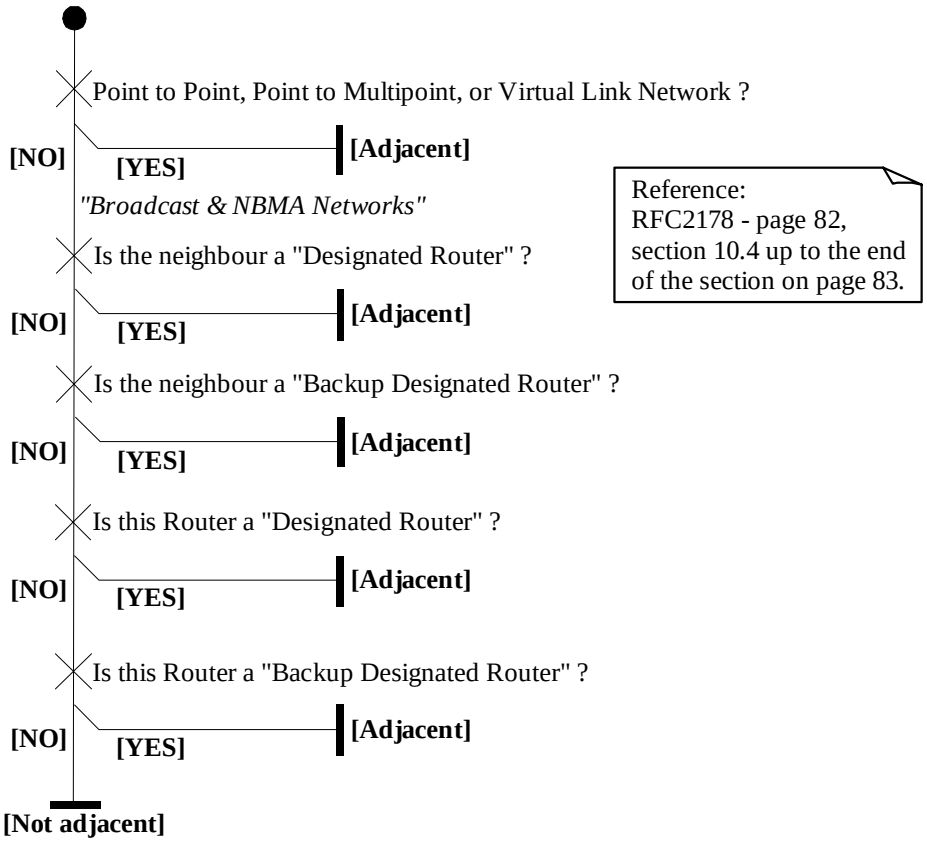


Figure 4.2 - OSPF UCM Plug-in "Whether to become adjacent"

To establish an adjacency (Figure 4.2), the two routers must perform some actions. If the routers are connected through a *Point-to-Point*, *Point-to-Multipoint* or *Virtual-link*, the adjacency is always established. But when connected to broadcast links or NBMA (Non-Broadcast Multi-Access) links, the adjacency upbringing is more complicated because several routers may be connected to one link. Therefore the protocol requires *electing* a *Designated Router*, and a *Backup Designated Router*. This is expressed in the standard and appears in Figure 4.3 (*Interface* data structure state machine) [46].

Interface data structure UCM

In Figure 4.3, we can see more details about the *interface* data structure because they are drawn in UCMs. We see that every time the interface is initialized, a **ResetData** event happens. When brought-up, the Hello protocol starts to periodically send out *Hello*

Packets through that interface. At any time the Hello start-point can be triggered by an incoming *Hello packet*. In this case, the receiving router must analyze the *Hello packet* and identify if it itself has been seen yet (**2-WayReceived**) by its neighbour (See also Figure 3.1 in chapter 3). More discussion on the behaviour of the interface would not be appropriate because the Use Case Map represents several pages of the standard. This step relies heavily on the understanding of the UCM designer, and his/her design decisions. We tried to depict as much information as possible in our UCMs. We believe this description of the interface data structure is accurate and valid, i.e., corresponding to the text, based on extensive manual cross-validation between actions/events in the UCMs and the standard.

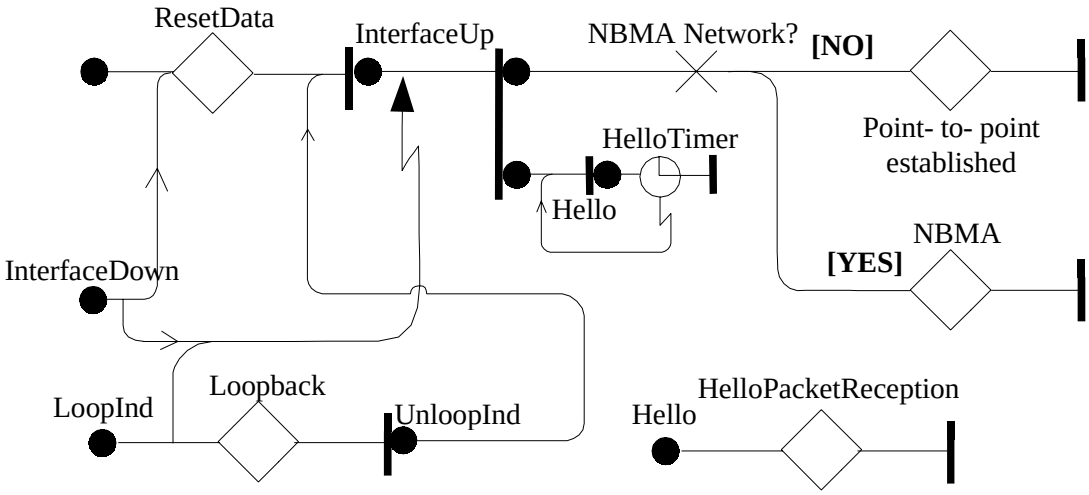


Figure 4.3 - OSPF Interface Data Structure Use Case Map

We may compare the UCM in Figure 4.3 with the state machine in Figure 4.3 as follows. State machines can clearly show the state the machine is in at a certain moment, but does not include any detailed information on the actions for each transition. The UCM can show the flow of events, but cannot show which state the machine is in.

Neighbour data structure UCM

Figure 4.4 shows in more detail the behaviour of the *neighbour* data structure. In a normal case, the following behaviour happens: the interface starts the neighbour; the neighbour negotiates if there is 2-Way visibility between them; the two neighbouring routers exchange database description packets; they exchange database discrepancies (*Link-State Request*, *Link-State Update* and *Link-State Acknowledgement* packets); then

they are ready to route Internet packets (traffic). The full description of a *neighbour* data structure requires several pages of textual description and will not be further detailed here. However, it is easy to understand the OSPF neighbour data structure by following the UCM flow depicted in Figure 4.4 (see also chapter 2 for a description and references on the UCM notation) together with its state machine in Figure 4.4.

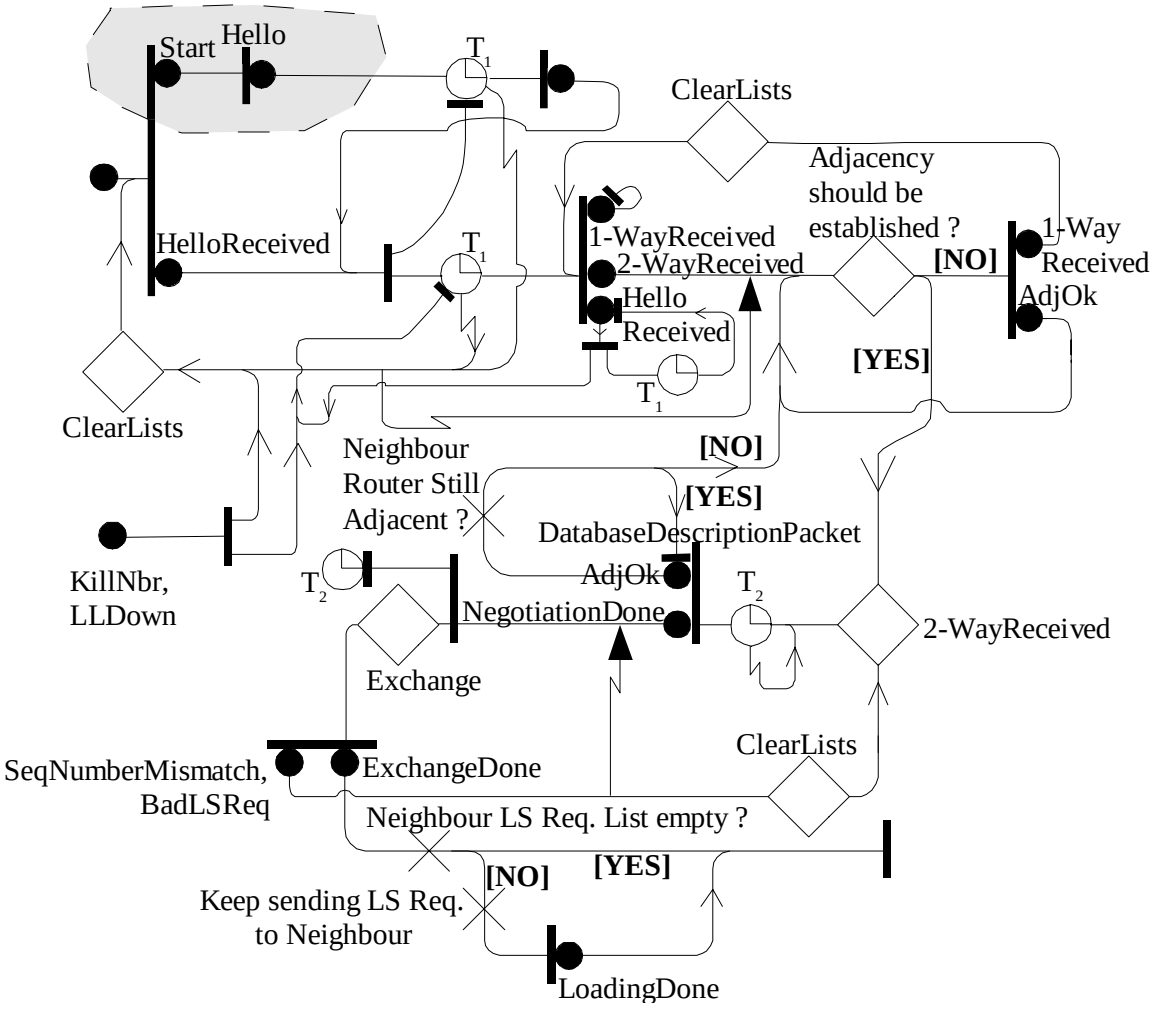


Figure 4.4 - OSPF Neighbour Data Structure Use Case Map

Note that in this UCM there are two timers. These timers are responsible for ensuring that the *neighbour* adjacency still exists, i.e., checking that there are incoming signals from the *neighbouring* router before timer expiration. These timers are not described in the state machine (Figure 4.4).

There are several other Use Case Maps derived from the standard. Many of these UCMs are shown in Appendix B. This concludes our brief description of OSPF. We now move to the specific behaviour from the *internet-draft* [65].

Link State Advertisement refreshment function (1)

The standard has been observed to contain a possible problem with the LSA refreshment function, namely the flooding of a large number of LSAs in a small time interval, which may cause network and router congestion. To correct this, an *internet-draft* was issued on the efficient LSA refreshment function [65]. The studied *internet-draft* [65] describes the network exhaustion possibilities in OSPF. We need to describe in UCMs what was meant in the *internet-draft*. We need to describe the normal behaviour (as in the RFC) and the behaviour proposed in the *internet-draft*. These are shown from Figure 4.5 to Figure 4.9.

Figure 4.5 - LSA Refreshment UCM Calculate Delay (Normal)

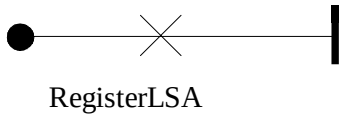


Figure 4.6 - LSA Refreshment UCM RegisterOriginatedLSA (Normal)

It is straightforward to understand the normal case behaviour. The delay to which an LSA will take to be refreshed (Figure 4.5) is always constant and equal to OSPF_LS_REFRESH_TIME (30 minutes by default). To register an LSA in the database (Figure 4.6), the procedure simply includes the LSA in the database. Every OSPF_LS_REFRESH_TIME seconds an LSA refresh occurs. This timing is not shown here because is part of the router behaviour, and not of the **RegisterOriginatedLSA** behaviour.

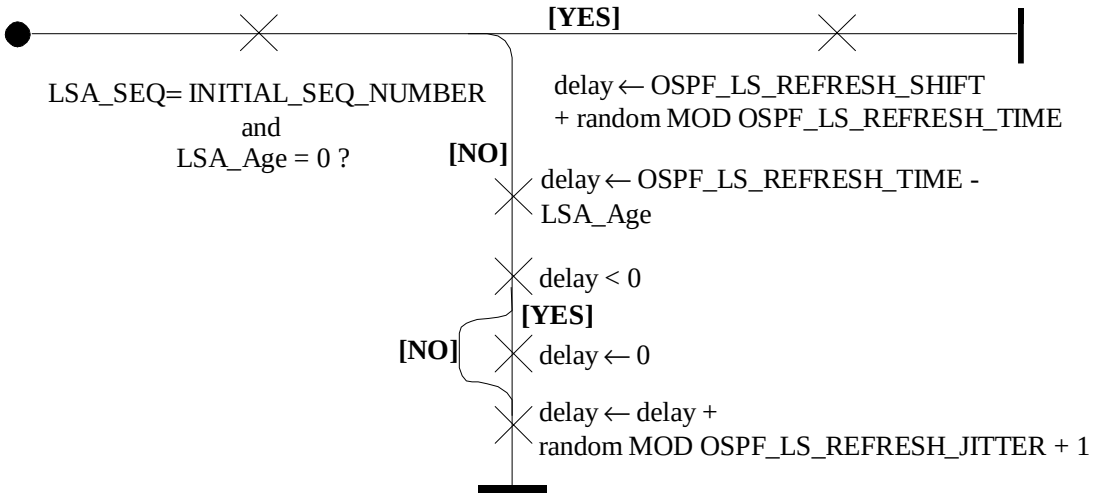


Figure 4.7 - LSA Refreshment UCM Calculate Delay (Zinin)

Figure 4.7 shows the delay calculation procedure proposed in the *internet-draft*. This procedure randomizes the first time an LSA is refreshed and then keeps refreshing it in OSPF_LS_REFRESH_TIME seconds intervals, plus or minus some time to avoid jitter. Another improvement of the RFC by the *internet-draft* is the addition of refreshment groups. Refreshment groups allow the originated LSAs to be grouped and limit the maximum number of LSAs in a flooding operation. In the *internet-draft* proposal, groups of LSAs are created to ensure a maximum throughput of flooded LSAs in the network. For example, not more than OSPF_REFRESH_GROUP_LIMIT LSAs can be grouped together, or no two LSAs in the same group can have more than OSPF_REFRESH_GROUP_AGE_DIF seconds in their LSA age difference. This is shown in Figure 4.8.

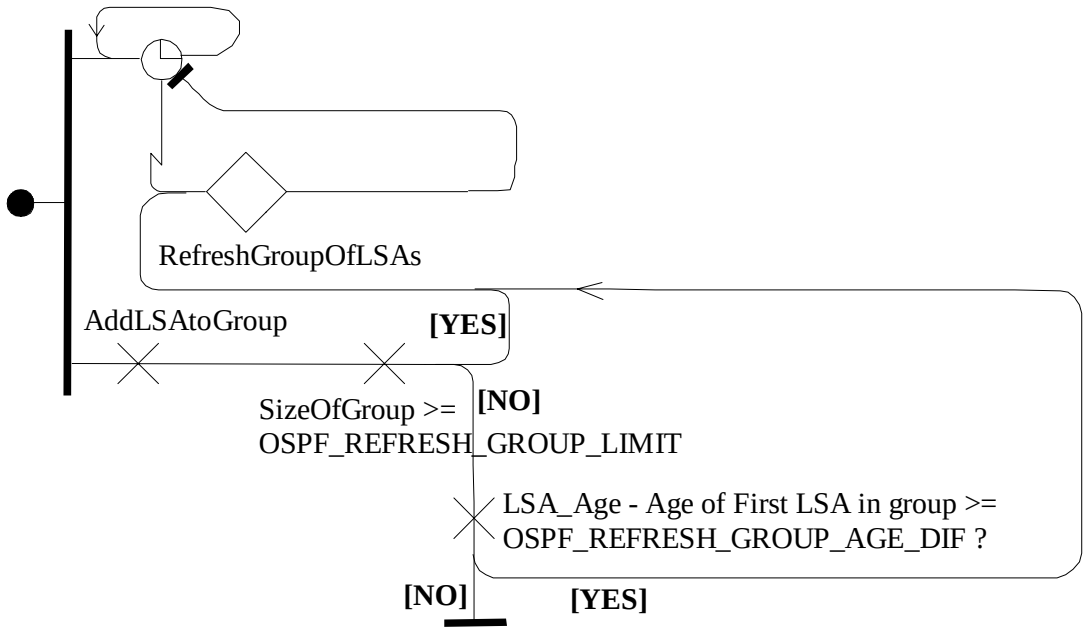


Figure 4.8 - LSA Refresh UCM Register Originated LSA (Zinin)

Every time an LSA is passed for registration (Figure 4.8) the procedure groups it together with other LSAs that fall in the same conditions and schedules the group refreshment. For more details on the groups refer to [65].

Figure 4.9 shows the procedure that floods the LSA groups. It calculates the delay time for a group to be flushed, sets the timer with that delay, and creates a new LSA group. When the timer goes off, the group of LSAs is flushed.

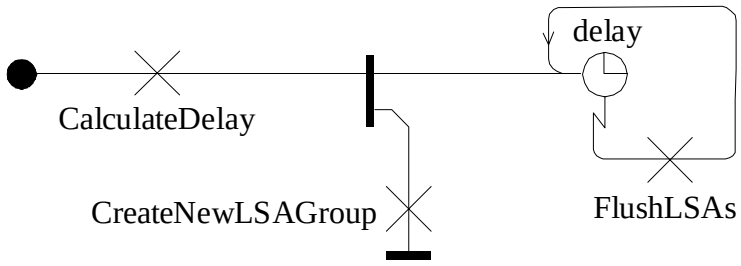


Figure 4.9 - LSA Refresh UCM Refresh Group of LSAs (Zinin)

Step 1 and Step 2 (next) overlapped. We found this overlapping was useful for the iteration cycles back to step 1. We think UCMs and SDL should be designed concurrently, having the SDL design start shortly after the design of UCMs. We believe it is not cost-effective to design all the UCMs and then move on to the SDL. The next step describes in more detail the step from UCMs to SDL.

4.3.3. Step 2: The Bridging from UCMs to SDL

In this project we built two SDL models, one for the *generic behaviour* of OSPF and one for the *Link State Advertisement refreshment function*. We did not have any prior knowledge about the OSPF routing protocol. Therefore we used the UCM semi-formal description technique to model the requirements and learned about OSPF on the go. As previously mentioned, we needed to get an understanding of OSPF prior to being able to specify only the necessary behaviour for the correct and complete LSA refreshment function.

The architectures of the SDL models were derived from our understanding of the OSPF protocol. Our UCMs are *unbound*, representing only a process or a procedure of the protocol per UCM or plug-in. Therefore, we could not derive the structure of the SDL from the UCMs. However, this did not pose a problem, since the structure of our SDL models is very simple.

In the *generic behaviour* the structure consists of a router process, *interface* processes and *neighbour* processes in a router block. The router blocks are linked together with the link blocks. A system consists of router blocks and link blocks. The *Link State Advertisement refreshment function* SDL architecture is simpler. It consists of router processes communicating with link processes, because there was not the need for *interface* or *neighbour* processes this time. To see these architectures please refer to Appendix C. In this case study we focus on the behaviour of the SDL processes.

Generic Behaviour (2)

Almost all of the behaviour described in SDL was derived from the set of Use Case Maps. The remainder of the SDL behaviour is the infrastructure necessary for a working specification. This infrastructure includes the *network topology set-up*, *point-to-point* link and *broadcast* link behaviours among other things. We start this section by showing the SDL behaviour in Figure 4.1, which was derived from the *interface* UCM (see Figure 4.3), and the SDL behaviour in Figure 4.2, which was derived from the *neighbour* UCM (see Figure 4.4).

Deriving Interface SDL process diagram

First, the process starts with some infrastructure SDL behaviour (**IamYourInterfaceRouter** and **IamYourInterfaceLink** calls). Then, as in the Use Case

Map (Figure 4.3), the process calls **ResetData**, and waits for the **InterfaceUp** signal after having been initialized. Then there is the *Hello packet* reception and the choice between going to the **PointToPoint** or **NBMA** states. To the left we can see some exceptional behaviour handling.

Using CG1 (start-point to input) we have defined the input signal **InterfaceUp** in the **Down** state. Then with CG4 (responsibility to stub) we can see a call to **ResetData** after the process **Initialization**. CG5 (or-fork to decision symbol) was used to check the result of the responsibility **NBMANetwork** and take the according path to **PointToPoint** or **NBMA**. To set the **HelloTimer** and make the call to see whether the link is **PointToPoint** or **NBMA** we used CG9 (simple and-fork). Finally to set the **HelloTimer**, reset it, and define the actions taken when the timer expires, we used CG12 (timer).

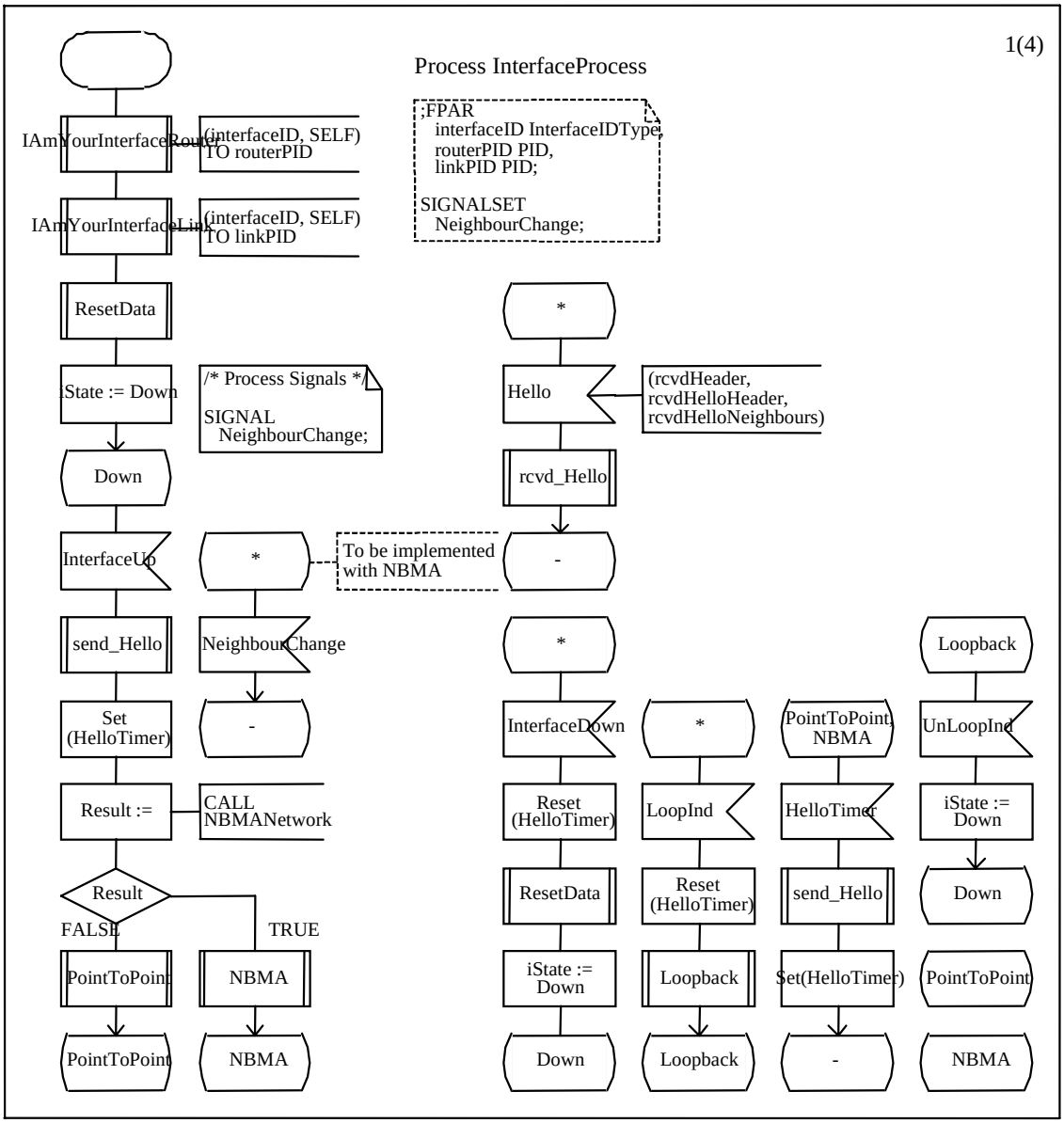


Figure 4.1 - SDL Primary Behaviour for OSPF Interface Data Structure

Deriving Neighbour SDL process diagram

Using CG2 (end-point to output) the *neighbour* sends **NeighbourStateChange** after the call to **NgbrLSReqListEmpty** in case the list is empty. Using CG6 (or-join) we have merged the exits from **2WayReceived** in the event of an **ExchangeDone** event. CG7 (causality preservation) was used throughout the whole process. We can see that by observing that the events in the SDL transitions take place in the same order as the ones in the Use Case Maps.

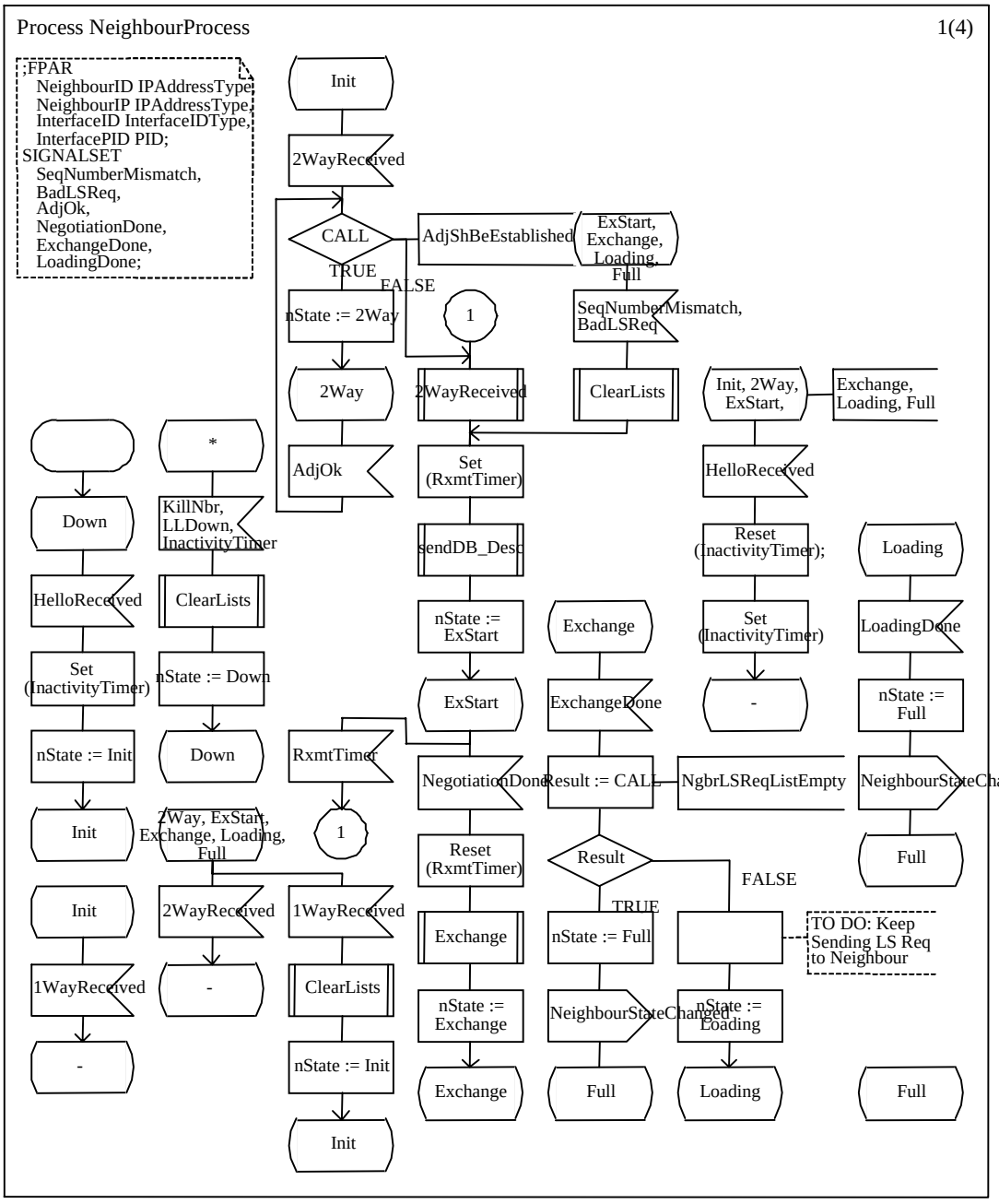


Figure 4.2 - SDL Primary Behaviour for OSPF Neighbour Data Structure

CG8 (and-fork with two simple paths), CG10 (and-fork with general paths) and CG11 (and-join) were not used in this case study. However, as mentioned before they were tested in a simple example, but not needed in any of the processes or procedures we designed for the OSPF model.

Using the Construction Guidelines mentioned earlier (chapter 3), we have created SDL processes and procedures that describe the generic behaviour of OSPF. The majority

of the OSPF SDL model was created from the UCMs describing the *Database Description* packets, *Link State Request* packets, *Link State Update* packets, and *Link State Acknowledgement* packets. These are not shown for their display complexity (see Appendix C).

Link State Advertisement refreshment function (2)

In order to generate a working model for the Link State Advertisement refreshment function alone, we would need to strip down many of the procedures included in the *generic behaviour*. However we decided it would be easier to create a new model and create the needed processes based on our knowledge of OSPF. Therefore we have created the process diagrams in Figure 4.3 (Normal router type) and Figure 4.4 (Zinin router type). This behaviour was not derived directly from any UCMs, but rather from our reasoning of what was needed from an OSPF router to work with the correct Link State Advertisement refreshment function. The transition on these two processes that are triggered by **LSA_Refresh** are the transitions that take care of the refreshment event. In the Normal (Figure 4.3) router type, every time there is a refreshment event, the process calculates the delay, updates the database registering the fresh LSAs and floods the LSAs to the network. In the Zinin (Figure 4.4) router type, the process is similar, but this time there are many more **LSA_Refresh** events happening, each one associated with a group of LSAs (**sentLSAs**).

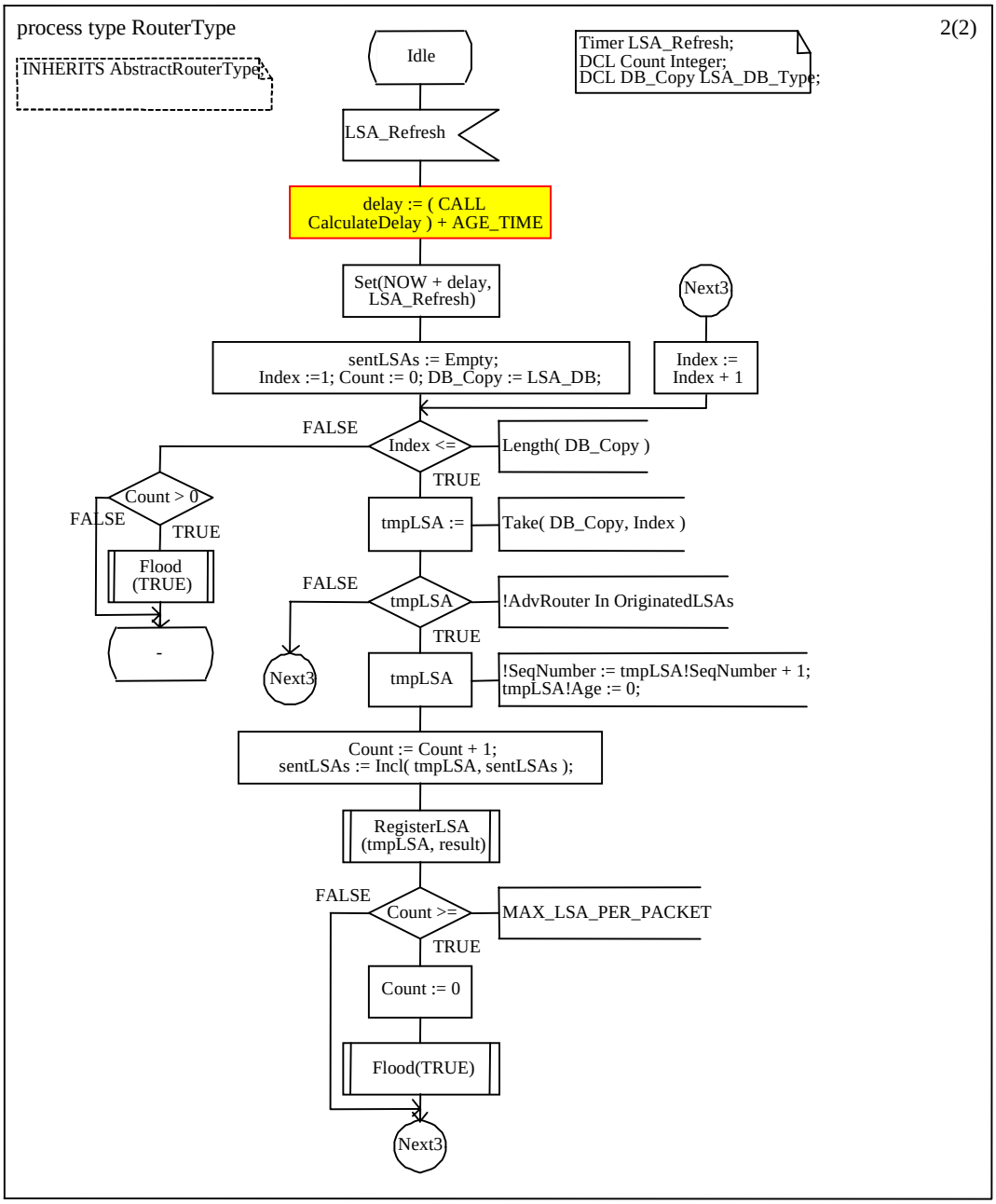


Figure 4.3 - SDL Behaviour of the NormalRouterType

The other SDL model behaviours represent the *LSA periodic refreshment* as follows. **Normal** represents the behaviour from the RFC2328, while **Zinin** represents the behaviour proposed in the *internet-draft*. Figure 4.5 through Figure 4.8 show the SDL created from the *LSA efficient refreshment* UCMs.

Figure 4.3 defines the router behaviour as in the RFC2328 on how to schedule its originated LSA refreshments. So in the event of an **LSA_Refresh** it sends as many of its

originated LSAs per *LS_Update* packet as possible. This process was defined with knowledge of OSPF, rather than an UCM. This is an example of infra-structural SDL behaviour.

Figure 4.4 shows a similar behaviour for the case where the Zinin router must refresh its LSAs. It floods the LSAs and re-originates them, which causes new groups to be formed and new refreshments to be scheduled.

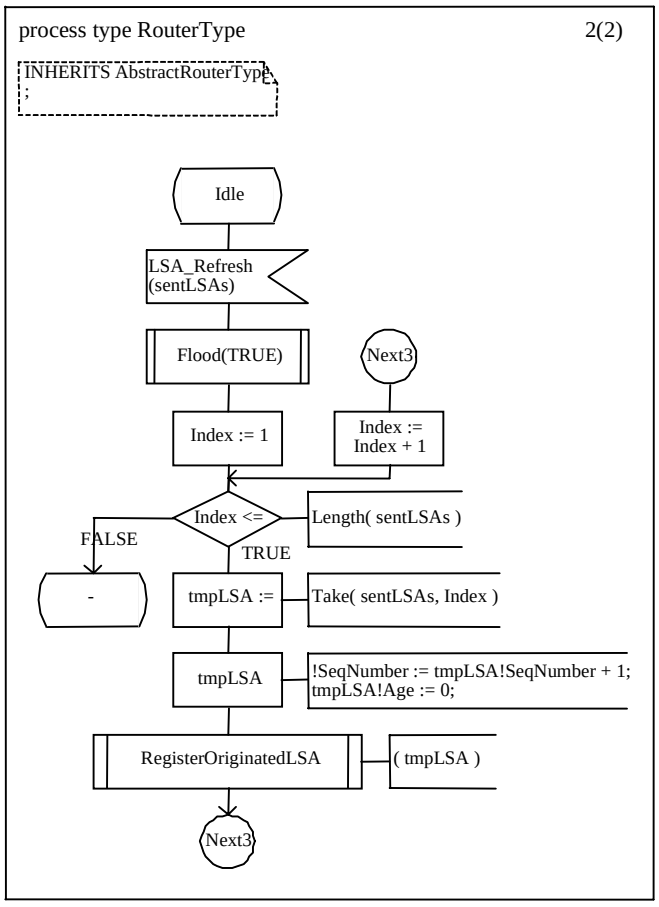


Figure 4.4 - SDL Behaviour for the ZininRouterType

Figure 4.5 shows the SDL corresponding to the UCM in Figure 4.5. The Normal Delay calculation is simply a lookup to a pre-set constant (OSPF_LS_REFRSH_TIME).

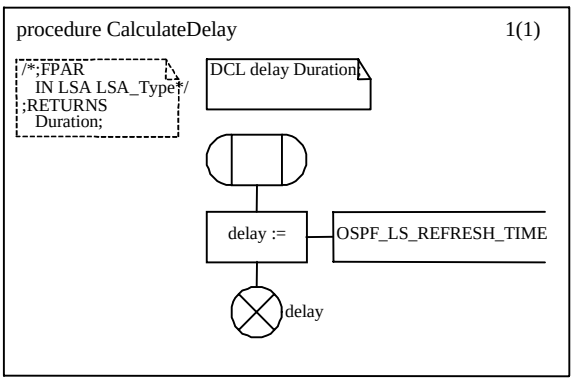


Figure 4.5 - SDL Behaviour for Normal Delay Calculation

Figure 4.6 is the SDL corresponding to the UCM in Figure 4.6. Note that the SDL procedure to Register an LSA includes a loop that analyzes the database. The UCM simply makes a call to this procedure. This procedure too is based on our knowledge of the generic OSPF behaviour, and goes far beyond the simple UCM.

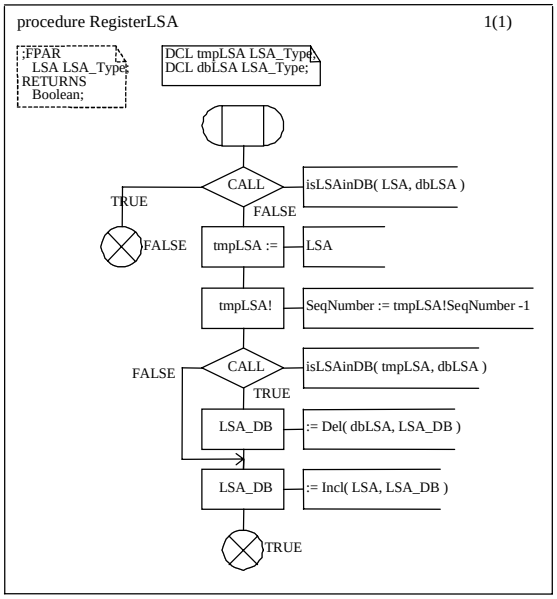


Figure 4.6 - SDL Behaviour for Register LSA

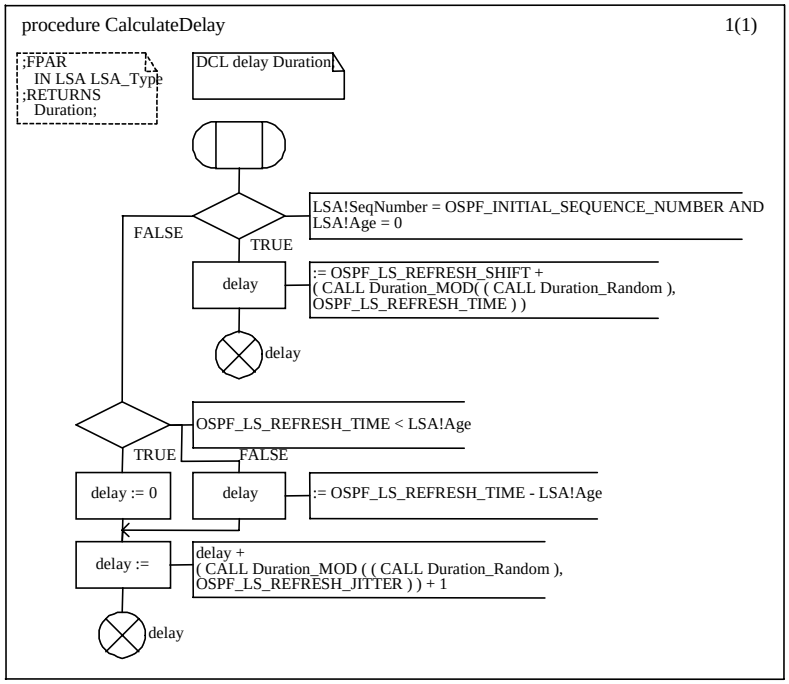


Figure 4.7 - SDL Behaviour for the Zinin Delay Calculation

Figure 4.7 is the SDL corresponding to the UCM in Figure 4.7. It is the Zinin Delay calculation. It calculates the first delay interval for a group of LSAs based on the groups first LSA's Age and Sequence number. If this is the first time an LSA is being originated (Age = 0 and Sequence Number = OSPF_INITIAL_SEQUENCE_NUMBER), the delay is set to a random value between 0 and OSPF_LS_REFRESH_TIME plus OSPF_LS_REFRESH_SHIFT. If not, it is set to OSPF_LS_REFRSH_TIME plus or minus some time to avoid jittery.

Figure 4.8 is the SDL corresponding to the UCM in Figure 4.8. This procedure registers the originated LSA in the router's database, and then includes the LSA in the refresh group if it satisfies the refreshment group conditions. As soon as one of the conditions is not satisfied, the group is closed and its refreshment scheduled. The conditions that close a group have already been discussed, but for more detail on these conditions see [65].

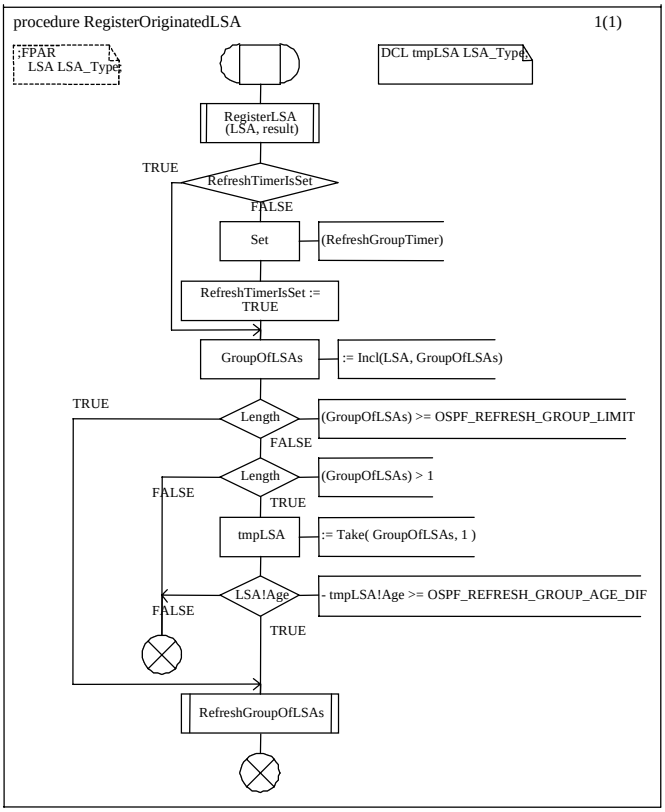


Figure 4.8 - SDL Behaviour for Register Originated LSA

We now present the third step of our methodology applied to our case study. This step is also done while designing UCMs and SDL. We believe it is a good idea to have the validation and simulation overlapping the design of the UCMs and SDL. This allowed us to validate the SDL and the UCM before having the UCM and SDL models completed.

4.3.4. Step 3: Validation via Simulation and Verification Steps

In these steps we validated our model against the requirements (standard). First we validated the *hand-written* UCMs against the texts and state machines. Then we created *digital* UCMs and validated them again. This extensive cross-checking forced us to ensure the UCMs we created conformed to the texts. Then we created the SDL models based on our validated UCMs.

Generic behaviour (3)

We validated the *generic behaviour* directly from the requirements and from the UCMs. We generated MSCs (not shown because of their display complexity) and compared the behaviours in the MSCs with the expected behaviours in the RFC2328 [46]. This

validation was not difficult because it was done in stages. For example, when we wanted to validate the Adjacency upbringing, we would generate a scenario up to that point and validate the messages being passed between routers, interfaces, neighbours, and link processes against what was expected according to what was described in the RFC. We could have used some MSCs from the standard if they were available, but OSPF being such a complex protocol, it is infeasible to have such MSCs in its description. However, we are confident that the extensive manual cross-checking between the SDL model and the UCM model assures us that these two correspond. Also, the same cross-checking from UCMs to the standard permits us to state that the SDL model also corresponds to the standard.

Link State Advertisement Refreshment Function (3)

In this step we found the *internet-draft* proposal was valid. In particular, Zinin's claim that the peaks of LSAs in the network every OSPF_LS_REFRESH_TIME seconds was found to be true in our model. We validated the implementation comparing MSCs generated from the SDL models with behaviours specified in the RFC. These MSCs are not shown here because of their display complexity. We had to observe if the LSAs were correctly being sent through the link processes, and if all the parameters were correct. We simulated three network topologies specified in SDL. Each time an LSA would be sent to the network, the simulation model would register the time it was sent. Counting the number of LSAs in the network at a particular moment led to the graphs in Appendix A. The simulation results revealed that there were peaks of LSAs in the RFC LSA refreshment behaviour, and that they were distributed along time with Zinin's predicted behaviour.

We wanted to show the advantage of using the internet-draft approach. For this we needed to show that there would be a ceiling on the number of LSAs at all times when compared to the RFC (Normal behaviour) refreshment. It can be seen that the Zinin (internet-draft) cases in the simulations graphs always (even the unrealistic cases) has a ceiling lower than the normal cases. For example, consider the graph in Appendix A, page 97. The network periodically flushes about 275 LSAs in a 30 minute interval. In the Normal case, there is a peak of 200 LSAs at a certain time, and for a long time no LSAs.

However, in the Zinin case there are no peaks, and they are equally distributed reaching a maximum of about 15 LSAs.

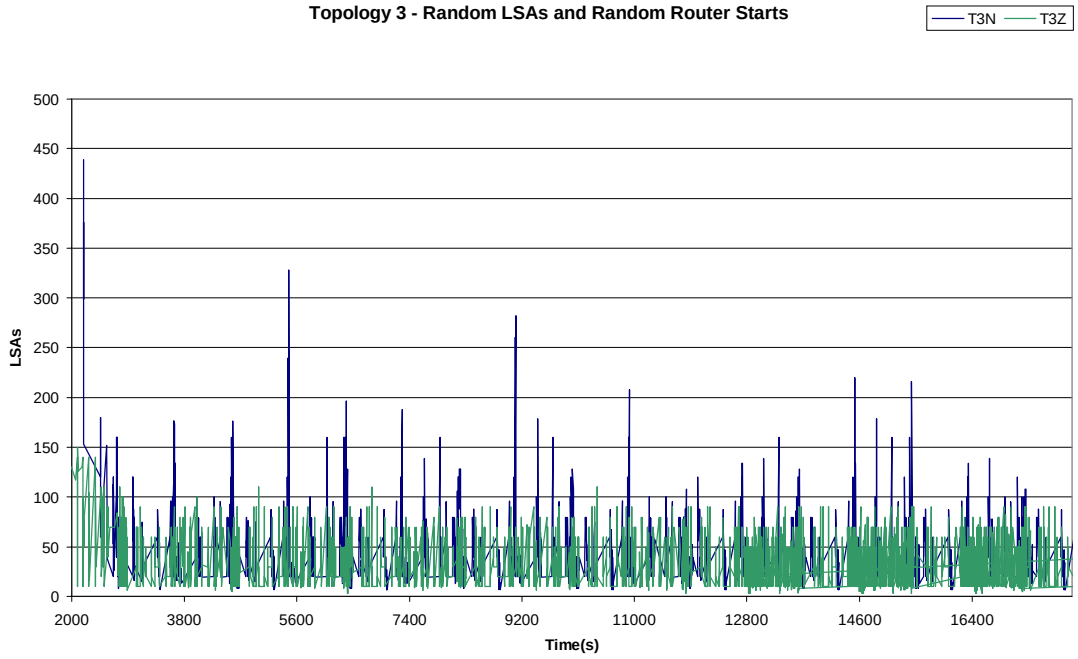


Figure 4.1 - Normal LSA refreshment vs. Zinin LSA refreshment

For example, from Figure 4.1 we can see a realistic comparison between the Normal behaviour and the Zinin behaviour. This case is realistic because it assumes that the routers start at random times, and each of them has a random number of originated LSAs in its database. We can clearly see that Zinin's approach does not allow peaks of LSAs in the network, while in the normal case there are several peaks of LSAs. We considered these results acceptable since for the extensive simulations carried out, the verdict was that the internet-draft proposal was considerably better in most cases, but never worse than the normal cases.

Validation and Verification

By validation we mean cross-checking the models with each other and the standard text, and also running scenarios (via simulation) and comparing them with the procedural descriptions in the standard. The network topology simulations are built by having the SDL models count the number of LSAs at a particular moment in time. The implementation of this involves simply printing the number of LSAs at each moment when there are LSAs travelling the network. In simulation terminology, one tick is 30

minutes (1800) seconds as shown in Figure 4.1. The measurement of goodness of the model was the maximum size of peaks. This measure is related to the reliability of the network. If the peak (number of LSAs) is too large, one or more routers may fail.

Another advantage of using SDL is the availability of tool support for carrying out extensive verification of the SDL design. We have run several times the automated bit-state search and random walk state exploration capabilities of the tool on our SDL models [61]. We achieved 100% coverage reports, and did not encounter any deadlocks or unspecified receptions. Note that the simulation model is built automatically for us by using the TAU tool code generator. This simulation model may be used within TAU's *simulator* tool.

After collecting the simulation and verification results we proceeded to the iteration decision point shown next.

4.3.5. Step 4: Iteration Decision Point (When to Stop)

Generic behaviour (4)

The first decision point when to stop occurred when we realized that specifying OSPF completely was not going to be cost-effective. We realized that it would require several more months to specify and validate all of OSPF before we would be able to change its refreshment function. Nevertheless, we had also realized we had enough understanding of the protocol that we could specify only its refreshment function apart from all its other behaviours.

Link State Advertisement refreshment function (4)

We established a new plan to specify only the LSA efficient refreshment function. We kept coming back and refining the UCMs and the SDL in iterations. We finally decided to stop again when we had the first set of simulation results from our SDL models. We presented these results to the OSPF IETF working group [42], where they required us to assign several LSAs per router instead of only one.

We came back to the process and specified the changes for more LSAs per router. This time the simulation results were sufficient to prove the feasibility of the *internet-draft* proposal [43]. This proposal is now in the process of becoming an Informational RFC. This Informational RFC includes the UCM and SDL models, as well as simulation

results and will be recognized as a contribution to the standard [66]. This recognition is an indication of the value of the contributions of this thesis to the IETF standards process.

This concludes the discussion of the details of conducting the case study. In the next chapter, we evaluate the effectiveness of the USHLTD bridging methodology based on a comparison with the other bridging methodologies, and the analysis of the results of our case study.

5. Results of Case Study and Assessment Of Bridging Methodology

5.1. Introduction

This chapter presents a preliminary assessment of the methodology. In section 5.2 we compare our bridging methodology and the others discussed in chapter 3. In section 5.3, we describe the observations and preliminary conclusions from the case study. Section 5.4 presents an overall assessment of the UCM/SDL HLTD bridging methodology. In section 5.5 we present recommendations for the improvement process of IETF standards.

5.2. Comparison of USHLTD Bridging Methodology to others

in Table 5-1, we summarize a comparison of the bridging methodologies described in chapter 3 with USHLTD, our proposed methodology. For a review of semi-formal and formal description techniques, please see section 2.4. First we present the comparison criteria.

5.2.1. Comparison Criteria

Here we present the criteria selected for this comparison and justify each selection.

We feel that it is important to use a *semi-formal* description technique as the starting point of the development bridge. Some of the other techniques do the same. One key benefit of starting with a *semi-formal* description technique is the ability to capture the essence of a requirement without becoming bogged down in detail which is not needed at this phase of development. Thus the level of formality and completeness is a criterion. More on *semi-formal* description techniques is presented in section 2.4.1.

The other criteria are described in section 3.2, and are briefly reviewed below and justified for application to Internet protocols.

A *scenario-driven* approach focuses on the scenarios as definitely representing the functional requirements. A *responsibility-driven* approach focuses on the functions performed and functional system components. A *hybrid* approach is a mixture of the *scenario-driven* and *responsibility-driven* approaches. This is an important comparison criterion because of the resemblance of the external events and reactions to scenarios, and because of the resemblance of the internals of the system under development to procedural descriptions. Moreover, the Internet RFCs emphasize procedures.

When a bridging methodology is an *iterative development methodology* it can be used as the part of development process for a project. More on *iterative development methodologies* is found in section 3.2. We believe this criterion is important because it helps to denote whether or not refinements are necessary in the bridging. Refinement methodologies are more realistic, because it is common to discover and repair omissions or inconsistencies during iterative development.

Synthetic and *Analytic* approaches were explained in chapter 3 (section 3.2). We feel it is an important criterion because it indicates whether the process allows on-the-fly design decisions or not. Such decision making respects the designer independence.

An *automated* methodology indicates whether the process is automated (tool supported) or not. More information on the definition of automated methodologies is found in section 3.2. We believe it is important to indicate whether a methodology is automated or not because automation reduces the number of faults injected during the bridging process, and saves time by requiring much less human intervention in the bridging process.

The *executability of a notation* of a bridging methodology indicates which of the bridging models can be simulated and validated. More on this key ability to run simulations on a model is described in 3.2. We believe it is important to include this in our criteria because of the current availability of industrial-strength tools for such notations. The benefits of using such tools is section 2.5.

5.2.2. Comparison and Discussion

Table 5-1 summarizes the properties of each of the bridging methodologies with respect to the comparison criteria. We now give the justification for our assignment of properties to the methods, and compare them to USHTLD. We also give examples where applicable.

Methodology	Parameter					
	i) Uses a Semi-Formal Description Technique for source mode?	**ii) S**cenario-driven **R**esponsibility-driven **H**ybrid approach	iii) Iterative Development methodology ?	**iv) S**ynthetic or **A**nalytic ?	v) Automated ?	vi) The Executable Destination Description Technique has Industrial-Strength tools ?
SPEC-VALUE	YES	H	YES	A	NO	NO
RT-TROOP	YES	S	YES	A	NO	YES
MOST	NO	S	NO	S	YES	YES
UCM/SDL HLTD	YES	R	YES	A	NO	YES

Table 5-1 - Comparison of the different bridging methodologies

i) The use of a *semi-formal* description technique: We believe this is necessary because it allows the designer to interpret and write the requirements in a more formal way than natural language. MOST uses Message Sequence Charts (MSCs) as the *source* notation, differently than all the other methodologies. With Use Case Maps we can define high-level flows of particular system components, and even the interaction between system components, without defining any signals between these components. However, with MSCs, we need to precisely define these signals in order for the MSCs to be useful. Finally, it can be seen from Figure 3.1 and from Figure 4.3 that a single UCM is much more expressive than an MSC.

ii) The use of a *responsibility-driven* approach: In a *scenario-driven* approach and in a *hybrid* approach, the designer must have many scenarios made available to him in order to complete the design. In a *responsibility-driven* approach this does not hold. Moreover, Internet standards are often poor in scenario descriptions and rich in procedural descriptions (better for responsibility-driven). For example, in [46] there are several procedural descriptions and only one scenario description.

iii) The use of an *iterative development methodology*: This is important to allow iterative refinement our requirements and design models until we obtain the desired models.

Without an *iterative development methodology* the designer is limited to narrowing the usability of the bridging methodology as simple transformation rules.

iv) The use of an *analytic* approach: A *synthetic* approach is limited to a set of transformation rules that does not leave room for design decisions. This way, if the *synthesized* model is not exactly what the designer expected, he/she may have to change the requirements model (*synthesis* source model). With an *analytic* approach the designer is involved in the construction of the target model at all times, and is directly responsible for all design decisions.

v) The use of a *manual* bridging approach: *Automated* bridging approaches save time because they automatically transform the *source* model into the *destination* model. However, if the *source* model is written in a semi-formal notation, this may lead to omissions in the *destination* model. A *manual* approach allows the designer to keep track of all the design decisions, and monitor and measure the project activities. We believe manual bridging approaches allow more control.

vi) The use of *industrial-strength* tools for executable description techniques: This guarantees that the executable model generated with a bridging methodology will have adequate support. Without such support, the generated models may lack facilities for validation, verification, and automatic code generation, as described in chapter 2.

We can see that our USHLTD bridging methodology, uses a *semi-formal description technique* for the high-level design, it is *responsibility-driven*, is an *interative development methodology*, is *analytic*, is *manual* and its *destination formal description technique* (executable) has *industrial-strength tools* available.

Therefore, by using USHLTD, we can depict the requirements in a high-level way without deciding design details very early in the design stages, we can derive models directly from an Internet standard (poor in scenarios and rich in procedural descriptions), we can affect the whole development process instead of only a notation translation phase, and may combine USHLTD with other methods, we can control and monitor design decisions, we can gather project data during the development process, and finally we can use *industrial-strength* tools for *validation, verification* and *simulation* of the models. Moreover, some of these tools support *automatic code generation*.

Thus, we believe our USHLTD methodology is more appropriate for IETF standards.

5.3. Experimental Results and Observations from Case Study

In the section we present observations (sub-section 5.3.1) and results (section 5.3.2) extracted from the case study.

5.3.1. Observations

This case study consists of constructing two OSPF models (one UCM model, and one SDL model) from RFC2328 [46] (244 pages of ASCII text) and two LSA efficient refreshment models (one UCM model, and one SDL model) from the internet-draft [65] (11 pages of ASCII text).

Constraints

The generic behaviour model was the first attempt to specify OSPF. We wanted to specify all of OSPF and then modify its LSA refreshment function. We decided to start a new specification with the LSA refreshment alone because:

- It would remove the overhead caused by extra OSPF behaviour. This extra behaviour is irrelevant to the LSA refreshment function.
- There would not be enough time to validate the whole OSPF specification and then specify the efficient LSA refreshment.
- It would create machine overhead, which may result in problems in the simulation results.
- It would be harder to identify and demonstrate the specific points where the efficient LSA refreshment has been changed.

Scope

For the generic behaviour:

- Semi-formal UCM model: 38 UCMs representing approximately 45% of the RFC2328 *functional requirements*. The smallest UCM has 2 symbols, and the largest UCM has 39 symbols. The average UCM has 12 symbols.
- Formal SDL model: 140 pages representing 100% of UCM, therefore about 45% of the RFC2328 *functional requirements*. There is 1 System, 3 blocks, and 3 process types. In average a process has 4 states and 12 transitions. The

smallest process has 1 state and 1 transition. The largest process has 7 states and 15 transitions.

- There are two versions of the Generic OSPF SDL model.

For the LSA refreshment behaviour

- Semi-formal UCM model: 5 UCMs representing 100% of the efficient refreshment *Internet-draft* proposal. The smallest UCM has 3 symbols. The largest UCM has 10 symbols. The average UCM has 6 symbols.
- Formal SDL model: 159 pages representing 100% of the UCMs, and therefore the whole *internet-draft*. There are 6 systems, 6 blocks, and 7 process types. In average a process has 1 state and 2 transitions. The smallest process has 1 state and 1 transition. The largest process has 2 states and 3 transitions.
- There are 4 versions of the LSA refreshment SDL models.

Skills of Project Personnel

Table 5-1 summarizes the personnel characteristics for this case study's project. This case study was conducted by 1 person, the author. Therefore 1 person-month = 1 month.

Project Personnel Characteristic	Rating
Semi-Formal Description Technique Knowledge (by author)	
UCM Knowledge	Very good
Formal Languages	
SDL knowledge	Excellent
OSPF Knowledge (by author, and before the case study)	
General Behaviour	Absent
LSA Refreshment function	Absent
Description Technique Tool Knowledge	
UCM Navigator	Poor
Telelogic TAU	Excellent

Table 5-1 - Case Study Project Characteristics at Start of Project

This summary may be used as an aid for others to estimate the complexity of modelling other IETF RFCs with UCMs and SDL. We state *very good* knowledge for UCMs because in previous projects we have worked with UCMs. We rate as *excellent*

knowledge of SDL because we have worked on several projects before where we wrote SDL models for several different systems.

Project Tables and Timeline

This case study was conducted as shown in Figure 5.1. This Gantt chart gives the reader a sense of the percentage of the time taken by each of the bridging methodology steps.

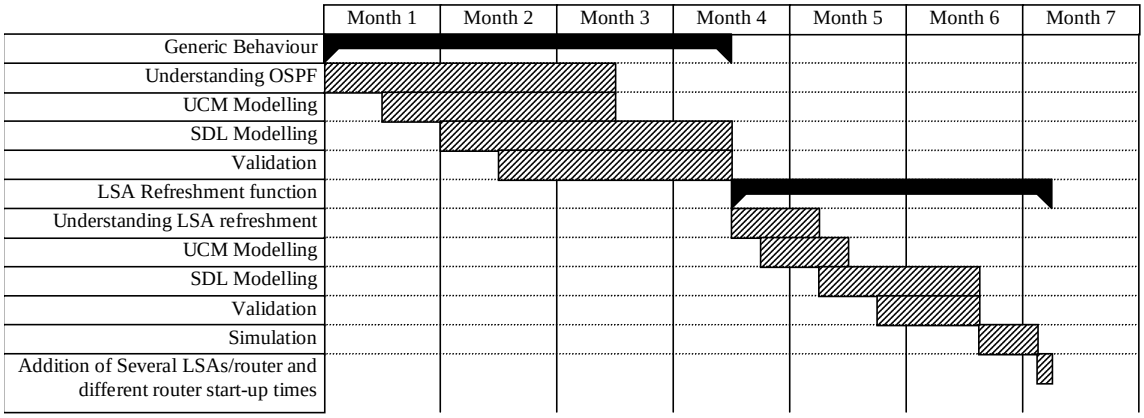


Figure 5.1 - Case Study Gantt chart

In the development of the generic OSPF router model, the cutover point was 3½ months after starting time. We then focused on the LSA refreshment because our expertise on OSPF was mature enough to specify the *internet-draft* proposal.

In the development of the LSA refreshment models, the UCM time is much shorter than in the previous milestone because:

- We had a good understanding of the OSPF protocol.
- We had relevant generic behaviour OSPF UCMs from the previous milestone.
- The UCM model was considerably smaller.

The SDL took some relatively more time than in the previous milestone because:

- Several parallel investigations in code generation and code integration were being done at the same time.
- Development of a solution for extracting data for the simulation results.
- Investigation and experimentation in the connection of SDL and network communication interfacing. These experiments took considerable time but are not relevant to the bridging methodology.

Simulations

For the generic behaviour we have set up a network topology with three routers linked in a triangle (with point-to-point links). For the LSA refreshment experiments we have set up three network topologies. Topology 1 with 3 routers linked in a triangle (with point-to-point links), Topology 2 with 3 routers linked in a broadcast link, and Topology 3 with 11 routers in a mixed topology. This model initially assumed one originating LSA per router, and was afterwards upgraded to accept many LSAs per router.

5.3.2. Results

The topologies with 1 LSA, which are not realistic, are the only ones that show that the *Internet-draft* approach has an equivalent performance to the normal case. Also simultaneous router starts are somewhat unrealistic too.

Methodology Results

- Learning curve: was low for understanding the protocol with the use of description techniques. We showed a decrease of about 50% in the learning time. Internet experts state that at least 8 months are required to have an understanding of OSPF, while we had a partially working model at 3½ months, with no prior knowledge of any routing protocol.
- Data about the project: As shown previously we have metrics on the actual improvement process.
- Activity Logging: There are 130 pages of detailed log documentation on case study.
- Traceability: The UCM models are fully traceable. The SDL models are traceable to the UCMs, therefore, traceable to the standard text.

Simulation Results

These comprise the results taken from the models we have built. There are 6 simulation programs generated completely from the SDL model by the toolset. These include 3 topologies, each with Normal and Zinin delays. We therefore named them T1N, T1Z, T2N, T2Z, T3N, T3Z. We ran these 6 programs with different parameters for the number of LSAs per router (1, 10, 30 and random between 1 and 100) and the Router start mode (simultaneous, equally spaced and random). The graphs in Appendix A show the results we obtained, and are discussed in the next section.

Standards Results

- Commenting: There are 25 **notes** regarding the OSPF standard and 3 **notes** regarding the *Internet-draft*. None were expected for the standard, since it is a well established protocol.
- IETF approval: We had the approval of OSPF Working Group chair to present our results in the 49th Internet Engineering Task Force meeting. These results had the specification with 1 LSA per router. Improvements were suggested for the inclusion of several LSAs per router.
- IETF approval: We again (50th Internet Engineering Task Force) had the approval of the OSPF Working Group chair to present our new results taken from the improvement from the internet-draft. This work was an improved version of the item above, including several LSAs per router and several different router start-up times. This improvement took 4 days.

5.4. Assessment of USHLTD Bridging Methodology

This section presents an analysis of the this assessment. We show advantages and disadvantages.

5.4.1. Assessment based on Case Study

We chose the OSPF protocol and the Link-State Advertisement refreshment function because it is an Internet protocol, it is current, and it is complex. This has already been said in the case study chapter, chapter 4. We used the TAU tool because it is of industrial-strength. It allows for validation, verification and simulation of SDL models, as well as code generation for our SDL models. We believe the activity logs are good because they reflect careful details of the case study during the case study execution. We believe our results are good because we have been approved twice for presentation in the IETF meetings. Our productivity was recognized as experts stated that we have developed our models in half the time they expected us. The evidence for this is the IETF OSPF working group acceptance of our results.

5.4.2. Other Assessment factors and Limitations

Our USHLTD approach is not very precise because it is open to on-the-fly design decisions. However, this openness is one of its main strengths since it allows a heavily

experienced UCM/SDL designer to quickly produce a working SDL model from a UCM model.

Our bridging methodology is more appropriate for Internet protocol standards compared to other methodologies because it was designed for this purpose. We warn the reader about the subjective and manual nature of our USHLTD approach, which makes the bridging approach rely heavily on the design decisions.

5.4.3. Overall Assessment of USHLTD Bridging Methodology

USHLTD has the following advantages:

- It uses both a semi-formal (source model) and a formal-description technique (destination model).
- It is an iterative development methodology, and therefore allows model refinements.
- Its *analytic* characteristic gives the designer more freedom to model the executable models, rather than depending on the constraints imposed by a *synthetic* approach or a tool.
- Our approach has been proven useful with an industrial-strength tool that several advantages as discussed in chapter 2.
- The case study showed that the theoretical process model is feasible and is practical throughout for developing Internet protocol models.

From the case study, these are the limitations we found in our USHLTD methodology:

- UCM Parallelism. We know our methodology is weak in exploring the UCM possibilities in parallel paths. However, we have not encountered any cases where this parallelism is really necessary. It may be the case that it will be useful in other internet protocols. We hope as UCMs acquire more semantics and SDL evolves, this limitation can be easily overcome.
- UCM *abort* construct. Our methodology does not provide a construction guideline for the *abort* construct. The designer should handle this manually. We only used this construct a few times, but during the SDL modelling it did not pose a problem. We find it is more useful for warning the designer than actually defining some relevant behaviour.

- SDL limitations: The SDL language has certain limitations that influenced some of the decisions in our design. One of them is that SDL'96 does not accept the dynamic creation of channels (or routes) between processes and blocks. This required us to create the infrastructure in terms of configuration processes for the different topologies. SDL'2000 promises to overcome this weakness.
- Tool limitations: Although we have generated graphs, we could not fully port the generated SDL code into a target platform because of tool limitations. We expect this will not pose a problem in a the very near future.

We believe our USHLTD bridging methodology is sound because we have showed that starting from reasonable UCMs, we can achieve a faithful SDL image of them. To support this fact we have shown that useful simulations were feasibly generated. We know there is some extra SDL modelling done, as described by the *infra-structural* SDL in chapter 4. However, nothing in the extra SDL contradicts the UCM specified behaviour.

5.5. Recommendations for Improvements of Internet Standards protocol development

We now give some recommendations for improving the RFC standards-track Internet protocol development process.

- We recommend the use of a semi-formal description technique in the early stages of the protocol development.
- We recommend the use of a bridging methodology to derive executable models from the semi-formal description technique models.
- We recommend that the Semi-Formal description technique (source notation) be UCMs, and that the Formal Description Technique (destination notation) be SDL.
- We recommend that the use of UCMs and SDL should be informative, and perhaps normative in the future.

The next chapter contains the conclusions of this thesis and some suggestions for future research.

6. Conclusions and Suggestions for Future Research

6.1. Summary and Conclusions

In this thesis, we identified an opportunity for improvements to the way that Internet protocol standards are developed by IETF. We defined the concept of a *bridging methodology* which transforms a higher-level, *source model* of a standard protocol under development (SPUD) into a less abstract, *destination model*. After reviewing other examples of bridging methodologies in the research literature, we proposed USHLTD (UCM to SDL High-Level To Design), a bridging methodology for Internet SPUDs which transforms a UCM (Use Case Map) source model into an SDL (Specification and Description Language) destination model. The methodology consists of a set of construction guidelines or rules for transforming constraints in UCMs, the source *Semi-Formal Description Technique* into SDL, the destination *Formal Description Technique*.

To validate this USHLTD bridging methodology, we applied it to the development of a real Internet routing protocol standard, OSPF (Open Shortest Path First), and particularly to a function of the OSPF protocol named LSA (Link State Advertisement) refreshment function, in a collaborative industrial research project. The USHLTD methodology was found to enhance the quality and time to standard of the protocol, and led to contributions to IETF to validate the Link-State Advertisement efficient refreshment function proposed in an Internet-draft. In addition, USHLTD compares well with other bridging methodologies currently under study. Thus, the USHLTD appears to be an effective and practical means of incorporating formal protocol engineering methods into the development process for Internet protocol standards.

6.2. Contributions, Benefits and Limitations

6.2.1. Contributions of the Thesis

The contributions of this thesis are:

1) The USHLTD bridging methodology (chapter 3), that transforms a source UCM model into a destination SDL model.
2) A Case Study (chapter 4) to validate USHLTD including a key Internet protocol and a report of related experience and results (chapter 5).

3) Recommendations for improvements to the IETF RFC standards-track process (section 5.5), and for refinements of USHLTD in the future (in section 6.3).

We now list the expected benefits from applying such methodology.

6.2.2. Benefits of using USHLTD

As a result of using USHLTD, we obtain:

1) *A Semi-Formal Description (source) model of the protocol under development*. Using a graphical notation (UCM) to represent the high-level behavioural description of the protocol, helps the understanding of the protocol by other individuals. The UCM SFDT helps to reduce the number of inconsistencies which may be created by natural language and the manual standards processes. Requirements in the RFC may appear distributed over many different pages.
2) *A Formal Description Technique (destination) model of the protocol under development*. By transforming to a formal (SDL) notation using a powerful (TAU) tool, we obtain many benefits (4 to 7 below).
3) *More Complete Project Documentation*. These are the side-effects of applying *traceability*, *commenting*, and *activity logging* to Internet standards. We have shown the benefits of *logging* for project chronology and progress tracking. This type of data can be used for report creation, for instance. *Traceability* helps us keep track of the diverse components of a standard, and their descriptions in text and in the various notations. The UCM notation enabled us to gather the functions in each requirement into a conceptual flow in a single UCM, for example the UCM describing the *Neighbour* data structure behaviour.
4) *Fast Prototyping*. Prototyping is part of our methodology as shown in Figure 3.1. With prototyping we can *validate*, and *verify* the SDL model (and therefore the standard). *Prototyping* allows us to quickly produce an executable version of an Internet standard. As in chapter 4, for validating the LSA refreshment function using different algorithms and network topologies.

5) *A Set of Possible Scenarios of the protocol under development*. As the protocol communicates messages between components, descriptions of any desired scenarios may be generated from the SDL model.
6) *Higher Quality Protocol Models*. This is gained with notations and tools. The industrial-strength tools permitting the execution of state-based models allows for *validation* and *verification*, which then permits us to certify that the requirements are complete, correct and consistent before implementation. For example, this approach allowed us to detect and raise a serious bug in the RFC namely, the lack of behaviour descriptions for the case when two or more routers start simultaneously. This would not have been systematically discovered without using SDL or some other formal description technique as we did. However, we needed to use UCMs to help understand the protocol before we could develop the state-based SDL model. This is why USHLTD uses both notations.
7) *Improvements on Time-to-market of Standard*. By increasing the quality and the precision of a standard, we are therefore reducing re-work and therefore the time for the standards development. Automatic code generation may also save time by quickly providing an executable implementation directly from the SDL model as required by IETF.

All these reasons contribute to improving quality and time to market for Internet protocol standards. It is recommended that the USHLTD bridging methodology be used to introduce standardized formal and semi-formal descriptions into the process of describing Internet protocol standards, where helpful. This will facilitate communication of technical details among industry vendors.

We now present the limitations found in our methodology.

6.2.3. Limitations of USHLTD

We list the main limitations as well as what may be done to overcome any problem, where possible.

1) *Difficulty*. The methodology we have presented in this thesis requires somewhat more effort to be put into a project than in the conventional IETF process. This is because one has to learn several new concepts, like

description techniques, tools, results produced by tools, and some self-discipline to change the process of Internet protocol standard development to adapt to USHLTD. However, our process is simpler than the others we studied.

2) *Reliance on manual aspects*. Even with the use of industrial-strength tools manually constructed artefacts are needed in our process. It can be tedious to systematically carry out *activity logging*, *commenting* and enforcing *traceability*. Nevertheless, this was found to pay off later. The Personal Software Process also describes this as an advantage [23].
3) *Need for Confidence in tools and notations*. It is absolutely necessary to have confidence in the tools and notations used. We feel it is impractical to conduct a project with two different approaches or tools because it will double the effort and the cost. Since the use of industrial-strength tools and notations are only starting to become popular, it is natural to have a lack of confidence in them. However, this can be overcome as we demonstrated in the case study.
4) *Requirement for defining and naming states*. This is a design-level consideration which cannot effectively be automated. It is up to the designer to make proper design decisions in this area based on a solid understanding of requirements. For this reason the methodology may be applied differently by different individual developers. Thus the state name, for example, is a designer's decision.
5) *Need for additional case studies to help USHLTD mature*. Even though we chose a difficult protocol to work with in the case study, we may need to test this methodology with other standardized Internet protocols to broaden its usability, to correct problems, and to fine tune the methodology.

We now present suggestions for future research.

6.3. Future research

We now propose some ideas for the continuation of this research. These are presented below.

1) Adding TTCN into the process. TTCN is currently the only formal international standardized language for testing. It should be used to describe

test suites for protocol development. But testing can be difficult. It would be interesting to study how to incorporate test development in TTCN directly into the standard development methodology. In other words, when a model prototype is ready, its test suite would be too. This would guarantee that even the earliest implementations would comply to the earliest test suites. A significant advantage of USHLTD is that TTCN test cases can be automatically derived from our SDL model [50]. In the final analysis, testing is the only widely accepted method for detecting implementation errors.

2) Applying our methodology to other behaviours of the OSPF protocol. This will build more confidence in the methodology and perhaps find other limitations. By removing any weaknesses, we will strengthen the methodology. We suggest some of the other IETF OSPF internet-drafts.
3) Applying our methodology to specify an entire abstract and executable model of the OSPF routing protocol. The reason behind this is to demonstrate the beneficial impact of such a methodology in an entire complex Internet protocol. This should show major drawbacks (if any) with notations and industrial-strength tools, besides contributing enormously to the standard itself.
4) Standardizing another IETF protocol with our methodology and TTCN. Since the IETF has not yet accepted the use of description techniques, and given its influence and the importance of Internet protocols, success in a new IETF protocol could lead to IETF adoption of description techniques. This would have a far reaching impact on Internet standards.
5) Code Integration. We performed some experiments trying to attach the generated SDL code from the LSA refreshment function onto a working specification. We found that code integration (especially with legacy code) is a field of research not very explored yet, and it presents great challenges for the use of description techniques automatically generated code.

We now present a brief summary of this thesis.

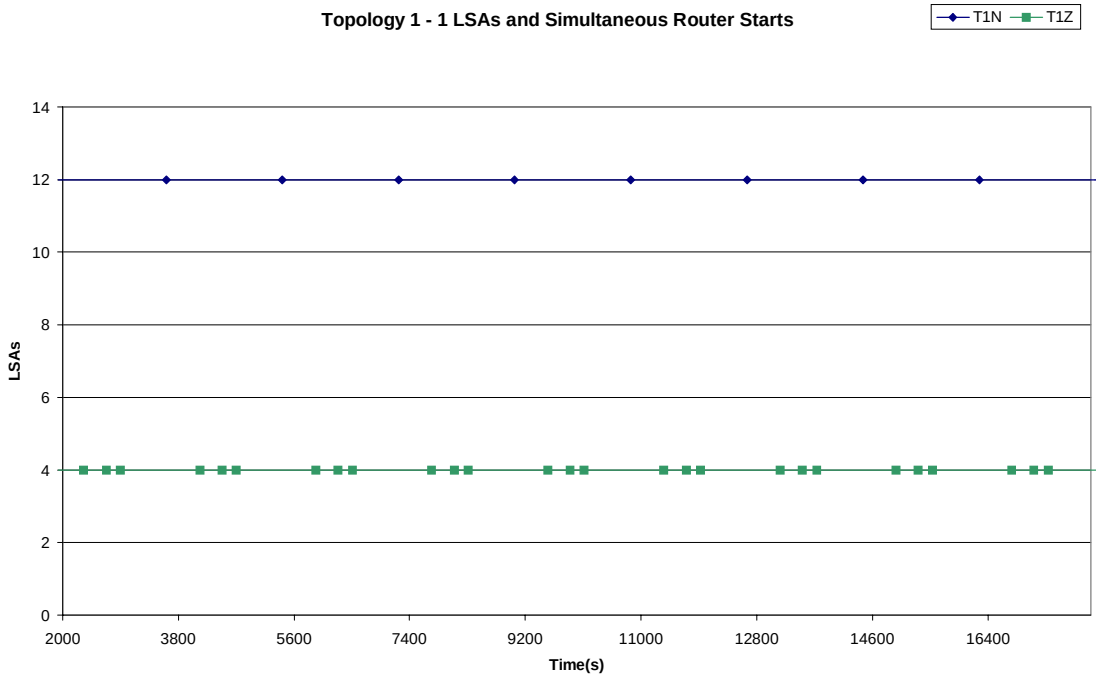
6.4. Final Summary

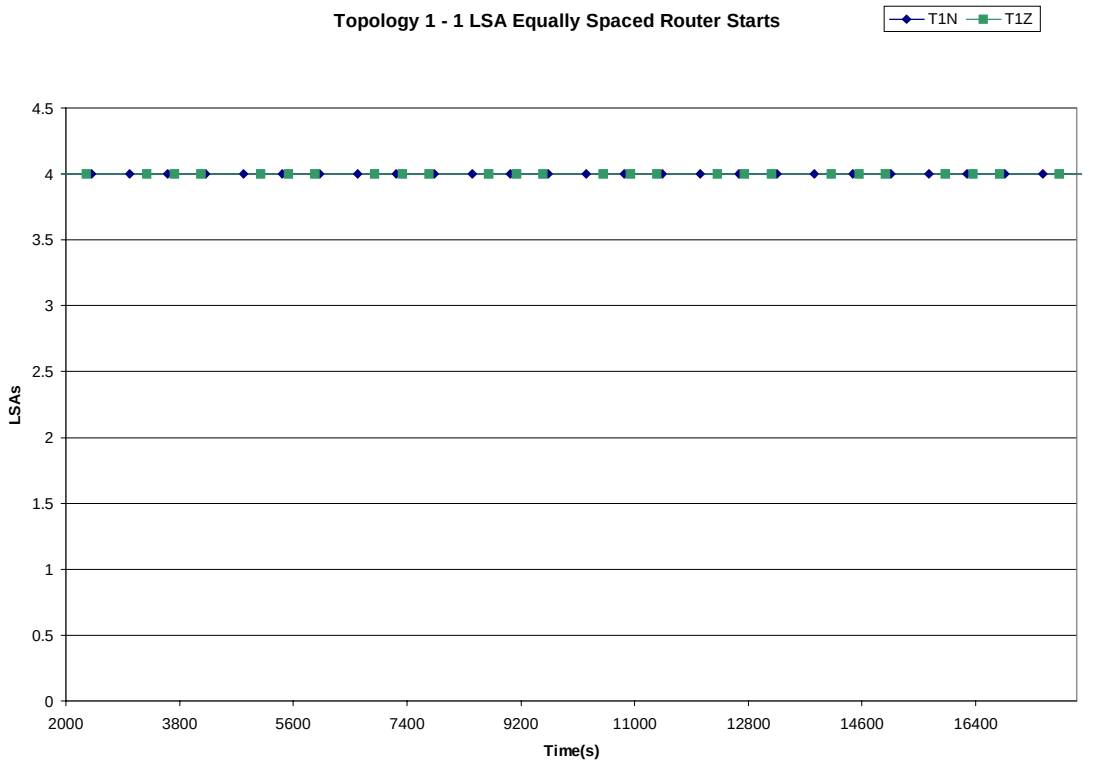
In summary, with our USHLTD bridging methodology we can easily go from an RFC (Request For Comments, Internet Standards text) to a requirements (source) model written in Use Case Maps, and then transform it into an SDL model. This SDL model provides several benefits including validation, verification, and automatic code generation. In parallel the designer can maintain the model traceability, insert comments, and keep track of the project activities. With automatic code generation it is possible to have an early interoperable implementation, as required by IETF.

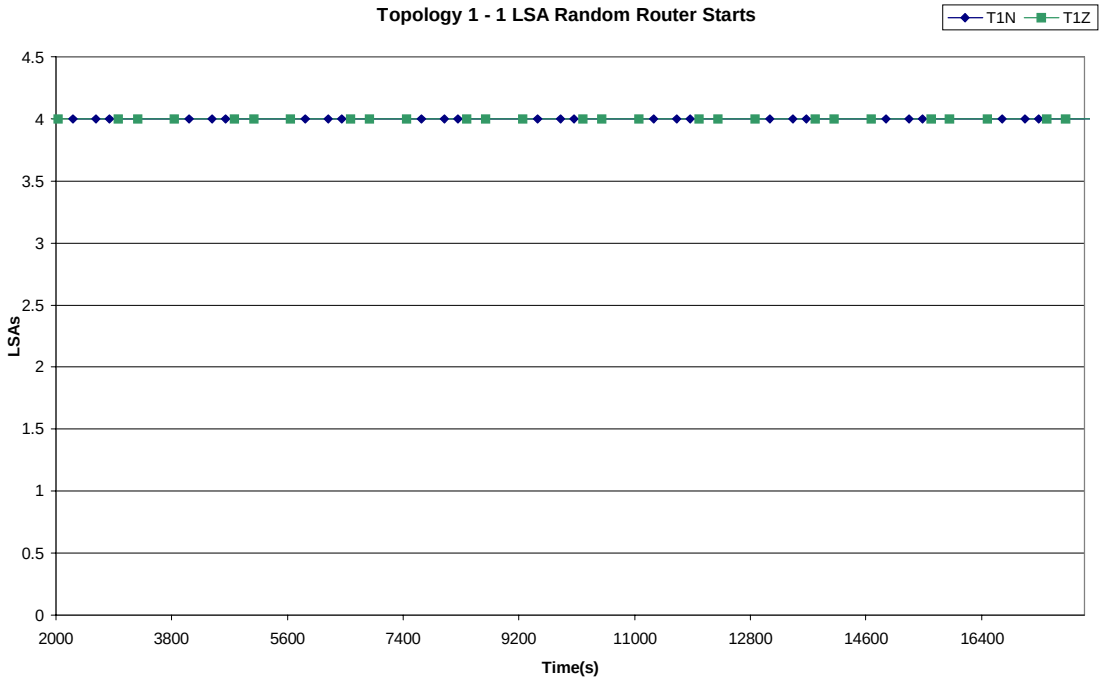
We believe, therefore, that by using USHLTD description techniques, high quality standards will result, improving the communication of Internet standards descriptions. This opens a wide gammet of near possibilities for the development of standards, while still precisely and consistently describing Internet protocol behaviour from the very high-level to the lower-levels of design. Methods such as our USHLTD have great potential for both technology improvements and the advance of public communication, understanding, and good will.

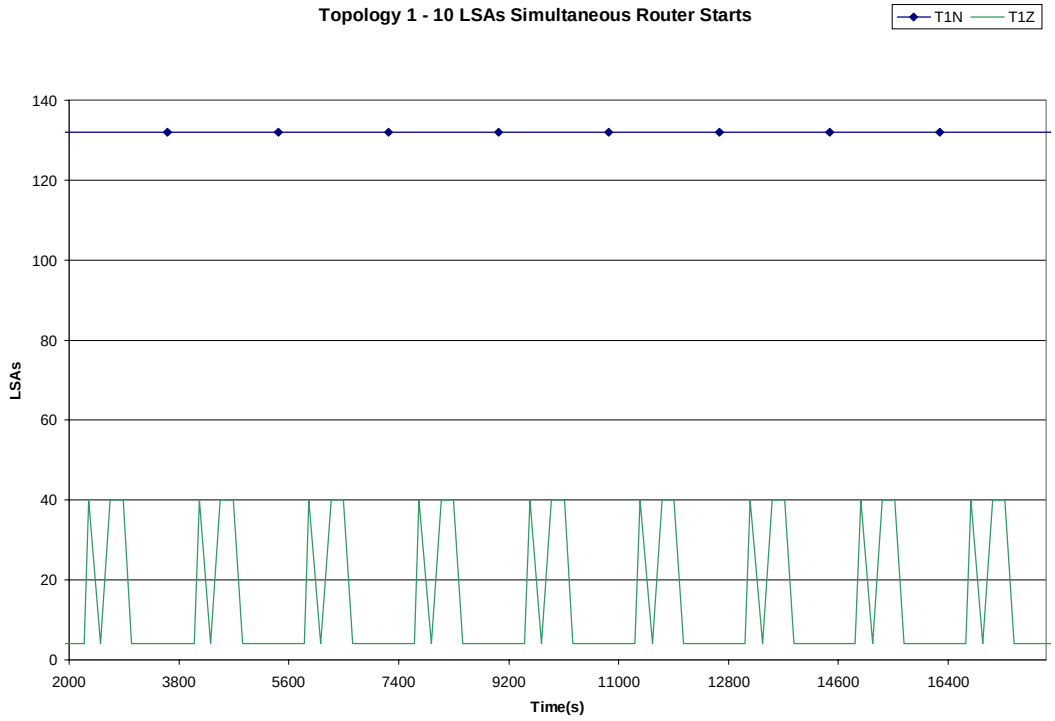
Appendix A – Graphs from LSA Refreshment Simulation

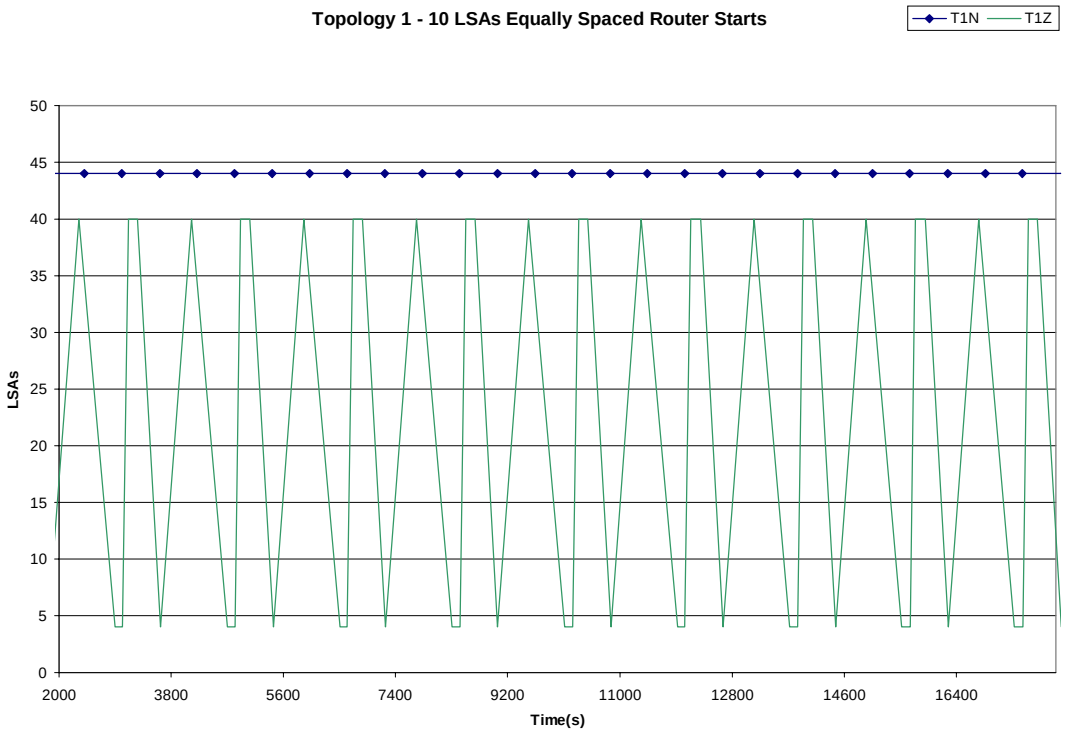
Note: Up to topology 3, the peaks of LSAs are denoted by discrete diamonds or squares. The line between these diamonds or squares is an artefact of the graphing program, and does NOT indicate LSA traffic in the intervals between successive points. In the topology 3 graphs they are denoted as peaks because of the density of information, and because the diamonds and squares severely clogged the graphs. Each diamond or square represents the number of LSAs in the network at that point in time. Each peak in the topology 3 graphs also represent the number of LSAs in the network at that point in time.

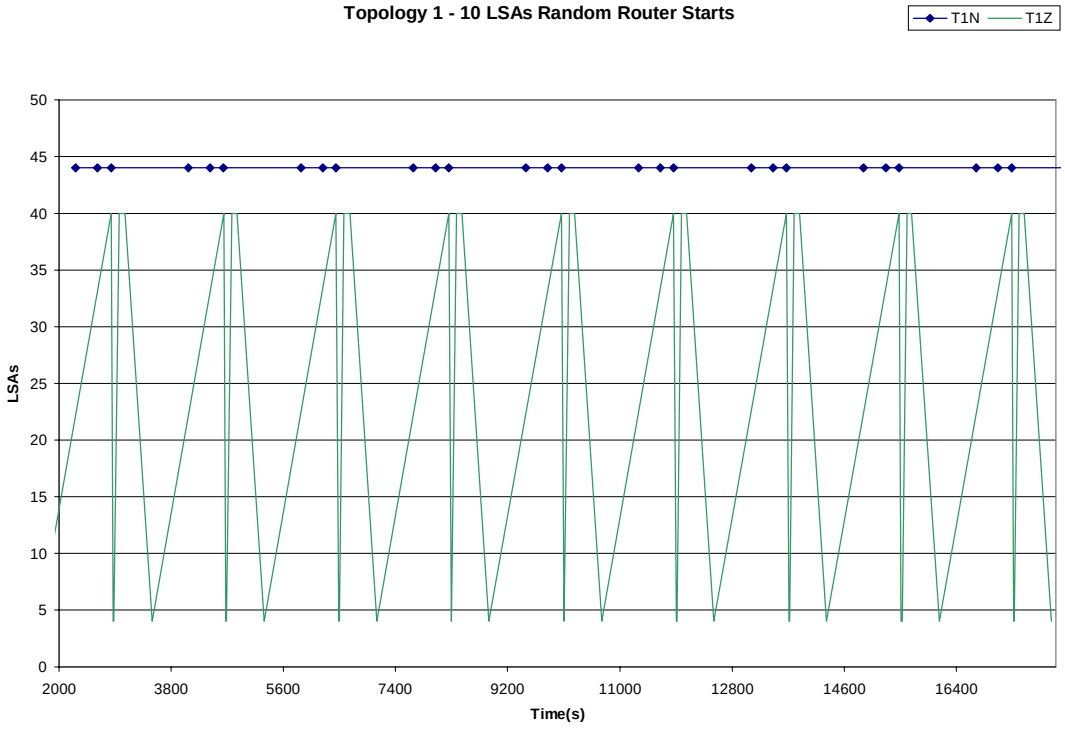

Topology 1 - 1 LSAs and Simultaneous Router Starts
T1N
T1Z
LSAs
Time(s)
0
2
4
6
8
10
12
14
2000
3800
5600
7400
9200
11000
12800
14600
16400

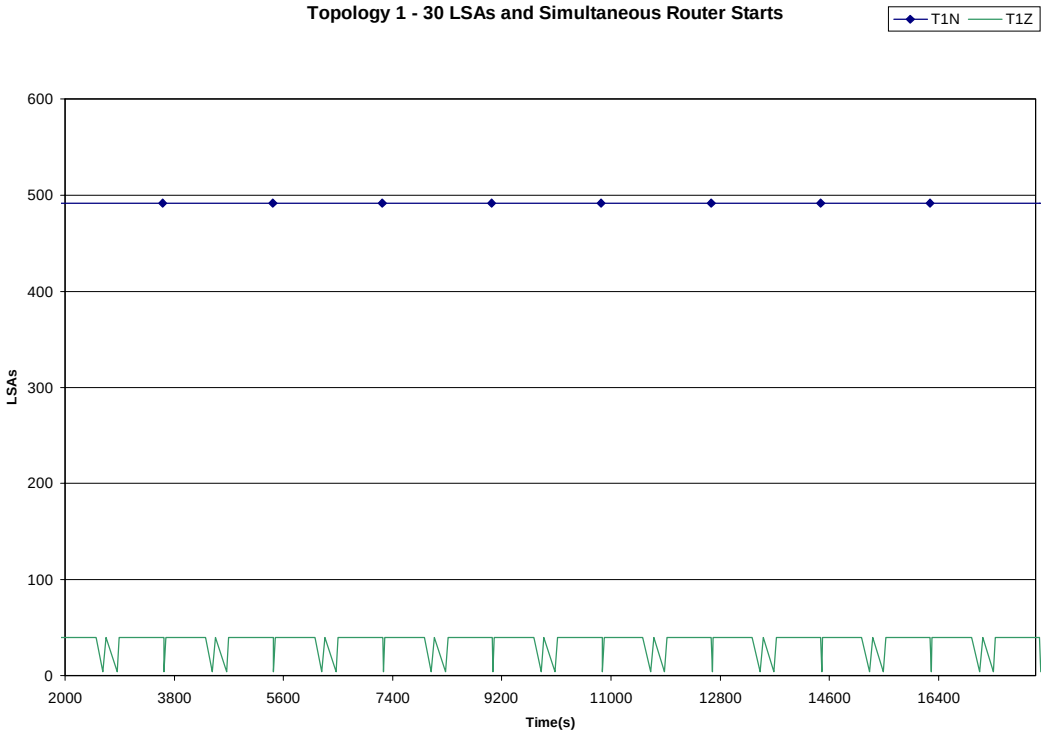

Topology 1 - 1 LSA Equally Spaced Router Starts
T1N
T1Z
LSAs
Time(s)
0
0.5
1
1.5
2
2.5
3
3.5
4
4.5
2000
3800
5600
7400
9200
11000
12800
14600
16400

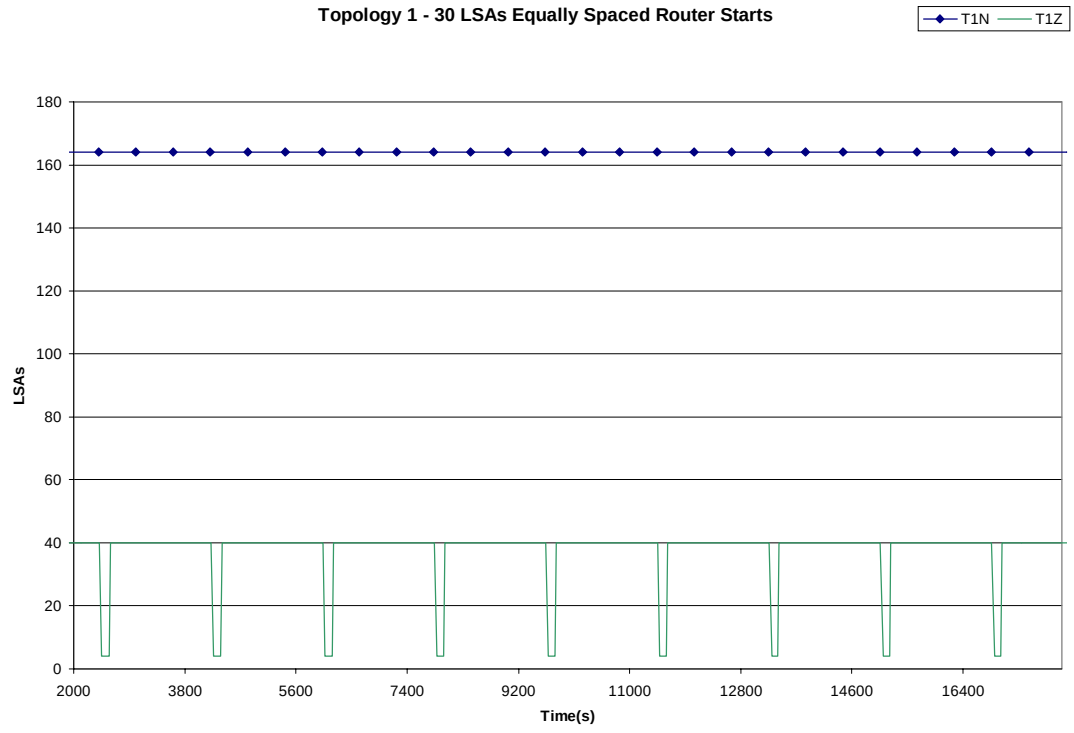

Topology 1 - 1 LSA Random Router Starts
T1N
T1Z
LSAs
4.5
4
3.5
3
2.5
2
1.5
1
0.5
0
2000
3800
5600
7400
9200
11000
12800
14600
16400
Time(s)

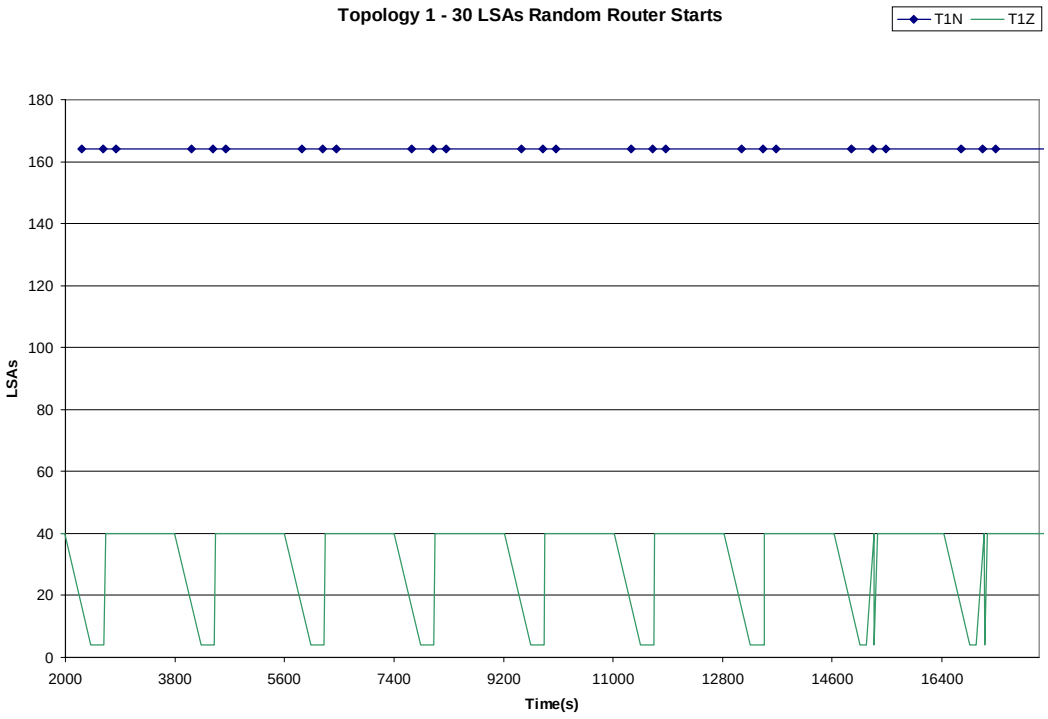

Topology 1 - 10 LSAs Simultaneous Router Starts
T1N
T1Z
LSAs
140
120
100
80
60
40
20
0
2000
3800
5600
7400
9200
11000
12800
14600
16400
Time(s)

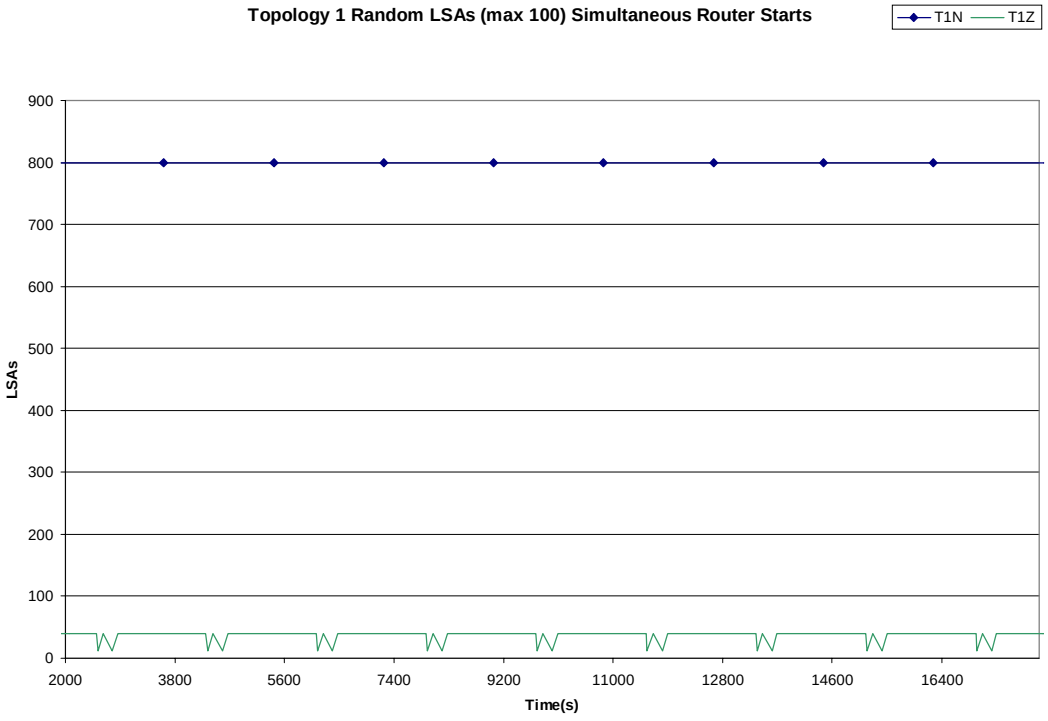

Topology 1 - 10 LSAs Equally Spaced Router Starts
T1N
T1Z
LSAs
Time(s)
0
5
10
15
20
25
30
35
40
45
50
2000
3800
5600
7400
9200
11000
12800
14600
16400

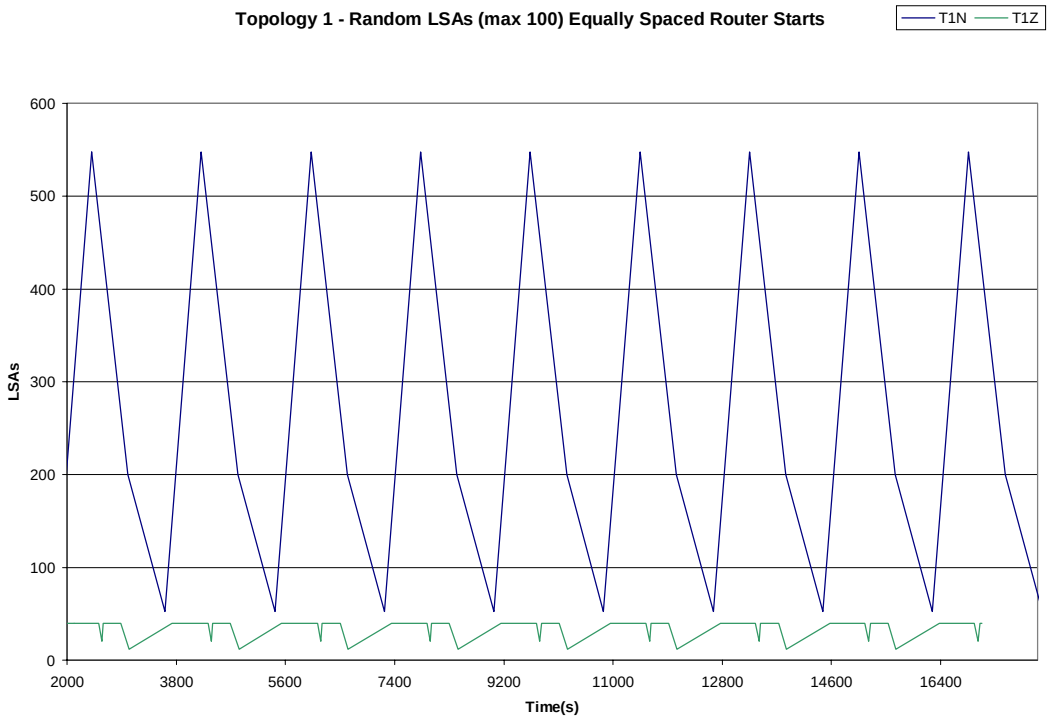

Topology 1 - 10 LSAs Random Router Starts
T1N
T1Z
LSAs
Time(s)
0
5
10
15
20
25
30
35
40
45
50
2000
3800
5600
7400
9200
11000
12800
14600
16400

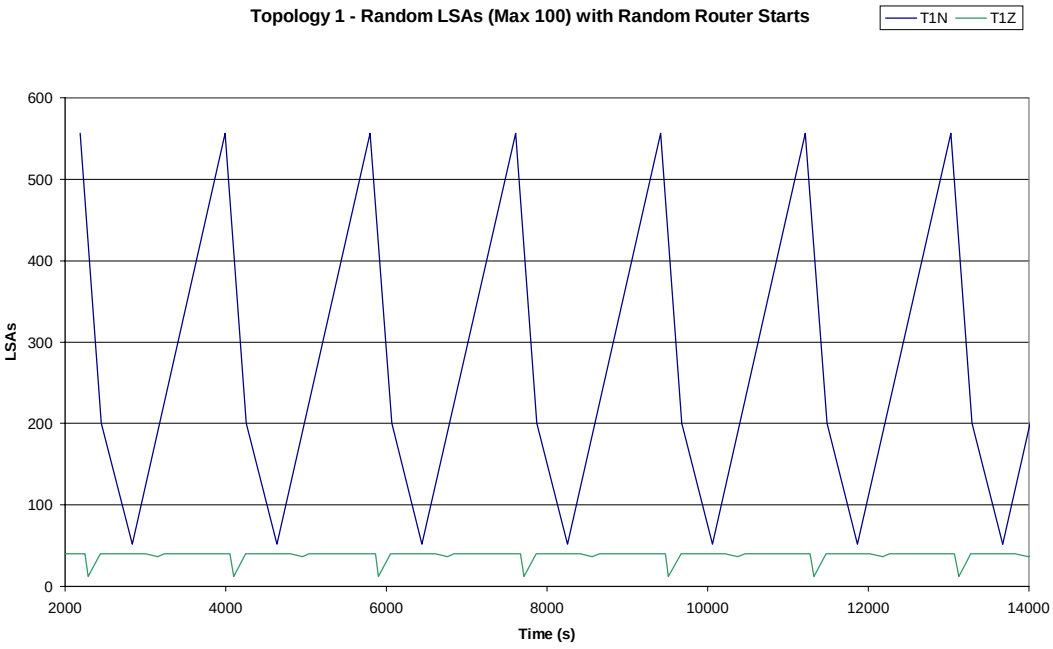

Topology 1 - 30 LSAs and Simultaneous Router Starts
T1N
T1Z
600
500
400
300
200
100
0
LSAs
2000
3800
5600
7400
9200
11000
12800
14600
16400
Time(s)

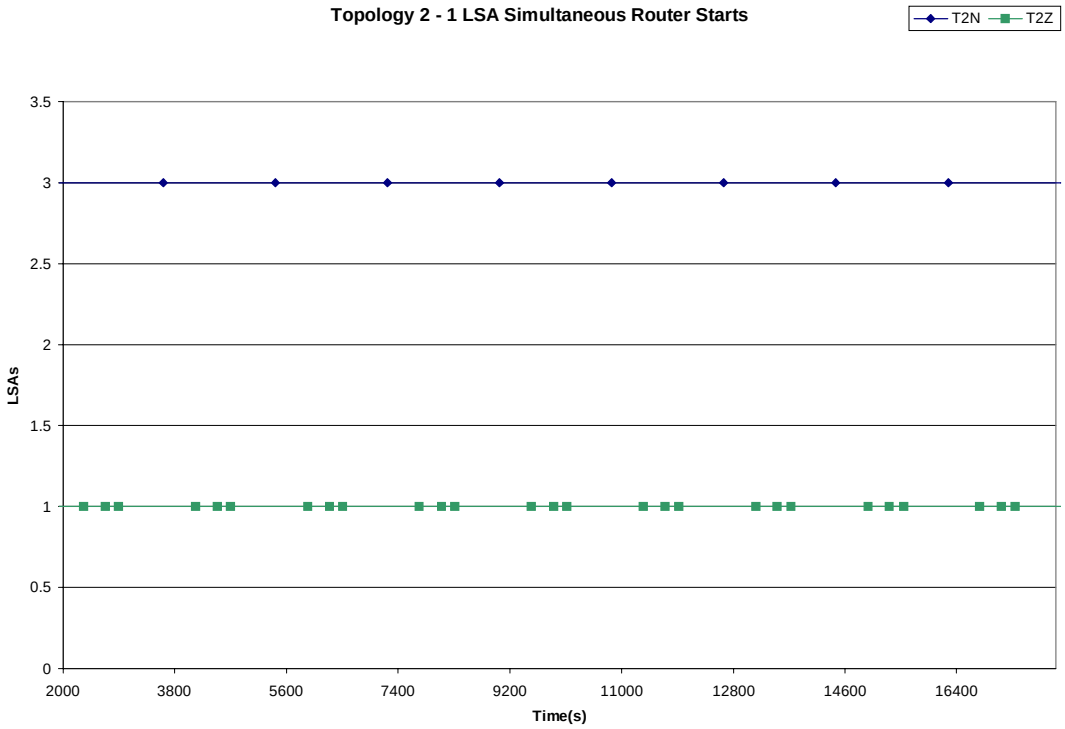

Topology 1 - 30 LSAs Equally Spaced Router Starts
T1N
T1Z
180
160
140
120
100
80
60
40
20
0
LSAs
2000
3800
5600
7400
9200
11000
12800
14600
16400
Time(s)

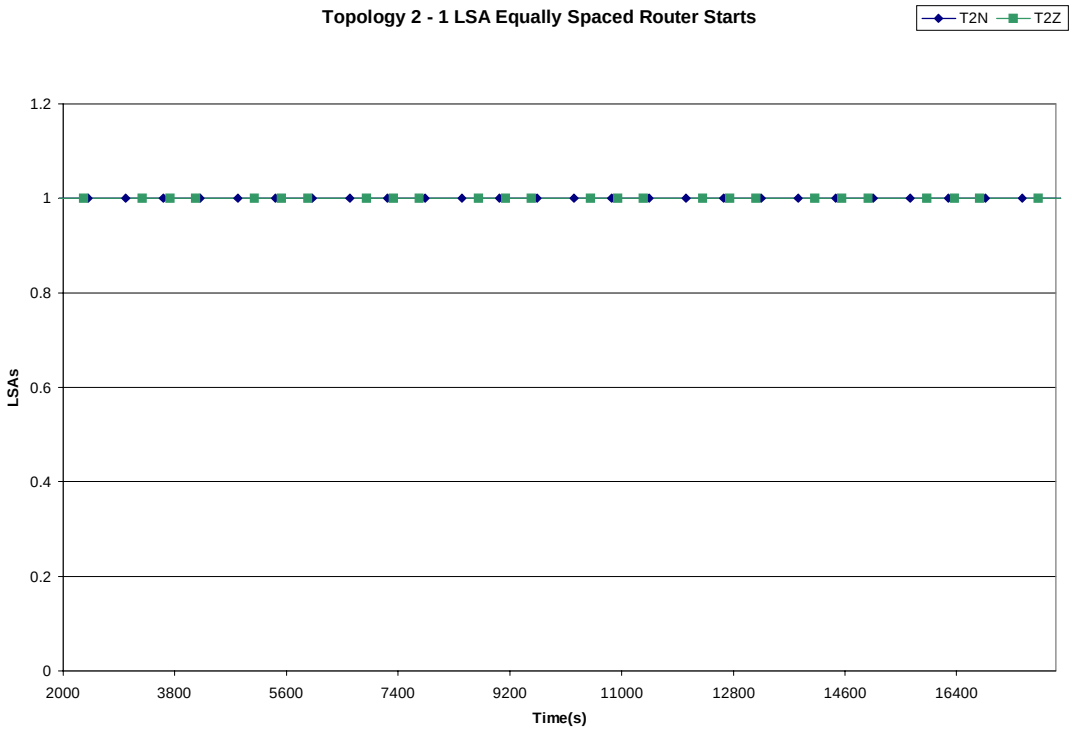

Topology 1 - 30 LSAs Random Router Starts
T1N
T1Z
180
160
140
120
100
80
60
40
20
0
LSAs
2000
3800
5600
7400
9200
11000
12800
14600
16400
Time(s)

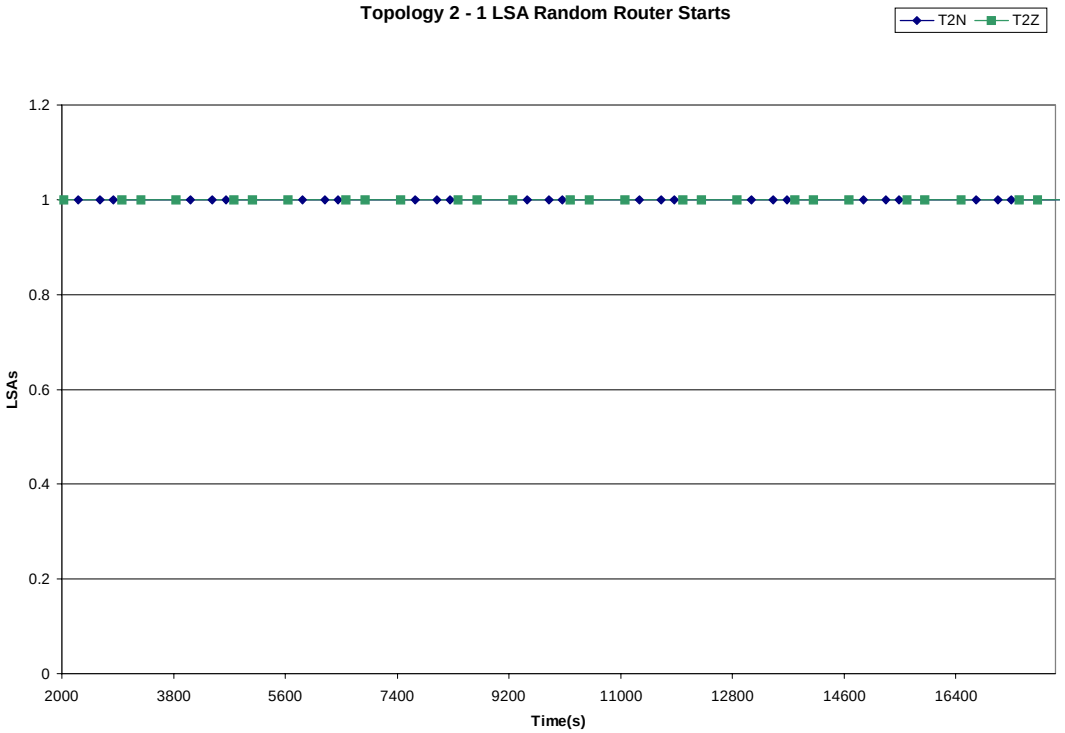

Topology 1 Random LSAs (max 100) Simultaneous Router Starts
T1N
T1Z
900
800
700
600
500
400
300
200
100
0
LSAs
2000
3800
5600
7400
9200
11000
12800
14600
16400
Time(s)

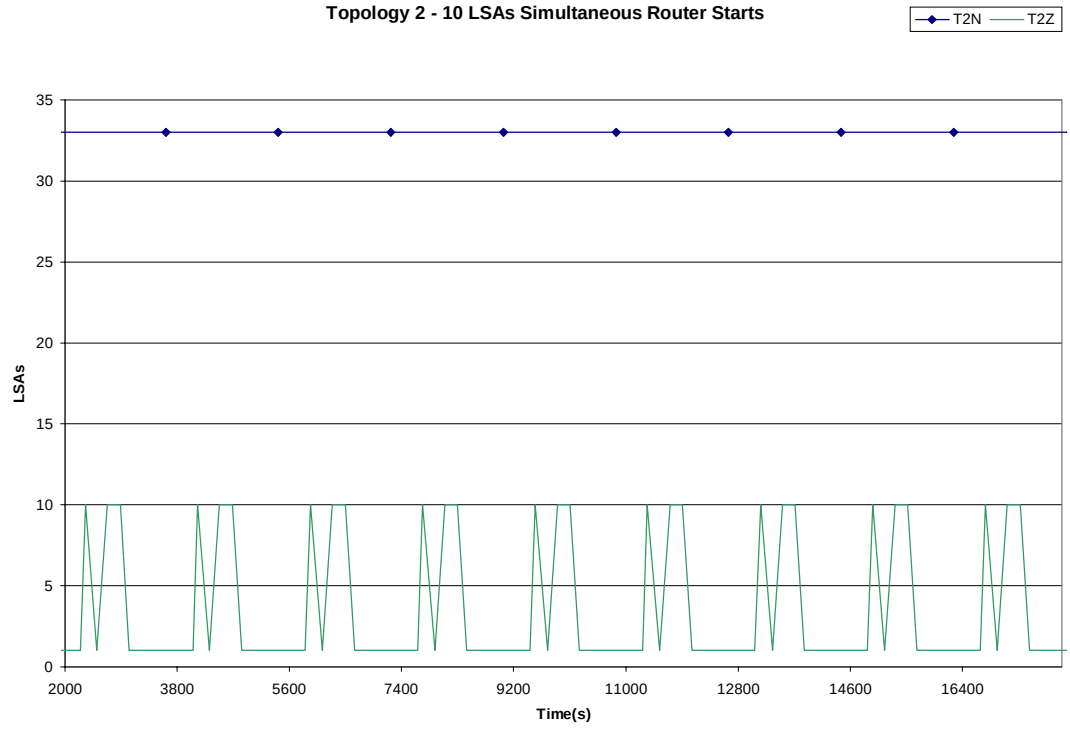

Topology 1 - Random LSAs (max 100) Equally Spaced Router Starts
T1N
T1Z
LSAs
600
500
400
300
200
100
0
2000
3800
5600
7400
9200
11000
12800
14600
16400
Time(s)

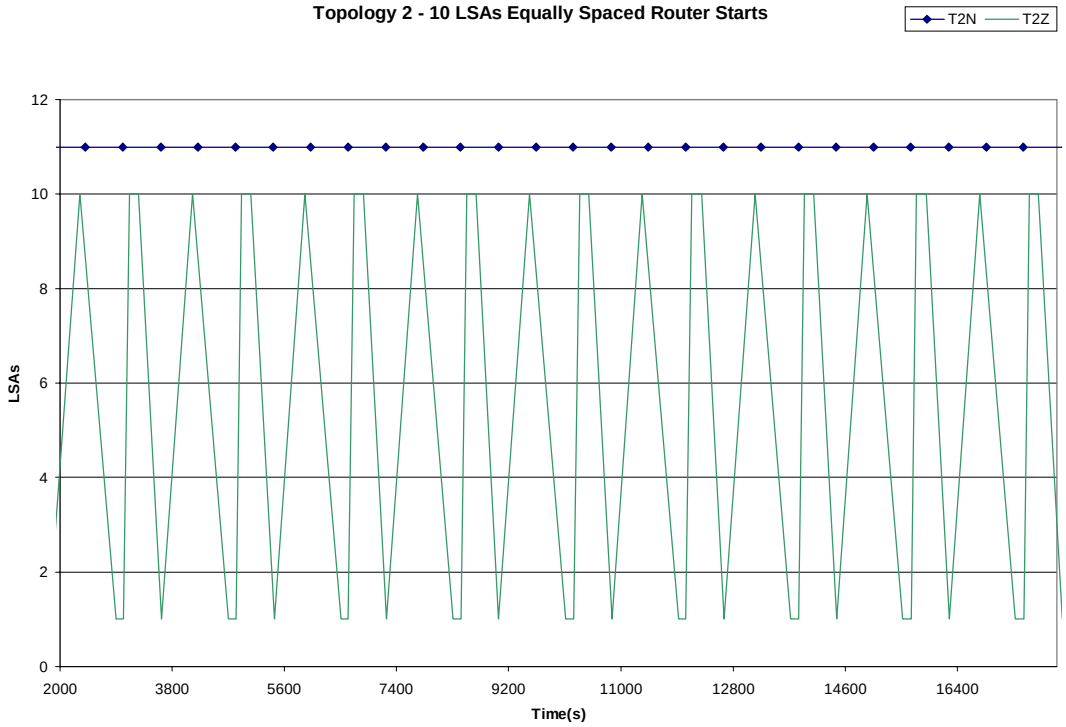

Topology 1 - Random LSAs (Max 100) with Random Router Starts
T1N
T1Z
LSAs
600
500
400
300
200
100
0
2000
4000
6000
8000
10000
12000
14000
Time (s)

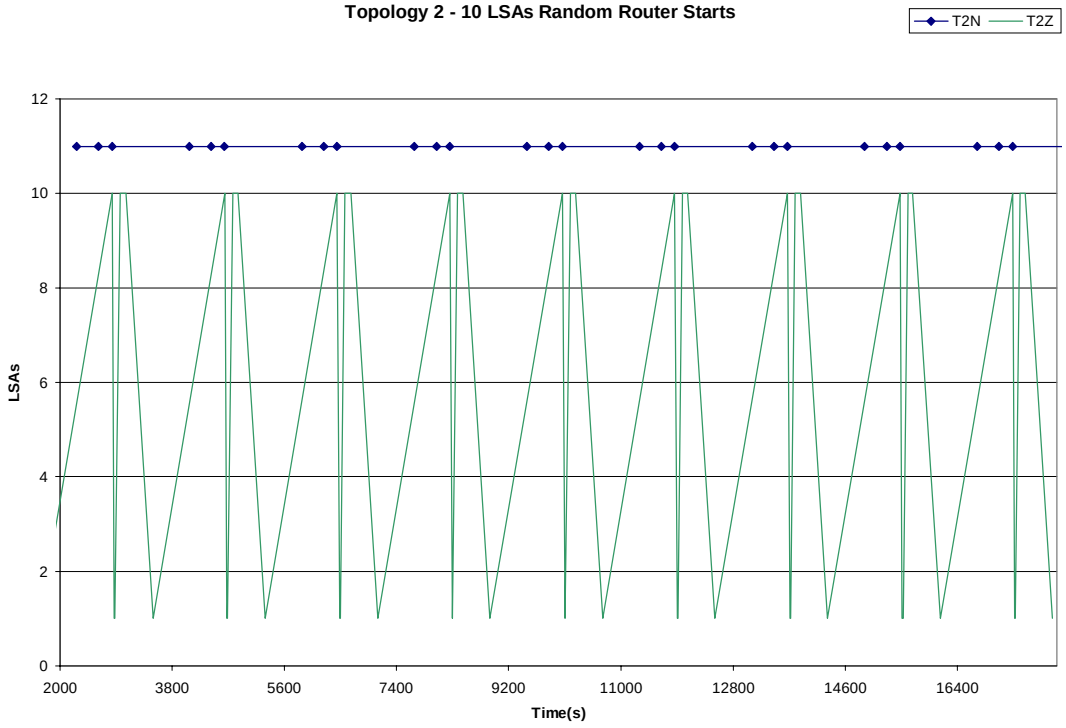

Topology 2 - 1 LSA Simultaneous Router Starts
T2N
T2Z
LSAs
3.5
3
2.5
2
1.5
1
0.5
0
2000
3800
5600
7400
9200
11000
12800
14600
16400
Time(s)

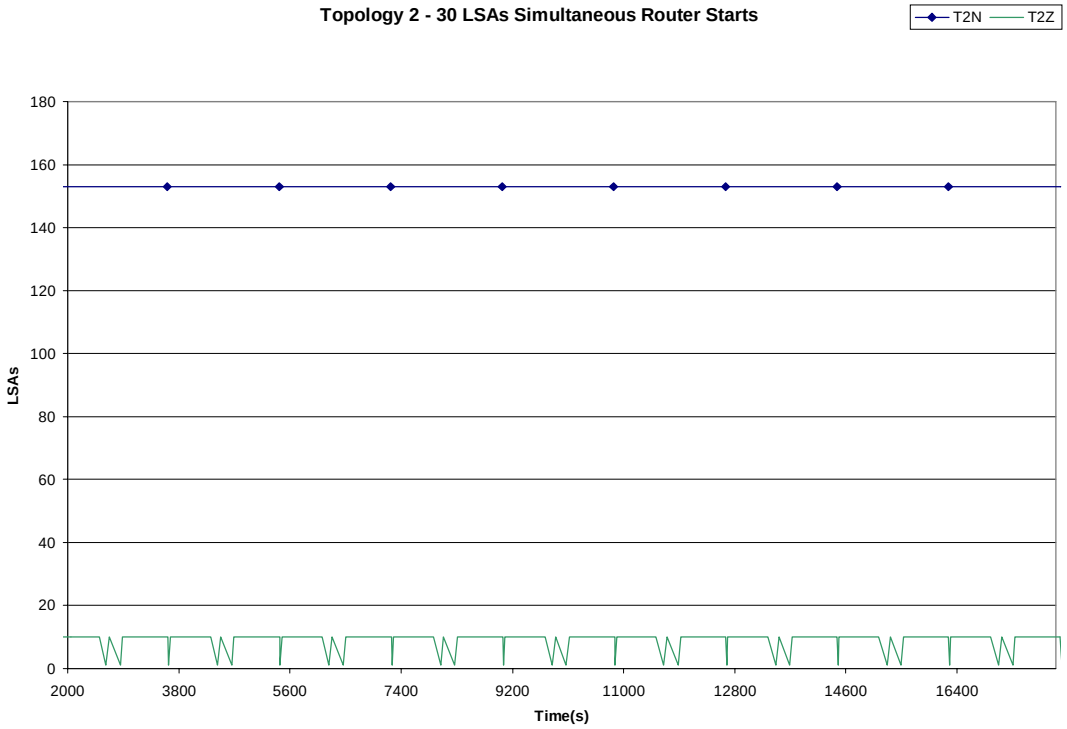

Topology 2 - 1 LSA Equally Spaced Router Starts
T2N
T2Z
LSAs
1.2
1
0.8
0.6
0.4
0.2
0
2000
3800
5600
7400
9200
11000
12800
14600
16400
Time(s)

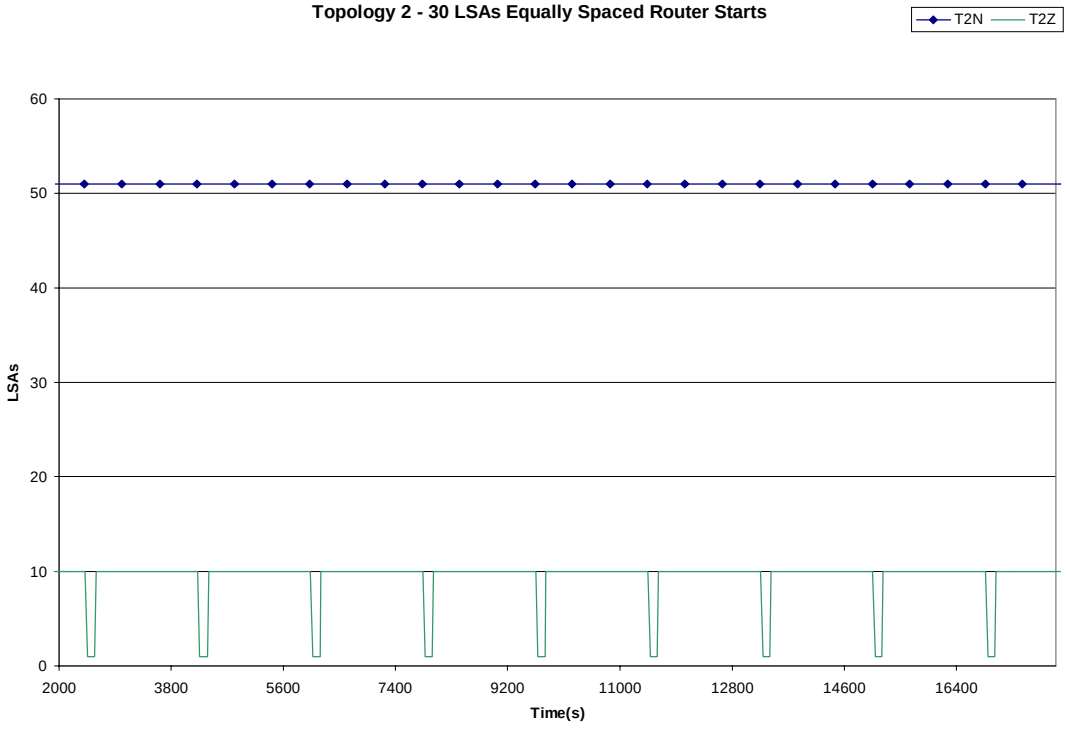

Topology 2 - 1 LSA Random Router Starts
T2N
T2Z
1.2
1
0.8
0.6
0.4
0.2
0
LSAs
2000
3800
5600
7400
9200
11000
12800
14600
16400
Time(s)

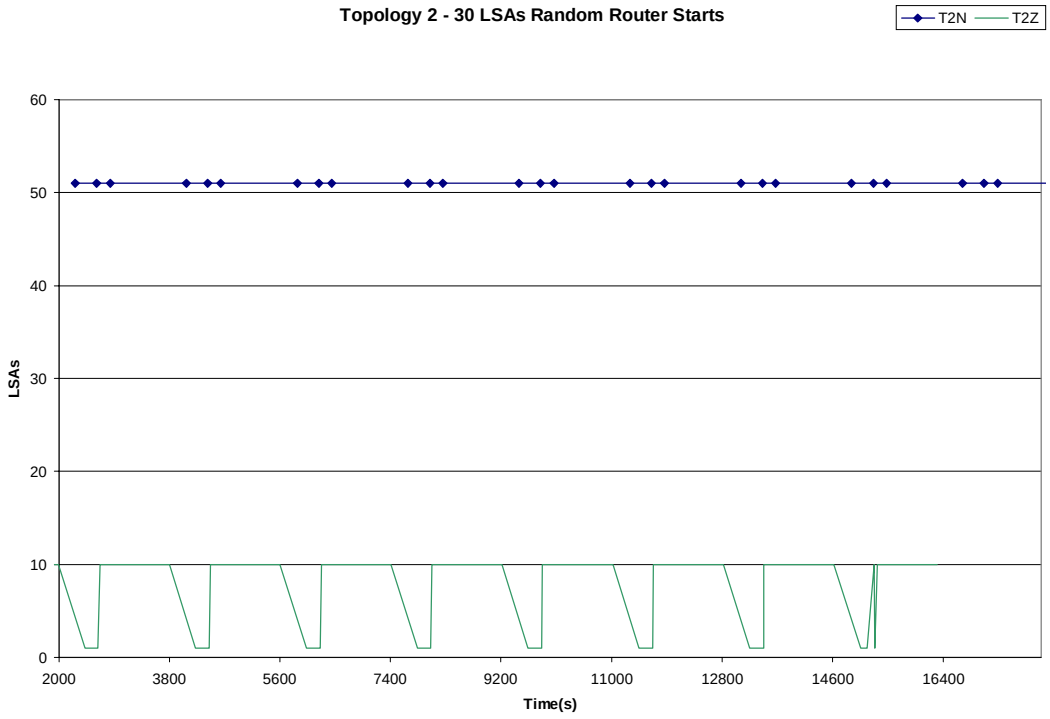

Topology 2 - 10 LSAs Simultaneous Router Starts
T2N
T2Z
35
30
25
20
15
10
5
0
LSAs
2000
3800
5600
7400
9200
11000
12800
14600
16400
Time(s)

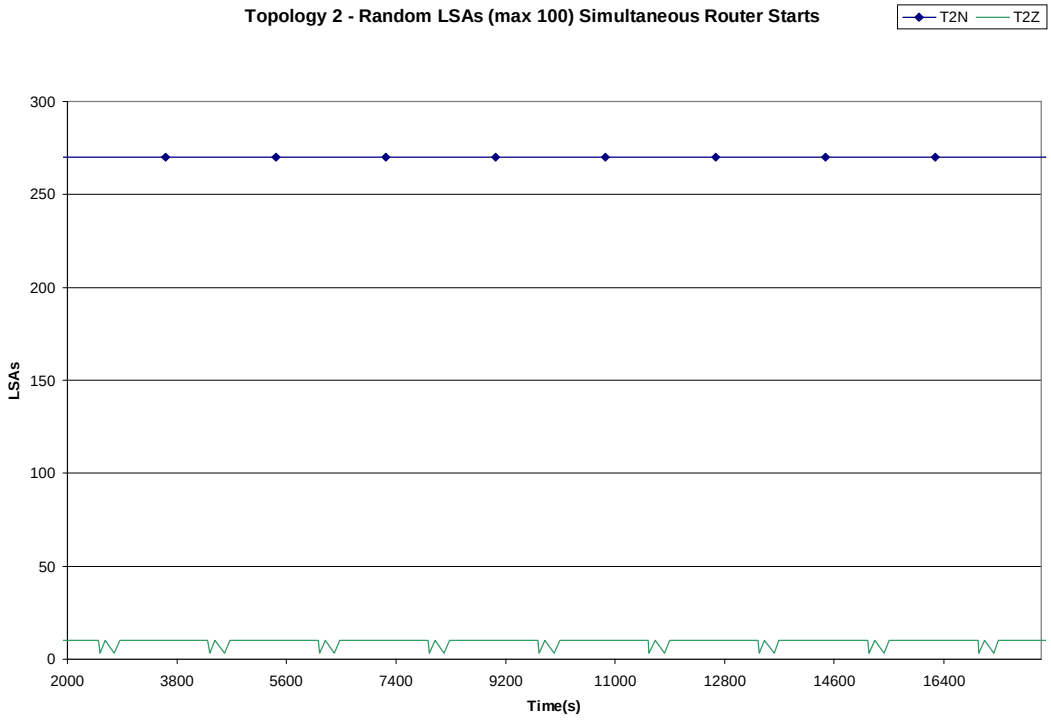

Topology 2 - 10 LSAs Equally Spaced Router Starts
T2N
T2Z
LSAs
Time(s)
0
2
4
6
8
10
12
2000
3800
5600
7400
9200
11000
12800
14600
16400

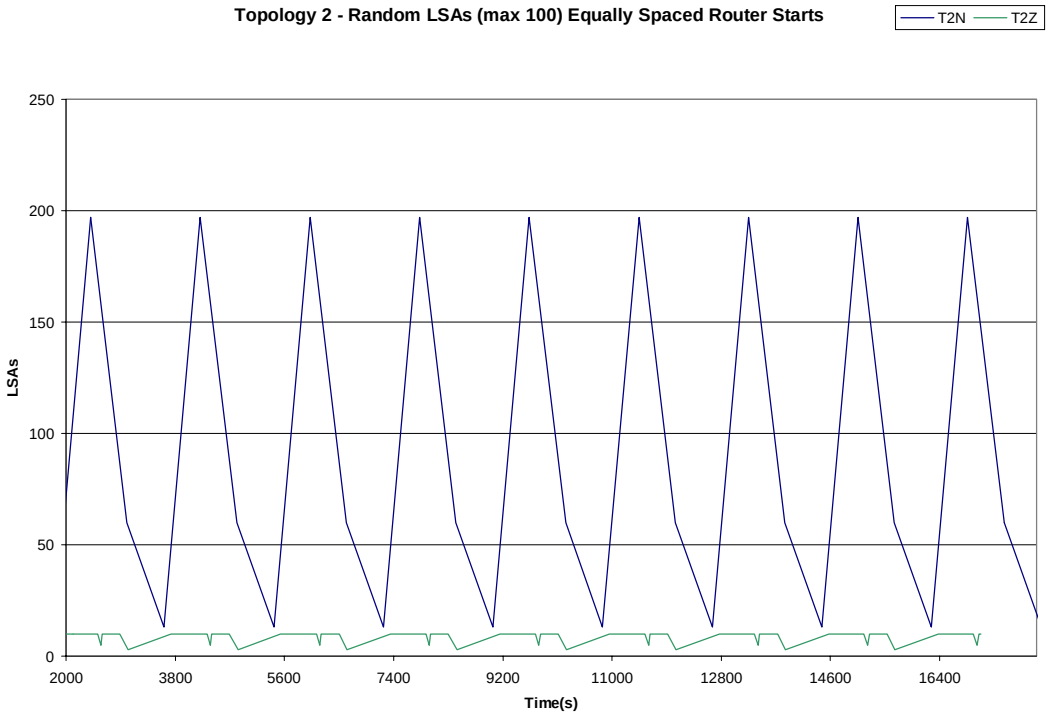

Topology 2 - 10 LSAs Random Router Starts
T2N
T2Z
LSAs
Time(s)
0
2
4
6
8
10
12
2000
3800
5600
7400
9200
11000
12800
14600
16400

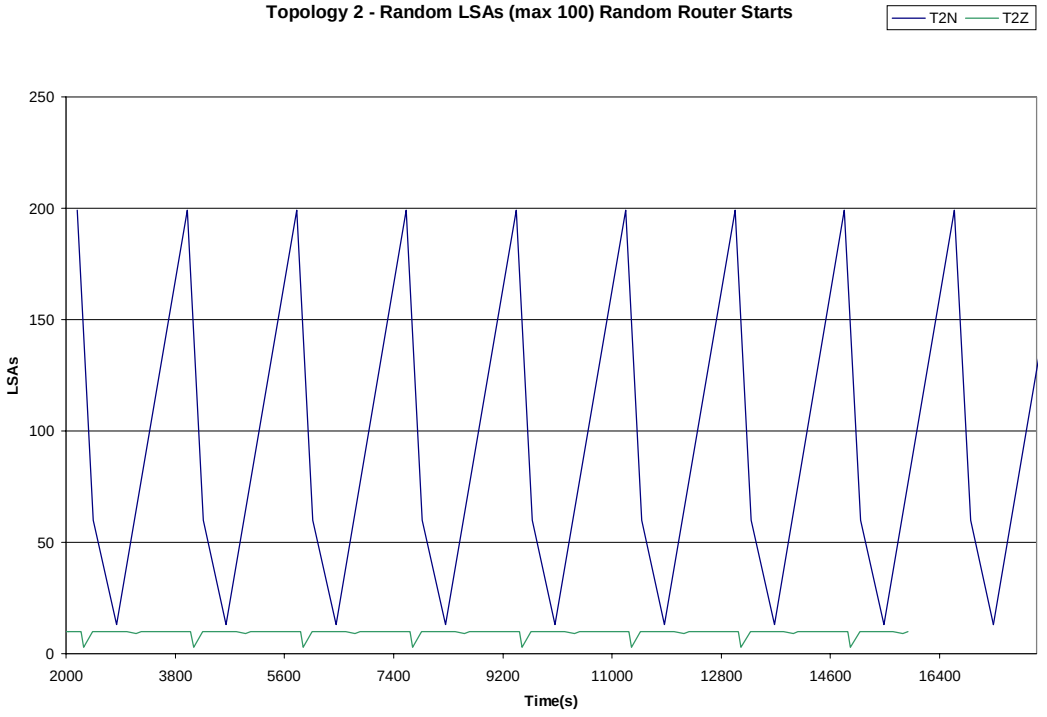

Topology 2 - 30 LSAs Simultaneous Router Starts
T2N
T2Z
LSAs
180
160
140
120
100
80
60
40
20
0
2000
3800
5600
7400
9200
11000
12800
14600
16400
Time(s)

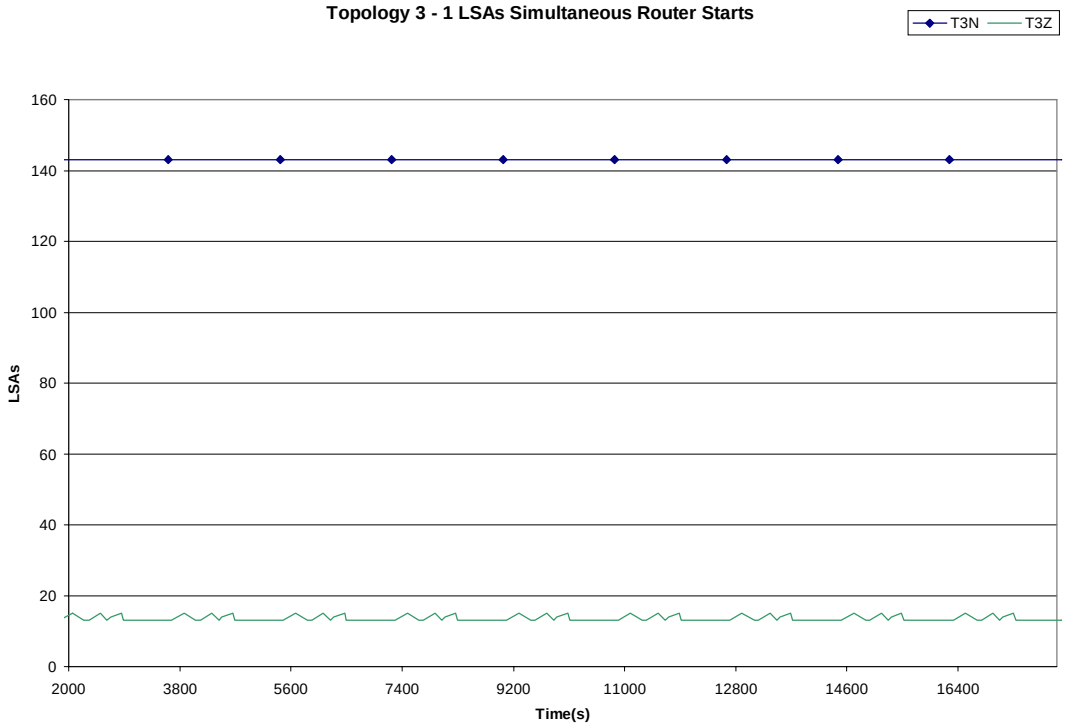

Topology 2 - 30 LSAs Equally Spaced Router Starts
T2N
T2Z
LSAs
60
50
40
30
20
10
0
2000
3800
5600
7400
9200
11000
12800
14600
16400
Time(s)

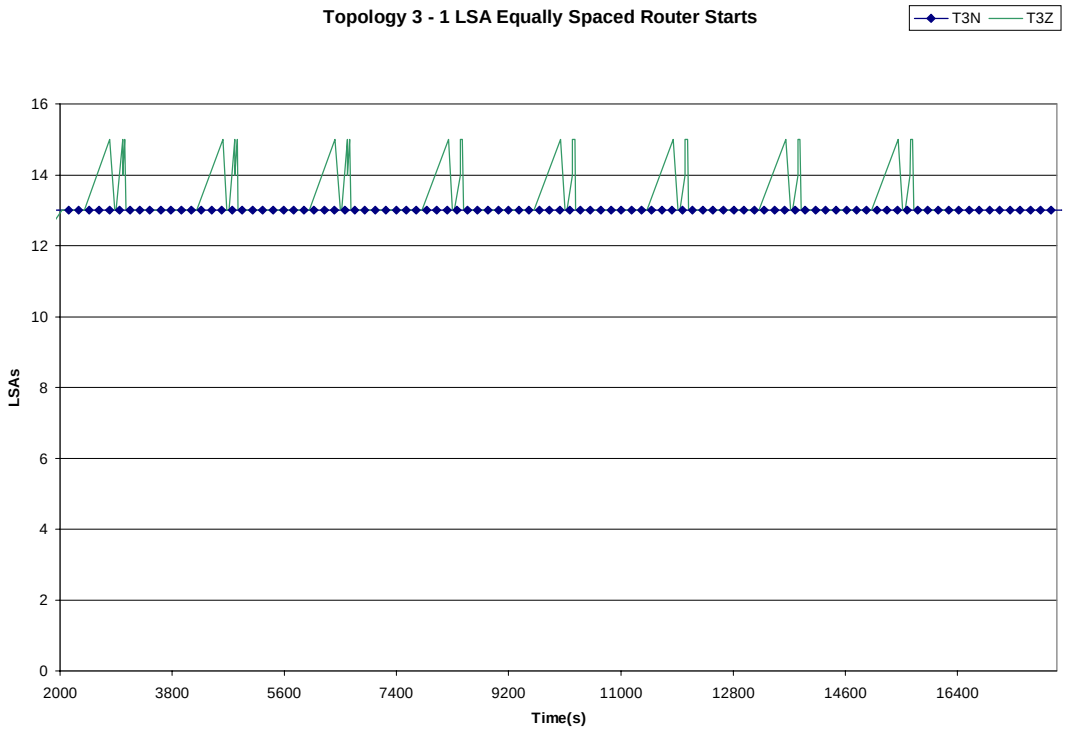

Topology 2 - 30 LSAs Random Router Starts
T2N
T2Z
60
50
40
30
20
10
0
LSAs
2000
3800
5600
7400
9200
11000
12800
14600
16400
Time(s)

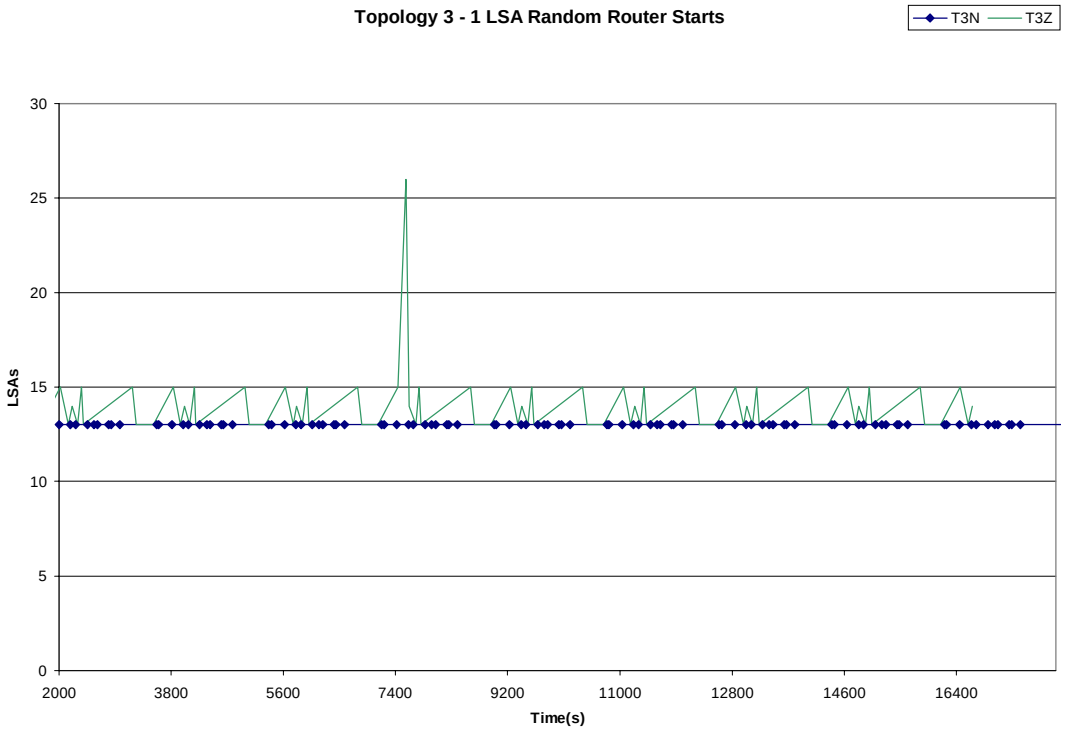

Topology 2 - Random LSAs (max 100) Simultaneous Router Starts
T2N
T2Z
300
250
200
150
100
50
0
LSAs
2000
3800
5600
7400
9200
11000
12800
14600
16400
Time(s)

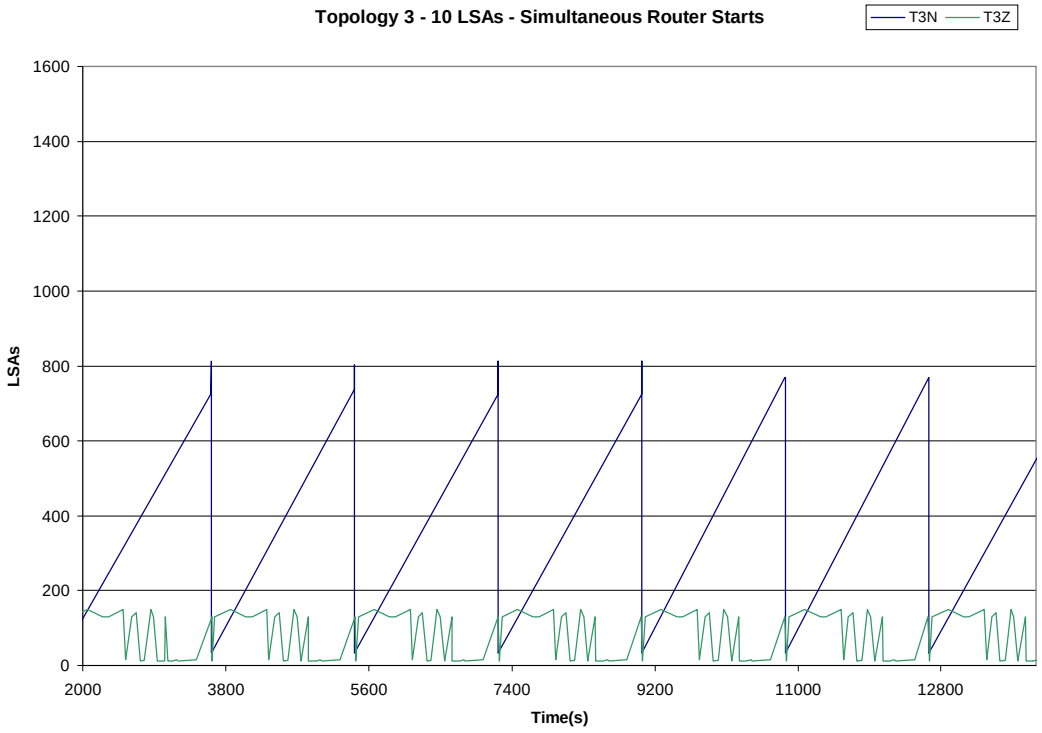

Topology 2 - Random LSAs (max 100) Equally Spaced Router Starts
T2N
T2Z
250
200
150
100
50
0
LSAs
2000
3800
5600
7400
9200
11000
12800
14600
16400
Time(s)

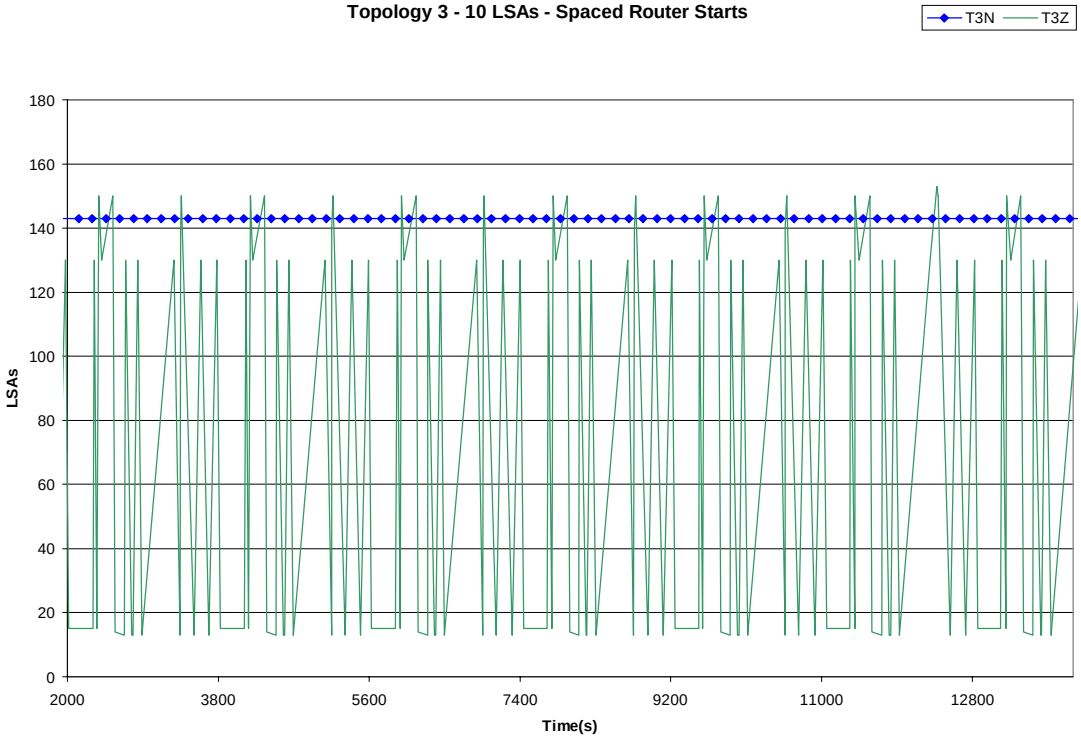

Topology 2 - Random LSAs (max 100) Random Router Starts
T2N
T2Z
250
200
150
100
50
0
LSAs
2000
3800
5600
7400
9200
11000
12800
14600
16400
Time(s)

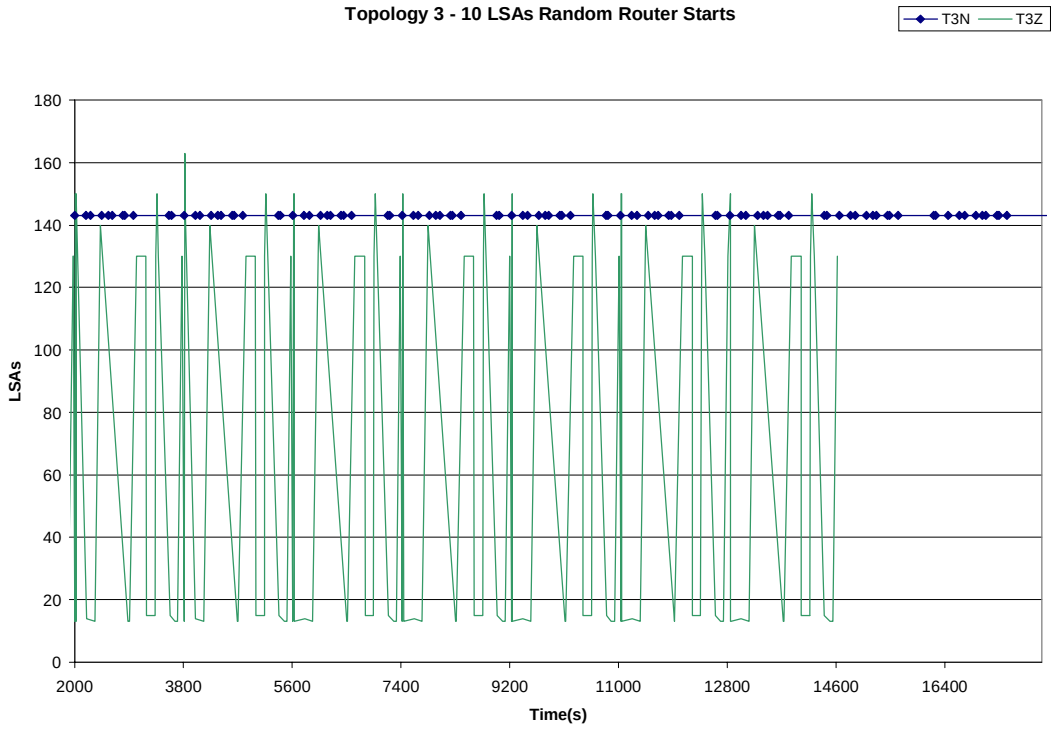

Topology 3 - 1 LSAs Simultaneous Router Starts
T3N
T3Z
LSAs
160
140
120
100
80
60
40
20
0
2000
3800
5600
7400
9200
11000
12800
14600
16400
Time(s)

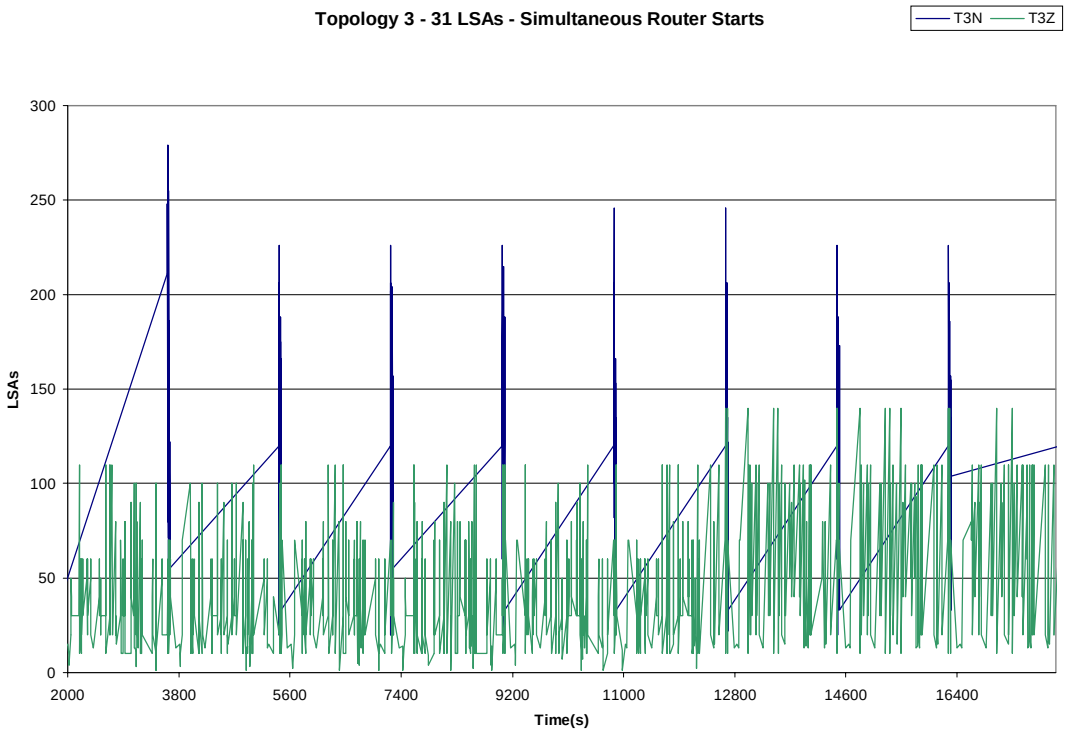

Topology 3 - 1 LSA Equally Spaced Router Starts
T3N
T3Z
LSAs
16
14
12
10
8
6
4
2
0
2000
3800
5600
7400
9200
11000
12800
14600
16400
Time(s)

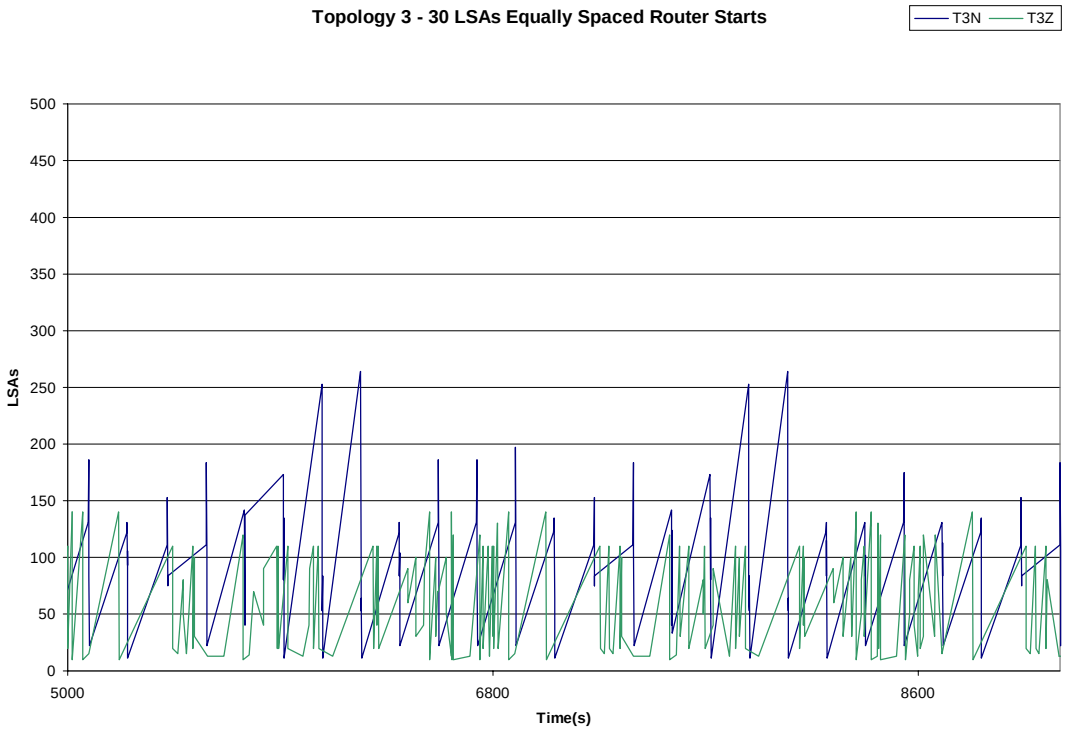

Topology 3 - 1 LSA Random Router Starts
T3N
T3Z
LSAs
30
25
20
15
10
5
0
2000
3800
5600
7400
9200
11000
12800
14600
16400
Time(s)

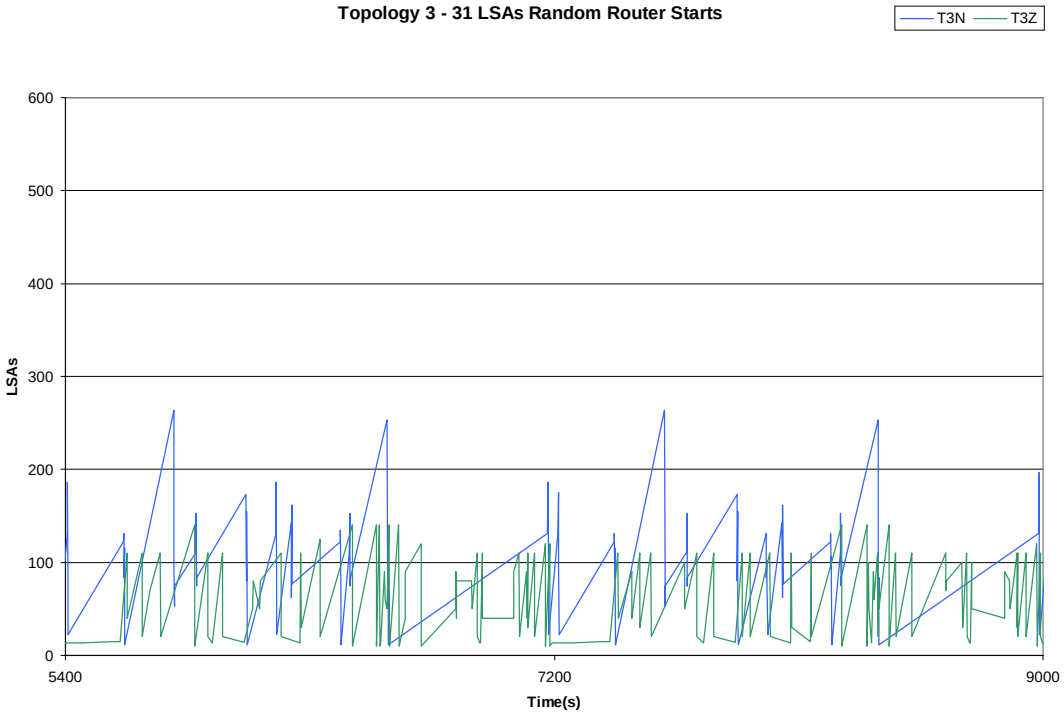

Topology 3 - 10 LSAs - Simultaneous Router Starts
T3N
T3Z
LSAs
1600
1400
1200
1000
800
600
400
200
0
2000
3800
5600
7400
9200
11000
12800
Time(s)

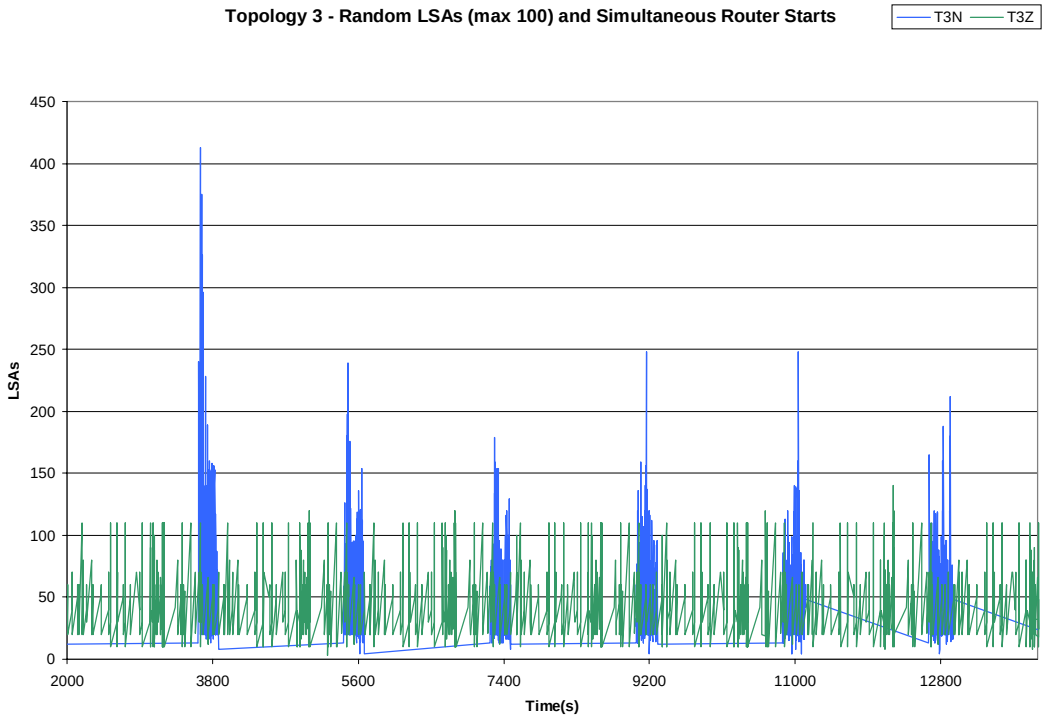

Topology 3 - 10 LSAs - Spaced Router Starts
T3N
T3Z
LSAs
Time(s)
0
20
40
60
80
100
120
140
160
180
2000
3800
5600
7400
9200
11000
12800

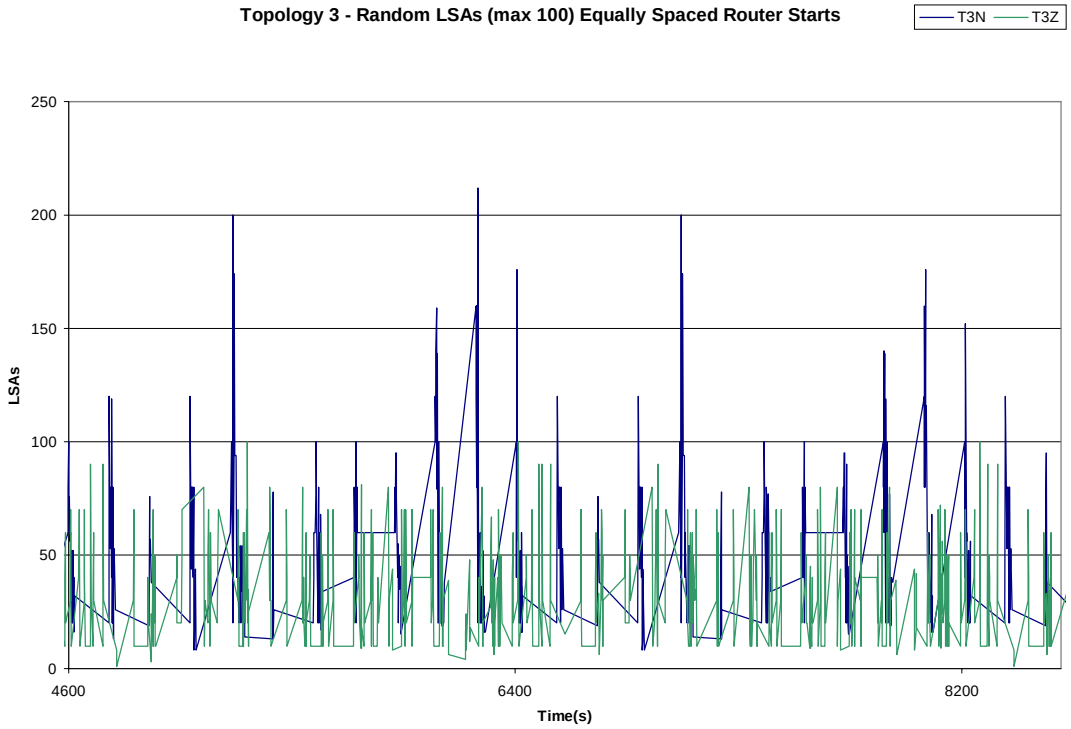

Topology 3 - 10 LSAs Random Router Starts
T3N
T3Z
LSAs
Time(s)
0
20
40
60
80
100
120
140
160
180
2000
3800
5600
7400
9200
11000
12800
14600
16400

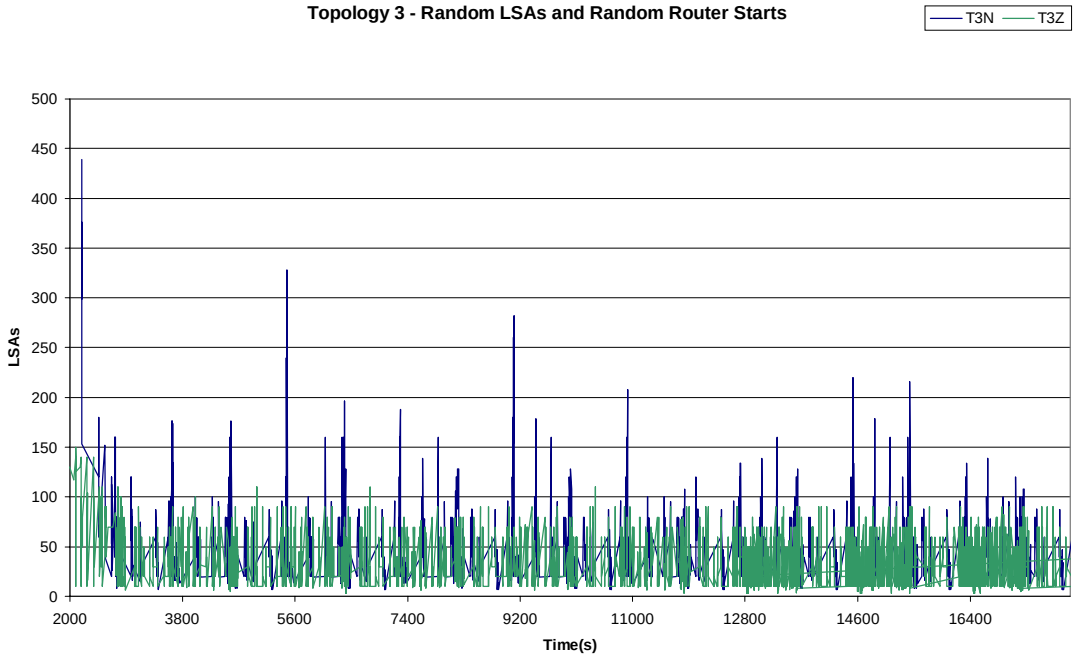

Topology 3 - 31 LSAs - Simultaneous Router Starts
T3N
T3Z
LSAs
300
250
200
150
100
50
0
2000
3800
5600
7400
9200
11000
12800
14600
16400
Time(s)


Topology 3 - 30 LSAs Equally Spaced Router Starts
T3N
T3Z
LSAs
500
450
400
350
300
250
200
150
100
50
0
5000
6800
8600
Time(s)


Topology 3 - 31 LSAs Random Router Starts
T3N
T3Z
600
500
400
300
200
100
0
LSAs
5400
7200
9000
Time(s)

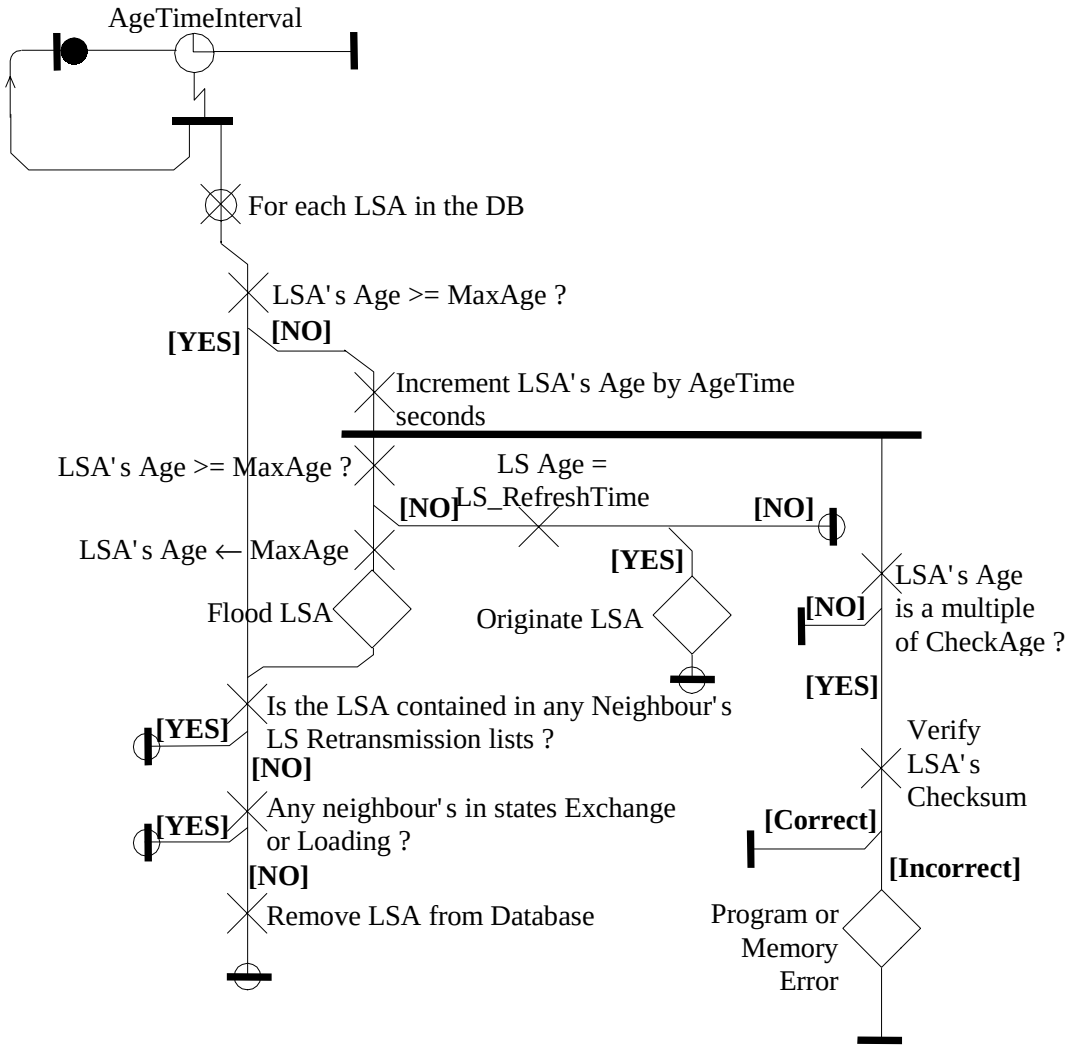

Topology 3 - Random LSAs (max 100) and Simultaneous Router Starts
T3N
T3Z
450
400
350
300
250
200
150
100
50
0
LSAs
2000
3800
5600
7400
9200
11000
12800
Time(s)


Topology 3 - Random LSAs (max 100) Equally Spaced Router Starts
T3N
T3Z
250
200
150
100
50
0
LSAs
4600
6400
8200
Time(s)

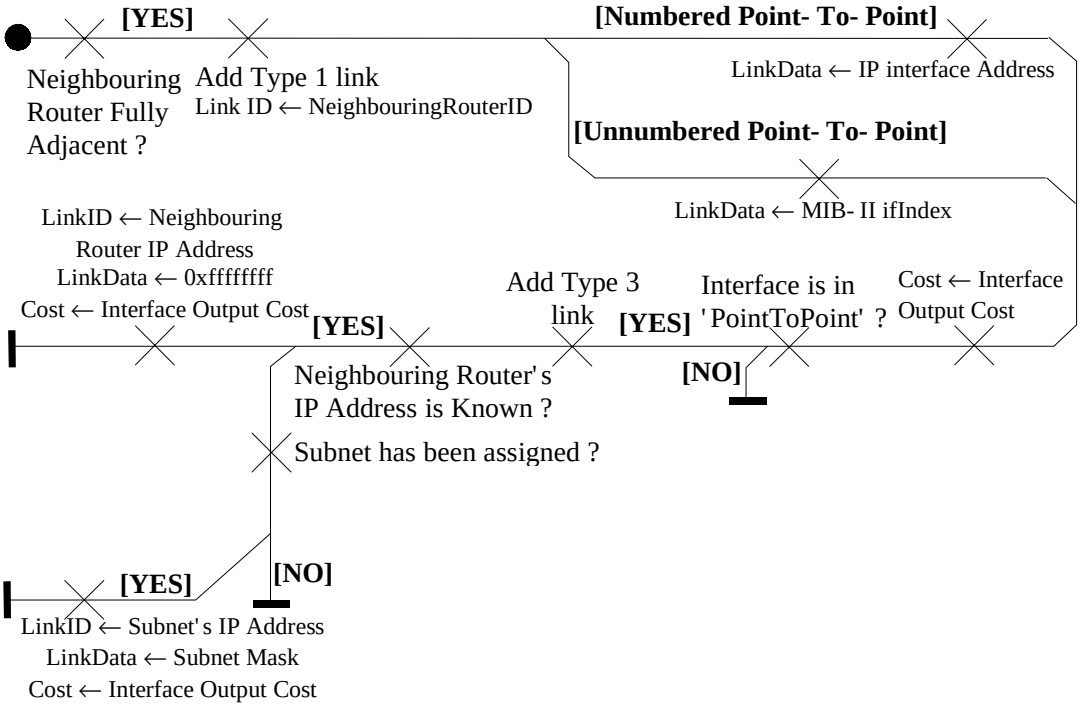

Topology 3 - Random LSAs and Random Router Starts
T3N
T3Z
500
450
400
350
300
250
200
150
100
50
0
LSAs
2000
3800
5600
7400
9200
11000
12800
14600
16400
Time(s)

Appendix B – Use Case Maps from Case Study

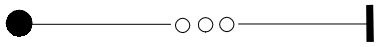
UCM Aging The Link State DB



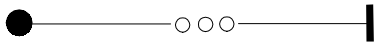
UCM Describe Link {link type = Point To Point} *section 12.4.1.1*



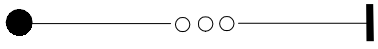
UCM Describe Link {link type = Broadcast | NBMA} *section 12.4.1.2*

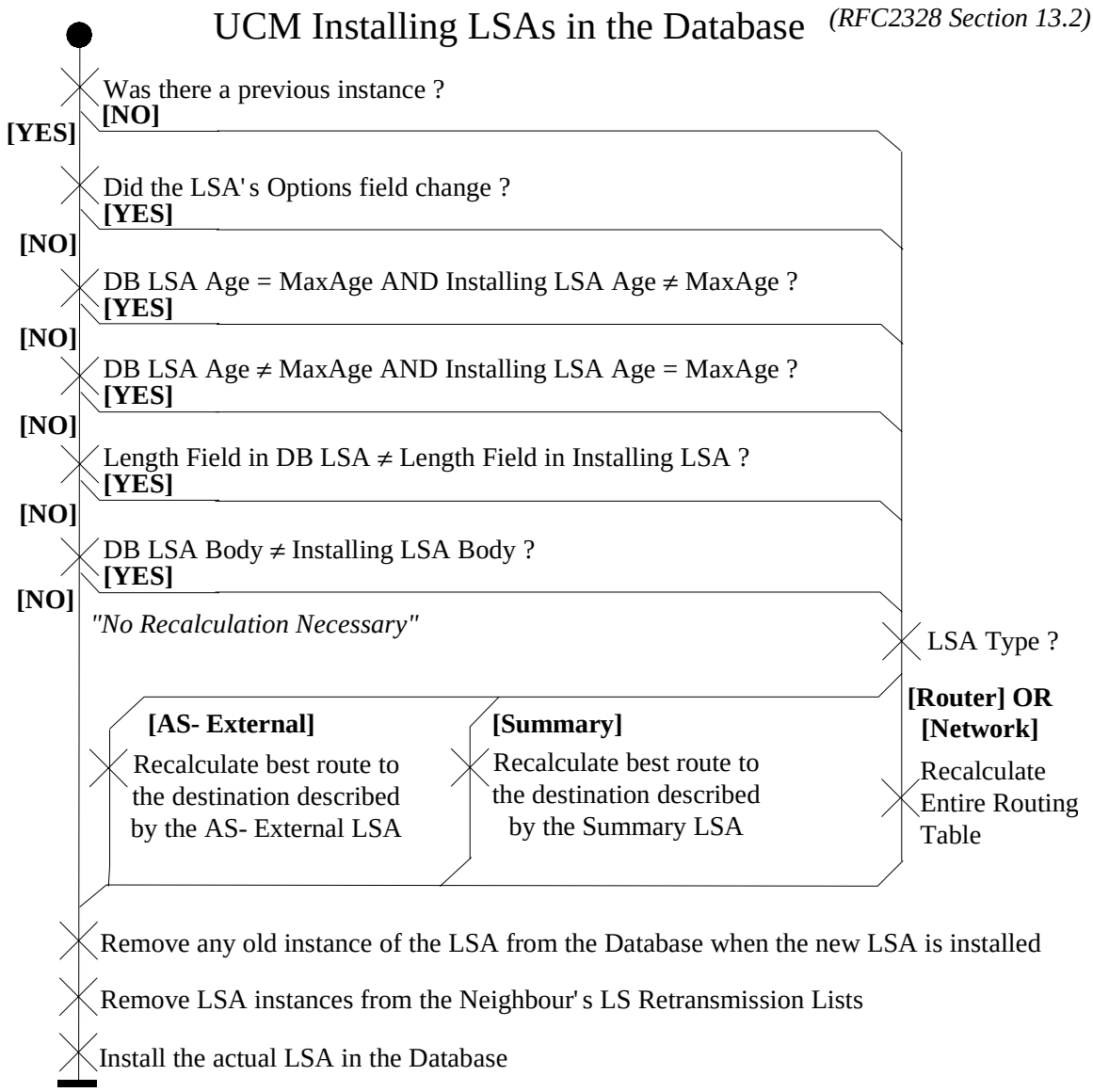


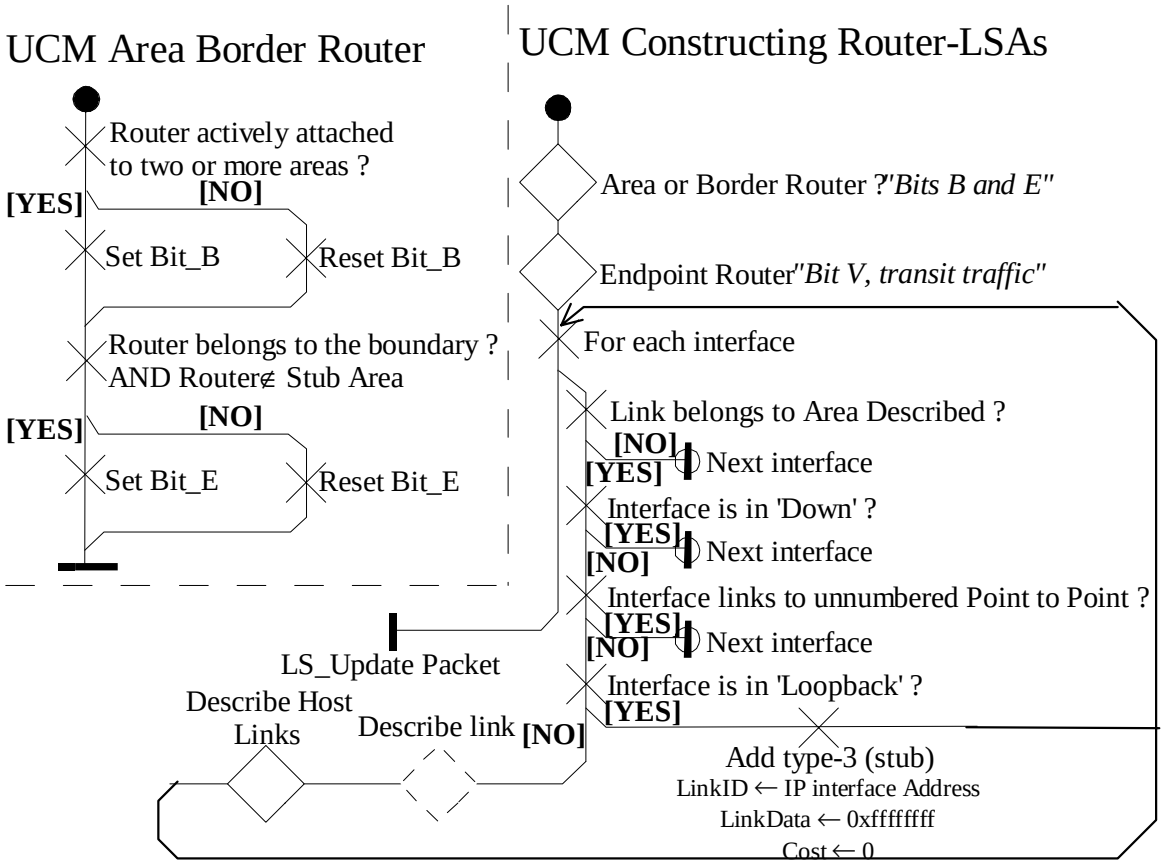
UCM Describe Link {link type = Virtual Link} *section 12.4.1.3*

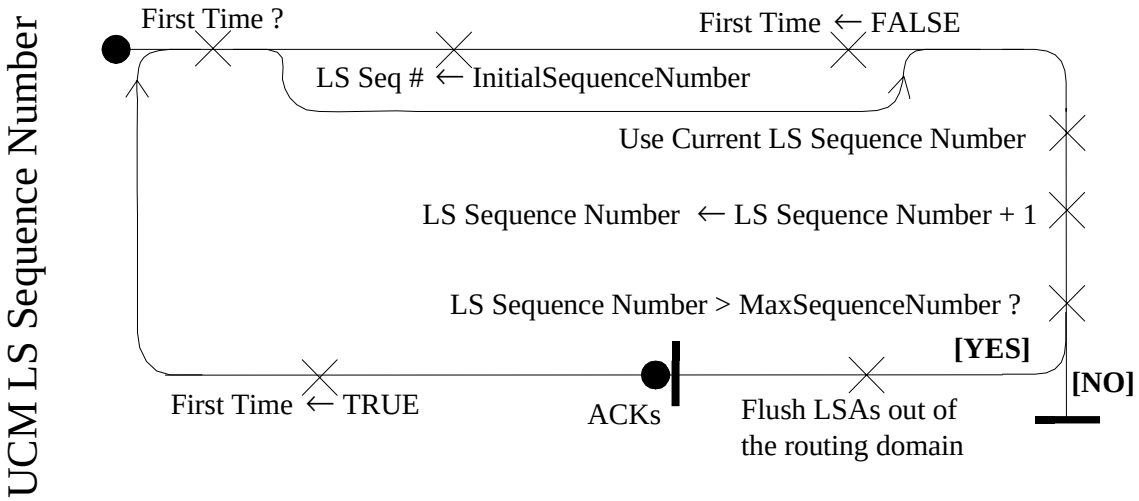


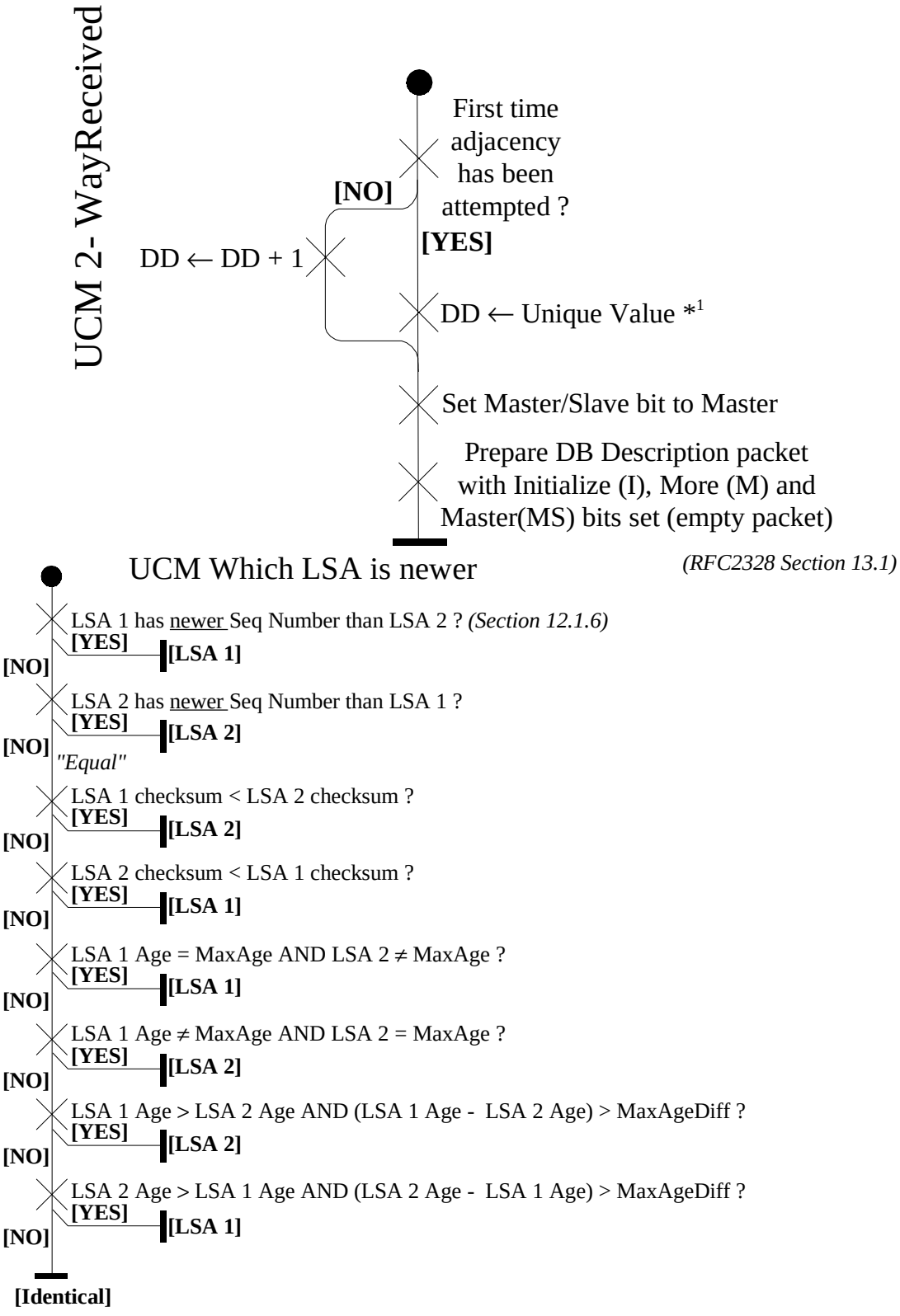
UCM Describe Link {link type = Point To Multicast} *section 12.4.1.4*



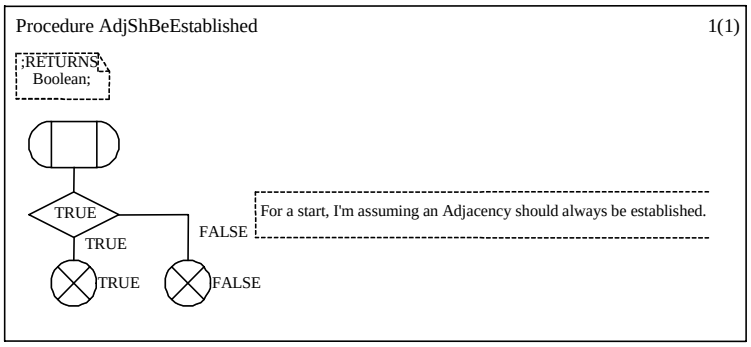

UCM Installing LSAs in the Database
(RFC2328 Section 13.2)
Was there a previous instance ?
[NO]
[YES]
Did the LSA's Options field change ?
[YES]
[NO]
DB LSA Age = MaxAge AND Installing LSA Age ≠ MaxAge ?
[YES]
[NO]
DB LSA Age ≠ MaxAge AND Installing LSA Age = MaxAge ?
[YES]
[NO]
Length Field in DB LSA ≠ Length Field in Installing LSA ?
[YES]
[NO]
DB LSA Body ≠ Installing LSA Body ?
[YES]
[NO]
"No Recalculation Necessary"
LSA Type ?
[AS- External]
Recalculate best route to the destination described by the AS- External LSA
[Summary]
Recalculate best route to the destination described by the Summary LSA
[Router] OR [Network]
Recalculate Entire Routing Table
Remove any old instance of the LSA from the Database when the new LSA is installed
Remove LSA instances from the Neighbour's LS Retransmission Lists
Install the actual LSA in the Database

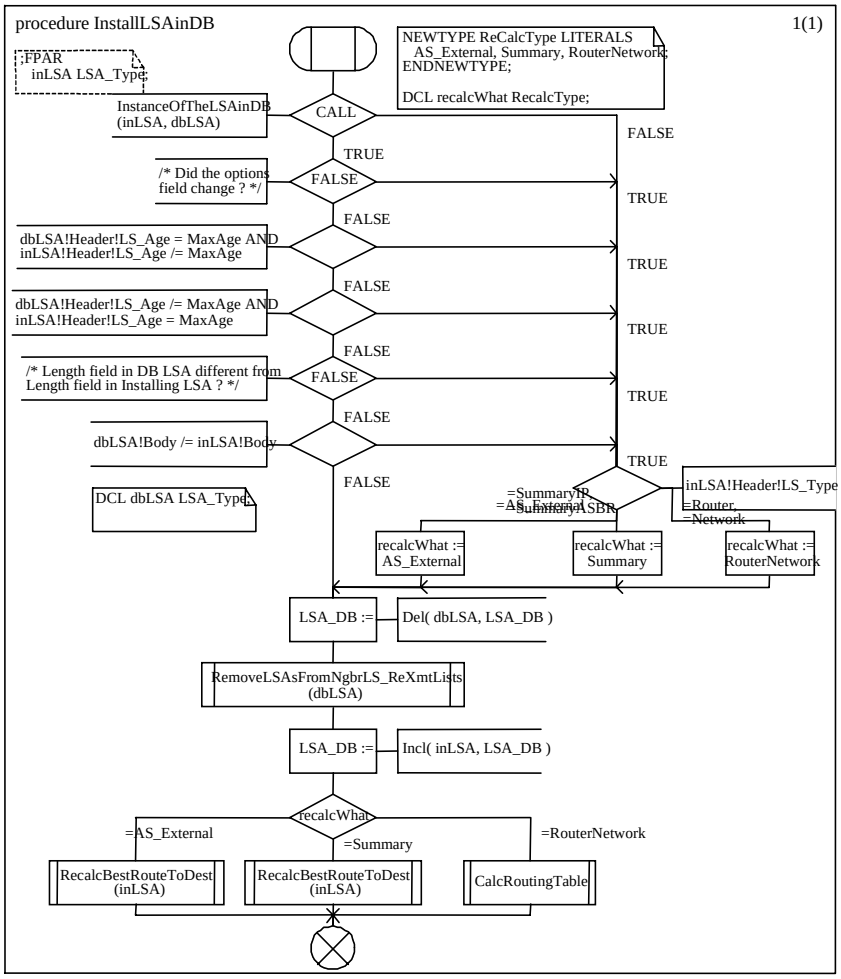

UCM Area Border Router
UCM Constructing Router-LSAs
Router actively attached to two or more areas ?
[YES]
[NO]
Set Bit_B
Reset Bit_B
Router belongs to the boundary ? AND Router ∉ Stub Area
[YES]
[NO]
Set Bit_E
Reset Bit_E
LS_Update Packet
Describe Host Links
Describe link
Area or Border Router ? "Bits B and E"
Endpoint Router "Bit V, transit traffic"
For each interface
Link belongs to Area Described ?
[NO]
[YES]
Next interface
Interface is in 'Down' ?
[YES]
[NO]
Next interface
Interface links to unnumbered Point to Point ?
[YES]
[NO]
Next interface
Interface is in 'Loopback' ?
[YES]
[NO]
Add type-3 (stub)
LinkID ← IP interface Address
LinkData ← 0xffffffff
Cost ← 0

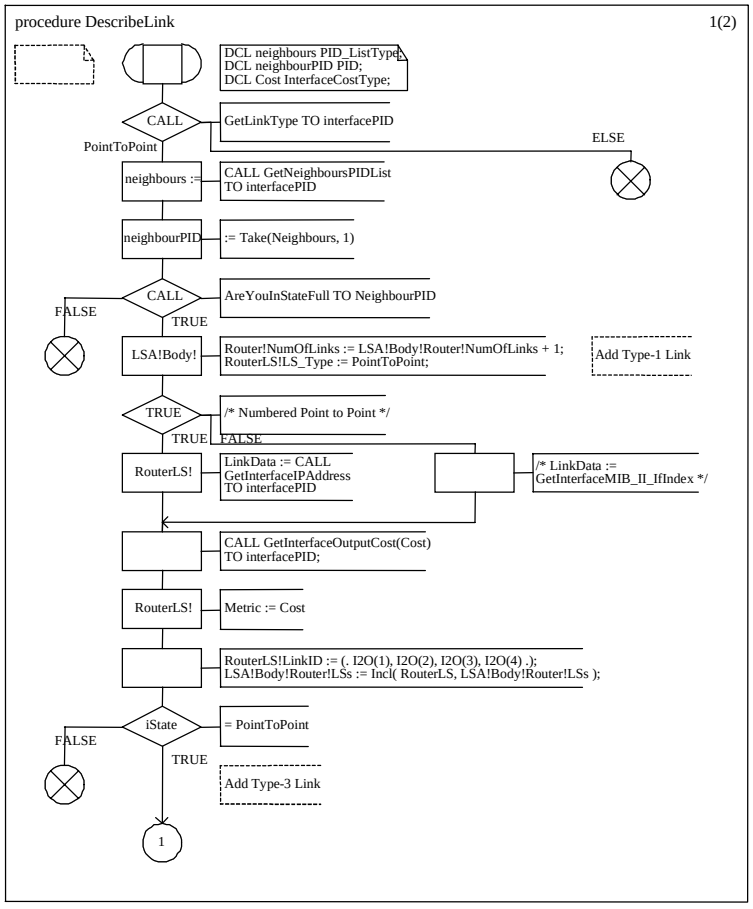

UCM LS Sequence Number
First Time ?
First Time ← FALSE
LS Seq # ← InitialSequenceNumber
Use Current LS Sequence Number
LS Sequence Number ← LS Sequence Number + 1
LS Sequence Number > MaxSequenceNumber ?
[YES]
[NO]
First Time ← TRUE
ACKs
Flush LSAs out of the routing domain

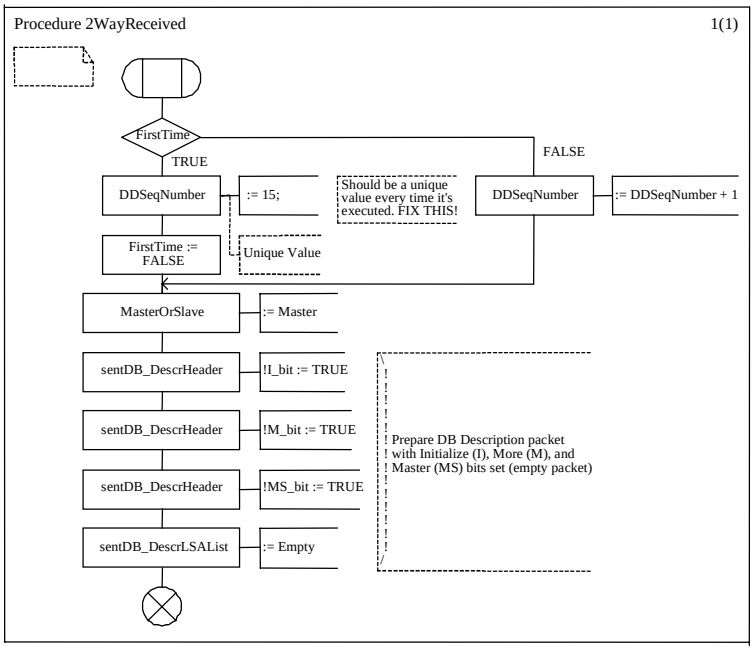

UCM 2- WayReceived
First time adjacency has been attempted ?
[NO]
[YES]
DD ← DD + 1
DD ← Unique Value *1
Set Master/Slave bit to Master
Prepare DB Description packet with Initialize (I), More (M) and Master(MS) bits set (empty packet)
UCM Which LSA is newer
(RFC2328 Section 13.1)
LSA 1 has newer Seq Number than LSA 2 ? (Section 12.1.6)
[YES]
[LSA 1]
[NO]
LSA 2 has newer Seq Number than LSA 1 ?
[YES]
[LSA 2]
[NO]
"Equal"
LSA 1 checksum < LSA 2 checksum ?
[YES]
[LSA 2]
[NO]
LSA 2 checksum < LSA 1 checksum ?
[YES]
[LSA 1]
[NO]
LSA 1 Age = MaxAge AND LSA 2 ≠ MaxAge ?
[YES]
[LSA 1]
[NO]
LSA 1 Age ≠ MaxAge AND LSA 2 = MaxAge ?
[YES]
[LSA 2]
[NO]
LSA 1 Age > LSA 2 Age AND (LSA 1 Age - LSA 2 Age) > MaxAgeDiff ?
[YES]
[LSA 2]
[NO]
LSA 2 Age > LSA 1 Age AND (LSA 2 Age - LSA 1 Age) > MaxAgeDiff ?
[YES]
[LSA 1]
[NO]
[Identical]

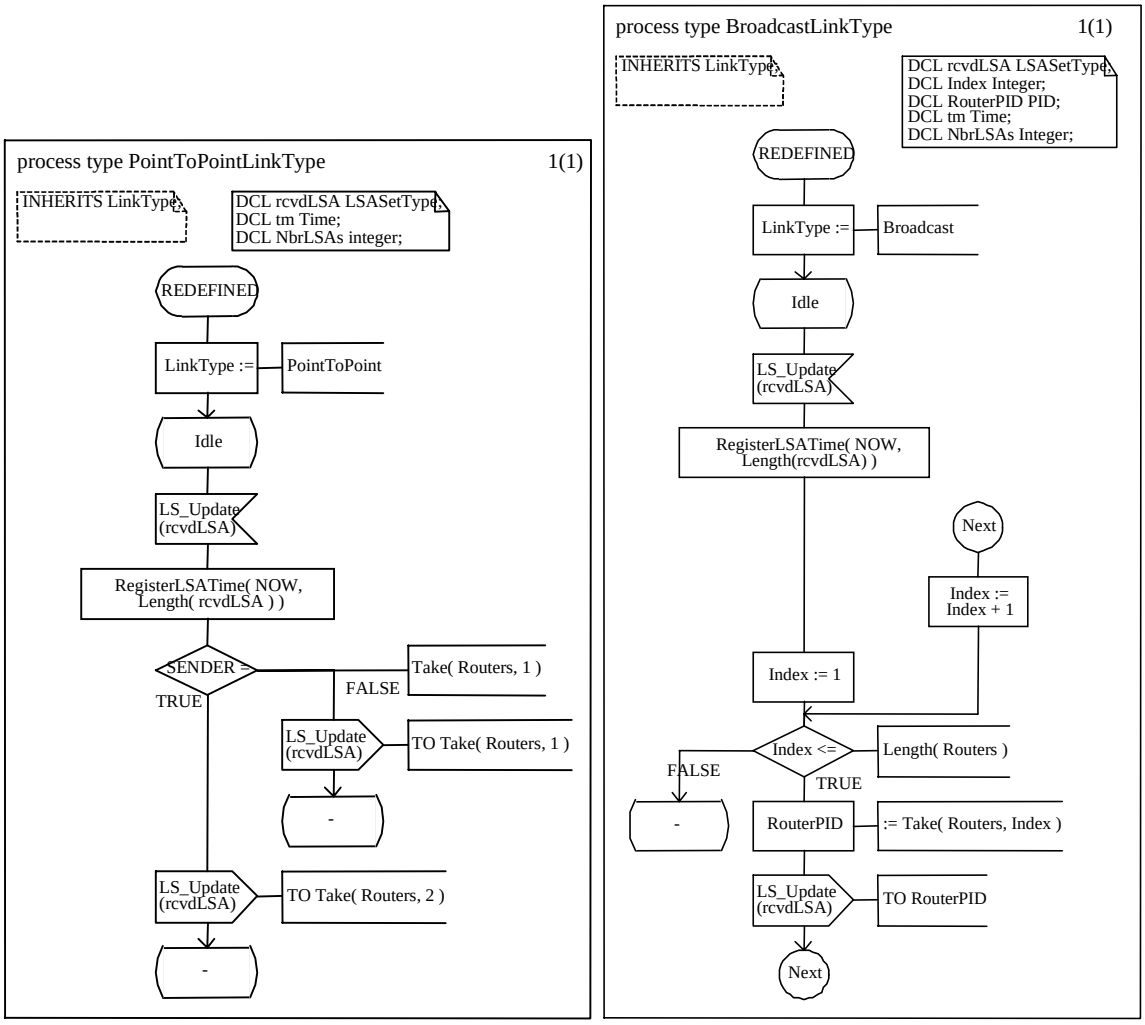
Appendix C – SDL Diagrams from Case Study

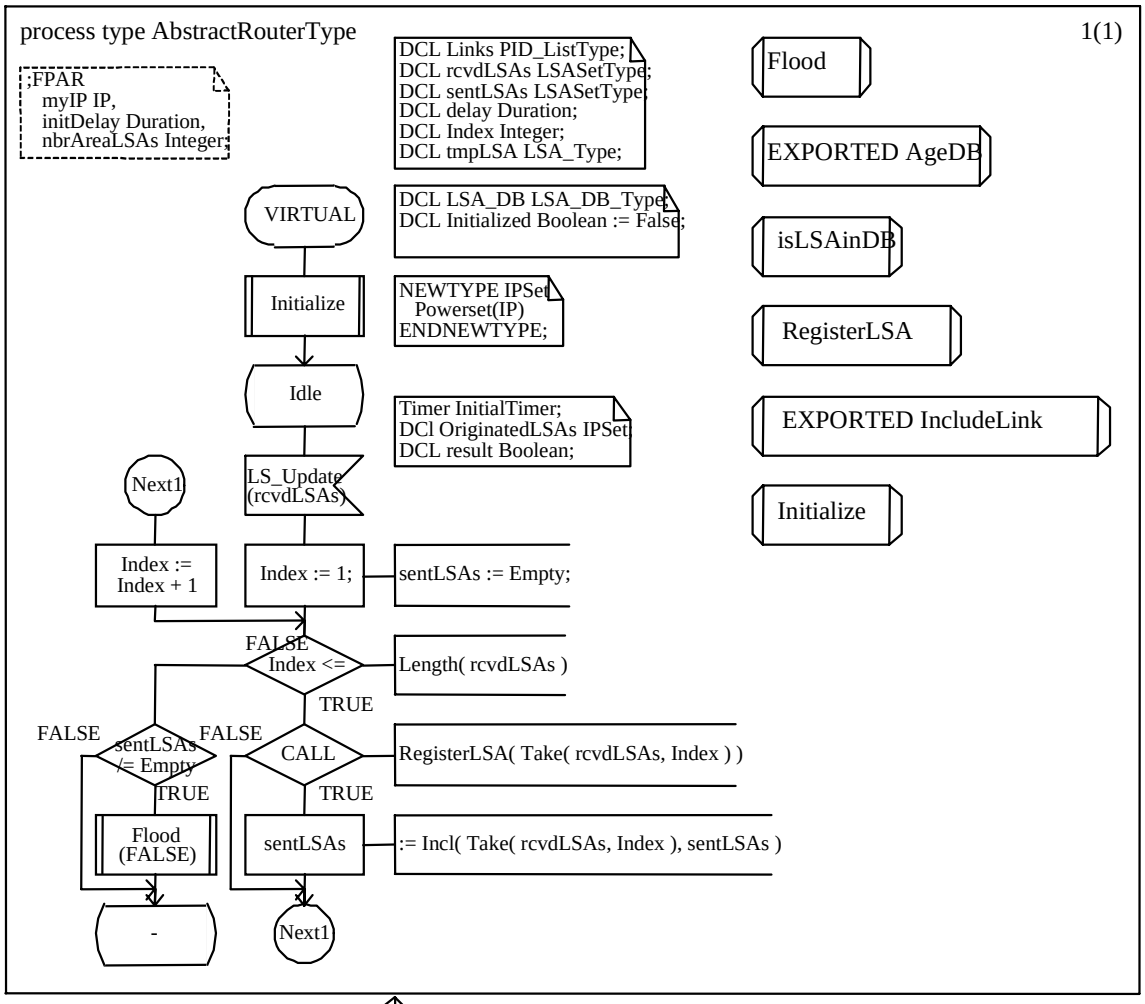


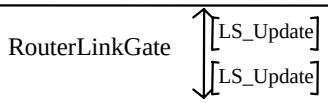


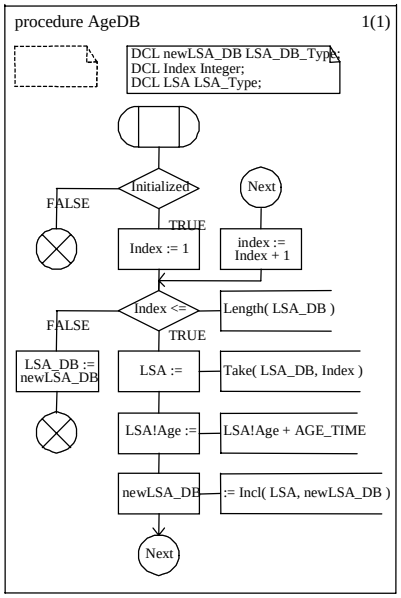

procedure DescribeLink
1(2)
DCL neighbours PID_ListType;
DCL neighbourPID PID;
DCL Cost InterfaceCostType;
CALL
GetLinkType TO interfacePID
PointToPoint
ELSE
neighbours :=
CALL GetNeighboursPIDList TO interfacePID
neighbourPID
:= Take(Neighbours, 1)
CALL
AreYouInStateFull TO NeighbourPID
FALSE
TRUE
LSA!Body!
Router!NumOfLinks := LSA!Body!Router!NumOfLinks + 1;
RouterLS!LS_Type := PointToPoint;
Add Type-1 Link
TRUE
/* Numbered Point to Point */
TRUE
FALSE
RouterLS!
LinkData := CALL GetInterfaceIPAddress TO interfacePID
/* LinkData := GetInterfaceMIB_II_IfIndex */
CALL GetInterfaceOutputCost(Cost) TO interfacePID;
RouterLS!
Metric := Cost
RouterLS!LinkID := (. I2O(1), I2O(2), I2O(3), I2O(4) .);
LSA!Body!Router!LSs := Incl(RouterLS, LSA!Body!Router!LSs);
iState
= PointToPoint
FALSE
TRUE
Add Type-3 Link
1

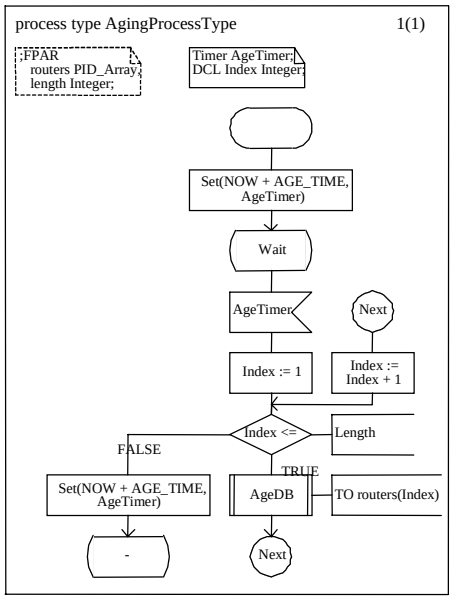

Procedure 2WayReceived
1(1)
FirstTime
TRUE
FALSE
DDSeqNumber
:= 15;
Should be a unique value every time it's executed. FIX THIS!
DDSeqNumber
:= DDSeqNumber + 1
FirstTime := FALSE
Unique Value
MasterOrSlave
:= Master
sentDB_DescrHeader
!I_bit := TRUE
sentDB_DescrHeader
!M_bit := TRUE
sentDB_DescrHeader
!MS_bit := TRUE
sentDB_DescrLSAList
:= Empty
! Prepare DB Description packet
! with Initialize (I), More (M), and
! Master (MS) bits set (empty packet)

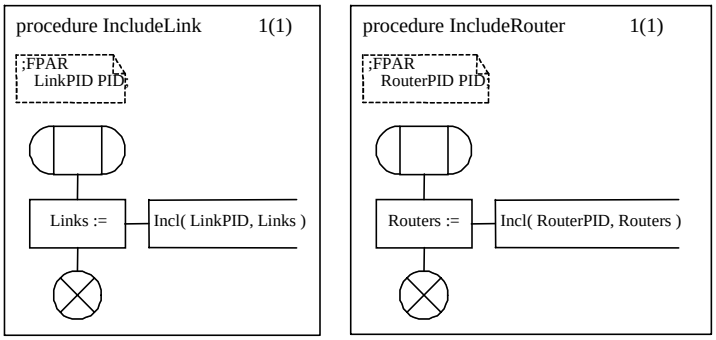

process type PointToPointLinkType 1(1)
INHERITS LinkType;
DCL rcvdLSA LSASetType;
DCL tm Time;
DCL NbrLSAs integer;
REDEFINED
LinkType :=
PointToPoint
Idle
LS_Update (rcvdLSA)
RegisterLSATime(NOW, Length(rcvdLSA))
SENDER =
Take(Routers, 1)
FALSE
TRUE
LS_Update (rcvdLSA)
TO Take(Routers, 1)
-
LS_Update (rcvdLSA)
TO Take(Routers, 2)
-
process type BroadcastLinkType 1(1)
INHERITS LinkType;
DCL rcvdLSA LSASetType;
DCL Index Integer;
DCL RouterPID PID;
DCL tm Time;
DCL NbrLSAs Integer;
REDEFINED
LinkType :=
Broadcast
Idle
LS_Update (rcvdLSA)
RegisterLSATime(NOW, Length(rcvdLSA))
Next
Index := Index + 1
Index := 1
Index <=
Length(Routers)
FALSE
TRUE
-
RouterPID
:= Take(Routers, Index)
LS_Update (rcvdLSA)
TO RouterPID
Next

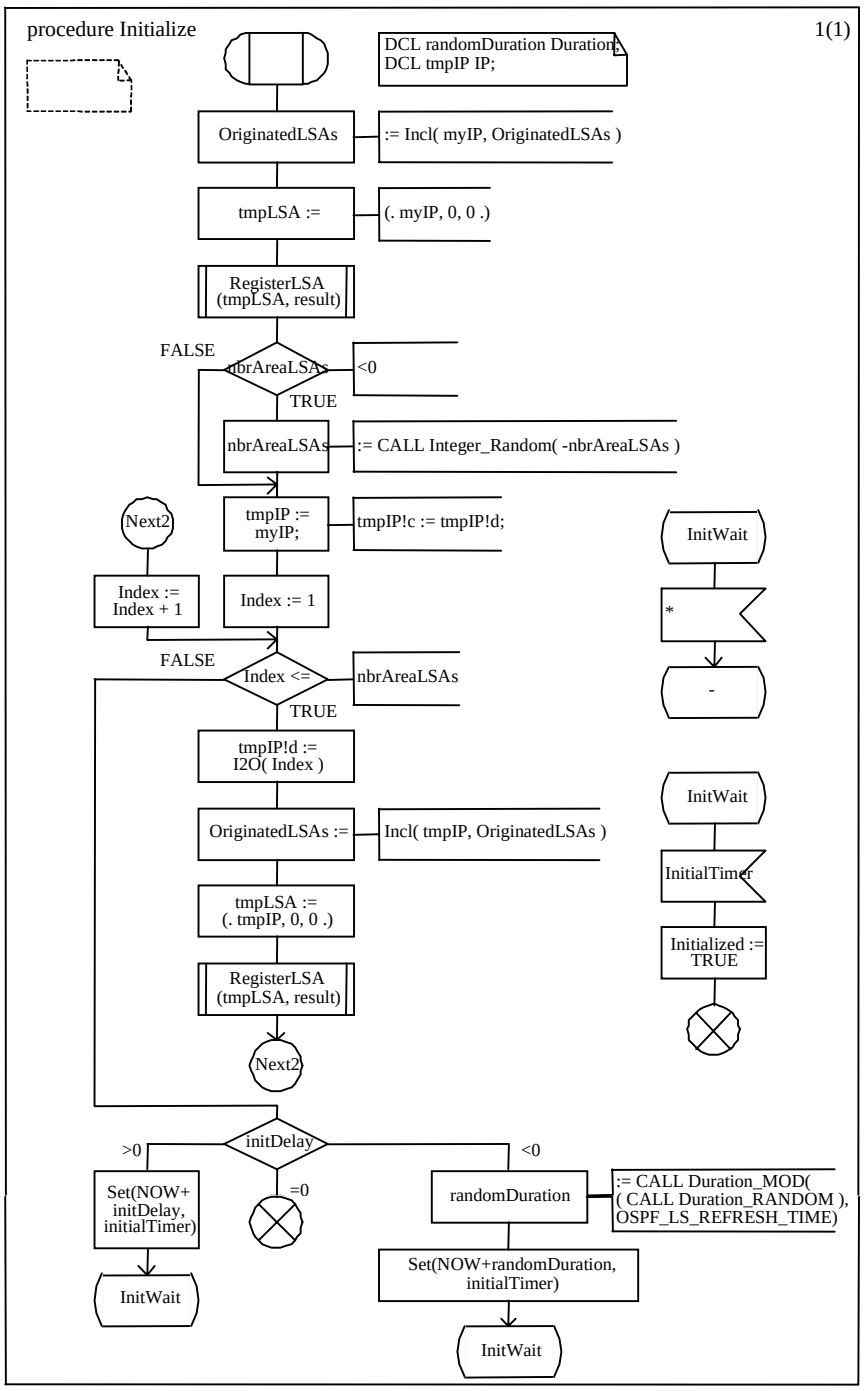

process type AbstractRouterType
1(1)
;FPAR
myIP IP,
initDelay Duration,
nbrAreaLSAs Integer;
DCL Links PID_ListType;
DCL rcvdLSAs LSASetType;
DCL sentLSAs LSASetType;
DCL delay Duration;
DCL Index Integer;
DCL tmpLSA LSA_Type;
DCL LSA_DB LSA_DB_Type;
DCL Initialized Boolean := False;
NEWTYPE IPSet
Powerset(IP)
ENDNEWTYPE;
Timer InitialTimer;
DCl OriginatedLSAs IPSet;
DCL result Boolean;
Flood
EXPORTED AgeDB
isLSAinDB
RegisterLSA
EXPORTED IncludeLink
Initialize
VIRTUAL
Initialize
Idle
LS_Update
(rcvdLSAs)
Next1
Index :=
Index + 1
Index := 1;
sentLSAs := Empty;
FALSE
Index <=
Length(rcvdLSAs)
TRUE
FALSE
sentLSAs
/= Empty
FALSE
CALL
RegisterLSA(Take(rcvdLSAs, Index))
TRUE
TRUE
Flood
(FALSE)
sentLSAs
:= Incl(Take(rcvdLSAs, Index), sentLSAs)
-
Next1


RouterLinkGate
LS_Update
LS_Update


procedure AgeDB
1(1)
DCL newLSA_DB LSA_DB_Type;
DCL Index Integer;
DCL LSA LSA_Type;
Initialized
Next
FALSE
TRUE
Index := 1
index :=
Index + 1
Index <=
Length(LSA_DB)
FALSE
TRUE
LSA_DB :=
newLSA_DB
LSA :=
Take(LSA_DB, Index)
LSA!Age :=
LSA!Age + AGE_TIME
newLSA_DB
:= Incl(LSA, newLSA_DB)
Next


process type AgingProcessType
1(1)
;FPAR
routers PID_Array;
length Integer;
Timer AgeTimer;
DCL Index Integer;
Set(NOW + AGE_TIME,
AgeTimer)
Wait
AgeTimer
Next
Index := 1
Index :=
Index + 1
Index <=
Length
FALSE
TRUE
Set(NOW + AGE_TIME,
AgeTimer)
AgeDB
TO routers(Index)
-
Next


procedure IncludeLink 1(1)
;FPAR
LinkPID PID;
Links :=
Incl(LinkPID, Links)
procedure IncludeRouter 1(1)
;FPAR
RouterPID PID;
Routers :=
Incl(RouterPID, Routers)


procedure Initialize
1(1)
DCL randomDuration Duration;
DCL tmpIP IP;
OriginatedLSAs
:= Incl(myIP, OriginatedLSAs)
tmpLSA :=
(. myIP, 0, 0 .)
RegisterLSA
(tmpLSA, result)
FALSE
nbrAreaLSAs
<0
TRUE
nbrAreaLSAs
:= CALL Integer_Random(-nbrAreaLSAs)
Next2
tmpIP :=
myIP;
tmpIP!c := tmpIP!d;
InitWait
*
-
Index :=
Index + 1
Index := 1
FALSE
Index <=
nbrAreaLSAs
TRUE
tmpIP!d :=
I2O(Index)
InitWait
InitialTimer
OriginatedLSAs :=
Incl(tmpIP, OriginatedLSAs)
tmpLSA :=
(. tmpIP, 0, 0 .)
Initialized :=
TRUE
RegisterLSA
(tmpLSA, result)
Next2
initDelay
>0
<0
=0
Set(NOW+
initDelay,
initialTimer)
randomDuration
:= CALL Duration_MOD(
(CALL Duration_RANDOM),
OSPF_LS_REFRESH_TIME)
Set(NOW+randomDuration,
initialTimer)
InitWait
InitWait

References

[1] Alhir, S., *UML in a Nutshell, A Desktop Quick Reference*, September 1998, O'Reilly and associates, Sebastopol California

[2] Amyot, D., Andrade, R., Logrippo, L., Sincennes, J., Yi, Z., *Formal Methods for Mobility Standards*, in proceedings of IEEE 1999 Emerging Technology Symposium on Wireless Communication & Systems, Dallas, USA, 1999, http://www.usecasemaps.org/UseCaseMaps/pub/ets99.pdf

[3] Amyot, D., *Formalization of Timethreads Using LOTOS*, Thesis M.C.S., 1994, University of Ottawa

[4] Amyot, D., *Specification and Validation of Telecommunications Systems with Use Case Maps and LOTOS*, Thesis (Ph.D.), School of Information Technology and Engineering, University of Ottawa, Canada, (Submitted January 2001)

[5] Amyot, D., Logrippo, L., *Use Case Maps and LOTOS for the Prototyping and Validation of a Mobile Group Call System*, in Computer Communications, Special Issue of Formal Description Techniques, Vol. 23, No. 8, 2000, pp. 1135-1157.

[6] Asynchronous Transfer Mode – ATM Forum, *website of the ATM Forum*, http://www.atmforum.com (Accessed April 2001)

[7] Black, U., *OSI: A Model for Computer Communications Standards*, Prentice-Hall, Englewood Cliffs, New Jersey, 1991.

[8] Bordeleau, F., *A systematic and traceable progression from scenario models to communicating hierarchical state machines*, Thesis (Ph.D.), Carleton University, 2000

[9] Bordeleau, F., Cameron, D., *On the relationship between Use Case Maps and Message Sequence Charts*, in Proceedings of The 2nd Workshop of the SDL Forum Society on SDL and MSC (SAM2000), Grenoble, France, June 2000

[10] Bradner S., *Internet Standards Process – Revision 3, Best Current Practice*, http://www.ietf.org/rfc/rfc2026.txt (Accessed April 2001), esp. pp. 6 and pp. 12.

[11] Bræk, R., Øystein, H., *Engineering Real Time Systems, An object-oriented methodology using SDL*, Prentice Hall, New York, 1993, esp. pp. 343-347

[12] Buhr, R., *Use Case Maps as Architectural Entities for Complex Systems*, in proceedings of IEEE Transactions on Software Engineering, Special Issue in Scenario Management, Vol. 24, No. 12, December 1998, pp. 1131-1155.

[13] Buhr, R., Casselman, R., *Use Case Maps for Object-Oriented Systems*, Upper Saddle River, New Jersey, 1995, Prentice-Hall.

[14] Charfi, L., *Formal Modeling and Test Generation Automation with Use Case Maps and LOTOS*, Thesis (M.C.S.), University of Ottawa (Submitted April 2001).

[15] Cinderella Inc., Cinderella SDL, http://www.cinderella.dk/csdl.html (Accessed April 2001)

[16] Ellsberger, J., Hogrefe D., Sarma A., *SDL: Formal Object-Oriented Language for Communicating Systems*, Second Edition, London, 1997, Prentice Hall.

[17] European Telecommunications Standardization Institute – ETSI, *Methods for testing the use of formal SDL as a descriptive tool,* ETSI EG 202 106 v1.1.1, October 1999, ETSI

[18] European Telecommunication Standardization Institute – ETSI, *Tree and Tabular Combined Notation, TTCN version 3*, http://www.ttcn-3.com (Accessed April 2001)

[19] European Telecommunications Standards Institute – ETSI, *Website of the European Telecommunications Institute*, http://www.etsi.org (Accessed April 2001)

[20] Garavel, H., Sifakis, J., *Compilation and verification of Lotos specifications*, in Proceedings of the 10th International symposium on Protocol Specification, Testing and Verification, ed. R. Probert, L. Logrippo, H. Ural, Ottawa, 1990. IFIP, North-Holland

[21] Gill, A., Introduction to the theory of finite-state machines, McGraw-Hill, New York, 1962

[22] Holzmann, G., *Design and Validation of Computer Protocols*, Prentice Hall, 1991, New Jersey, esp. pp. 217-244.

[23] Humphrey, W., *Introduction to the Personal Software Process (Sei Series in Software Engineering),* 1996, Addison-Wesley Publishing Company.

[24] International Federation for Information Processing – IFIP, ed. H. Ural, R. Probert, G. Bochmann, *Testing of Communicating Systems, 13th International Conference on*

Testing of Communicating Systems (TestCom 2000), Ottawa, Canada, 2000, Kluwer Academic Publishers

[25] International Federation for Information Processing – IFIP, *Protocol Specification, Testing and Verification, XIII*, ed. A. Danthine, G. Leduc, P. Wolper, Proceedings of the IFIP TC6/WG6.1 13th International Symposium on Protocol Specification, Testing and Verification, Liège, Belgium, 1993, North-Holland Publishers

[26] International Federation for Information Processing – IFIP, *Formal Description Techniques – FORTE, VI*, ed. R. Tenney, P. Amer, M. Uyar, Proceedings of the IFIP TC6/WG6.1, 6th International Conference on Formal Description Techniques – FORTE'93, Boston, U.S.A, October 1993, North-Holland Publishers.

[27] Institute for Systems Programming, CASE tool department, *MOST: The Moscow Synthesizer Tool*, http://case.ispras.ru/most/index.html (Accessed April 2001)

[28] International Organization for Standardization - ISO, Information Processing Systems, Open Systems Interconnection, *LOTOS – A Formal Description Technique Based on the Temporal Ordering of Observational Behaviour*, IS 8807, Geneva, 1989

[29] International Organization for Standardization – ISO, *Website of the International Organization for Standardization*, http://www.iso.ch (Accessed April 2001)

[30] International Telecommunication Union - ITU, *ITU-T Recommendation A.8: Alternative Approval Process for new and revised Recommendations* http://www.itu.int/itudoc/itu-t/approved/a/a8_ww9.doc (Accessed April 2001), October 2000

[31] International Telecommunication Union - ITU, *Recommendation Z.100: Specification and Description Language (SDL)*, ITU, Geneva, 1996

[32] International Telecommunication Union - ITU, *Recommendation Z. 120: Message Sequence Chart (MSC)*, ITU, Geneva, 1996

[33] International Telecommunication Union - ITU, *Website of the International Telecommunication Union*, http://www.itu.int/home/index.html (Accessed April 2001)

[34] International Telecommunication Union, ITU-T Sector, *Information paper for Participants, Rapporteurs and Chairmen (January 2001)*, http://www.itu.int/ITU-T/info/participants_info.html#3 (Accessed April 2001)

[35] International Telecommunication Union – ITU-T Sector, Study Group 10, Question 12, *URN: User Requirements Notation*, http://www.itu.int/ITU-T/com10/questions-sg10.doc (Accessed April 2001)

[36] Internet Engineering Task Force – IETF, *Website of the Internet Engineering Task Force*, http://www.ietf.org (Accessed April 2001)

[37] Knightson, K., *Osi Protocol Conformance Testing : IS 9646 Explained (McGraw Hill Series on Computer Communications)*, New York, 1993, McGraw Hill

[38] Mansurov, N., Zhukov, D., *Automatic synthesis of SDL models in Use Case Methodology*, in SDL'99: The Next Millenium, ed. R.Dssouli, G.Bochmann, Y.Lahav, proceedings of the ninth SDL Forum, Montréal, Québec, Canada, 1999, Elsevier Science.

[39] Miga, A., *Application of Use Case Maps to system design with tool support*, Thesis M.Eng, Carleton University, 1998

[40] Miga, A., *UCMNAV – A Use Case Maps Navigator*, http://www.usecasemaps.org/tools/ucmnav/index.html (Accessed April 2001)

[41] Monkewich, O., *Conformance ATS for Available Bit Rate (ABR) Source and Destination Behaviours*, af-test-rm-0157.000, 2001, ATM Forum.

[42] Monkewich, O., Sales, I., *Verification of Alex Zinin's Proposal for Efficient LSA Refreshment*, Presentation to the 49th IETF meeting, OSPF Working Group, San Diego, California, United States, December 2000.

[43] Monkewich, O., Sales, I., *SDL Simulations of Alex Zinin's Proposal for Efficient LSA Refreshment*, Presentation to the 50th IETF meeting, OSPF Working Group, Minnesota, United States, March 2001.

[44] Moy, J., *OSPF: Anatomy of an Internet Routing Protocol*, Addison-Wesley Longman, 1998

[45] Moy, J., *Open Shortest Path First version 2*, http://www.ietf.org/rfc/rfc2178.txt (Accessed April 2001)

[46] Moy, J., *Open Shortest Path First version 2*, http://www.ietf.org/rfc/rfc2328.txt (Accessed April 2001)

[47] Probert et al., *FAST Spec-To-Test Project*, Report to MITEL Corporation, confidential, April 2000.

[48] Probert, R., Lew, N., *Protocol quality engineering: addressing industry concerns about formal methods*, in Computer Communications Special issue on Protocol Engineering, pp. 1258-1267, vol. 19, Elsevier publishers, 1996.

[49] Probert, R., Williams, A., *Fast Functional Test Generation Using an SDL model*, in proceedings of the 12th International Conference on Testing Communicating Systems, IWTCS'99, Kluwer Publishers, Hungary, 1999, pp.

[50] Probert, R., Ural, H., Williams, A., *Rapid generation of functional tests using MSCs, SDL and TTCN*, in Computer Communications, Vol. 24, pp. 374-393, Elsevier publishing, 2001.

[51] Quemada, J., Fernández, A., Mañas J., LOLA: Design and Verification of Protocols, Lisbon, May 1987. Also in Computer Communication Systems, Cerveira, A., (ed) North-Holland (1988).

[52] Rational Corporation Inc., *Rational Rose v2001: Visual Modeling UML, Object-Oriented, Component-Based Development with Rational Rose*, http://www.rational.com/products/rose/index.jsp (Acccssed April 2001)

[53] Sales, I., Probert, R., *From High-Level Behaviour to High-Level Design: Use Case Maps to Specification and Description Language*, in proceedings of the 18th Brazilian Symposium on Computer Networks, SBRC'2000, pp. 457-471, Belo Horizonte, Minas Gerais, 2000

[54] Sarikaya, B., *Principles of Protocol Engineering and Testing*, New York, Ellis Horwood, 1993

[55] Schneider, G., Winters, J., *Applying Use Cases, a Practical Guide*, 1998, Addison Wesley Longman

[56] Selic, B. et al, *Real-time Object-Oriented Modelling*, New York, 1994, John Wiley and Sons.

[57] Specification and Description Language (SDL) Forum, *Specification and Description Language Forum 99 – SDL'99 – The next millenium*, ed. R. Dssouli, G.

Bochmann, Y. Lahav, Proceedings of the 9th SDL Forum, Montréal, Québec, Canada, 1999, Elsevier Science.

[58] Stallings, W., *Data and Computer Communications*, 8th Edition, Upper Saddle River, New Jersey, Prentice Hall, 2000.

[59] Tanenbaum, A., *Computer Networks*, 3rd Edition, Upper Saddle River, New Jersey, Prentice Hall, 1996.

[60] Telelogic Inc., *Telelogic Object GEODE*, http://www.telelogic.com/objectgeode (Accessed April 2001)

[61] Telelogic Inc., *Telelogic TAU v4.0*, June 2000, http://www.telelogic.com/SDL/default.asp (Accessed April 2001)

[62] Turner, K., *Using Formal Description Techniques: An introduction to Estelle, LOTOS, and SDL*, New York, John Wiley and Sons, 1993.

[63] Turner, K., *An engineering approach to formal methods*, in Dathine, A., Leduc, G., Wolper, P., editors, proceedings of Protocol Specification, Testing and Verification XIII, Amsterdam, Netherlands, 1993, North Holland Publishers, pp. 357-380.

[64] Use Case Maps User Group, *Website of Use Case Maps*, http://www.usecasemaps.org (Accessed April 2001)

[65] Zinin, A., *Guidelines for Efficient LSA Refreshment in OSPF*, http://community.roxen.com/developers/idocs/drafts/draft-ietf-ospf-refresh-guide-01.txt (Accessed April 2001)

[66] Zinin, A., *Informational RFC on the efficient LSA refreshment function*, to be published on the IETF website.