NTNU – Trondheim
Norwegian University of
Science and Technology

Automated routing confiiguration

Marija Gajić

Submission date: December 2019
Responsible professor: Finn Arve Aagesen, NTNU IIK
Supervisor: Otto Jonasen Wittner, NTNU IIK

Norwegian University of Science and Technology
Department of Communication Technology and Information Security

Abstract

Nowadays Internet Service Providers are facing increase in usage of different services and growing number of customers. Network Management has become very important secondary network functionality. One of the network management tasks is routing configuration. Traditional IP networks are required to be dynamic, flexible and react in real time to the events that might impact its users (e.g. link failure, overload, flapping etc.). In case of such events, changes in routing configurations often require manual interaction and adjustments of static link metrics. This lowers the efficiency of the network management procedures in terms of utilization of both time and available resources. Work presented here introduces an Adaptive Network Routing Configuration framework for OSPF/IS-IS based IP networks. The framework stands on XML Equivalent Transformation logic applied as reasoning procedure for route choices. This reasoning procedure is a way towards automatic and adaptive routing decision making process. One example case scenario of network enforcing the framework was described and simulated. We investigated how router´s buffer size and interarrival times of the traffic flows impact the performance of the framework. Finally, we compared performance of the framework with the current traditional IP networks setup.

Keywords: *adaptive IP networks, automated routing configuration, IS-IS, OSPF, XML Equivalent Transformation, reasoning engine.*

To my brother, Veljko

Preface

This master thesis was conducted as an introduction to my integrated Ph.D studies research work.

Hereby I would like to say thank you to my supervisor, professor Finn Arve Aagesen. Thank you for choosing me to be your Ph.D candidate, thank you for believing in me, and thank you for all the help and fruitful discussions we have had during this work.

I would also like to thank to my co-supervisor Otto J. Wittner for his suggestions, ideas, and for giving me another perspective about things we were working on. I owe a huge gratitude to Chutiporn Antariya for explanations and help related to the reasoning machine she created. Furthermore, I would like to say thanks to Svein Ove Undal from Uninet, for hints which lead me to the main idea of this project.

Big thanks to my colleges and friends - Poul, Katina, Besmir and Romina for giving me motivation and making me laugh every day at work.

My deepest gratitude goes to VEGA d.o.o company, its owner and my friend Stanko Jesić who supported me since my early teenage years. You have made a huge impact on my life. Thank you for being such a perfect role model.

Mama i Tata, tako je teško napisati nesto dostojno Vas. Kada me pitaju šta je tajna uspeha, kažem im da nema tajne. Ja sam samo imala sreće da budem Vaša ćerka. Vi ste moja tajna, Vi ste moj jedini trik. Ne postoji jedno hvala dvoljno veliko za to.

At the very end, I would like to dedicate this thesis to my brother, Veljko. My dearest little one, this is for all the days I was not there with you.

Contents

List of Figures

List of Tables

List of Acronyms

ECP Equal - Cost MultiPath Routing.

LDM Load Distribution in MPLS.

LSA Link State Advertisment.

MPLS Multiprotocol Label Switching.

NOC Network Operation Center.

PEMS Periodic Multi-Step MPLS traffic engineering technique.

PER Prediction of Effective Reparation.

QoS Quality of Service.

RM Reasoning Machine.

SLA Service Level Agreement.

TAPAS Telematics Architecturefor Play-based Adaptable Service System.

XDD XML Declerative Description.

XET XML Equivalent Transformation.

Chapter 1
Introduction

Network management is secondary network functionality consisting of OAM&P - Operation, Administration, Maintenance and Provisioning of network and services. Network providers want to provide their services at a satisfactory level most commonly expressed through Quality of Service(QoS), while keeping the cost required to achieve this at the optimum level. This comes with many challenges that can be grouped under the umbrella of FCAPS - Fault management, Configuration management, Accounting, Performance and Security. Configuration management includes setting and tuning various network devices. [3] The work presented here is related to routing configuration management in traditional IP networks. Devices of interests are routers or Level 3 switches.

Main tasks in routing configuration are configuration of IP addresses, choice and implementation of routing protocols, and security related challenges. We focus on routing configuration of the Interior Gateway Routing (IGR) protocols such as Open Shortest Path First (OSPF) and Intermediate System-Intermediate System (IS-IS). These are link-state based routing protocols designed to be used within one autonomous system (AS). Most large IP-networks are functioning with these two routing protocols. TCP/IP protocol stack is to a high extent succeeding in making the network robust and reliable. However it does not necessarily provide efficiency. This means that selected route for one IP packet might not be the optimum one, e.g. the one with the lowest latency. This is related to traffic engineering. [4]

Nowadays there is strong tension that Internet Service Provides (ISPs) have to deal with, caused by constant growth of number of users as well as intensive usage of services such as VoIP or Video on demand. In order to deal with high competition while keeping the QoS level conformed with offered Service Level Agreements (SLAs) ISPs have to introduce efficient traffic engineering methods instead of using only the "best-effort" approach. Traffic engineering aims for better utilization of network resources. Work presented here focuses on one of the techniques for traffic engineering which includes adaptation of routing configuration. In link state based routing

protocols path selection is governed by link weights/"cost". Link weight is a metric used to indicate to what extent one route should be preferred over its alternatives. For example, routes with higher capacity will normally have lower weights and are as such more often selected as best path.

Despite various traffic engineering techniques available, doing this in real time still remains a big challenge. Networks are very dynamic and complex, and in order for provided services to fulfill SLA requirements network should immediately react to the changes in topology. Very often, this requires manual changes in configuration files.

Let´s consider one specific network autonomous system, presented in Figure 1.1. Network topology shown is very similar to UNINETT core network with few additional nodes and links. Purpose of added links is to have more available node-to-node connections and alternative paths for router to choose among. IS-IS routing protocol is used. Topology consists of 7 routers and links between them. Node oslo_qw1 is connected to the Internet. Only three links have 100 Gb/s capacity (marked in the figure), the rest are 10 Gb/s. Weight on the three links with higher capacity are smaller than for 10 Gb/s links. Thus, if customers connected to node trd_gw1 want to send a lot of traffic to the Internet, they will by default use the 100 Gb/s direct link to oslo_gw1.

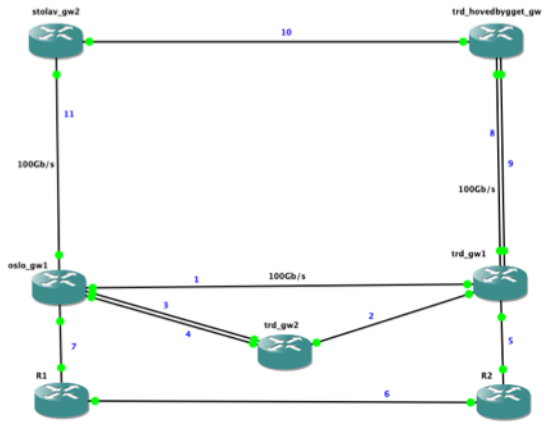

Figure 1.1: Network topology for case scenario of interest

What happens when link number 1 gets broken or is flapping i.e. has significant percentage of packet loss due to cable instability or damage? The traffic is re-routed to alternative path. Based on the routing table, the best alternative is the one with shortest path, via trd_gw2. This path consists of two links: link number 2 and link

number 3 (see Figure 1.1 for link numbers). Routing protocols immediately react to changes in topology and network continuous to function as desired. However, if utilization of the link 1 at the point of breakage was higher than 10%, a link overload at link 2 will occur. This happens because alternative link has 10 times less capacity than the main one. Packets are being buffered on the interface, causing delay or significant package loss when the buffer gets full. In order to prevent this, the engineers managing the network have to do traffic engineering by manually altering the values of link weights. Weights are manipulated in such a way that load distribution is achieved, resulting in prevention or decrease in number of packets lost.

Manual intervention requires the engineers to have detailed overview and knowledge about the network and to work under time pressure. Network provider needs to ensure human resources as well as coverage for cost of possible SLA requirements deviations. It would be much better if providers could replace these manual reconfigurations with a framework that is capable of achieving nearly the same result but autonomously and efficiently. The ultimate goal is to have an adaptable service system which by definition in [5] is *a service system capable of dynamically adaptation to changes related to users, nodes, status of capability and service performance measures, changed service requirements and policies.* This is where we center our research around.

In order to replace manual link weights configuration on which forwarding decisions are based, the new framework needs to have knowledge about network topology obtained through routing table. It can thus be ran as a software architecture on top of the existing router operating system. Furthermore, while doing reasoning it has to be capable of choosing the correct optimal alternative path. Work presented in this thesis is about an Adaptive Network Routing Configuration framework which is supposed to encapsulate and provide all this functionalities. The framework is governed by a Reasoning Machine based on XET (XML Equivalent Transformation) policy - driven reasoning engine, with the goal of reducing human interaction and provider´s costs related to amount of packets lost due to previously described link overload scenarios.

Several research questions are being addressed. Is it possible to implement such a framework and how would it look like? What data is required as an input to the framework so that it can fulfill the automated routing configuration tasks? How the framework would preform, more precisely how the traffic intensity and size of the router´s buffer impact the amount of packets lost? Lastly, we tackle on how does this framework´s performance compare to the network scenario without application of the reasoning machine.

The rest of the sections are organized as follows; Section 2 is about background and related work where the reasoning machine was developed and applied. Section 3 describes how the reasoning machine is functioning. Section 4 describes how the reasoning machine can be used as a part of Adaptive Network Routing Configuration framework. Section 5 is about how it was implemented and tested in simulation, Section 6 presents results of the simulation. Finally, Section 7 centers about results, conclusion, and future research work.

Chapter 2
Background and Related Work

Routing protocols of interest are link - state based IGRPs such as IS-IS and OSPF, and these are not fully dynamic. This means that they are not capable of real-time adaptation to all kinds of different events since the link weights which serve as basics for routing table formation are static. Link cost is specified and kept static for a long period of time meaning that traffic is always forwarded via one, same path. Furthermore these types of protocol do not have any knowledge about performance objectives. Also, routers only have overview of their neighbors´ link states, and not having a centralized approach makes it challenging to do the optimal end-to-end traffic engineering.

One would argue that one of the obvious way to do traffic engineering is to replace the traditional network setup with Software Defined Networks (SDN). In SDN control plane and data plane is decoupled, and control plane is where all the network intelligence is centered. Control plane is responsible for making forwarding decisions, while data plane only executes the commands. Centralized SDN network controller enables efficient network management and provisioning including traffic engineering solutions. Some of the possible techniques and their advantages have been discussed in [6]. However, majority of networks nowadays are traditional IP based networks and switching to SDN would require a change in complete architecture which even if possible and applicable to the specific scenario, would this bring a lot of cost. Focus is therefore kept on TE in traditional IP networks, more specifically on state dependant methods where traffic is manipulated depending on different metrics reflecting the network state at the moment of calculation. We aim for reducing the packet loss in peak hours by routing this traffic as per metrics stored in configuration of the applied routing mechanism. [7]

One of the basic techniques for traffic engineering in OSPF/IS-IS based networks is Equal - Cost MultiPath Routing (ECP), where router has several next-hop stations for given destination and must then choose one of them based on a certain method.[8] Most nowadays methods applying hashing approach to select the best route among

all the available ones. Hashing is a stateless approach to traffic engineering which distributes the traffic with the usage of hash function. A hash function is commonly enforced on a subset of five elements - source and destination addresses, ports and protocol ID. Computation is efficient and straightforward. It enables flow based distribution of traffic load but this distribution is always done evenly. Furthermore, hashing based approach does not manage and update the state of the network which consequently makes dynamic traffic engineering impossible. [9] If 16-bit CRC (Cyclic Redundant Checksum) is used for hashing calculations network becomes less imbalanced compared to classical hashing, but this comes with a cost of high computational complexity. In addition to complexity, problems with packets from one flow arriving with different delay times and being re-ordered is a non-trivial challenging issue for ISPs.[7]

Apart from hashing, several different techniques for traffic engineering based on packet loss and delay have been developed. In [10] a technique based on Multiprotocol Label Switching (MPLS) called Periodic Multi-Step (PEMS) was introduced. MPLS solutions are efficient because forwarding decisions do not require unpacking the packet up to network headers layer. Only the MPLS label is checked which results in scalable mechanism. PEMS has three phases, path selection, allocation, and dynamic adaptation of the metrics based on the network state. In the first phase all available paths are fetched for each source-and-destination pairs. Upon traffic arrival, Prediction of Effective Reparation (PER) is used to select one Label Switched Path (LSP) which will be allocated for the current packet. PER is an advanced type of Load Distribution in MPLS (LDM) network which takes into account path capacity of the available path and distributes the traffic optimally using the formula that includes number of hops, bandwidth and number of paths currently allocated to the existing traffic in the network. Third phase of PEMS is dynamic adaptation of the distribution coefficient according to the change in state of important elements in network topology. PEMS enables differentiated services (DiffServ) and was proved to be scalable in terms of number of nodes in the network but it requires MPLS as an additional construct introduced to the basic IP network.

According to the survey paper done by Singh et al. [7], most popular techniques for IP traffic engineering either include MPLS or hashing. However these techniques have some drawbacks mentioned above (complexity, per packet calculations, no dynamism, re-ordered packet arrival, even distribution of traffic only etc.). There exists another approach introduced by Fortz et al. in [4] where OSPF based network is monitoring traffic pattern and then changing the weights accordingly. A "centralized" approach was introduced. Topology and packets circulating within the network are being monitored, and the link weights are adjusted accordingly to achieve the specific goal for provided services. They have built so called "Traffic Engineering Framework", which is applied on the top of operational network. In order to change

link weights and make routing decisions it requires Network Operation Center (NOC) for acquiring the information about network topology and traffic statistics. The framework proposes usage of an intermediate construct (for example Simple Network Management Protocol - SNMP) for polling or getting the data via traps. These traffic demands are then taken as an input to routing model, which will compute possible paths and set the link weights accordingly. Decisions can be made by several different approaches: link weight proportional to link capacity, physical distance, being random, optimized to traffic conditions etc. The authors concluded that is is not possible to find an algorithm that is guaranteed to be both fast and produce close to optimal weight settings. With this heuristic approach defining and meeting SLA requirements comes as a big challenge for ISP.

Our goal is to develop a new framework which would eliminate the mentioned drawbacks of existing traffic engineering techniques. In order to create adaptable network for case scenario and similar challenges described in Section 1 the framework should be operating dynamically, per flow, with uneven traffic distribution based on the routing metrics of link. Our framework is similar to Fortz et al. approach explained previously, but without the need for NOC or SNMP, thus based exclusively on data from routing tables. One way to achieve that is by using the data and applying a Reasoning Machine (RM) which enables policy based specification and operation of traffic engineering. The reasoning machine used is based on XML Equivalent Transformation - declarative programming language for manipulation of XDD (XML Declarative Description) expressions as introduced in [2]. While details of the Reasoning Machine functioning and adaptation will follow in next section, this approach found its application many different scenarios. It was used in three doctoral thesis for different purposes: For example, in [11] and [12] it was a part of capability-related adaptation framework for TAPAS - Telematics Architecture for Play-based Adaptable Service Systems. TAPAS is a service execution platform and prototype for adaptable service system supporting capability-, functionality-, and concept-related adaptation of services. In [13] it was used for home energy management system for electricity cost savings and comfort preservation and creation of SMASH - SiMulated Adaptable Smart Home (an environment for case-based simulations from users perspective in power systems).

Chapter 3
Reasoning Machine

Main artifact of this work is policy-based framework which enables adaptable routing configuration of OSPF/IS-IS based routing in traditional IP networks. This architecture consists of the network as such, policy-based adaptation mechanism and data model. Shortly, it is about applying already existing concepts of reasoning machine to specific adaptive network routing configuration scenario.

3.1 Policy-based adaptation mechanism

Adaptation mechanism embodies a Reasoning Machine (RM) responsible for making automated forwarding decisions. Decision making process is based on the data from routing table and traffic statistics serving as an input. End result is RM being capable of dynamical adaptation of routing configuration and changing the original routing protocol´s best path choice. The RM to be used was in its original format first introduced in [1], and we will apply the same logic and data model - XML Equivalent Transformation (XET) rule based language that stands on XML Declarative Descripiton (XDD) expressions.

As defined in [1], RM R can be described as follows:

$$R \equiv \{Q, F, P, T, E, \Sigma\} \tag{3.1}$$

$$P \equiv \{X, A\} \tag{3.2}$$

where :

- Q is the set of query&reply messages exchanged between the managed system and the reasoning engine,
- F is a generic reasoning procedure,

- P is a policy system which consists of set of rules X and set of actions A ,
- T is a set of system constraints,
- E is a set of status data and
- Σ is a set of reasoning conditions.

Status data represent current values of the variables of interest while system constraints are restraints and relationships between variables. Policy system is based on system constraints. Σ is a set of reasoning conditions that consists of trigger condition and goal condition. RM begins execution when trigger condition is matched end end when goal condition is achieved. These connections and relations between these elements is shown in Figure 3.1. Service System Adaption Manager takes *Rules*, *Actions*, *System constraints* and *Status data* as inputs required to make decision about the action to be applied on the associated Managed Service System.

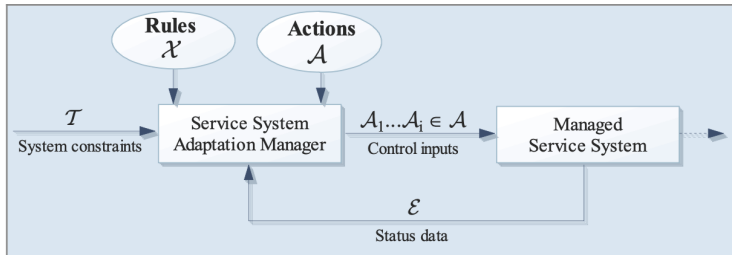

Figure 3.1: Policy-based adaptation architecture taken from [1]

Now, all concepts described have to be adjusted and mapped to routing configuration case. It is easier to imagine and explain them if we think about one specific scenario, for example the one explained in Figure 1.1. If we are to prevent link overload on the link which has 10 times less link capacity than the original broken link, RM needs to achieve that would function as follows:

- Q can be thought of a set of query&reply messages, where a query is an XDD formatted file (details will follow) consisting of flow related information such as source and destination address and an unresolved variable representing best path for that variable. Response message should then return the same document with instantiated values for the best path, consisting of a forwarding decision about to which interface to forward the current flow. It is per flow processing for two reasons: 1. less delay and computations then in per-packet approach and 2. prevents packets travelling different routes and arriving re-ordered to the destination,

- F in our case is a process of determining the best route based on the data from routing table,
- P as our policy system is constituted by rules and actions are also written in XDD. Every rule comprises of rule "HEAD" and "BODY" atoms which will be further explained later on in this chapter,
- T is a set of system constraints such as maximum size of the interface output buffer on each interface in the router,
- E as set of status data in our case is for example a routing table data and current interface output buffer size,
- Σ when it comes to conditions, trigger condition can be arrival of the flow´s first packet, while goal condition and end of the execution is reached when the best route for the flow currently being processed is determined.

We will start with explaining the Reasoning Procedure F which is similar to *logic programming*. Reasoning Procedure requires at least two input files, one is file with rules and another one is file with the query clause. Both rules and queries have "HEAD" and "BODY", and body of the query is matched against the head of each rule. If these are matched, rule execution continues. The principle applied here is XET, where problem is transformed through repetitive applications of equivalent transformation rules.

Policy rules and specification is enforced through Equivalent Transformation rules and clauses. Problem must be formulated as a query clause that allows for further transformation. An ET clause has the form:

$$\underbrace{\text{Head atom}}_{\textbf{Head}} \leftarrow \underbrace{\text{Body atom}_1, \ldots \text{Body atom}_n}_{\textbf{Body}}$$

Head or head atom has a message that incorporates a problem which requires an answer or action. This problem will be equivalently transformed by matched rules´ body atom(s) until it eventually contains an answer to the initial problem. Policies for routing decision making procedure are defined as XET rules. They are therefore easy to manipulate and change if needed.

3.2 Data model

As mentioned previously, data model used by the RM is XET rule based language. XML Equivalent Transformation is a tool which one can use to model and run application rules and logic, data, queries and requests. [2] These components are represented in XML Declarative Description (XDD).

XDD is XML based data modelling language that allows for precise and formal representation of all the required concepts. It is a language with two constituents: words and sentences. XML elements are words and can be explicit/implicit, simple or complex, while XML clauses are sentences used to depict conditional relationships or constraints between them. The biggest advantage of XDD compared to basic XML or RDF (Resource Description Format) is in increased expressive power of relationship between XML expressions which include *symmetry, composition-of* and *inverse*. [14] This section handles XML expressions´structure defined abstractly with *XML Specialization System* and syntax of XDD clauses.

3.2.1 XML Specialization System

The difference between XML elements and expressions is that expression can carry variables representing implicit information and thus have enriched expressive power. XML expression can be *ground* - basic XML element without variables or *non-ground* meaning that they carry variables. Similar to XML elements, expressions also have tag names, attributes and their associated values, and main contents which can carry variables. All well defined types of variables and their representations are shown in Table 3.1. We have *N-, S-, P-, E-, I-* and *Z- variables* representing basic XML element name, String variable, zero or more attribute-value pairs, variable consisting of zero or more XML expression, parts of XML expressions and sets of XML expressions respectively. Each variable type begins with "$" sign and has a generic form of *(v,w)* where v is the name of the variable and w is its value. In this work we will focus only on two variables types from the table: *S-variables* and *E-variables*. The first is a simple implicit representation of a variable which can be replaced by a String, for example *S-variable* **$Svar_name** in the following XML expression:

```
<name>$Svar_name</name>
```

can be replaced with

```
<name>Ola</name>
```

since "Ola" is a String. *E-variables* are constructed type of variables consisting of zero, one or more XML expressions. For example *S-variable* **$Svar_person** in the following XML expression:

```
<person>$Evar_person</person>
```

can be specialized with for example two ground XML expressions:

```
<person>
    <name>Ola</name>
    <surname>Nordmann</surname>
</person>
```

These two variable types will play the key role for definition of rules and queries in the reasoning machine for routing configuration adaptation.

Table 3.1: XML expression variable types [2]

Variable Type	Variable Names Beginning with	Instantiation to
N-variables: Name-variables	$N	Element types or attribute names
S-variables: String-variables	$S	Strings
P-variables: Attribute-value-pair-variables	$P	Sequences of zero or more attribute-value pairs
E-variables: XML-expression-variables	$E	Sequences of zero or more XML expressions
I-variables: Intermediate-expression-variables	$I	Parts of XML expressions
Z-variables: Set-variables	$Z	Sets of XML expressions

3.2.2 XML/XET caluses

XML clauses are ground for ET clause and have identical formulation. As described before, clauses represent implicit and conditional relationship where each clause is of the form

$$H \leftarrow B_1, B_2, ..., B_n \tag{3.3}$$

where H is the head XML expression, and $B_1, B_2, ..., B_n$ are body atoms. When $n = 0$ we have *unit-clause* written simply as H. If we re-write the clause from 3.3 with XDD syntax, we get an XET/XDD clause shown in 3.1. XML representation of clause is equivalently transformed until all the body atoms are resolved.

Listing 3.1: XET/XDD clause

```
<xet:Clause>
    <xet:Head>...</xet:Head>
    <xet:Body>
        Body atom1, Body atom 2, ...
    </xet:Body>
</xet:Clause>
```

How do XML clauses work with ET rule and reasoning procedure? As seen from equation 3.1 we need a set of messages between the system and a reasoning machine which is modeled as an XML clause looks :

$$msg(...) \leftarrow msg(...) \tag{3.4}$$

Relating it to XET rule and XDD clause, this is is an axiom interpreted as: head of the message is true if body of the message is true. Reasoning procedure will be transforming the head of the message clause until there are no body atoms left.[1]

If we make now another example, where a set of conditions C that have to be fulfilled in order for the statement to be true, it looks as follow:

$$msg(...), C \leftarrow B_1, B_2, ..., B_n \tag{3.5}$$

then the rule will transform original *msg(...)* into n body atoms but only in the case when conditions are fullfiled. The action is recursive; once a body atom is matched with head of the rule it is further transformed into the transformation rule's body atoms. This means that each of the body atoms becomes a head of new sub-clause and it continuous with transformation through another existing rule with whose head it matches. Transformation may include a set of different actions and the process ends when there are no more body atoms or where there is no rule to be matched with the remaining body atom(s).

Chapter 4
Adaptive Network Routing Configuration

This section is about designing query and rule file for network routing configuration scenario. Reasoning Engine that implements described reasoning procedure and matches the query against the rules is contained in Java code called NxET reasoning engine. Full implementation details are described explained in technical report [15]. It takes query and rule files as an input and does complete XET.

Policy based framework is to be enforced within routers in network of interest. We assume that our router has the knowledge of network topology, data from routing table and OSPF Link State Advertisement (LSA) (or similarly Link State PDUs in IS-IS). All these data has specific format which will be shown and discussed in Section 4.3. Even though in practice routers in traditional IP network can only see its neighbors link, only one XML file is required to represent the whole network topology and provide topology overview for the reasoning machine. Trigger condition for activation of reasoning procedure is flow packets arrival. Packets are part of the flow, and forwarding decision is done per flow and not per packet for two reasons, first one is that TCP requires packet to arrive to the destination in right order and the second is time/delay it would take to do the reasoning for every packet. Goal condition is known route for the arrived packet. Messages exchanged with the RM are query&reply, and reply is based on the XET processing of the query clause and rules. Head of the each defined rule is matched against body of the query atoms.

4.1 Query

Query clause is specific type of a XET clause, because of two reasons. Its head specifies how the reply should look like using XML expressions carrying variables, and it has only one body atom to be matched against the set of rules.

Upon arrival of the first packet in the flow trigger condition is met and new query clause is created. Router should be able to fetch information needed to forward the

table such as source and destination addresses. These two pieces of data are used and sent in a query which has the following format:

Listing 4.1: Query message sent from managed system-network router to the RM

```
<xet:XET xmlns:xet="http://xet.sf.net" xmlns:rdf="RDF">
    <xet:Query>
        <xet:Clause>
            <xet:Head>
                <Request>
                  <from>Svar_from</from>
                  <to>Svar_to</to>
                  <utilization maxvalue=Svar_utilization/>
                  <route sum="Svar_sum" nodecrossed="Svar_crossed">
                        Evar_result
                  </route>
                  <best>
                        Evar_best
                  </best>
                </Request>
            </xet:Head>
            <xet:Body>
                <Request>
                    <from>trd_gw1</from>
                    <to>oslo_gw1</to>
                    <utilization maxvalue="80" />
                    <route sum="Svar_sum" nodecrossed="trd_gw1">
                        Evar_result
                    </route>
                    <best>
                        Evar_best
                    </best>
                 </Request>
            </xet:Body>
        </xet:Clause>
    </xet:Query>
 </xet:XET>
```

From Listing 4.1 we see that query clause consists of one head and one body atom. Namespace for these is xet indicating that this is XET clause. Body of the query has the same skeleton as the head, but instantiated with some concrete values for one specific packet flow. Head atom has only one XML expression named <Request> which is structured and non-ground since it carries variable. Expressions <from> and <to> carry one *S-variable* each, representing the name of their originating and destination nodes. Expression named <utilization> is optional, but can be used to select a maximum value of the link utilization allowed - Svar_utilization. For

example, if one link has its utilization value higher than defined acceptable value (e.g. 90%) utilization then we do not want to send packets on that link.

Expression <route> is an auxiliary expression which containing an *E-variable* Evar_result which will during the execution of XET be instantiated in such a way that it will represent all the possible routes between from and to nodes. It has two attributes, sum and nodecrossed. Sum will be instantiated to the value equal to the lowest cost among all available routes and nodecrossed is there just to help XET know which next node has already been visited when determine the possible paths. Variable of interest and is Evar_best, contained in expression <best>. This one will contain the best route chosen among all the possible routes contained in Evar_result.

Body of the query instantiates all these variables in such way that it represents a packet coming from node called "trd_gw1" and aiming to reach node "oslo_gw1", using only link whose ultization is less then specified acceptable maximum. The first nodecrossed is the origin. After being transformed and matched against the XET rules, variable Evar_best will contain the answer about the best route, provided by the reasoning procedure. This will be contained in a reply message sent from RM to the node forwarding the arrived packet.

4.2 Rules

In order for XET transformations to lead us to the best alternative route, set of rules is needed. NxET reasoning engine supports two types of rules, *N*- and *D*- rules. N-rule is a declarative description of a problem where the sequence of Equivalent Transformation executions does not matter - the final result is the same no matter the order of resolving the body atoms in each rule. This gives a lot of flexibility, and is close to human language specification of the static policies to be applied.

Our set of rules consists of three rules, these are "AllOklinks","Set" and "RuleMInimum" as shown in listing A. All rules are of a type N-rule, meaning that order of the Body atoms specification does not affect the final result, but we will discuss them in the order in which they are written for the sake of easier explanation. First two rules together are responsible for creating a set consisting of all possible paths between <to> and <from> nodes of currently processed packet. Third rule is responsible for choosing one best path from the result of the execution of XET of the first two rules.

How is then query clause matched against this set of rules? As explained in previous chapter, body of our query from listing 4.1 is matched against the set of rules, more precisely heads of the rules. Head of the rule "Set" is successfully matched with the body of our query, and that is the only rule in our set of rules that has the initial match. Execution starts from there, but we will first explain the rule

"AllOkLinks" because functioning of that rule is required for understanding the "Set" rule.

Rule "AllOklinks" is a rule which returns all possible routes between <from> and <to> nodes that fullfil the conditions specified. Condition for a route to be returned is that state of the link is "ok" and the utilization of that link to be less than a maximum specified value. Querying the router is done through NxET built-in function called xfn:FactQuery. FactQuery queries the router´s data base to get the links that are up and not overloaded above the percentage specified as attribute "maxvalue" inside <utilization>. Router´s data base contains two files as data sources - ds://ALL-ROUTES and ds://UTILIZATION all XML files. The first file has a list of all links in the network topology, and related data (such as IP addresses, interface names etc.) while the second file keeps data related to utilization of this links. Appendix 4.1 contains example files for scenario presented in Figure 1.1.

There are two basic cases that has to be distinguished when looking for the possible routes. The first one is when we have a direct link between the two nodes, and the second one is when routes have 1 or more intermediates. That is why this rule has two body atoms. For direct link between the source and destination nodes, application of this rule sounds straight forward. In data base file definition of <LINK> you are searching for the links whose attributes node1/2 have the same value as Svar_from/Svar_to. Once you find it you save data related to that direct link as one answer to the query.

If the link is indirect, every neighbor of the originating node becomes new <to> element and again its neighbors are considered and checked if one of them is the original Svar_from, from the query clause. If yes the route is added to the answer and this node is added to the String list of node crossed in order to avoid looping. All intermediate links of one path are join in one XML expression which becomes one answer to the query. This is done with the built-in function xfn:MatchD which is used for instantiation of *E-variables*. Namespace :xfn is connected to all the core functions of the reasoning engine. For example, a path from node node trd_gw1 to oslo_gw1 via nodes R1 and R2 will start from the link trd_gw1-R2. Then E_var routeok will through xfn:MatchD become representation of that link, and then when links R2-R1 and R1-oslo_gw1 are traversed they will be added as ground expressions to the*E-variable*. That is how the first rule ET is executed.

As said before, rule number 2 is the one whose head matches the body of the query. By rules of ET, next step after the match is resolving body atoms of the rule "Set". There is only one body atom from this rule, but it has two parts. The first body part starts with XML expression named xfn:SetOf which is NxET reasoning machine built in function. XFN function SetOf merges all answers from rule "AllOkLinks"

into one answer. It has a Condition for becoming the member of the set, which then becomes a new body atom. As seen from the rule file, condition for this set is basically equal to the head of the rule "AllOkLinks" . The first rule is then executed returning resolved body atoms as separate answers. Every new answer becomes a new clause and in case there are further rules that are matching we get a lot of branching. SetOf will join all answers in one, making only one potential close for further transformation. Not only that, having the routs and their cost in one answer make it possible and easier to compare the routes.

After xfn:SetOf of the "Set" rule, the body atom is not yet fully resolved, it has an XML expression called "FindMinimum". As the name says, the aim of this part id to find the path with the minimum cost/weight given the network conditions form the input files. This part of the body atom will tried to be matched against our rules set. The rule that matches it is "RuleMinimum". This rule will instantiate the *E-variable* Evar_best with the path which has the smallest cost value from all the available paths contained in the Set obtained after execution of the "Set" rule.

Since it is declarative description of code execution, the rule to choose path with minimum cost is quiet complex and long. In object oriented programming it could take just one function or one *for loop*. Here however we have to distinguish three cases, having zero available paths between <to> and <from> node, or having one path, two path or having more paths. To support these three alternatives the rule has three body atom. The first one is when there is no available path between the nodes. In that case Evar_best remains unresolved while the value for the cost is assigned high value, for example 10000 because that one is hardly reached since metrics values are usually around 10. If there is only one path, then that one automatically becomes Evar_best and minimum cost is equal to the cost of that path. If there are two, then these two are compared and the Evar_best becomes more than one, then they are compared and the one with lower cost becomes the insantiation of the best path. If there are three or more, then recursion is needed. Last two paths are taken and processed by body for only two paths available, result is compared with the rest of the paths and so until until all body atoms have been traversed and resolved.

Final result to the query is sent back to the router, in the same format as head of the body, but with an instantiated values for <best> XML expression in the query. Example of the answer clause is shown in Appendix C.

4.3 Data base files

It is required from the node/router executing the reasoning procedure to have two files - ALL LINKS and UTILIZATION as data input to the engine. Example files for our case scenario network are shown in Appendix B. Both files are of course XML

files, the first file contains list of all links available in the network. Every link is in the file represented as shown in Listing 4.2. Data required is very similar to routing table data of link-state based protocols such as OSPF/IS-IS. We have name of the two nodes, IP addresses, and interface names(e.g. interface GigabitEthernet 1/0 is shortly g1/0), as well as metric from routing configuration. Data for our specific network case scenario was made up and the values are chosen arbitrarily.

Listing 4.2: Required format of the LINK entry in ALL-LINKS file in router´s data base

```
<LINK node1="Svar_value1" int1="Svar_value2" ip1="Svar_value3"
      node2="Svar_value4" int2="Svar_value5" ip2="Svar_value6"
      metric="Svar_value7" />
```

The second file, UTILIZATION just adds utilization value of each link so the entry there becomes as shown in Listing 4.3. The IP value of one of the link ends is the cross-reference between the two data base files, so for simplicity of rule files two entries are required for each link. State of the link can have two values, "ok" or "down", and rules will not take into account links that are down basing the choice on the value of this state attribute. Utilization represents current value of the link utilization which if higher than maximum specified utilization value will also lead to link been discarded from list of all possible links.

Listing 4.3: Required format of the LINK entry in UTILIZATION file in router´s data base

```
<LINK ip1="Svar_value3" state="ok/down" utilization="Svar_utilization"/>
<LINK ip1="Svar_value6" state="ok/down" utilization="Svar_utilization"/>
```

Chapter 5
Simulation

Application scenario considered was presented in Figure 1.1. The services of interest are Voice Over IP (VoIP) services – e.g. Viber, Skype, Whats-up voice calls only. We focus on one part of the traffic in this network, and that is a traffic between all the users connected to node "trd_gw1" making their calls over the Internet via node "oslo_gw1". This example case scenario can then be generalized to other traffic cases. So, traffic data will be originating from main Trondheim node, targeting Oslo as destination.

In order to simulate the example case scenario, Java application has been written and ran on top of the NxET reasoning engine. All data explained previously is required for proper functioning of the simulated framework. We opted for time-driven simulation. It has a disadvantage of using real time, but in our case it was a good choice for several reasons. Firstly, our simulation environment has to be written in Java for the sake of enabling interoperability with NxET reasoning engine which is also described in Java. The best approach would be to have event-based simulator in Java. The initial intention was to use Java for DEMOS library which unfortunately is not available anymore since the project was shut down. Essential construct for the event-based simulation is event list handler. After searching for the one that could fit our research questions well, we figured out that no such exists and new Java based event list handler would have to be programmed. That as such requires high effort which would exceed the time limit. Furthermore, our scenario of interest when 10Gb/s link gets overloaded only happens for a in a short interval of peak-hour traffic. That is why time-based simulation was selected.

There are three basic constructs of our simulation: application for packet generation, application for packet processing (deciding on which interface to send the packet), and packet sending (sending out the packet on the interface). These three constructs are implemented as synchronized thread objects, started at the same point in time.

Packet generator generates the flow packets following Poisson distribution, meaning with negatively exponentially distributed (n.e.d) interarrival times. We will vary the intensity in order to see how it impacts the percentage of packets loss. In addition, we will vary the the size of the interface output buffer with the same purpose. Source and destination for flow packets will be "trd_gw1" and "oslo_gw1" respectively. Flow size is assumed to be 1400 kB, meaning that each flow has 7 packets of 200 kB which according to [16] is the average size of one VoIP packet. Packet generator will therefore create flows of packets following n.e.d with varied intensity and send them to router´s input buffer queue from where packet processor takes over.

Packet processor is using Java class BlockingQueue for router´s input and interfaces´ output queues. BlockingQueue is used because its implementation is done in such a way that it does not require any thread synchronization. Once packets are in the input buffer of the router "trd_gw1", the NxET reasoning engine will be activated and fetch the interface name on which it is supposed to forward the packet. For every flow, a query message will be sent to the reasoning machine, and the interface would be read from response after all the rules body atoms are resolved. Targeted interface will be contained in the first element of the E-variable $Evar_best. Once the interface is identified, packet will be put into output buffer queue of the interface and will wait to be sent on the link. In order to quickly reach rare event condition - full output buffer, we chose low values for output buffer´s capacity. When the interface is chosen, the application will check the current size of its output buffer. Trigger condition for activation of the reasoning machine happens when that value is higher than specified maximum allowed value for buffer usage. For example, if we chose 90% once the buffer capacity is 90% occupied, the application will consider the link connected to that interface as overloaded. Reasoning procedure will be invoked again and again until it finds the link with lower utilization.

Flow packets residing in the interfaces´ output buffers are further processed by the packet sender which will put the packets on the link and forward them to the next node. This happens with a transmission delay equal to the quotient between flow size and link capacity. Since our links are 100 and 10 Gbps, again for simulation purposes we chose links with lower capacities in order to reach full output buffer. After delay time expires packet is removed from the queue.

In the case where there is no interface available with buffer occupation being less than 90%, the last chosen interface will be selected again. Once its buffer gets full, packets will be dropped. Main goal of the simulation is to see the impact of intensity and buffer size variations on percentage of packets dropped, as well as to try to compare it to network without the reasoning machine. Results will follow in the next section.

Chapter 6
Results & Discussion

First part of simulation had the goal to check the impact of the intensity of flow arrivals on the percentage of packets lost. For the sake of getting many runs of our time based simulation, simulation was run for 100s, in 20 repetitions. Intensity was varied from having 1 packet every 20ms, to 1 packet flow every 100ms having 8 steps with 10ms increment in between. Buffer size was limited to 20 flows (224 000) bits, and was kept fixed trough out this part of the simulation. Average values of percentage of flow packets lost are shown in Table 6.1 and Figure 6.1. Maximum deviation for the percentage loss was less then 5% of the value of percentage of packets dropped.

Table 6.1: Percentage of flow packets dropped versus traffic interarrival time

Interarrival time [ms]	**Percentage of flow packets dropped**
20	3.09 %
30	2.79 %
30	2.72 %
40	2.49 %
50	2.33 %
60	2.18 %
70	2.08 %
80	1.53 %
90	1.39 %
100	1.06 %

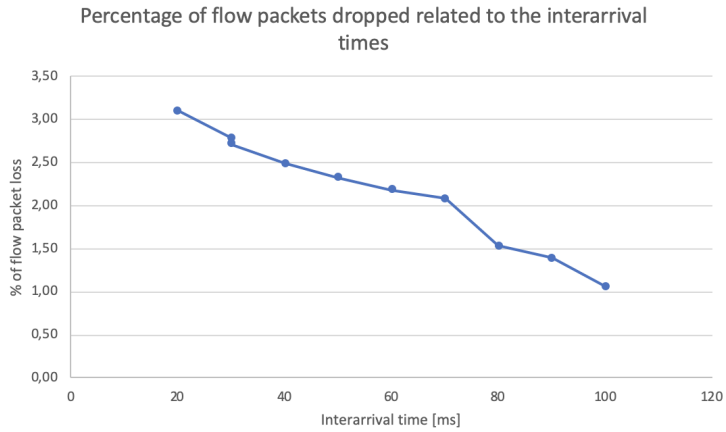

Figure 6.1: Percentage of flow packets dropped related to traffic interarrival time

From both the table and the graph we can see that the percentage of packet loss is decreasing with the increase in interarrival time. The whole system can be described as two M/M/1 queues, where the first server is Packet Processor and the second server is Packet Sender. Thus, having less requests arriving to the queue and being processed would result in less flow drop. During the simulation, we could see that the reasoning machine correctly made all the decisions about what is the best path. The path with the minimum sum of link weights was always chosen correctly, and in case there were several the first one on the list was selected. This showed that it is practically possible to implement the adaptive network routing configuration framework working in a described way.

The same scenario was also tested without reasoning machine, but since our chosen buffer size was chosen to be extremely small we ended with a huge amount of flow items being dropped (more than 1/3 of generated flows). It is thus hard to create a time based simulation which would fit both scenarios. This is because for the simulation of the reasoning engine application we have to use small, quickly full buffers of all links that can lead from trd_gw1 to oslo_gw1 while for the case without reasoning machine we need a buffer big enough not to drop a lot of flows in chosen alternative link.

We quickly figured out that buffer size plays an important role in the percentage of packets loss. Thus, in next rounds of simulation we also used 1.5 minutes long simulation where we kept the intensity value constant - 1 flow every 100ms. We varied buffer size from being able to contain 50 flows up until 250 flows, having 3 intermediate steps with increment of 50 flows.

Flow size was also kept constant at the value of 1400 kB. Average values of percentage of flow packets lost are shown in Table 6.2 and Figure 6.2. Maximum deviation for the percentage loss in 20 simulation repetitions was less then 10% of the value of percentage of packets dropped.

Table 6.2: Percentage of flow packets dropped versus router´s interface buffer size

Buffer size [number of flows]	Percentage of flow packets dropped
50	9.368770764 %
100	4.711425206 %
150	2.975557917 %
200	3.676078849 %
250	1.003344482 %
500	0 %

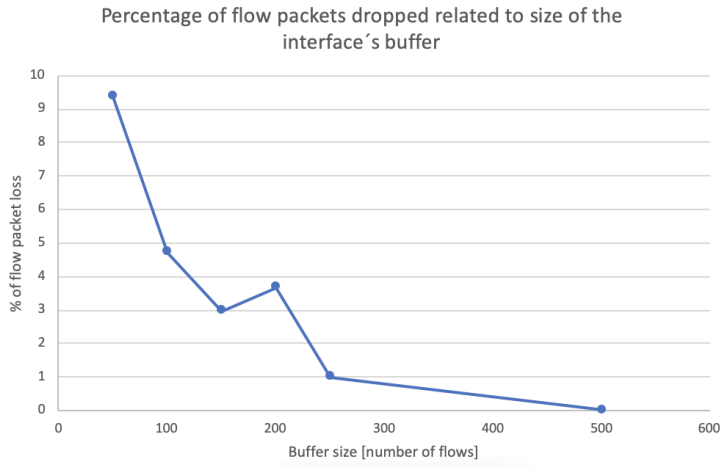

Figure 6.2: Percentage of flow packets dropped related to the router´s interface buffer size

As expected, increasing the interfaces´output buffers sizes resulted in decrease of flow packets drop. Our buffer size represents maximum size of the queue in M/M/1 process and increase here causes less requests to be dropped.There is however an exception in the decreasing trend, when the buffer size is 200 but if we take into account the 10% deviation it still can be considered to follow the trend.

After buffer size was increased to 250 flows, we tried also with 500 got zero flows dropped which makes sense because in this case approximately 1000 flows were generated and processed and this can be shared between two interfaces with output buffer capacity less than or equal to 500 flows. Furthermore, with a buffer size of

500 flows we simulated the network scenario without enforcement of the reasoning machine since this buffer size is close to the buffer size of CISCO router C7200. [17] After 20 repetitions of simulations we got average of 10.7% of flow packets dropped with maximum percentage deviation equal to 3% of the average value. This value is already way higher compared to 0 flow packets lost in the case of adaptive network routing configuration framework application.

Furthermore, we tried to see what size of a buffer is needed for raw network scenario (aka without adaptive routing configuration framework) to reach no flow packet dropped under the same conditions. Of course, higher the buffer size is less the percentage of lost flow packets were until after 700 it reached the value of 0 flow packets dropped. How quickly the average values were falling towards zero % is shown in Table 6.3 and Figure 6.3 , again with maximum percentage deviation equal to 3% .

Table 6.3: Percentage of flow packets dropped versus router´s interface buffer size without the application of the Reasoning Machine

Buffer size [number of flows]	% of packet loss
50	9,368770764 %
100	4,711425206 %
150	2,975557917 %
200	3,676078849 %
250	1,003344482 %
500	0 %

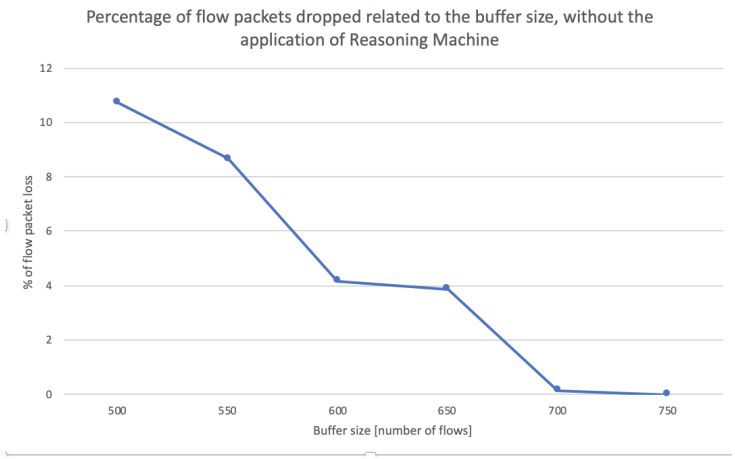

Figure 6.3: Percentage of flow packets dropped related to router´s interface buffer size without the application of the Reasoning Machine

Chapter 7
Conclusion and Future Work

7.1 Conclusion

From what we have seen, it is possible to create high level Adaptive Network Routing Configuration framework based on NxET Reasoning Machine engine which enforces the reasoning procedure based on XML Equivalent Transformations on XDD XML expressions and clauses. A node has to exchange messages with the reasoning engine, which are of query&reply format where query is asking for a best path between two nodes specified as source and destination. Reasoning engine needs a set of rules describing the policy of how this choice is to be made. Body of the query atom is recursively equivalently transformed through body of the rule(s) whose head it matched. Finally an answer is obtained and returned to the node as reply to the query clause.

Simulated in simple UNINET-like network topology we realized that percentage of flow packets dropped is decreasing with increase of interarrival time and increase in buffer size. When Adaptive Network Routing Configuration framework is applied, in order to reach 0 flow packets dropped interfaces are required to have output buffer size capable of hosting about 500 flows. Under the same conditions, without the framework more than 700 flows in the output buffers are needed.

7.2 Future work

This master thesis work was conducted as part of my integrated Ph.D studies. The purpose was to get to know the area of interest and get ideas for the direction of the Ph.D research work. Focus of this work was in understanding and developing the reasoning procedure capable of doing adaptive routing configuration. Simulation was done in order to show how this artifact can interact in the context of one specific network scenario.

At this point there are several ideas for improvements and future research work ideas for Ph.D research directions. Firstly, an event-based simulator or more precisely event-list handler in Java is to be implemented. This would take away a lot of drawbacks of the time-based simulation approach such as time consuming simulations leading to a decision to have very short and not fully representative intervals and results. Further on, with event-based handler it would be possible to directly observe and compare two approaches - with and without the reasoning machine. Another parameter of interest - delay in packets arrival is of a high importance for QoS and might as well be checked in simulation. In addition to that, packets coming and destined to other nodes should also be simultaneously included, as well as enforcement of the framework in each of the nodes in the described scenario. Another type of services could also be considered in addition to voice calls. For example Video on Demand (VoD)- this service has different traffic characteristics - bigger data and different interarrival time distribution from VoIP.

Next step is the actual implementation of the described framework in real hardware running on the top of currently implementing routers´logic. This furthermore comes with the challenge of having vendor dependent routing configuration files and operating system. One way to unify this is OpenConfig with common data models written in YANG. YANG is a network configuration data modelling language for management protocols. [18] Potential implementation of Adaptive Network Routing Configuration framework through an OpenConfig extension would enable dynamic and programmable architecture which would be vendor independent and shared between network operators. This is a direction to center my further Ph.D research around.

References

[1] P. Supadulchai and F. A. Aagesen, "Policy-based adaptable service systems architecture," in *21st International Conference on Advanced Information Networking and Applications (AINA '07)*, pp. 656–665, May 2007.

[2] C. Anutariya, V. Wuwongse, K. Akama, and V. Wattanapailin, "Semantic web modeling and programming with XDD," in *The Emerging Semantic Web, Selected papers from the first Semantic web working symposium, Stanford University, California, USA, July 30 - August 1, 2001*, 2001.

[3] M. Subramanian, *Network Management: Principles and Practice*. The address: Pearson Education India, 2 ed., 11 2012.

[4] B. Fortz, J. Rexford, and M. Thorup, "Traffic engineering with traditional ip routing protocols," *IEEE Communications Magazine*, vol. 40, pp. 118–124, Oct 2002.

[5] P. Supadulchai and F. A. Aagesen, "Policy-based adaptable service systems architecture," in *21st International Conference on Advanced Information Networking and Applications (AINA '07)*, pp. 656–665, May 2007.

[6] A. Guleria, "Traffic engineering in software defined networks: A survey," *Journal of Telecommunications and Information Technology*, vol. 4, pp. 3–14, 12 2016.

[7] R. Singh, N. Chaudhari, and K. Saxena, "Load balancing in ip/mpls networks: A survey," *Communications and Networks*, vol. 4, 01 2012.

[8] D. Thaler and C. Hopps, "Multipath Issues in Unicast and Multicast Next-Hop Selection," RFC 2991, November 2000.

[9] Zhiruo Cao, Zheng Wang, and E. Zegura, "Performance of hashing-based schemes for internet load balancing," March 2000.

[10] A. Toguyeni and O. Korbaa, "Diffserv aware mpls traffic engineering for isp networks: State of art and new trends," *Journal of Telecommunications and Information Technology*, 01 2009.

[11] P. Supdalcuai, *Reasoning based Capability Configuration Management in Adaptable Service Systems*. PhD thesis, NTNU, 2007.

[12] P. Thongrta, *A Service Framework for Capability - based Adaptation in Adaptable Service Systems*. PhD thesis, NTNU, 2012.

[13] K. Dittawit, *Smart Grid Demand Response with Mutual Utility-Consumer Benefits*. PhD thesis, NTNU, 2016.

[14] C. Anutariya, K. Akama, and E. Nantajeewarawat, "Xml declarative description: a language for the semantic web," *Intelligent Systems, IEEE*, vol. 16, pp. 54 – 65, 06 2001.

[15] P. Supadulchai, "Nxet reasoning engine, *Plug & Play Technical Report*," tech. rep., Department of Telematics,NTNU, 2008.

[16] L. H. Do and P. Branch, "Real time voip traffic classification," tech. rep., Centre for Advanced Internet Architectures, Technical Report 090914A, July 2009.

[17] "Buffer tuning for all cisco routers." https://www.cisco.com/c/en/us/support/docs/routers/10000-series-routers/15091-buffertuning.html. Accessed: 2019-10-20.

[18] M. Bjorklund, " The YANG 1.1 Data Modeling Language," RFC 7950, August 2016.

Appendix A
Rules

Listing A.1: Rules

```
<xet:XET xmlns:xet="http://xet.sf.net" xmlns:xfn="http://tapas.item.ntnu.no/
    NxET/built-in/corefunctions"
        xmlns:xfn2="http://tapas.item.ntnu.no/NxET/built-in/string">
    <xet:Meta>
        <xet:SpecificationVersion>0.1</xet:SpecificationVersion>
    </xet:Meta>
    <xet:Rule xet:name="AllOklinks" xet:priority="1">
        <xet:Meta>
            <xet:RuleDescription>Rule for returning all links between
                specified nodes with the state OK
            </xet:RuleDescription>
        </xet:Meta>
        <xet:Head>
            <Request>
                <from>Svar_from</from>
                <to>Svar_to</to>
                <utilization maxvalue="Svar_maxutilization"/>
                <path sum="Svar_sum" nodecrossed="Svar_crossed">
                    Evar_routeok
                </path>
            </Request>
        </xet:Head>
        <xet:Body>
            <xfn:MatchD xfn:mode="Set">
                <X>Evar_routeok</X>
                <X>
                    <nexthop interface="Svar_int1" nextNode="Svar_to"
                        utilization="Svar_utilization" metric="Svar_sum"/>
                </X>
            </xfn:MatchD>
```

```
        <xfn:FactQuery xfn:uri="ds://ALL-ROUTES" xfn:mode="Set">
            <LINK node1="Svar_from" int1="Svar_int1" ip1="Svar_ip" node2="
                Svar_to" int2="Svar_otherint"
                 ip2="Svar_otherip" metric="Svar_sum"/>
        </xfn:FactQuery>
        <xfn:FactQuery xfn:uri="ds://UTILIZATION" xfn:mode="Set">
            <LINK ip1="Svar_ip" state="ok" utilization="Svar_utilization"/
                >
        </xfn:FactQuery>
        <xfn:LessThanOrEqual xfn:number1="Svar_utilization" xfn:number2="
            Svar_maxutilization"/>
        <xfn:Not>
            <xfn:StringIsMember xfn:string="Svar_to" xfn:list="
                Svar_crossed"/>
        </xfn:Not>
    </xet:Body>
    <xet:Body>
        <xfn:MatchD xfn:mode="Set">
            <X>Evar_routeok</X>
            <X>
                <nexthop interface="Svar_int1" nextNode="Svar_any"
                    utilization="Svar_utilization"
                        metric="Svar_metric"/>
                Evar_routeok2
            </X>
        </xfn:MatchD>
        <xfn:FactQuery xfn:uri="ds://ALL-ROUTES" xfn:mode="Set">
            <LINK node1="Svar_from" int1="Svar_int1" ip1="Svar_ip" node2="
                Svar_any" int2="Svar_otherint"
                 ip2="Svar_otherip" metric="Svar_metric"/>
        </xfn:FactQuery>
        <xfn:FactQuery xfn:uri="ds://UTILIZATION" xfn:mode="Set">
            <LINK ip1="Svar_ip" state="ok" utilization="Svar_utilization"/
                >
        </xfn:FactQuery>
        <xfn:Not>
            <xfn:StringIsMember xfn:string="Svar_any" xfn:list="
                Svar_crossed"/>
        </xfn:Not>
        <xfn:Not>
            <xfn:StringEqual xfn:string1="Svar_any" xfn:string2="Svar_to"/
                >
        </xfn:Not>
        <xfn:LessThanOrEqual xfn:number1="Svar_utilization" xfn:number2="
            Svar_maxutilization"/>
```

```
        <xfn:Add xfn:number1="Svar_metric" xfn:number2="Svar_sum2" xfn:
            result="Svar_sum"/>
        <xfn2:ConcatString xfn2:string1="Svar_crossed" xfn2:string2=" "
            xfn2:result="Svar_temp"/>
        <xfn2:ConcatString xfn2:string1="Svar_temp" xfn2:string2="Svar_any
            " xfn2:result="Svar_crossed2"/>
        <Request>
            <from>Svar_any</from>
            <to>Svar_to</to>
            <utilization maxvalue="Svar_maxutilization"/>
            <path sum="Svar_sum2" nodecrossed="Svar_crossed2">
                Evar_routeok2
            </path>
        </Request>
    </xet:Body>
</xet:Rule>
<xet:Rule xet:name="Set" xet:priority="1">
    <xet:Meta>
        <xet:RuleDescription>Rule for set</xet:RuleDescription>
    </xet:Meta>
    <xet:Head>
        <Request>
            <from>Svar_from</from>
            <to>Svar_to</to>
            <utilization maxvalue="Svar_maxutilization"/>
            <route sum="Svar_sum" nodecrossed="Svar_crossed">
                Evar_result
            </route>
            <best>Evar_shortestpath</best>
        </Request>
    </xet:Head>
    <xet:Body>
        <xfn:SetOf xfn:mode="Set">
            <xfn:Set>Evar_result</xfn:Set>
            <xfn:Constructor>
                <path sum="Svar_sum">
                    Evar_routeok
                </path>
            </xfn:Constructor>
            <xfn:Condition>
                <Request>
                    <from>Svar_from</from>
                    <to>Svar_to</to>
                    <utilization maxvalue="Svar_maxutilization"/>
                    <path sum="Svar_sum" nodecrossed="Svar_crossed">
```

```
                    Evar_routeok
                </path>
            </Request>
        </xfn:Condition>
    </xfn:SetOf>

    <FindMinimum>
        <data value="Svar_minimun">
            Evar_result
        </data>
        <best>Evar_shortestpath</best>
    </FindMinimum>
</xet:Body>
</xet:Rule>

<xet:Rule xet:name="RuleMinimum" xet:priority="1">
    <xet:Meta>
        <xet:RuleDescription>Rule for returning minimum value</xet:
            RuleDescription>
    </xet:Meta>
    <xet:Head>
        <FindMinimum>
            <data value="Svar_valueminimum">
                Evar_allpaths
            </data>
            <best>Evar_shortestpath</best>
        </FindMinimum>
    </xet:Head>

    <xet:Body> <!-- Body1 OK-->
        <xfn:MatchD xfn:mode="Set">
            <X>Evar_allpaths</X>
            <X></X>
        </xfn:MatchD>
        <xfn:AssignString xfn:variable="Svar_valueminimum" xfn:value="
            100000"/>
        <xfn:MatchD xfn:mode="Set">

            <X></X>
            <X>Evar_shortestpath</X>

        </xfn:MatchD>
    </xet:Body>

    <xet:Body> <!-- Body2 one element -->
```

```
    <xfn:MatchD xfn:mode="Set">
        <X>
            <path sum="Svar_value1">Evar_whatever</path>
        </X>
        <X>Evar_allpaths
        </X>
    </xfn:MatchD>
    <xfn:AssignString xfn:variable="Svar_valueminimum" xfn:value="
        Svar_value1"/>
    <xfn:MatchD xfn:mode="Set">
        <X>Evar_shortestpath</X>
        <X>
            <path sum="Svar_value1">Evar_whatever</path>
        </X>
    </xfn:MatchD>
</xet:Body>

<xet:Body> <!-- Body3 two or more element-->
    <xfn:MatchD xfn:mode="Set">
        <X>
            <path sum="Svar_value1">Evar_whatever</path>
            Evar_rest_paths
        </X>
        <X>
            Evar_allpaths
        </X>

    </xfn:MatchD>
    <FindMinimum>
        <data value="Svar_minimum2">
            Evar_rest_paths
        </data>
        <best>Evar_rest_shortestpath</best>
    </FindMinimum>

    <xfn:LessThanOrEqual xfn:number1="Svar_value1" xfn:number2="
        Svar_minimum2"/>
    <xfn:AssignString xfn:variable="Svar_valueminimum" xfn:value="
        Svar_value1"/>

    <xfn:MatchD xfn:mode="Set">
        <X>
            <path sum="Svar_value1">Evar_whatever</path>
        </X>
        <X>Evar_shortestpath</X>
```

```
            </xfn:MatchD>
        </xet:Body>

        <xet:Body> <!-- Body4 three or more element-->
            <xfn:MatchD xfn:mode="Set">
                <X>
                    <path sum="Svar_value1">Evar_whatever</path>
                    Evar_rest_paths
                </X>
                <X>
                    Evar_allpaths
                </X>
            </xfn:MatchD>
            <FindMinimum>
                <data value="Svar_minimum2">
                    Evar_rest_paths
                </data>
                <best>Evar_rest_shortestpath</best>
            </FindMinimum>

            <xfn:LessThan xfn:number1="Svar_minimum2" xfn:number2="Svar_value1
                "/>
            <xfn:AssignString xfn:variable="Svar_valueminimum" xfn:value="
                Svar_minimum2"/>
            <xfn:MatchD xfn:mode="Set">
                <X>Evar_rest_shortestpath</X>
                <X>Evar_shortestpath</X>
            </xfn:MatchD>
        </xet:Body>
    </xet:Rule>
</xet:XET>
```

Appendix B

Router´s Data Base Files

Listing B.1: ALL-LINKS

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?><xet:XET xmlns:xet="
    http://xet.sf.net" xmlns:contact="http://www.w3.org/2000/10/swap/pim/
    contact#" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
   <xet:Fact>
      <LINK int1="g1/0" int2="g1/0" ip1="1.1.1.1" ip2="1.1.1.2" metric="10"
           node1="oslo_gw1" node2="trd_gw1"/>
      <LINK int1="g5/0" int2="g4/0" ip1="1.1.5.1" ip2="1.1.5.2" metric="10"
           node1="oslo_gw1" node2="stolaw_gw2"/>
      <LINK int1="f0/0" int2="f0/0" ip1="1.1.7.1" ip2="1.1.7.2" metric="15"
           node1="oslo_gw1" node2="R1"/>
      <LINK int1="f2/0" int2="f2/0" ip1="1.1.2.1" ip2="1.1.2.2" metric="15"
           node1="oslo_gw1" node2="trd_gw2"/>
      <LINK int1="f3/0" int2="f3/0" ip1="1.1.3.1" ip2="1.1.3.2" metric="15"
           node1="oslo_gw1" node2="trd_gw2"/>
      <LINK int1="g1/0" int2="g1/0" ip1="1.1.1.2" ip2="1.1.1.1" metric="10"
           node1="trd_gw1" node2="oslo_gw1"/>
      <LINK int1="g4/0" int2="g5/0" ip1="1.1.5.2" ip2="1.1.5.1" metric="10"
           node1="stolaw_gw2" node2="oslo_gw1"/>
      <LINK int1="f0/0" int2="f0/0" ip1="1.1.7.2" ip2="1.1.7.1" metric="15"
           node1="R1" node2="oslo_gw1"/>
      <LINK int1="f2/0" int2="f2/0" ip1="1.1.2.2" ip2="1.1.2.1" metric="15"
           node1="trd_gw2" node2="oslo_gw1"/>
      <LINK int1="f3/0" int2="f3/0" ip1="1.1.3.2" ip2="1.1.3.1" metric="15"
           node1="trd_gw2" node2="oslo_gw1"/>
      <LINK int1="f1/0" int2="f1/0" ip1="4.4.1.1" ip2="4.4.1.2" metric="15"
           node1="trd_hovedbygget_gw" node2="stolaw_gw2"/>
      <LINK int1="f0" int2="f1/0" ip1="1.1.14.2" ip2="1.1.14.1" metric="15"
           node1="R2" node2="R1"/>
      <LINK int1="f2/0" int2="f1/0" ip1="2.2.4.1" ip2="2.2.4.2" metric="10"
           node1="trd_gw1" node2="trd_gw2"/>
```

```
        <LINK int1="g3/0" int2="g2/0" ip1="2.2.2.1" ip2="2.2.2.2" metric="10"
             node1="trd_gw1" node2="trd_hovedbygget_gw"/>
        <LINK int1="f4/0" int2="f3/0" ip1="2.2.3.1" ip2="2.2.3.2" metric="15"
             node1="trd_gw1" node2="trd_hovedbygget_gw"/>
        <LINK int1="f2/0" int2="f2/1" ip1="2.2.5.2" ip2="2.2.5.1" metric="15"
             node1="R2" node2="trd_gw1"/>
        <LINK int1="f1/0" int2="f1/0" ip1="4.4.1.2" ip2="4.4.1.1" metric="15"
             node1="stolaw_gw2" node2="trd_hovedbygget_gw"/>
        <LINK int1="f1/0" int2="f010" ip1="1.1.14.1" ip2="1.1.14.2" metric="
            15" node1="R1" node2="R2"/>
        <LINK int1="f1/0" int2="f2/0" ip1="2.2.4.2" ip2="2.2.4.1" metric="10"
             node1="trd_gw2" node2="trd_gw1"/>
        <LINK int1="g2/0" int2="g3/0" ip1="2.2.2.2" ip2="2.2.2.1" metric="10"
             node1="trd_hovedbygget_gw" node2="trd_gw1"/>
        <LINK int1="f3/0" int2="f4/0" ip1="2.2.3.2" ip2="2.2.3.1" metric="15"
             node1="trd_hovedbygget_gw" node2="trd_gw1"/>
        <LINK int1="f2/1" int2="f2/0" ip1="2.2.5.1" ip2="2.2.5.2" metric="15"
             node1="trd_gw1" node2="R2"/>
    </xet:Fact>
</xet:XET>
```

Listing B.2: UTILIZATION

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?><xet:XET xmlns:
    xet="http://xet.sf.net" xmlns:contact="http://www.w3.org/2000/10/
    swap/pim/contact#" xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
    syntax-ns#">
    <xet:Fact>
        <LINK ip1="1.1.1.1" state="down" utilization="10"/>
        <LINK ip1="1.1.1.2" state="down" utilization="10"/>
        <LINK ip1="1.1.5.1" state="ok" utilization="20"/>
        <LINK ip1="1.1.5.2" state="ok" utilization="20"/>
        <LINK ip1="1.1.7.1" state="ok" utilization="30"/>
        <LINK ip1="1.1.7.2" state="ok" utilization="30"/>
        <LINK ip1="1.1.2.1" state="ok" utilization="50"/>
        <LINK ip1="1.1.2.2" state="ok" utilization="50"/>
        <LINK ip1="1.1.3.1" state="ok" utilization="50"/>
        <LINK ip1="1.1.3.2" state="ok" utilization="50"/>
        <LINK ip1="2.2.2.1" state="ok" utilization="50"/>
        <LINK ip1="2.2.2.2" state="ok" utilization="50"/>
        <LINK ip1="2.2.3.1" state="ok" utilization="50"/>
        <LINK ip1="2.2.3.2" state="ok" utilization="50"/>
```

```
        <LINK ip1="2.2.4.1" state="ok" utilization="50"/>
        <LINK ip1="2.2.4.2" state="ok" utilization="50"/>
        <LINK ip1="2.2.5.1" state="ok" utilization="50"/>
        <LINK ip1="2.2.5.2" state="ok" utilization="50"/>
        <LINK ip1="1.1.14.1" state="ok" utilization="50"/>
        <LINK ip1="1.1.14.2" state="ok" utilization="50"/>
        <LINK ip1="4.4.1.1" state="ok" utilization="50"/>
        <LINK ip1="4.4.1.2" state="ok" utilization="50"/>
    </xet:Fact>
</xet:XET>
```

Appendix C

Example Answer of the NxET Reasoning Engine

Listing C.1: Answer to the query message from listing 4.1

```
<Request>
  <from>trd_gw1</from>
  <to>oslo_gw1</to>
  <utilization maxvalue="80"/>
  <route sum="Svar_sum" nodecrossed="Svar_crossed">
    <path sum="10">
      <nexthop metric="10" nextNode="oslo_gw1" utilization="10" interface="g1
          /0"/>
    </path>
    <path sum="25">
      <nexthop metric="10" nextNode="trd_gw2" utilization="50" interface="f2
          /0"/>
      <nexthop metric="15" nextNode="oslo_gw1" utilization="50" interface="f2
          /0"/>
    </path>
    <path sum="35">
      <nexthop metric="10" nextNode="trd_hovedbygget_gw" utilization="50"
          interface="g3/0"/>
      <nexthop metric="15" nextNode="stolaw_gw2" utilization="50" interface="
          f1/0"/>
      <nexthop metric="10" nextNode="oslo_gw1" utilization="20" interface="g4
          /0"/>
    </path>
    <path sum="40">
      <nexthop metric="15" nextNode="trd_hovedbygget_gw" utilization="50"
          interface="f4/0"/>
      <nexthop metric="15" nextNode="stolaw_gw2" utilization="50" interface="
          f1/0"/>
```

```
      <nexthop metric="10" nextNode="oslo_gw1" utilization="20" interface="g4
          /0"/>
    </path>
    <path sum="45">
      <nexthop metric="15" nextNode="R2" utilization="50" interface="f2/1"/>
      <nexthop metric="15" nextNode="R1" utilization="50" interface="f0"/>
      <nexthop metric="15" nextNode="oslo_gw1" utilization="30" interface="f0
          /0"/>
    </path>
    <path sum="25">
      <nexthop metric="10" nextNode="trd_gw2" utilization="50" interface="f2
          /0"/>
      <nexthop metric="15" nextNode="oslo_gw1" utilization="50" interface="f3
          /0"/>
    </path>
  </route>
  <best>
    <path sum="10">
      <nexthop metric="10" nextNode="oslo_gw1" utilization="10" interface="g1
          /0"/>
    </path>
  </best>
</Request>
               <- .
```