Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática

Dissertação de Mestrado

Mestrado em Engenharia Informática

Simple and Stable Dynamic Traffic Engineering for Provider Scale Ethernet

António Edgar Carvalho Teixeira (aluno nº 28086)

2º Semestre de 2009/10

Setembro de 2010

Universidade Nova de Lisboa
Faculdade de Ciências e Tecnologia
Departamento de Informática

Dissertação de Mestrado

Simple and Stable Dynamic Traffic Engineering for Provider Scale Ethernet

António Edgar Carvalho Teixeira (aluno nº 28086)

Orientador: Prof. Doutor José Legatheaux Martins

Trabalho apresentado no âmbito do Mestrado em Engenharia Informática, como requisito parcial para obtenção do grau de Mestre em Engenharia Informática.

2º Semestre de 2009/10
Setembro de 2010

Abstract

The high speeds and decreasing costs of Ethernet solutions has motivated providers' interest in using Ethernet as the link layer technology in their backbone and aggregation networks. Provider scale Ethernet offers further advantages, providing not only an easy to manage solution for multicast traffic, but also transparent interconnection between clients' LANs. These Ethernet deployments face altogether different design issues, requiring support for a significantly higher number of hosts. This support relies on hierarquization, separating address and virtual network spaces of customers and providers.

In addition, large scale Ethernet solutions need to grant forwarding optimality. This can be achieved using traffic engineering approaches. Traffic engineering defines the set of engineering methods and techniques used to optimize the flow of network traffic. Static traffic engineering approaches enjoy widespread use in provider networks, but their performance is greatly penalized by sudden load variations. On the other hand, dynamic traffic engineering is tailored to adapt to load changes. However, providers are skeptical to adopt dynamic approaches as these induce problems such as routing instability, and as a result, network performance decreases.

This dissertation presents a Simple and Stable Dynamic Traffic Engineering framework (SSD-TE), which addresses these concerns in a provider scale Ethernet scenario. The validation results show that SSD-TE achieves better or equal performance to static traffic engineering approaches, whilst remaining both stable and responsive to load variations.

Keywords: Ethernet, Traffic Engineering, Routing

Contents

1. Introduction

In everyday life, choosing a path between two points is one of the most common problems people face. For instance, assume that someone wants to drive his/her car from point A to point B, using a network of highways. Often that driver will select the path between A and B which appears to be the best. Note that the definition of best path can be highly variable: it either spans the smallest extension (in kilometres/miles), has the smallest time interval for traversal, or is the shortest by any other metric the driver deems suitable.

Furthermore, the driver can select the path empirically or by using a suitable tool (e.g. gps, google maps). However, most approaches can lead to a path which is not the best for the considered metric. The main cause for this problem is the current (and variable) amount of traffic present in the highway network, which more often than not, is not accounted by the driver/tools. As a result, some highways will become congested and others will become underused, leading to overall user dissatisfaction.

This behaviour can be transposed to backbone networks of Internet Service Providers (ISPs). Assume that the highway points of entry are now Points of Presence (POPs) of an ISP, where the traffic enters and exits the network, and that the cars are now network packets. The same problem occurs: the routing of network information using shortest path algorithms might lead to network congestion and underperformance. Therefore, ISPs requiring high network performance resort to *traffic engineering* in lieu of shortest path algorithms. Traffic engineering can be defined as the set of engineering paradigms and techniques that are aimed at optimising the flow of traffic in their network, considering the network's traffic pattern. This dissertation presents a traffic engineering framework, aimed at Ethernet-based provider networks.

Ethernet is a set of computer network technologies which constitute the *de-facto* standard in residential/small business networks. On top of that, Ethernet is greatly increasing its relevance in larger networks, such as Internet Service Provider (ISP) networks. This chapter comprises a description of the context and motivation of the dissertation, detailing the topics of provider-scale Ethernet networks and Traffic Engineering. The remainder sections go on to present an overview of the goals and philosophy of the developed framework. Finally, the chapter closes with an outline of the dissertation.

1.1 Motivation

Provider Scale Ethernet

Data link layer protocols play an important role in communication systems, being responsible for data transfer between directly connected nodes or in the same subnetwork (in TCP/IP parlance). In the context of Local Area Networks (LAN's), Ethernet enjoys widespread use as the underlying link-layer technology. Ethernet follows a connectionless transfer model, based on the switching of data units known as Ethernet frames.

Ethernet's success in LAN scenarios can be explained by several factors. Firstly, mechanisms such as the Spanning Tree Protocol (STP) and Virtual LAN's (VLAN) provide ease of configuration and management to Ethernet deployments. The former, coupled with a frame filtering process (learning host addresses by the reverse path), allows for dynamic configuration of bridged network segments in a loop-free configuration and establishes forwarding tables. The latter, VLAN's, allow for traffic isolation between sets of network hosts, effectively setting up a separate local area network for them. Moreover, Ethernet's constant increasing transmission speeds (10Gbps is the current standard) and decreasing costs have promoted its widespread market availability.

As opposed to LAN's, traditional providers have followed different approaches in their aggregation and backbone networks. These networks frequently used connection-oriented protocols in the data link layer, which required the setup of virtual/logical circuits in the network in order to transfer data. Examples of such technologies are the Asynchronous Transfer Mode (ATM) and Synchronous Digital Hierarchy networks (SDH).

Nevertheless, customers such as large enterprises may desire to connect geographically separate Ethernet LAN segments transparently, relying on their providers to offer Ethernet services on their backbone network. Furthermore, multicast traffic is becoming of primary importance in provider networks due to IP-TV deployments, among other applications. Ethernet offers simple solutions to handle multicast traffic, where frames of this type are broadcast onto a VLAN comprising the group members. What is more, switches can dynamically join the group using techniques such as IGMP snooping [45]. These requirements, alongside with the aforementioned increasing speeds and decreasing costs, have motivated the implementation of Ethernet-based provider-scale networks [9].

This application scenario greatly differs from LAN's since the number of connected hosts and traffic volume is significantly higher. These differences put to light the scalability issues of the previously seen Ethernet mechanisms. In the first place, the address tables resulting from the learning process grow to unbearable sizes and impair the forwarding process. Secondly, the maximum number of allowed VLAN's is limited to 4096, which is a rather small figure considering that each customer may be associated with several VLAN's simultaneously. Finally, the forwarding of frames based on STP with filtering can be far from optimal, potentially resulting in network congestion and poor performance for high traffic volumes. On top of that, legacy STP converges in tens of seconds in case of network reconfigurations.

Solutions addressing the first and second problem are fixed in Ethernet large scale deployments by separating the address and VLAN spaces of customers and providers. One of such solutions is Q-in-Q tagging of Ethernet frames [4]. It allows the stacking of multiple VLAN headers on a frame, effectively extending the maximum number of available VLAN's. The Provider Bridges standard (IEEE 802.1ad) builds upon Q-in-Q, creating the notion of separate customer and service VLAN identifier spaces. On top of that, the Provider Backbone Bridges (PBB) architecture achieves separation not only of VLAN spaces but also of customer and provider address spaces. To this end, PBB relies on the encapsulation of customer Ethernet frames in service Ethernet frames (with different bridge addresses and VLAN identifiers) [9].

The last issue faced by provider-scale Ethernet, related with its inherent forwarding inefficiency, is fixed by disregarding STP (and filtering mechanisms) and resorting to the use of more conventional routing architectures (e.g. shortest path routing) or to traffic engineering.

Traffic Engineering

Traffic engineering concerns the adaptation of routing to the network conditions, allowing for performance enhancements and efficient use of network resources [15]. Furthermore, it delays the need for network capacity improvements, decreasing expansion costs [21]. This advantage comes from the fact that traffic engineering optimizes the distribution of traffic onto a network, reducing the need for over-provisioning of link capacities.

Traffic engineering typically requires that specific forwarding paths be assigned to frames (or packets, in network layer architectures) when entering the network, which contrasts with the usual hop-by-hop forwarding paradigm which is followed in Ethernet LAN's or by common interior gateway routing protocols.

A common mechanism used to provide this path assignment is Multiprotocol Label Switching, also known as MPLS [25]. MPLS works by stacking a label on top of packets entering the network (and correspondly removing it at the exit), switching the packets in the network core according to their labels. Thus MPLS requires that a correspondence between labels and forwarding paths is set onto the network. These paths are known as Label Switched Paths (LSP's), which effectively tunnel traffic between ingress and egress routers (Label Edge Routers, or LER's). Moreover, multiple LSP's can be created between the same ingress and egress node pair. This set of paths can then be used for load balancing or quality of service purposes (and ultimately traffic engineering).

A similar scheme targeted at Ethernet-based architectures is PBB-Traffic Engineering (PBB-TE) [9]. Whereas standard Ethernet used STP and frame filtering to establish forwarding tables, PBB-TE uses static forwarding entries which support PBB-TE trunks. These trunks work in a similar fashion to LSP's, tunneling traffic from source to destination bridges.

Apart from traffic engineering supporting architectures, the approach used to optimize the network routing is a decisive factor contributing to the goals of traffic engineering. Most of these approaches follow a static paradigm, that is to say, the distribution of traffic flows[1]is computed in a centralized machine having as input previous measurements of network statistics. The resulting traffic distribution is then mapped to the network using a suitable mechanism, such as MPLS LSP's or PBB-TE trunks (depending on the architecture being used).

The aforementioned measurements are used to estimate a network traffic matrix. This matrix comprises the traffic load for all ingress and egress nodes (bridges or routers) pairs. A new traffic distribution should be calculated whenever the traffic matrix is expected to change significantly and then applied to the network. Assuming that traffic matrices remain largely constant during significant periods of time, this type of approach is feasible and yields near-optimal results.

[1]In the context of this work, a flow comprises all traffic with the same ingress and egress nodes

However, this assumption does not always hold true, mainly for two reasons. In the first place, the dominance of file sharing applications in the Internet has led to the increasing prevalence of long duration TCP connections. Due to the transmission rate oscillations induced by the end-to-end congestion control mechanism of TCP, such long-lived traffic follows an elastic pattern and thus traffic matrices are expected to show greater variability over time [21].

Secondly, the existence of phenomena like flash crowds (e.g. the flooding of news websites when an important event occurs) leads to unexpected and sudden changes in the network traffic matrix [14]. This type of events renders static traffic engineering useless, as the changes occur in a smaller time frame than the required to run the traffic engineering process and reconfigure the network.

In practice, both presented problems are fixed by capacity over-provisioning and calculating traffic distributions that give priority to low utilisation of networks' links instead of trying to route by the shortest path. These fixes attempt to reserve what could be described as a capacity buffer, thus guarding against changes to the estimated traffic matrix.

In light of the above, static approaches to traffic engineering have the significant drawback of slow reactivity in the presence of traffic matrix modifications. Another shortcoming relies on the fact that these approaches require centralised management, as opposed to the ease of configuration and management that traditional Ethernet and IP routing solutions offer. Dynamic traffic engineering schemes attempt to provide solutions to both these problems, albeit still optimising routing and efficient use of the network resources.

Dynamic traffic engineering adapts the routing of traffic on-the-fly as the network conditions change, namely its traffic matrix. Solutions following this approach attempt to minimise centralized computations, placing the burden of calculating the traffic distribution on the network nodes. In most cases, this means that nodes must acquire information about the network characteristics through some sort of signaling mechanism in order to load balance incoming traffic appropriately.

Despite the apparent advantages, dynamic traffic engineering solutions have had limited practical acceptance. This is due to the fact that most dynamic approaches induce network instability, which is rather grievous to TCP traffic, because of the resulting packet re-ordering and effects on TCP's congestion control mechanism. The said instability comes from the likelihood of constant changes in the traffic distribution any time there is a variation (even if slight) in the traffic matrix. This is especially true if the nature of elastic TCP traffic is considered, which is prone to cause small but constant modifications to the matrix.

In fact, experiences with the routing metric used in ARPANET (the precursor of Internet) put to light the above difficulty [33]. For a time, ARPANET relied on a shortest path algorithm to compute network routes, using the average value of packet delay over a 10 second period as the link metric. This approach caused high routing instability whenever the traffic volume was significantly high, as well as wasted link bandwidth and node processing power. Thus the metric was changed in periods of high traffic load, instead relying on the fixed link capacity. The same approach to estimate link costs is also known to be responsible for high instability if used with link state protocols, such as OSPF or IS-IS.

Besides stability concerns, other difficulties impair the applicability of dynamic traffic engineering. To start with, most solutions are quite disruptive from today's protocols and architectures, such as those presented previously in this section. Naturally, coupling the disruptive nature of solutions with earlier practical applications (such as ARPANET's) makes providers skeptical about adopting entirely new network schemes and architectures for dynamic traffic engineering, notwithstanding how adequate their proposals may seem.

As a closing remark, traffic engineering paradigms, both static and dynamic, largely ignore the existence of multicast traffic, focusing only on unicast traffic. This can be explained by the obstacles faced in accounting for this type of traffic in networks, which makes it hard to estimate and react to the network traffic conditions, let alone optimise routing for multicast traffic.

Problem Definition

In light of the above motivation, several issues are faced by dynamic traffic engineering approaches. These problems can be abstracted as design requirements and define a problem to be solved by the traffic engineering framework presented in the dissertation. The requirements are described as follows.

Optimality Optimal routing should be guaranteed in respect to a given optimality criteria (e.g. path length, in shortest path routing). The achieved optimality should not only apply to unicasting but also to multicasting. In fact, multicast traffic engineering faces several open challenges, such as accounting or routing optimisation, which must be addressed.

Routing Stability The lack of stability is largely responsible for the low adoption of dynamic traffic engineering approaches. The traffic flow distribution should not vary needlessly in the presence of high traffic volumes or small changes in the network load. All the same, excess routing stability may lead to low convergence times whenever the traffic distribution must change to optimize routing.

Fault Tolerance Failures of network components cannot be set aside even in highly resilient provider networks, albeit happening in a significantly larger timescale than the changes in network traffic load. As a consequence, the implications of failures in the signaling and computational mechanisms of dynamic traffic engineering schemes must be taken into account.

Implementation Compatibility Dynamic traffic engineering schemes should not force providers to change their network architecture and use untested protocols. The compatibility of scheme's implementations with proven and robust network standards increases their practical applicability and success. Furthermore, the achieved compatibility should not result in costly and complex management or configuration procedures.

Scalability Aside from these design requirements, dynamic traffic engineering schemes must be scalable to the high traffic volume expected in provider networks. Low signaling and computational complexities are vital for the success of such solutions.

1.2 Traffic Engineering Framework

The presented design requirements led to development of a traffic engineering framework, the Simple and Stable Dynamic Traffic Engineering Framework, or SSD-TE for short.

In this dissertation, priority was given to the following issues: unicast optimality, stability, implementation compatibility and scalability. The problems related with the support for multicast traffic engineering and dealing with node or link failures were considered to be out of scope. The former was disregarded due to multicast traffic accountability and routing optimality being open problems. On the other hand, although node/link failures have a degree of impact on the behaviour of dynamic traffic engineering approaches, these failures often happen on a much larger timescale than network load changes and can thus be separated from the aims set above. Moreover, it would have been unlikely that, in the available period of time, it would have been possible to obtain valid results regarding these two challenging design concerns.

The philosophy of SSD-TE is that a simple ideal method to operate a network would be provisioning it in a way that shortest path routing of the expected traffic would not increase packet loss or delay. In the event of network congestion however, the unexpected surplus load should be balanced using either a provisioned extra amount of capacity or available capacity in underused network paths.

To this end, the framework contains three different components: a provisioning method, a forwarding scheme, and distributed adjustment algorithm.

SSD-TE's provisioning method assures the following guarantee: all normally expected traffic fits in the network if routed through the shortest path, i.e. there is no packet loss. Moreover, the provisioning method allows the operator to overprovision the network so that it accomodates a specific range of surplus load. In this range, all possible variations of network load fit the network, assuming that the surplus load is balanced using the SSD-TE forwarding scheme.

The forwarding scheme of SSD-TE is based on a straightforward concept. It ensures that as much network traffic as possible is forwarded by the shortest path, whilst avoiding congestion. Remaining traffic is distributed among load balancing paths, using the ELB algorithm [43].

The last component of SSD-TE is a distributed adjustment algorithm. This algorithm adapts the load amounts that are either forwarded by shortest path or balanced among paths, between each pair of network nodes. The distributed adjustment algorithm relies on the exchange of end-to-end messages between nodes in order to collect link load statistics, which are then used to dynamically adapt the traffic distribution as the network load changes.

In accordance to its design requirements, the main contribution of SSD-TE is providing a dynamic traffic engineering framework that yields results similar to static traffic engineering

approaches for provisioned traffic matrices, while at the same time handling sudden and unexpected changes to the traffic matrix.

Finally, the framework validation results presented in the dissertation support the claim that SSD-TE achieves the stated contributions. The dissertation goes on to discuss possible SSD-TE adaptations that would be able to tackle the design requirements of fault tolerance and multicast traffic optimality.

1.3 Outline of the Dissertation

The remainder of the dissertation is structured as follows. Firstly, chapter 2 describes the related work, containing state-of-the-art studies and approaches related with traffic engineering and other similar techniques, as well as their critical analysis in respect to this thesis' aims.

Secondly, chapter 3 presents the SSD-TE framework in detail, describing its components and laying out their respective theoretical background. Furthermore, it discusses the implementation of the framework in an Ethernet scenario.

Chapter 4 specifies the validation model, tools and process required to evaluate the developed traffic engineering scheme. This section also comprises the description of the test cases and their results, along with their respective analysis and summary.

Finally, chapter 5 ends the dissertation, outlining the conclusions and giving insight onto the future work to be done on the dissertation field of study.

2. Related Work

In the previous chapter, it has been seen that influencing the routing of flows in a network can improve its overall performance, reducing packet loss and delay. If no topology modifications are involved, the measures used to this end are known as traffic engineering.

Depending on whether traffic engineering approaches are adaptable to changes in network conditions, they fall in two different categories. On the one hand, static traffic engineering relies on an off-line procedure to calculate an improved distribution of network flows. On the other hand, dynamic traffic engineering adjusts the traffic distribution autonomously as network conditions vary.

In this chapter both static and dynamic traffic engineering state-of-the-art approaches are presented. In particular, the dynamic solutions are analysed in further depth, having in mind their relevance to the goals set in the introduction of this report. Finally, this chapter closes with a discussion on alternative paradigms with the same aims of traffic engineering.

2.1 Static Traffic Engineering

Static traffic engineering is the most commonly used form of traffic engineering. Unlike dynamic solutions, static traffic engineering approaches are not affected by network instability concerns, which makes providers much less skeptical to adopt them.

Current traffic engineering frameworks following this paradigm are typically organised as follows. First, they comprise a method to estimate the network traffic matrix. A network (point-to-point) traffic matrix contains the set of traffic rates for all network flows. In this context, a flow designates the network traffic with a given ingress and egress node pair.

Second, an algorithm is necessary to compute the available paths and the distribution of network flows among them. This algorithm further requires a metric to evaluate its optimality. Finally, frameworks must ensure that traffic follows the calculated distribution on the network through the use of a routing approach.

The remainder of this section comprises a description of the said components and finishes with a review of the existing protocols that allow the mapping of traffic engineering solutions to networks.

2.1.1 Traffic Matrix Estimation

Two main approaches exist for estimating a network's traffic matrix [20]. The first relies on the direct measurement of flow information and corresponding routing tables, which requires a posterior off-line analysis of the recovered data. This approach is computationally intensive, owing to the processing and storage of the said data.

On the other hand, the traffic matrix can be derived from online link measurements, which can be obtained using the Simple Network Management Protocol (SNMP). SNMP is a protocol

which can be used for managing and monitoring the state of network nodes [28].

However, using link measurements does not suffice to estimate a traffic matrix. Consider that in a network of N nodes, $O(N)$ links exist and the traffic matrix size is $O(N^2)$. It follows the number of constraints is $O(N)$ and the number of variables is $O(N^2)$. Therefore, the estimation problem is not sufficiently constrained to be solved. In order to solve it, additional constraints must be assumed, specifically about the routing or traffic types.

Solutions approaching this problem include statistical tomographic methods and optimization-based tomographic methods. The former approach creates additional constraints for the problem using higher-order statistics of link load, such as covariance or skew. The latter uses the optimisation of a linear or quadratic programming model to select an estimate matrix among the solution space.

2.1.2 Algorithms and Metrics

Having as input the estimated traffic matrix for the network, a general-purpose static traffic engineering algorithm optimizes the distribution of flows on network's links according to a pre-selected objective function [21]. In addition, several objective functions can be combined to form a multi-objective optimisation problem, either setting weights to each individual objective or by prioritizing one over another.

2.1.2.1 Metrics

An important criteria that must be taken into account while designing proper objective functions for the said optimisation problem is the network performance metric being considered. The following paragraphs describe some of the possible metrics.

Average Path Length The minimisation of this metric corresponds to the common objective of interior routing protocols, which is providing shortest path routing for all flows. Although it has been stated that shortest path routing does not necessarily yield optimal network performance, it remains as an important metric to be considered in order to achieve low delay.

Maximal Utilisation This metric represents the maximum ratio between the load and capacity for any network link, corresponding to the top bottleneck link in the network. Minimizing this metric will avoid network bottlenecks, providing a capacity buffer on each link that allows some degree of protection against unestimated traffic increases.

Average Utilisation The use of average utilisation relies on the consideration of the mean load-capacity ratio (i.e. the utilisation) of all networks links, as opposed to the earlier metric.

Average Load This metric improves on the previous one by weighing the link's utilisation with its capacity. This is due to the earlier metric disregarding the number of flows that each

link is allowed to carry, which has a decisive influence on the overall performance.

Congestion Costs The design of this metric takes into account that the average of both loss and delay parameters should be influenced by traffic engineering [21]. This is achieved by defining the metric as a non-linear convex function of the link's utilisation or load. For simplicity (although remaining sufficiently similar), the said function can be based on the relation between the loss and delay parameters and the link's utilisation, assuming a M/M/1/B queueing model. An example of such a congestion cost function can be seen in figure 2.1. In accordance to the motivation specified in the previous metric, the congestion costs may also be weighted by the link's capacity or load.

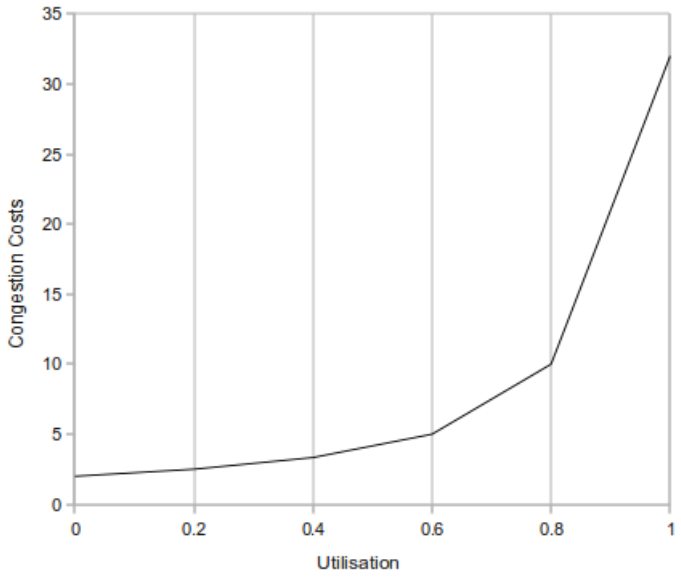

Figure 2.1 Congestion Cost Function Example

2.1.2.2 Flow-Path Selection Algorithms

Other than the considered metrics, the strategies used to solve the presented optimisation problem may vary. The more straightforward of these approaches consists in the explicit routing technique. The network's topology is modelled using sets of incoming and outgoing links for each node, whereas the estimated traffic matrix is mapped to variables representing the size, ingress node and egress node for each flow. The objective function takes into account the value of one of the previous metrics (or a combination of thereof) in respect to the distribution of flows on the network's links, and is subject to a series of constraints.

The described model can support both singlepath and multipath routing for a single flow. The difference between the former and the latter resides on the definition of a constraint: the

variable that represents the attribution of a flow to a given link can be either binary or any value between 0 and 1, respectively. This greatly influences the algorithm chosen to provide a solution. The multipath problem is a linear programming model and can be solved with the simplex algorithm, whereas the singlepath problem must be solved with mixed integer programming solutions (such as Branch and Bound) which are not efficient (specially considering the total number of binary variables is $O(N^3)$, N being the number of nodes).

An approach proposed by [21], known as path selection, reduces the complexity of the previous model. This scheme separates the optimisation problem from the computation of the set of possible paths for the flows. The computation of paths can resort to a k-shortest path algorithm which runs in polynomial time. The resulting problem models (for singlepath and multipath) can be solved with the same techniques used by the earlier approach, but its complexity is reduced to $O(N^2)$.

Finally, since the path computation procedure does not provide all possible network paths, the solution provided by this model may not be optimal. However, the results of [21] show that the distance to the optimal value is not significant and that the path selection method further offers a degree of control over the selected paths (e.g.: only selecting paths for a given flow within a reasonable hop count distance from the shortest path).

2.1.2.3 Link Weight Manipulation Algorithms

There are different approaches to static traffic engineering other than modeling a problem that attempts to optimise the networks performance using a series of paths. Instead, the problem can be defined as the computation of the optimal link weights, thus adapting the overall routing to a desired performance criteria (e.g. one of the presented metrics). The latter has the advantage that an explicit routing architecture such as MPLS or PBB-TE is not necessary to implement traffic engineering.

The proof presented in [16] shows that such computation is NP-hard and it is thus not a feasible approach. The same source proposes an approximation using local-search heuristics, which may not be optimal. However, experimental results, using OSPF as the underlying routing protocol, show that the chosen heuristic settings provide a solution that only deviates a small percentage from a classic explicit routing approach (like the one presented earlier in this section). This result makes this traffic engineering approach viable.

The PEFT protocol proposal [47] also relies on the manipulation of link weights (using a link-state protocol) in order to provide traffic engineering. PEFT achieves optimal traffic engineering feasibly: unlike the previous approach, the computational complexity of the link weight calculation is that of convex optimization (as opposed to NP-hard). However, albeit remaining as a hop-by-hop forwarding protocol, this approach requires that PEFT is used as the underlying routing scheme. Naturally, this is a disadvantage towards using a widely implemented and robust protocol such as OSPF.

2.1.3 Network Mappings

The results from last section's algorithms must be mapped to all network's nodes. In the case where the chosen optimization method relied on the setting of optimal link weights, this mapping must be done in a manual fashion, setting the appropriate link weight values at each router. However, if paths need to be set up in the network instead of link weights, there are protocols which can deal with this process autonomously.

The Label Distribution Protocol (LDP) [30] allows for the setting of MPLS label switched paths between network nodes, relying on the IP forwarding tables. Consequently, the nature of the chosen paths is in accordance to the underlying routing algorithm. The CR-LDP (Constraint Routing LDP) [27] builds upon the former protocol, enabling selection of paths other than the ones chosen by the routing protocol. This is achieved by supporting explicit route constraints (specifying all the hops that belong to a given path). CR-LDP thus allows the setting of paths in accordance to path selection algorithms earlier specified.

Furthermore, RSVP-TE [26], an extension of the Resource Reservation Protocol for Traffic Engineering in MPLS networks, allows the explicit setting of label switched paths. This is accomplished by incorporating a simple EXPLICIT_ROUTE object into RSVP Path messages. The EXPLICIT_ROUTE object comprises the hops of the intended path. Using this object, the paths taken by label-switched RSVP-MPLS flows can be pre-determined, independent of conventional IP routing.

Moreover, traffic engineering extensions to the OSPF routing protocol have been proposed as an RFC [29]. This RFC specifies the addition of traffic engineering information in link state advertisements (LSA's), which are then used to build an extended link state database. This traffic engineering database is used by each node for constraint-based source routing or for explicit route traffic engineering.

2.2 Dynamic Traffic Engineering

As opposed to its static counterpart, dynamic traffic engineering requires no previous estimation of the network traffic. Instead, approaches of this type adapt the routing of traffic flows on-the-fly, using measurements of the current (or recent) network conditions, such as packet loss or queueing delay.

Thus, dynamic traffic engineering schemes comprise solutions for two separate concerns. Firstly, *computation of network paths*. Solutions to this problem not only define the topology information maintained and exchanged at each node, but also constrain the possibilities for load balancing due to the nature of the computed paths. The latter possibilities influence the solutions for the second dynamic traffic engineering concern: the *distribution of traffic* amongst the available set of paths.

2.2.1 Path Computation

The problem of dynamically computing paths between network nodes has been of historical importance in the design of protocols for both interior gateway routing and local area network forwarding.

Developed solutions vary on several criteria. In the first place, the computational complexity of the path computation algorithm, related with the space and time overhead induced at each node. The required signaling (message exchanges between nodes), known as control complexity, further distinguishes the performance of the approaches. Above all, the convergence speed and the behaviour of the algorithm during topology changes (e.g.: due to node or link failure) have a decisive impact on its pratical applicability.

The next sections group some of the state-of-the-art approaches to this problem.

2.2.1.1 Spanning Tree

Spanning trees have been the legacy solution for computing paths and forwarding in Ethernet scenarios, namely through the use of the Spanning Tree Protocol (STP) [5]. STP defines a distributed approach to organize the network in a shortest-path tree, rooted at the bridge with the lowest identifier (defined by MAC address and bridge priority). This tree is built through the periodical exchange of BPDU messages. Additionally, forwarding is based on flooding on this tree, optimized by a filtering mechanism (built by learning through the reverse path of frames). Moreover, the STP protocol suffers from slow convergence in the case of topology changes, which effectively blocks forwarding for periods of time that may range up to one minute – far from optimal. The Rapid Spanning Tree Protocol (RSTP) [5] lowers the convergence times to roughly 1 second, thanks to changes made to BPDU message exchanges and the bridge port roles. Additionally, there is another interesting STP-based approach standardized, the Multiple Spanning Tree Protocol (MSTP) [4], providing for the creation of distinct multiple spanning tree instances per network VLAN, which allows for better use of network paths.

Another Ethernet-based approach, The Global Open Ethernet [31], or GOE for short, relies on a multiple spanning tree protocol (PD-MRSTP) to compute spanning tree instances rooted at each network edge bridge. On the other hand, backward compatibility is guaranteed by using Q-in-Q encapsulation of a special GOE tag, while simultaneously maintaining a legacy spanning tree to ensure communication between GOE and legacy bridges. This approach does not require filtering as GOE uses a sinktree (rooted at the destination) to forward unicast frames, whereas it selects a source-rooted tree to forward unknown or multicast traffic. This ensures shortest-path bridging for both unicast and multicast traffic.

2.2.1.2 Distance-Vector

Distance-vector schemes typically rely on a distributed Bellman-Ford graph algorithm, each node using partial knowledge of the network topology to compute the available paths. Several routing protocols take advantage of this approach, such as the historically known Routing

Information Protocol (RIP) [22]. Although providing ease of configuration, RIP (and most distance-vector protocols) react with instability after topology changes and suffer from poor convergence and scalability.

The Enhanced Interior Gateway Routing Protocol (EIGRP) [37], a proprietary solution from Cisco, is based on distance-vector routing protocol, albeit optimized in order to improve its scalability. The main optimization consists in the use of the Diffusing Update Algorithm (DUAL), a diffusing computation algorithm that avoids inconsistencies (loop-free routing) and reduces the convergence times.

Coupling both the spanning tree and distance vector concepts, STAR (short for Spanning Tree Alternate Routing [36]) provides an advanced bridge forwarding architecture, granting backward compatibility with legacy STP bridges by creating an overlay graph (with STAR-capable bridges only) on top of the shortest path tree. This approach, coupled with a distance-vector procedure, allows for the computation of shorter paths between the STAR bridges. However, due to the support for older bridges, STAR must use a bridge learning process analogous to that of STP.

2.2.1.3 Link-State

Link-state protocols enjoy wide-spread use as interior gateway routing protocols, namely the Open Shortest Path First (OSPF) [24] and the IS-IS [23]. These approaches rely on the propagation of a node's link state to all the other networks nodes, resulting in complete topology knowledge at each node. This allows for path computation using a graph algorithm such as Dijkstra's shortest path. Scalability can be achieved through network segmentation in order to reduce the signaling and increase reachability.

Link-state approaches have been proposed for layer 2 scenarios. Rapid Bridges (RBridges) [38] makes use of a link-state protocol to propagate the learned MAC address associations of a given bridge to the network, namely by the use of the IS-IS. Furthermore, each bridge learns the MAC addresses in its subnet using the ARP protocol. A form of encapsulation/network zone segmentation is used to prevent the MAC explosion issues. Naturally, the routing of unicast frames between RBridges relies on the information obtained via the routing protocol (with multi-paths to the same destination being available), whereas multicast or unknown traffic is propagated through a spanning-tree. Temporary inconsistencies are resolved in RBridges through the use of TTL-based approaches.

2.2.1.4 Diffusing Computation

The diffusing computation technique consists in starting a distributed computation at a given network node (known as the initiator) which spreads requests to its neighbours, which will recursively propagate the computation to the rest of the network, having a reliable termination criteria. The Smartbridge approach [39] uses diffusing computation to build the network topology, relying on epoch numbers to prevent the formation of transient loops. Notwithstanding, frame forwarding is blocked during topology acquisition, which spans tens of milliseconds.

The forwarding of unicast frames with known destinations (stored in a MAC address - segment table) is done through the shortest path, measured in number of hops. Otherwise, if the destination is unknown (or a multicast address), the frame is forwarded through a spanning tree rooted on the source segment, not the first bridge. Additionally, the convergence times facing a topology change are low (around 20 milliseconds), which is explained by the use of diffusing computation.

2.2.2 Traffic Distribution

The last section presented several approaches suitable for generating a set of paths for all source-destination pairs. Most of them additionally include procedures to select the path (from the available set) to a given flow. Typically this is done either by setting the path permanently at the ingress network node or distributing the path selection choice as the flow traverses the network nodes. The criteria used to select paths range from round-robin based decisions to approximations using measurements of current network conditions, such as load. In the latter case, the approaches further differ on the methods used to provide network load awareness for each node.

2.2.2.1 No Load Awareness: Round-robin

Round-robin strategies are employed in a variety of situations in order to achieve load balancing in a simple fashion. These include packet scheduling (round-robin scheduling) and equal cost multipath routing. The latter application to routing can be extended if not only equal cost paths are used to split traffic.

An approach of this type is the Valiant Load Balancing routing architecture [48]. VLB provides a suitable approach to distribute traffic onto a backbone network, based on the Valiant processor interconnection scheme [42]. The goal of this architecture is to provision the minimum amount of link capacity while assuring that all valid traffic matrices fit the network. For this purpose, VLB assumes the network comprises a full mesh of logical links between all n nodes, and that each node has a maximum ingress traffic rate of r.

VLB works by using a two-hop routing scheme: the ingress node forwards a flow to an intermediate node, chosen in a round-robin fashion from the set of all nodes, and then the intermediate node forwards it to the destination node. If each logical link is dimensioned with a capacity equal to $\frac{2*r}{n}$, then all traffic matrices fit the network.

On top of that, the Ethernet Load Balancing (ELB) [43] architecture builds upon VLB, splitting traffic in a round-robin fashion to spanning-trees rooted at the different network nodes. What is more, this approach still guarantees maximum throughput for valid traffic matrices, albeit requiring that each node stores information for all network spanning trees.

2.2.2.2 Load-aware Approaches

Round-robin approaches equally split traffic among a set of available paths. Another alternative consists in distributing flows in an informed way, using adaptive and optimized scheduling. Furthermore, the required information, typically concerning network traffic load, must be exchanged between network nodes.

One of such proposals is the Adaptive Multipath Routing (AMP) architecture [17], which attempts to extend the mapping of flows to any outgoing link which provides a smaller cost to the destination, thus increasing the size of the available path set. However, possible destinations do not share equal parts of the flows' load. AMP uses a non-conventional dynamic hashing method to split traffic amongst available paths. Hash space for each destination is adjusted having in mind a load-aware algorithm.

This algorithm takes into account the equivalent load measurement for all links of a given node, which is calculated from exchanged load statistics and traffic history at the node. The load signaling is done locally through neighbor exchanges of backpressure messages – BM's. A BM sent from node X to node Y via link *xy* notifies Y of its influence on the congestion situation at link *xy*, mapped to simple scalar value, reducing the signaling overhead. Finally, load balancing through available paths avoids network oscillation by introducing exponential shifting of flows – this mechanism works by switching a very small amount of traffic from congested paths at first and then increasing this amount exponentially as the path remains congested.

Accordingly, the REPLEX traffic engineering scheme [14] works by adjusting the weights of available outgoing links based on collected network statistics. The algorithm works by using a game theory model, observing the concept of Wardrop equilibrium. Analogously to a Nash equilibrium, at the Wardrop equilibrium, all agents (which can be seen as flows) would not gain performance by picking any other path than their current one. Not unlike AMP, the statistics, namely the expected latency for a link, are gathered using message exchanges between neighbour nodes in a distance-vector fashion. Network oscillation is avoided, both by increasing flow shifting probability as network load changes increase, as well as by fine tuning the weight shift factor and EMA weight parameters of the algorithm.

The MPLS adaptive traffic engineering (MATE) [13] method requires the existence of multiple virtual paths (label switched paths) between all pairs of nodes. These paths are used to define a traffic engineering model analogous to that seen in section 2.1. The resulting constrained optimization problem is solved using gradient projection, where routing is iteratively adjusted in the opposite direction of the defined gradient. Additionally, MATE uses probe packets in order to measure the delay and packet loss ratio of a given label switched path, which influence the path cost in the said optimization problem and therefore the traffic distribution among the available paths.

Likewise, an approach known as Routing with Potentials [8] works by viewing the entire network as a terrain, avoiding the obstacles caused by high congestion. These obstacles are mapped by networks zones with high potential, which the algorithm avoids by using the steepest-gradient method - forwarding to the network node that allows for a greater reduction

Scheme	Signaling Complexity	Spatial Complexity	Load Awareness
Flooding	High	High	Total
Aggregation	Low	Low	Partial
End-to-end probing	High	Low	Total

Table 2.1 Load signaling analysis

of the potential field. The potential at each node can be defined by a function that maps its distance to the destination: this method allows for simple shortest path routing. Moreover, the algorithm is provably loop-free and simulation results show that it remains stable under very rapid traffic load changes. Traffic-awareness is further achieved by combining the potential function with network load information, resulting in load balancing. The necessary load and routing information is propagated resorting to the use of a link-state protocol.

2.2.3 Load Signaling

Most traffic distribution approaches require that network nodes have information about the network load. However, load awareness has a significant toll in the overall network performance. This is due to the control message-exchanges necessary for load signaling.

One signaling method consists in using flooding to spread global link load information to all network nodes (e.g. used in Potentials routing). This method commonly suffers from rather poor scalability. This is explained by the number of messages greatly increasing in periods of high network load oscillation, in addition to the spatial overhead caused by storing complete load knowledge at each node.

Aggregation-based mechanisms attempt to minimise both problems. This is achieved by signaling aggregated amounts of load-information (such as the distance-vector propagated scalar values in the AMP architecture) and keeping partial load awareness at each node. Predictably, while solutions of this type improve scalability, they may lead to sub-optimal traffic distributions due to incomplete load information.

Finally, a common solution for load signaling used in schemes supported by explicit routing architectures (e.g. MATE) relies on end-to-end delay and packet loss measurements, also known as probing. The number of control messages required by this type of approaches is significantly high, although the spatial complexity at each node can be greatly reduced. Furthermore, the proposal stated in [34] optimises the number of needed measurements for these types of approaches, allowing for more scalable solutions. A similar alternative is computing end-to-end delay and packet loss using the current network traffic, e.g. by indirect estimation of the behaviour of TCP connections [49].

Table 2.1 comprises a summary of the presented load signaling analysis.

2.3 Other Related Work

The previous sections have described the static and dynamic approaches for traffic engineering. Nonetheless, there are different paradigms that attempt to optimize the use of network's resources.

First of all, some routing control can be shifted from the network's nodes to its end peers, achieving a degree of end-to-end load balancing. On top of that, a new logical network may be built on top of the underlying physical structure, optimizing routing and distributing the load between its peers.

The former approach is typically known as congestion-aware routing (or multipath TCP), whereas the latter relates to resilient overlay networks (RON's, for short). The next sections comprise their description.

2.3.1 Congestion-aware Routing

Earlier in this report, it has been seen that elastic TCP traffic poses important challenges to the design of traffic engineering approaches. It followed that this phenomenon is due to its congestion control mechanism, which coupled with multipath routing may lead to unexpected (and more often than not, underperforming) behaviour.

However, recent work [18] shows it is possible to alter the congestion control algorithm in order to retain stability when splitting a TCP flow to multiple network paths. The considered model assumes that either end nodes are either multi-homed by two or more ISP's or participate in an overlay network. Furthermore, the results of the cited work prove that a system working under this criteria holds stability when the average load through each link remains under the link's capacity, that is to say, below the congestion threshold.

Moreover, the Distributed Adaptive Traffic Engineering (DATE) approach [19] attempts to consider both the congestion control behaviour of TCP and traffic engineering requirements, offering not only stability but also optimality. Although dynamically adapting the percentage of traffic that follows each network path (which requires measurements of link utilisation statistics using the SNMP protocol), DATE still requires that all paths are previously computed in the network (such as the previously seen MATE proposal). Finally, congestion-aware routing is performed by keeping track of TCP sessions at edge nodes and adjusting their routing accordingly.

These solutions follow a recent trend in network architecture and routing research, which more and more values the availability of several end-to-end paths from source to destination, aiming to grant end-systems the ability to choose routing paths. This scenario is increasingly common since devices with two or more distinct connection interfaces are now in widespread use (e.g.: cellphones with both a regular GSM antenna and an external 3G Internet device).

2.3.2 Resilient Overlay Networks

Overlay networks are a common solution when some degree of routing control or isolation needs to be achieved among a set of participants. Their use has grown accordingly with the growth of global Internet applications, such as file sharing systems (e.g.: gnutella, kazaa) or communication software (e.g.: skype).

The case for resilient overlay networks [6] proposes a somewhat more ambitious set of goals: recover from node failures and periods of degraded performance in shorter time than traditional IP routing algorithms (with convergence times lowering from tens of seconds to seconds). RON's offer control over the selected network paths using client specified metrics, which further allows advanced routing schemes (such as two-hop schemes similar to Valiant Load Balancing, seen in section 2.2.2).

One of the most significant drawbacks of RON's use is their low scalability due its inherent full mesh connectivity, often being limited to few more than 50 participant nodes. Nevertheless, the proposal presented in [40] significantly lowers the amount of signaling required to compute paths in a RON (the per-node communication is reduced from $O(n^2)$ to $O(n^{1.5})$). This routing approach limits the propagation of link-state updates to subsets of nodes while simultaneously being provably optimal.

2.4 Summary

This chapter described state-of-the-art approaches for both static and dynamic traffic engineering, as well as introduced related work in congestion-aware routing and resilient overlay networks.

Static traffic engineering was shown to comprise traffic estimation techniques and optimisation algorithms. The former included statistical tomographic methods and optimisation-based tomographic methods, whereas the latter included path selection and link weight manipulation algorithms.

Several optimisation metrics were presented. From these, the congestion costs appears to be the most suitable metric by accounting the convex impact of link utilisation on the network load. Furthermore, two or more optimisation metrics can be combined to achieve further control over the traffic distribution.

Finally, static traffic engineering solutions can be implemented using explicit routing architectures such as MPLS, using RSVP-TE as the signaling protocol to setup the desired network paths. Another approach relies on using the traffic engineering extensions to the OSPF routing protocol (OSPF-TE).

In the previous sections, several state-of-the-art approaches for dynamic traffic engineering have been presented. Three separate concerns were identified: path computation, traffic distribution and load signaling.

It has been seen that a surplus of effective (in regard to the set of computed paths and behaviour during network reconfigurations) solutions exist for the first problem, either by using novel spanning-tree methods, conventional routing protocols (link-state or distance-vector) or diffusing computation. What is more, dynamic traffic engineering schemes such as MATE can rely on statically computed network routes.

Assuming that load signaling is done as efficiently and optimally as possible, the distribution of traffic becomes the vital problem when designing suitable dynamic traffic engineering schemes.

Section 2.2.2 presented five state-of-the-art approaches with this aim, the first of which was ELB. ELB's reliance on a round-robin distribution unaffected by the network load grants it perfect stability and independence of load signaling methods. Additionally, ELB can be implemented on top of current spanning tree protocols developed for Ethernet scenarios, achieving very good compatibility. However, the use of longer than the shortest network paths, particularly during periods of low congestion, leads to sub-optimal routing.

On the other hand, the AMP architecture uses partial load awareness to adapt routing, achieving a near-optimal distribution of traffic. In addition, the aforementioned exponential shifting mechanism makes AMP stable enough under load oscillation. That being said, AMP requires the adoption of a new routing architecture in order to support its load signaling and forwarding processes.

REPLEX routing can be run on top of both hop-by-hop or explicit routing path computation architectures, but requires nonetheless a distance-vector signaling protocol to grant its load awareness, reducing its compatibility. Its wardrop equilibrium model coupled with the available parametrization make REPLEX very stable during load changes, and it provides near-optimal results by adjusting network link weights.

Deriving from the gradient projection algorithm used in the scheme, optimality and stability can be proven for MATE under the assumptions seen in [13]. Nonetheless, in spite of its signaling and routing mechanisms being fully supported in MPLS-based network, the traffic distribution model is highly complex and unfeasible for practical implementation.

The routing with potentials is optimal, its criteria relying on the metrics being used to define the traffic potential. Furthermore, it is provably stable assuming that the mean packet propagation delay within a routing domain is smaller than the rate at which the traffic potential surface changes within a domain [8]. Even so, routing with potentials requires both a new signaling and forwarding architecture, rendering it incompatible with current network standards.

Table 2.2 summarises the presented analysis.

Unlike any of the presented dynamic traffic engineering alternatives, the SSD-TE framework presented in this dissertation attempts to address both stability and implementation compatibility concerns, whist remaining very close to optimal and responsive to network load variations.

Scheme	Optimality	Stability	Compatibility
ELB	Not Optimal	Stable	High
AMP	Near Optimal	Stable	Low
REPLEX	Near Optimal	Stable	Low
MATE	Optimal	Stable	Low
Potentials	Optimal	Stable	Low

Table 2.2 Summary of dynamic traffic engineering schemes

The next chapter of the dissertation goes on to present the framework, introducing its underlying philosophy, as well as a description of its architecture and implementation.

3 . Traffic Engineering Framework

This chapter describes the traffic engineering framework which this dissertation proposes. After a brief introduction, along with the motivations and assumptions which steered its design, the several components of the framework are detailed. Finally, the implementation requirements of the framework are discussed.

3.1 Introduction

The naïve way to run a successful network mixes shortest path routing with overprovision. With such network, by construction, all packets get to the destination in the shortest possible delay. Unfortunately, in large networks this method is rather uneconomical and cannot be used by a competitive provider. The next empirical alternative can be to iteratively adjust network capacity to the observed load, and fine tune routing in a way that the available paths are better explored. A more sophisticated alternative, quite suitable when the network load is stable and predictable, is to use static traffic engineering methods to provision the network and fine tune the load distribution.

From the simple naïve alternative to that based on static traffic engineering, complexity of network operations has grown dramatically, which in turn increases its operational cost. Moreover, real provider networks are often not stable at all, due to flash crowds phenomena and variations related with the evolution of costumers habits and the popularity of applications and sites [14]. Thus, in practice, complex networks operations go hand in hand with overprovision due to load variations, flash crowds and the difficulty to measure or quickly adapt to the expected load. Very small and very large providers live quite well at the two extremes of the presented spectrum. However, this state of affairs is cumbersome in medium backbone networks.

A simple ideal way to operate a network could consist in provisioning it in a way that shortest path routing of the "commonly expected load" brings no congestion or stress to the network. However, whenever needed, the network should automatically resort to some form of load distribution of any (un) expected surplus load, using not only an already provisioned spare capacity, but also using any available capacity in whatever path could be used to route packets to their destination.

In this chapter, the Simple and Stable Dynamic Traffic Engineering framework is presented in detail. SSD-TE aims to support the provisioning and routing control of an Ethernet-based provider backbone close to the ideal simple way introduced above. The chapter starts by giving a bird's-eye view of the framework. As any other traffic engineering framework, SSD-TE comprises a provisioning method, routing algorithms and a network status monitoring mechanism.

With SSD-TE, a given network is provisioned in a two-step way. First, an estimated traffic matrix (ETM) should be defined in a way that all packets belonging to traffic fitting in ETM must be routed by shortest paths and with no losses. ETM can be seen as the fraction of guaranteed

high quality traffic the network should be able to transport. The actual fraction of the real expected traffic that will fit in ETM is an operator option, e.g. the high quality operator will define ETM in a way that almost all traffic at all times will fit in ETM.

ETM allows a first computation of the ingress traffic in each network ingress node. In accordance to SSD-TE, the operator is then required to define an incoming traffic surplus for each ingress node. Given ETM and the surpluses per ingress node, SSD-TE provides tools allowing the operator to compute the capacity of all links in the network in a way that the routing and load distribution method used by a SSD-TE managed network will guarantee the following two conditions:

1. At any moment, the rate of the packets flowing from any ingress node to any egress node by the shortest path and without loss, is grater or equal to the one that is deemed to occur according to ETM for the same ingress - egress pair;

2. If the packet incoming rate at any ingress node do not violate the rate foreseen in ETM plus the surplus rate anticipated for that ingress node, all packets entering the network will be routed to the egress node without loss, by one the available paths (not necessarily the shortest one).

To achieve these guarantees, SSD-TE uses shortest path routing at the base, complemented with load distribution according to the ELB algorithm, presented in section 2.2.2.1. In fact, an SSD-TE engineered network by default routes packets by the shortest path, but continuously evaluates the state of packet transport in each ingress / egress nodes pair, e.g. (n_i, n_j). Whenever some form of congestion is deemed to occur from n_i to n_j "surplus" packets will be forwarded using ELB. These monitoring and forwarding methods, complemented with the SSD-TE provided provisioning tools, guarantee that the above conditions will be respected.

After this preliminary presentation of SSD-TE we will present the framework in a more formal way in next section. After that, the remainder of the chapter will elaborate on several aspects comprising the framework, hypothesis will be clarified and formal definitions and proofs will be presented.

Several aspects are out of scope of the discussion in this chapter, namely the occurrence of node and link faults and a more elaborate characterization of network traffic.

3.2 Architecture

The SSD-TE framework has the objective of optimising the traffic distribution in a network. The framework comprises three components: a provisioning method, a forwarding scheme and a traffic monitoring and distribution adjustment algorithm.

The first component provides a *method which assigns the link capacities* of a network. To this end, it requires both the network topology and an estimated traffic matrix. The topology contains the network's node and link sets, each link being weighted by its respective latency.

On the other hand, the ETM traffic matrix comprises the ETM rate for each distinct *ingress node – egress node* pair, which from hereafter will be called *flow*. The ETM rate of a given flow f, with ingress node n_i and egress node n_j, describes a traffic rate r_f such that the provisioning method assures that at least r_f traffic entering the network at node n_i can be forwarded through the shortest path to node n_j without packet loss.

Thus, the method assigns enough capacity per link to fully accommodate this matrix, assuming that the traffic is forwarded by the shortest path. Based on the traffic ingress surpluses, the method assigns extra capacity to the links in order to tolerate deviations to the ETM matrix.

Secondly, the *forwarding scheme* specifies the paths followed by traffic in the network. This scheme distributes network traffic either through the shortest path between nodes or using the ELB algorithm [43], presented in chapter 2. This distribution is done on a per-flow basis, i.e. the traffic rates routed by either approach are defined for each network flow.

Moreover, using this forwarding scheme with a network provisioned with the first component ensures the two guarantees specified in the previous section. This is equivalent to say that any flow which actual rate does not exceed its ETM rate plus its surplus rate fits the network. What is more, the traffic rate up to the ETM rate is routed by shortest path, whereas the surplus rate is distributed by the ELB algorithm.

The forwarding scheme is fully *compatible* with state-of-the-art Ethernet solutions and current standards. This comes from the fact that it relies on shortest path routing and traffic distribution among a set of spanning trees (used by ELB), both widely known and implemented in Ethernet scenarios. Furthermore, section 3.7 provides more details on how to implement the scheme.

Finally, the third component is a *traffic monitoring and distribution adjustment algorithm*, which dynamically changes the flow traffic rates used by the previous component in order to reach *optimality*. The optimality criteria for this algorithm can be loosely defined by stating that, in an optimal distribution, no more network traffic can be routed by the shortest path without causing congestion. As a result, this criteria leads to the minimisation of end-to-end delay while simultaneously avoiding packet loss. The component relies on a distributed optimisation algorithm, known as the gradient projection algorithm, in order to optimise the traffic distribution for the said criteria. Furthermore, this algorithm requires that each ingress node is aware of network traffic on the paths used by its flows. Such load awareness is granted by end-to-end signaling messages.

SSD-TE's traffic adjustment algorithm is rather important in two different scenarios. In the first, the network traffic matrix changes significantly and unexpectedly from the estimate used in the provisioning method, for instance, during flash crowds. When this network behaviour occurs, it is unlikely that the distribution used by the forwarding scheme is well suited to the new traffic matrix. As a result, packet loss is bound to occur. In the second scenario, the forwarding scheme is being used on a network that was not provisioned using the respective framework component. Consequently, the appropriate traffic distribution rates can only be calculated by the adjustment algorithm, since no estimate traffic matrix exists.

Furthermore, as one of the goals of the proposed framework is *stability*, and that, as stated

in chapter 1, dynamic load-aware forwarding algorithms are more often than not causes of network instability. The SSD-TE forwarding scheme uses both shortest path routing and the ELB algorithm, which are both stable types of forwarding if used standalone. On the one hand, ELB grants perfect routing stability when distributing a certain amount of traffic. This stability comes from the fact that the traffic is distributed among all source trees in the network, using a weighed round-robin distribution. On the other hand, shortest path routing is inherently stable in the forwarding scheme, because only one shortest path per network flow (an ingress-egress node pair) is considered.

Nevertheless, using two stable types of routing does not necessarily imply that SSD-TE is a stable framework. The variations in the traffic rates assigned to each approach might be a cause of instability, causing oscillations and underperformance.

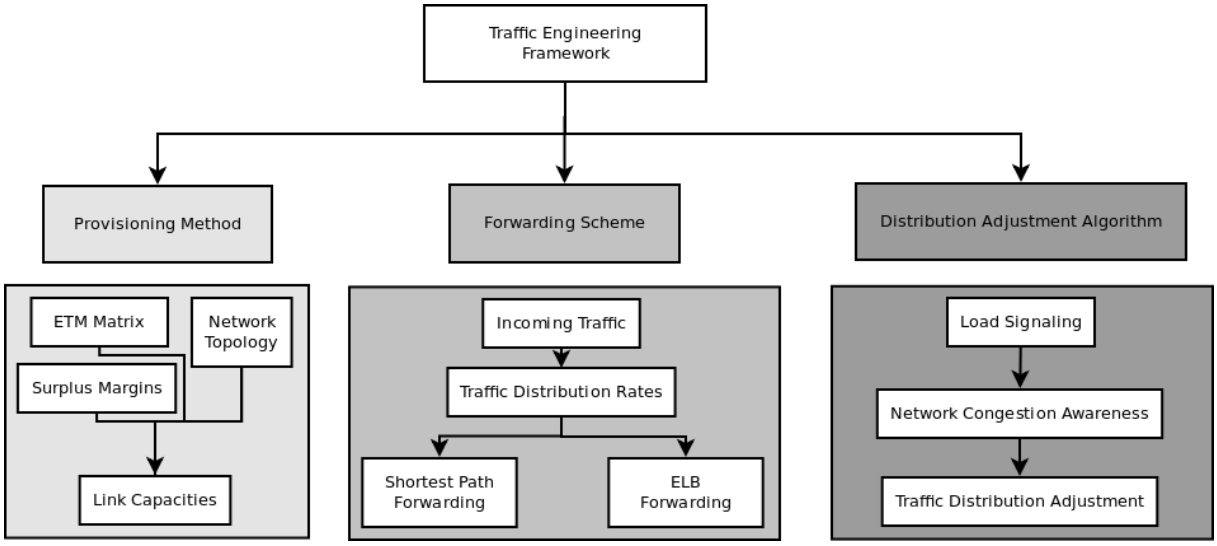

Figure 3.1 Traffic Engineering Framework Diagram

However, the traffic rate distribution is dynamically adjusted by the last component, which is deemed to accomplish a degree of stability. This claim is supported by the proof that the algorithm converges to optimality, presented in section 3.6.7, provided that the traffic matrix remains constant for a period of time. Moreover, the convergence speed and interval between rate adjustments can be parametrised, which increases the forwarding scheme's potential stability.

What is more, the last component's signaling mechanism is sufficiently scalable for provider network implementation. The signaling complexity amounts to $O(n)$ messages per node, where n is the number of network nodes. The required spatial complexity per node is $O(n^2)$, which is also deemed to be feasible.

The diagram in figure 3.1 comprises a visual summary of the traffic engineering framework. The following section presents the assumptions regarding the network and routing schemes which had to be made to guarantee the presented properties of the framework.

3.3 Preliminary Assumptions and Notation

The following assumptions are held throughout the remainder of the chapter:

1. Multicast traffic and node/link failures are out of scope, as discussed in chapter 1. Thus the network configuration is stable and only unicast traffic is transported.

2. When entering the network, the inter-arrival time of traffic pertaining to a given flow follows a constant distribution. That is to say, the time between the arrival of each frame of a particular flow at its ingress node is fixed. Thus assumption is taken in order to facilitate analytical proofs and results. In the validation chapter, which is based on simulation, a more realistic traffic model is used.

Furthermore, consider the following variables:

N and n	the node set and its size, respectively.
L and c_l	the set of links and the capacity of link l, respectively.
S	the set of flows, each comprised of a distinct ingress-egress node pair.
r_s	the ETM traffic matrix rate for flow s.
η_i	the ingress traffic surplus margin specified for node i.

The next sections go on to describe the various framework components.

3.4 Provisioning Method

This section introduces a provisioning method for the assignment of link capacities. This model assures that any traffic matrix under certain assumptions, and using a routing approach that makes a compromise between shortest path forwarding and ELB, fits the network. First, consider the following properties of ELB.

3.4.1 ELB Properties

Path Length An interesting property of ELB is that it may achieve equal or lower delays than Valiant Load Balancing (VLB – introduced in section 2.2.2.1), since the length of its network paths are shorter or equal to those of VLB. Let sp_{ij} designate the shortest path between nodes i and j in the network graph, whereas tp_{ijr} specifies the path between nodes i and j in the network shortest paths tree with root r. Furthermore, $l(p)$ is the length of path p.

From graph theory, the following property holds,

$$\forall i, j, r \in N : l(tp_{ijr}) \leq l(sp_{ir}) + l(sp_{rj}) \tag{3.1}$$

This property states that, using a shortest-paths tree with root r, the shortest path between nodes i and j has a length shorter or equal to that of a path between i and j traversing the root.

In other words, only one loop free path exists between any two distinct nodes, for any tree graph. Moreover, there is only one path between two distinct nodes which traverses the root. It follows that this path is either the shortest or it contains a loop. The latter case implies that the path is necessarily lengthier than the loop free path between the nodes.

Consider therefore, that each VLB path between the end nodes consists of a distinct VLB logical link pair, whereas ELB uses a direct path on a tree rooted at the intermediate VLB node. A VLB logical link is mapped by a shortest path between the end node and the intermediate node. It then follows, using the above property, that any ELB tree path has a length less or equal to the corresponding VLB path.

Figure 3.2 shows an example of the difference between ELB and VLB paths.

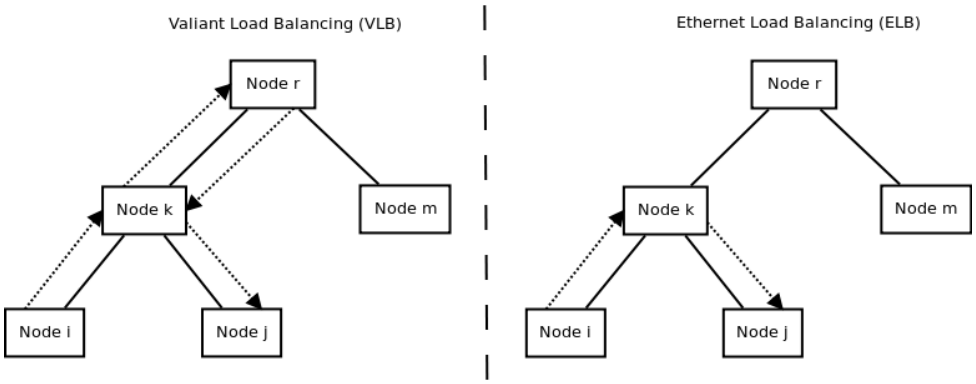

Figure 3.2 The VLB and ELB paths between nodes i and j, using node r as the intermediate hop and tree root, respectively

Capacity Provisioning Complementing the earlier property, the capacity required by the VLB scheme is an upper bound of that required by ELB. This claim is supported as follows.

Considering that VLB logical links between two network nodes follow the shortest path between them, it follows that the VLB scheme can be mapped onto an modified ELB scheme – *mELB*.

This mapping can be achieved by specifying that *mELB* tree paths traverse the tree root. Effectively, this means that a tree path between two nodes may cross one (or more) links more than once in order to reach the root of the tree.

Thus, a VLB two-hop path from node i to node j, using r as the intermediate node, is directly mapped to an *mELB* r-rooted tree path from i to j. Specifically, we can split the *mELB* path in two: the ascending path a_{ir} from the source node i to the tree root r and the descending path d_{rj} from the tree root r to the target node j. As said earlier, the intersection of the link sets of a_{ir} and d_{rj} may not be empty.

From the earlier section, it also follows that the load balancing strategy used by the two schemes is identical. Since the network paths and traffic distribution are identical, it follows that *mELB* and VLB require the same link capacity provisioning. Recall that, for a network with n nodes and a maximum ingress rate θ per node, VLB – and therefore, *mELB* – requires a full mesh of logical links with capacity $\frac{2\theta}{n}$.

Finally, the concept of ascending and descending paths of *mELB* is applied to ELB. ELB can map a VLB two-hop path from node i to node j, using r as the intermediate node, and distribute traffic in a similar fashion. This is done by routing traffic up the path a_{ir} and then down the path d_{rj}. As said earlier, the intersection of the link sets of a_{ir} and d_{ir} may not be empty. Unlike *mELB* however, the intersected links are not traversed in ELB, meaning that the switch from the ascending to the descending path may be done in a node other than the tree root. This difference only implies that part of the provisioned paths are unused; the required capacity per path is the same as in VLB and *mELB*.

3.4.2 Capacity Assignment

The provisioning method works by assigning enough capacity to network links in order to ensure that the ETM matrix fits the network. Furthermore, it considers that each node must handle a surplus ingress traffic rate of η_i.

To this end, the capacity assignment is done in the following manner:

1. Provision enough capacity to forward the ETM matrix by the shortest path.
2. Provision enough capacity to forward the surplus ingress rates η_i , $i \in N$ using the ELB algorithm.

For the first part, the method needs to guarantee that each flow "reserves" a capacity equal to its ETM rate in all links of its shortest path. That is to say, the capacity of a link is equal to the sum of the ETM rates for all flows that traverse it.

$$c_l = \sum_{l \in shortestPath(s)} r_s \tag{3.2}$$

For the second part however, both VLB and ELB assume that the maximum ingress rate to be balanced is equal for all network nodes, in order to guarantee support for all valid traffic matrices. Thus, the capacity assignment and load balancing used by the ELB scheme must be adapted to be able to support different added traffic margins per node.

One non-disruptive method for this adaptation is the Gravity Full Mesh scheme [48]. Proposed for VLB, it considers that the access capacity per node is variable. Consider first that $R = \sum_{i \in N} \eta_i$ is the sum of all access capacities. A small enough network constant η can be specified, so that for any node $i \in N$, we have an integer value k_i that makes the following equation hold true:

$$\eta_i = k_i \times \eta \tag{3.3}$$

It follows that the physical node i can be treated as k_i logical nodes, each with access capacity η. Then, the total amount of logical nodes in the network can be given by:

$$M = \sum_{i \in N} k_i = \frac{R}{\eta} \tag{3.4}$$

For the VLB or ELB schemes to properly distribute traffic, a full mesh of logical links between all logical nodes should exist, with capacity equal to $\frac{2\times\eta}{M}$. Therefore, between physical nodes i and j, we have $k_i \times k_j$ logical links, each with capacity $\frac{2\times\eta}{M}$. Therefore, the capacity c_{ij} of the logical link between physical nodes i and j is given by:

$$c_{ij} = \frac{2\,\eta}{M} \times k_i\,k_j = \frac{2\,\eta^2}{R} \times k_i\,k_j = \frac{2\,\eta_i\,\eta_j}{R} \tag{3.5}$$

Figure 3.3 presents a diagram of this approach. As opposed to VLB or ELB however, the gravity full mesh scheme does not evenly distribute traffic amongst intermediate nodes or trees, respectively. Instead, a network node balances a fraction $\frac{\eta_j}{R}$ of its traffic rate to node j.

All in all, the total provisioning of a physical network link is given by the sum of two different amounts.

First, the aggregate estimated ingress rate of all flows which shortest path traverse the link. This guarantees that the estimated traffic matrix fits the network using shortest path forwarding.

Second, the total ELB logical link rate of all logical links that traverse the physical link. In conclusion, the following equation gives the total capacity of physical link l.

$$c_l = \sum_{l \in shortestPath(s)} r_s + \sum_{l \in logicalPath(i,j)} \frac{2\,\eta_i\,\eta_j}{R} \tag{3.6}$$

Furthermore, since the logical path between i and j is equivalent to the shortest path of the correspondent flow s with ingress node i and egress node j, it follows that:

$$c_l = \sum_{l \in shortestPath(i,j)} r_{ij} + \frac{2\,\eta_i\,\eta_j}{R} \tag{3.7}$$

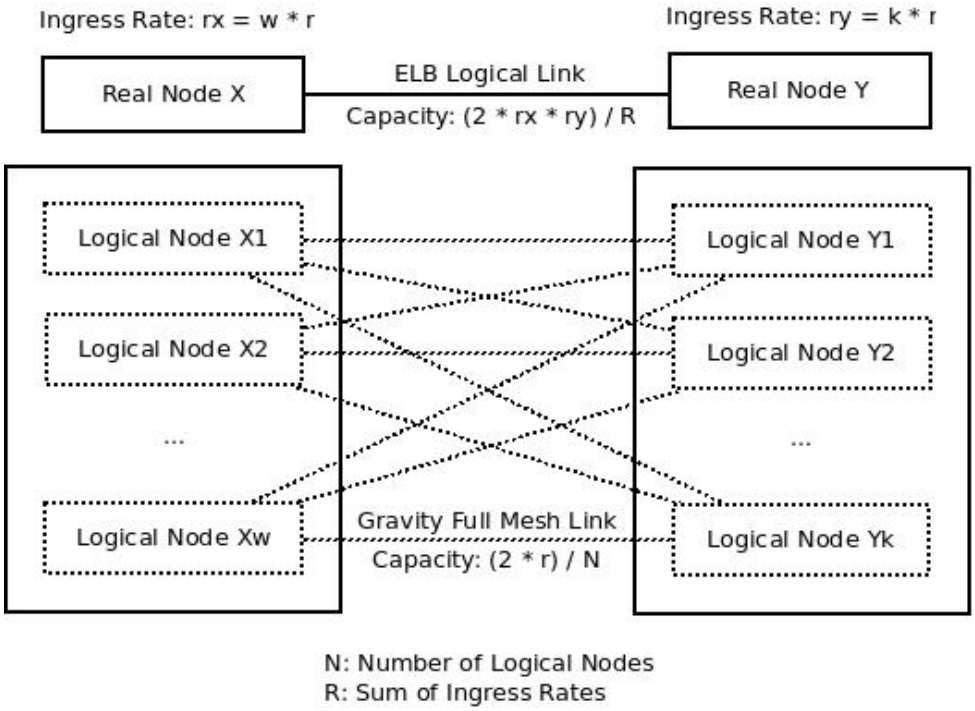

Figure 3.3 Gravity Full Mesh Diagram

3.5 Forwarding Scheme

The forwarding scheme of the framework specifies two distinct rates per flow s, x_{sa} and x_{sb}. These rates indicate the amount of traffic being forwarded by the shortest path and by ELB, respectively.

The traffic forwarded by ELB is split in a weighted round robin fashion among all network shortest-path trees. Specifically, for a flow s with ingress node i and egress node j, the amount of traffic being distributed onto the ELB path with root r is equal to $\frac{x_{sb} \times \eta_j}{R}$.

The x_{sa} and x_{sb} rates can be set either statically or dynamically. The former (static) approach is possible if used with a network provisioned with the framework's method. Let r'_s designate the current ingress rate of flow s. At the ingress node i of s, a traffic rate up to r_s is routed through the shortest path to its egress node. The remaining traffic of the flow, with a rate up to a maximum of η_i, is routed using the gravity full mesh ELB approach. Formally, this means:

$$x_{sa} = \begin{cases} r'_s & r'_s \leq r_s \\ r_s & r_s < r'_s \leq r_s + \eta_i \end{cases} \tag{3.8}$$

$$x_{sb} = \begin{cases} 0 & r'_s \leq r_s \\ r'_s - r_s & r_s < r'_s \leq r_s + \eta_i \end{cases} \tag{3.9}$$

The dynamic approach to set the rates relies on the framework's distributed rate adjustment component, which is described in the next section.

3.6 Traffic Distribution Adjustment Algorithm

The previous section showed a possible static setting of the framework's traffic distribution rates. However, this distribution might not be optimal depending on the network current traffic matrix. In other words, the amount of traffic routed by the shortest path can be superior to the rates set by the provisioning method, without causing the traffic matrix not to fit the network. Specifically, this amount of traffic can be computed by solving the following linear programming model.

3.6.1 Centralised Optimality Model

The objective of this model is to minimise the total amount of network traffic that is distributed using ELB without overflowing any links.

Sets and Parameters

- n the number of nodes / load balancing trees.
- S the set of flows.
- L the set of links.
- r_s the current ingress rate of flow s.
- c_l the capacity of link l.
- k_{sb} the number of load balancing trees such that their path for flow s traverses link l.

Variables

- x_{sa} the rate of traffic from flow s being routed on the shortest path.
- x_{sb} the rate of traffic from flow s being distributed among all load balancing trees.

Objective Function

$$Min \sum_{s \in S} x_{sb} \tag{3.10}$$

Constraints

$$x_{sa} + x_{sb} = r_s, \forall s \in S \tag{3.11}$$

$$\sum_{sa \in l} x_{sa} + \sum_{sb \in l} k_{sb} \cdot \frac{2 \cdot x_{sb}}{n} \leq c_l, \forall l \in L \tag{3.12}$$

$$x_{sa}, x_{sb} \geq 0, \forall s \in S \tag{3.13}$$

As it can be seen from (3.11), this model needs information about all the current flow rates. As a result, it needs to be solved in a centralised fashion which is not feasible for the goals set earlier. However, it is possible to define an alternative model, suitable for distributed optimization of the flow routing rates.

3.6.2 Distributed Optimality Model

The model defines the minimization of a continuous convex cost function in order to optimize the flow routing rates.

Sets and Parameters

$\alpha \in [1, \infty[$	a constant value that increases the priority of shortest path forwarding over load balancing.
γ	a stepsize sufficiently small to adjust traffic rates.
ε	a sufficiently small value to ensure convergence.
p_l	an estimate of the propagation delay at link l.

Variables In addition to the previous model variables:

- x_l – the rate of traffic at link l.
- $x_s = (x_{sa}, x_{sb})$ – the pair comprising both traffic rates (shortest path and tree distributed) of flow s.
- $x = (x_{sa}, x_{sb}, s \in S)$ – the vector with all traffic rates.

Cost Function

$$C_l(x_l) = \frac{p_l}{\alpha (c_l - x_l)} \tag{3.14}$$

Objective Function

$$Min\, C(x) = \sum_{l \in L} C_l(x_l) \tag{3.15}$$

Constraints

$$x_{sa} + x_{sb} = r_s, \forall s \in S \tag{3.16}$$

$$x_{sa}, x_{sb} \geq 0, \forall s \in S \tag{3.17}$$

This model can be solved by the gradient projection algorithm used by the MATE approach [13]. This algorithm can be executed asynchronously without node coordination. It works by iteratively modifying the traffic rates per flow. These modifications are processed in the opposite direction of the gradient of the cost function $C(x)$, which is described in section 3.6.4. The new traffic rates are then projected onto the feasible space (which is defined by the specified constraints).

3.6.3 Gradient Projection Algorithm

Let $\nabla C(i)$ designate the gradient vector of function C in respect to the model variables $x_{sa}, x_{sb} \in S$. Let i designate an iteration number and $[k]^+$ the projection of a vector k onto the feasible space. An iteration is done by reducing each traffic rate in x by a calculated amount and then projecting the resulting vector onto the feasible space. This amount is obtained by multiplying the respective first derivative in $\nabla C(i)$ by the stepsize γ. The following equations summarise the described iteration:

$$\nabla C(i) = \left(\frac{\partial C}{\partial x_{sa}}, \frac{\partial C}{\partial x_{sb}}, s \in S \right) \tag{3.18}$$

$$x(i+1) = [x(i) - \gamma \nabla C(i)]^+ \tag{3.19}$$

The algorithm stops when no significant change has been made to the traffic rate values. This is equivalent to say:

$$\| x(i+1) - x(i) \| < \varepsilon \tag{3.20}$$

Furthermore, as specified in [12], the derivative of the objective function in respect to a path traffic rate, $x_{sp}, s \in S$, is equal to the sum of the first derivative lengths of the links it traverses. That is to say:

$$\frac{\partial C}{\partial x_{sp}} = \sum_{l \in p} C_l'(x_l) \tag{3.21}$$

The extended traffic engineering model comprises two distinct *path* traffic rates per flow s: the shortest path rate x_{sa} and the distribution trees rate x_{sb}. However, the latter rate does not

follow a mere network path; in fact, the distribution path *sb* can be seen as a logical path that comprises all the distinct tree paths it uses.

Therefore, the derivative of the objective function in respect to a tree path traffic rate is given by the an average of the first derivative length of all the tree paths. This average is weighed by the fractions $\frac{\eta_i}{R}$, $s = (i, j)$, given by the provisioning algorithm:

$$\frac{\partial C}{\partial x_{sa}} = \sum_{l \in shortestPath(i,j)} C'_l(x_l) \tag{3.22}$$

$$\frac{\partial C}{\partial x_{sb}} = \sum_{treePath_k \in sb} \left[\frac{\eta_k}{R} \times \sum_{l \in treePath_k} C'_l(x_l) \right] \tag{3.23}$$

Note that $treePath_k$ designates the path for flow s on the shortest path spanning tree with root k.

3.6.4 Cost Function

The function C_l has the main objective of steering the traffic distribution towards a desired state by attributing a cost value to a given link l. With this goal in mind, consider the expected waiting time of a M/M/1 queue, where the service rate is c_l and the arrival rate is x_l, is given by:

$$t = \frac{1}{c_l - x_l} \tag{3.24}$$

Considering that, as seen before, c_l is the capacity of link l and x_l the current traffic rate at the said link, this waiting time t represents an approximation of the queuing delay at link l. Consider that the estimate of the propagation delay p_l of link l further influences the link delay. We can derive a convex cost function C_l for link l as follows:

$$C_l(x_l) = t * p_l = \frac{p_l}{c_l - x_l} \tag{3.25}$$

This approach yields the lowest values of $C_l(x_l)$ for short (in terms of propagation delay) and uncongested links. In turn, this means the algorithm steers the traffic distribution towards shortest path forwarding, whenever the network is not congested. However, links approaching their capacity return increasing values of $C_l(x_l)$, which makes sure that no link is overrun.

Control of the congestion sensitiveness of $C_l(x_l)$ can be achieved by multiplying the denominator of the function by the α parameter, as specified below.

$$C_l(x_l) = \frac{p_l}{\alpha\,(c_l - x_l)} \tag{3.26}$$

Higher values of α increase the concavity of the function, which in turn delays the reaction to link congestion. In practice, this means further preference of shortest path forwarding over load balancing.

3.6.5 Distributed Optimization Algorithm

Since the derivatives per flow only require information regarding the links their respective paths traverse, it is possible to solve the presented model distributively by each ingress-egress node pair. According to this observation, the iteration per flow becomes:

$$\nabla C_s(i) = \left(\frac{\partial C}{\partial x_{sa}}, \frac{\partial C}{\partial x_{sb}} \right) \tag{3.27}$$

$$x_s(i+1) = [x_s(i) - \gamma \nabla C_s(i)]^+ \tag{3.28}$$

However, since in a distributed model the network traffic rates cannot be instanly updated and coordinated among nodes, the model is extended to take this into account. The following procedure is identical to that of MATE, of which some details are omitted.

Let $T_s \subseteq \{1, 2, \cdots\}$ be a set of times at which the flow s has its rates updated at its ingress node. In addition, let $\lambda_s(t)$ be the vector comprising the estimate of the first derivative lengths of paths sa and sb. At a time $t \in T_s$, the iteration is expressed as:

$$\lambda_s(t) = (\lambda_{sb}, \lambda_{sb}) \tag{3.29}$$

$$x_s(i+1) = [x_s(i) - \gamma \nabla C_s(i)]^+ \tag{3.30}$$

The first derivative lengths λ_{sa} and λ_{sb} are computed by calculating a weighted average over the current and past values of $C'_l(x^t_l(t))$, where $x^t_l(t)$ is itself a weighted average of the values of x_l up to time t. Note that only the set of the k latest values of x^t_l are used in the computation of $C'_l(x^t_l(t))$ and that the weight vectors are respectively $cw_{sa}(t')$ and $cw_{sb}(t')$. Formally, this means:

$$\lambda_{sa} = \sum_{t' \in \{t'_1, \ldots, t'_k, t\}} \left[\sum_{l \in sa} cw_{sa}(t') \cdot C'_l \left(x^t_l(t') \right) \right] \tag{3.31}$$

$$\lambda_{sb} = \sum_{t' \in \{t'_1, \ldots, t'_k, t\}} \left[cw_{sb}(t') \cdot \frac{2}{n} \cdot \sum_{treepath \in sb} \sum_{l \in treepath} C'_l \left(x^t_l(t') \right) \right] \tag{3.32}$$

3.6.6 State and Cost Estimation

Naturally, older values of C'_l must be maintained for each flow at each ingress node. Furthermore, each node is responsible for $n-1$ flows, each having two distribution paths (either shortest

path or tree balancing). Consider that for a node x, d_{xsa} is the average number of links traversed by its shortest paths' to every destination and d_{xsb} is the average number of links traversed by its tree paths to every destination. Thus, it follows that its spatial complexity O_x, as a function of d_{xsa}, d_{xsb}, k and n, is:

$$O_{xsa} = k\, d_{xsa} \tag{3.33}$$
$$O_{xsb} = k\, n\, d_{xsb} \tag{3.34}$$
$$O_x = (n-1)\,(O_{xsa} + O_{xsb}) \Leftrightarrow \tag{3.35}$$
$$O_x = k\,(n-1)\,(d_{xsa} + n\, d_{xsb}) \Leftrightarrow \tag{3.36}$$
$$O_x = k\, d_{xsa}\,(n-1) + k\, d_{xsb}\, n\,(n-1) \tag{3.37}$$

This means that, other than the average link length and the number k of last values to be stored, the model requires roughly n^2 state information to be stored at each node.

Furthermore, the values of x_l cannot be computed. Such computation would require that each ingress node knew the values of all network path rates. Since that approach is deemed unfeasible due to the resulting coordination and signaling problems, the values of x_l must then be estimated. Using end-to-end probing per path sa or sb, all x_l values, where $l \in sa$ or $l \in sb$ respectively, can be estimated. In turn, the signaling complexity required by the algorithm to dynamically adjust the split of traffic between forwarding approaches is:

$$O_{xsa} = 1 \tag{3.38}$$
$$O_{xsb} = n \tag{3.39}$$
$$O_x = (n-1)\,(O_{xsa} + O_{xsb}) \Leftrightarrow O_x = n\,(n-1) \tag{3.40}$$

To sum up, the approximate signaling complexity of the traffic engineering extended model is n^2 per node. This complexity can be deemed unfeasible: the guarantees established by the provisioning method might be overruled due to the extra network load induced by excessive signaling.

However, the O_{xsb} complexity can be greatly reduced if instead of end-to-end probing through all sb tree paths, a round-robin probe through all sb paths is used. Specifically, at each probing event, only a probe is launched for a path sb. This probe results are then averaged with earlier probe values for other trees, and the average value is used in the dynamic adjustment algorithm.

As such, using this approach O_{xsb} is constant. This results in an overall approximate linear signaling complexity per node, trading off measurement accuracy for scalability. This is the modeled approach in the validation chapter of the dissertation.

3.6.7 Optimality Properties

Optimality Criteria Naturally, the optimal solution to the presented problem is the traffic rate vector x which minimises its objective function. As a result of the Kuhn-Tucker theorem [12], it follows that at the optimal rate, each flow s splits traffic only among the paths that have minimum and equal first derivative lengths.

Convergence to Optimality In order to prove that the dynamic traffic monitoring and adjustment algorithm converges to the optimal rate set by the above criteria, theorem 2 of [13] is used. First, consider the following:

Theorem 3.6.1. *For any link l, if $x_l \in [0, c_l[$ then:*

1. *$C_l(x_l)$ is twice continuously differentiable and convex.*
2. *$C_l'(x_l)$ is Lipschitz continuous over any bounded sets.*
3. *For any constant c, the sets $\{z | C_l(z) <= c\}$ are bounded.*

Proof. From (3.6.4), $C_l(x_l) = \frac{p_l}{\alpha (c_l - x_l)}$. Hence, the first and second derivatives of C_l can be computed as follows:

$$C_l'(x_l) = \frac{p_l' \cdot \alpha (c_l - x_l) - p_l \cdot [\alpha (c_l - x_l)]'}{[\alpha (c_l - x_l)]^2} = \frac{p_l}{\alpha (c_l - x_l)^2} \tag{3.41}$$

$$C_l''(x_l) = \left[\frac{p_l}{\alpha (c_l - x_l)^2}\right]' = \frac{p_l' \cdot [\alpha (c_l - x_l)^2] - p_l \cdot \left[\alpha (c_l - x_l)^2\right]'}{\alpha^2 (c_l - x_l)^4} = \frac{2\, p_l}{\alpha (c_l - x_l)^3} \tag{3.42}$$

Since $c_l - x_l = 0 \Leftrightarrow x_l = c_l$, the first and second derivatives of C_l are continuous in $\Re \setminus \{c_l\}$. Therefore, the first and second derivatives are twice continuosly differentiable for $x_l \in [0, c_l[$.

Moreover, if $C_l''(x_l) \geq 0$, $\forall x_l \in [0, c_l[$, then $C_l(x_l)$ is convex in the said interval. Hence,

$$C_l''(x_l) \geq 0 \Leftrightarrow \frac{2\, p_l}{\alpha (c_l - x_l)^3} \geq 0 \Leftrightarrow \tag{3.43}$$

$$\Leftrightarrow \left[2\, p_l \geq 0 \wedge (c_l - x_l)^3 \geq 0\right] \vee \left[2 p_l \leq 0 \wedge (c_l - x_l)^3 \leq 0\right] \Leftrightarrow \tag{3.44}$$

$$\Leftrightarrow 2\, p_l \geq 0 \wedge c_l - x_l \geq 0 \Leftrightarrow \tag{3.45}$$

$$\Leftrightarrow c_l \geq x_l \tag{3.46}$$

Because $x_l \in [0, c_l[$, $c_l \geq x_l$, and thus $C_l(x_l)$ is convex in $[0, c_l[$. As a result, condition 1. is proven.

Furthermore, if for a function $f(x)$, $f'(x) < k_f'$ and $f'(x)$ is continuous, $\forall x \in [a, b]$, then $f(x)$ is Lipschitz continuous for $x \in [a, b]$. It follows that,

$$C_l''(x_l) < k_c \Leftrightarrow k_c > \frac{2}{(c_l - x_l)^3} \tag{3.47}$$

$C_l''(x_l)$ is continuous for $x \in [0, c_l[$ and a k_c exists such that $C_l''(x_l) < k_c$. Therefore, $C_l'(x_l)$ is Lipschitz continuous over any bounded set $B \subset [0, c_l[$, and condition 2. is proven.

Since $C_l(x_l)$ is convex and defined in the interval $x_l \in [0, c_l[$, it follows that:

$$C_l(x_l) \leq C_l(b) \Rightarrow x_l \in [0, b] \tag{3.48}$$

Therefore, for any constant $c = C_l(b)$ and $b \in [0, c_l[$, the set $\{x_l \mid C_l(x_l) \leq c\}$ is bounded by 0 and b. This proves condition 3. and concludes the proof of the theorem. □

It is proven that the traffic engineering system distributed optimisation model follows the conditions 1., 2. and 3. specified by theorem 3.6.1. Therefore, if it is taken that the time interval between updates is bounded, *starting from any initial vector $x(0)$, there exists a sufficiently small stepsize ψ such that any accumulation point of the sequence $\{x(t)\}$ generated by the section 3.6.5 algorithm is optimal* [13].

This is equivalent to say that SSD-TE distributed rate adjustment algorithm converges to optimality, provided the previous conditions are met and that the traffic matrix remains constant for a period of time.

3.7 Implementation

One of the aims of the framework presented in this dissertation is to have low implementation complexity in a real case scenario. In this section, possible implementation mappings of the framework are discussed.

3.7.1 Provisioning Method

The computation of the required network link capacities is processed by an algorithm that receives as input the network topology with link weights (e.g. latencies), the estimated traffic matrix, and the desired ingress surplus traffic rates per flow. The output is the set of all link capacities. See algorithm 1 for the pseudo-code.

The time complexity of the algorithm is $O(n^2 \times k)$, where n is the network node set size and k is the average shortest path length in the network. This complexity is deemed to be feasible, considering that in practice this algorithm should only be used once per topology, matrix estimation and surplus margin adjustments. That is to say, once per network capacity adjustment provisioning.

3.7.2 Routing Management

The network must handle the two routing types used by the framework: shortest path routing and ELB. The former is implemented in an Ethernet scenario by forwarding frames onto their destination sink tree, i.e., a shortest path spanning tree with the egress node as the root. It follows that each network node needs to maintain state regarding all distinct node-rooted trees.

Algorithm 1 Provisioning Algorithm

for all Node $i \in N$ **do**
  for all Node $j \in N, i \neq j$ **do**
    $path \leftarrow shortestPathBetween(i, j)$
    for all Link $l \in path$ **do**
      $l.capacity \leftarrow l.capacity + ETM\_rate(i, j)$ // Provision the rate for shortest path
      $l.capacity \leftarrow l.capacity + \frac{2 \times surplus\_rate_i \times surplus\_rate_j}{sum\_surplus\_rates}$ // Provision the traffic margin for ELB
    end for
  end for
end for

In this case, no forwarding table per tree is necessary: all that needs to be kept is the path towards the tree root.

On the other hand, ELB requires that traffic is equally split (at the ingress node) among all network shortest path spanning trees, each having a different node as the tree root. Thus, the requirement is similar to that of shortest path routing: each network node must maintain state per network spanning tree. However, nodes must have appropriate forwarding tables for each tree. This happens because frames traversing a network tree may not be targeted towards its root, but to one of its leaves.

All in all, routing with either approach requires the existence of n shortest path spanning trees in the network, with proper routing tables per tree being known at each node. A straightforward method to implement this state relies on configuring n VLANs comprising all nodes, each being mapped by a shortest-path spanning tree rooted at a distinct node. Alternatively, if a link-state approach is used, all the required information is already available, namely the network description.

Since at provider scale networks, MAC learning and spanning tree protocols are undesired (see chapter 1), the topology of these VLAN trees along with their respective forwarding tables should be pre-calculated and then stored onto the nodes. A possible implementation mapping for this approach is PBT / PBB-TE.

As seen in the introduction chapter of this dissertation, the Nortel Provider Backbone Transport (PBT) – standardized as Provider Backbone Bridges - Traffic Engineering (PBB-TE) – discards flooding, MAC learning and spanning tree protocols. Forwarding tables, known as Forwarding Database (FDB) entries, are set statically. Forwarding works by selecting the appropriate FDB entry given the VLAN identifier (B-VID) and the backbone bridge destination address (B-DA).

Therefore, PBB-TE could be used to maintain the trees' forwarding tables as FDBs, statically set after a centralised calculation. Furthermore, a frame entering the backbone network could be assigned to a spanning tree by being encapsulated in a PBB-TE header as follows.

If the routing method is by the shortest path, assign the B-VID corresponding to the destination node tree to the frame. On the other hand, if the frame is to be forwarded using ELB, select a shortest path spanning tree from the set and assign the corresponding B-VID to the frame. The trees should be selected, per destination at the ingress node, in a weighted round-robin fashion from the tree set. Figure 3.4 shows a flowchart of this behaviour.

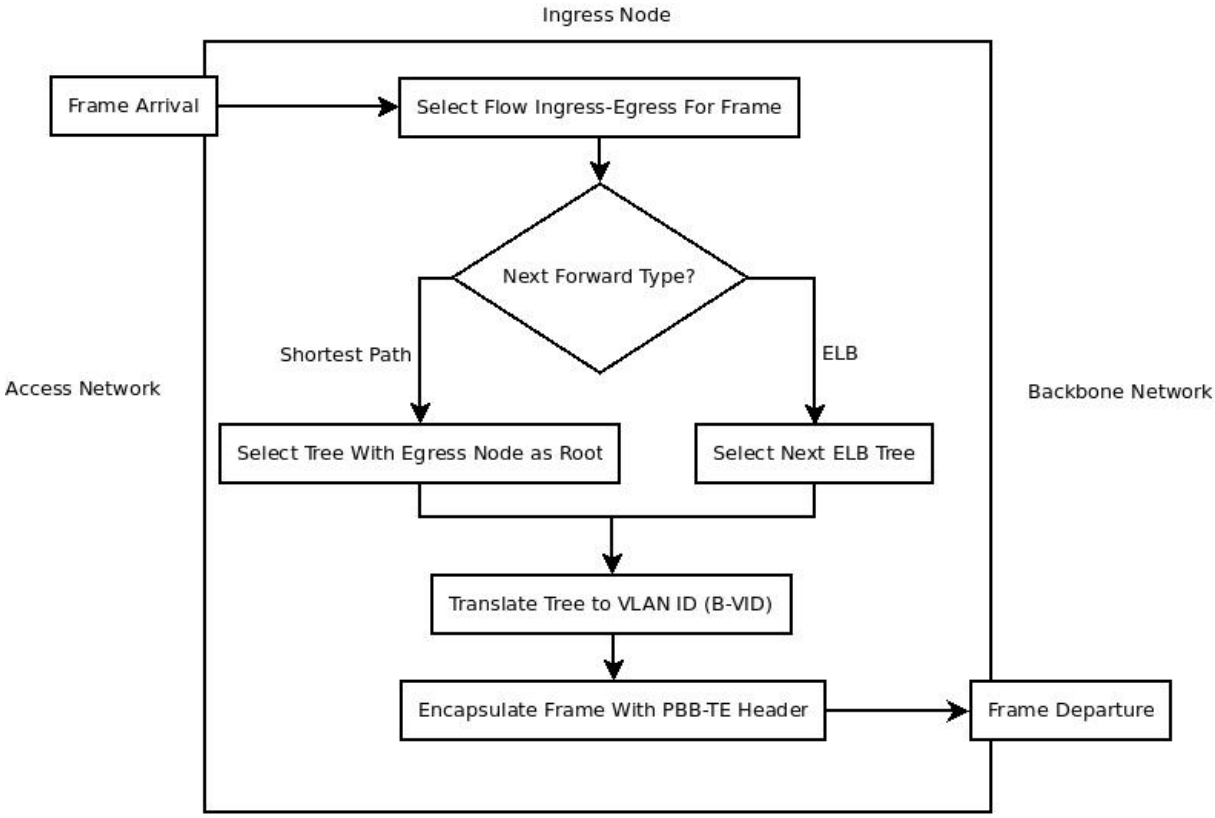

Figure 3.4 Ingress Node Flowchart

In practice however, this process would not be feasible to apply directly to incoming frames. This is because frames are often part of several micro flows (e.g. TCP flow). Frames belonging to the same micro flow should not traverse different network paths, in order to avoid jitter and packet re-ordering, which decrease performance. Therefore, in a real scenario, the selection of a B-VID should be done on a flow-per-flow basis.

3.7.3 Forwarding Selection

The framework is designed to use the dynamic adjustment algorithm to adapt the traffic rates distribution. However, as seen in section 3.5, this distribution can be preset statically. This mode of operation procedure is well suited when a failure of the dynamic algorithm occurs or in presence excessive network instability.

In both types of operation, ingress nodes should be capable of encapsulating incoming traffic inside PBB-TE headers. In addition, for a given flow, they should alternatively assign B-VIDs corresponding to shortest path forwarding or ELB forwarding, depending on the respective distribution traffic rates.

The use of the adjustment algorithm further requires that load values are obtained using end-to-end signaling.

This algorithm is fairly straightforward to implement at the ingress nodes, given the model presented in section 3.6.5. Algorithm 2 lists the pseudo-code of a simplified rate adjustment procedure.

Algorithm 2 Rate Adjustment Procedure

$oldX_{sa} \leftarrow spFlowRates[flow]$
$oldX_{sb} \leftarrow elbFlowRates[flow]$
$newX_{sa} \leftarrow oldX_{sa} - stepsize \times C'_{sa}$
$newX_{sb} \leftarrow oldX_{sb} - stepsize \times C'_{sb}$
$projFactor \leftarrow ingressRates[flow] / (newX_{sa} + newX_{sb})$
$spFlowRates[flow] \leftarrow projFactor \times newX_{sa}$
$elbFlowRates[flow] \leftarrow projFactor \times newX_{sb}$

Nevertheless, signaling the network load presents a more delicate situation. This challenge arises from the fact that there is no signaling protocol in the network which can be used for piggybacking load information – recall that the spanning tree protocol is disabled in PBT/PBB-TE architectures. Furthermore, there is no possibility of inferring network load using the network traffic, such as by inspecting TCP connections [49]. This happens because frames in symmetric flows (with reverse ingress-egress node pairs) may not follow the same network path using this framework, which in turn makes it impossible to calculate statistics, e.g. end-to-end delay.

As a result, a possibility to implement load signaling relies on the sending of end-to-end messages as the payload of Ethernet frames. The payload can be modified by the nodes to change the appropriate load values as it traverses a network path. It follows that this approach is relatively simple to implement and maintain; however, it comes at the expense of injecting further traffic into the network.

Note that this extra traffic may or may not be negligible depending on the network size and the periodicity of the signaling messages. Fine tuning of the periodicity parameter is required to make sure the overhead is not significant. Further discussion of this topic and other implementation concerns can be read in the Results section of the next chapter.

3.8 Summary

This chapter presented Simple and Stable Dynamic Traffic Engineering framework, targeted at Ethernet-based provider backbone networks.

The philosophy of SSD-TE was presented as forwarding as much traffic as possible by the shortest path, whilst distributing the remaining traffic. To this end, SSD-TE comprises a provisioning method, a forwarding scheme and a distributed rate adjustment algorithm.

The framework's provisioning method was shown to ensure that at least the commonly expected load fits the network. What is more, all load variations within a range defined by preset surplus ingress rates are tolerated without loss.

Moreover, the distribution adjustment algorithm was proven to converge to optimality. This algorithm is also fairly scalable and a degree of control over its stability can be achieved by regulating its stepsize parameter.

Finally, this chapter closed with a discussion of the framework's compatibility with current provider scale Ethernet standards, suggesting an implementation with the PBB-TE architecture. The next chapter will present the validation of SSD-TE's framework in a network simulation environment.

4. Validation

In addition to analytical validation, engineering artifacts must be validated using models, simulation, empirical studies and real test cases.

This chapter presents the validation work that has been done on the SSD-TE framework. It starts by introducing the validation model, as well as a definition of the simulation process and tools that were used. Finally, the defined test cases and respective results are presented and discussed.

4.1 Network and Traffic Model

This section shows the network and traffic model that will serve as the basis for the validation of dynamic traffic engineering schemes. Additionally, the method for assigning link capacities given a network topology is described, as well as the criteria used to evaluate the optimality of tested approaches.

4.1.1 Network Definitions

The network model used in the validation resembles a provider backbone packet-switched network, comprising a set of edge-nodes, a set of core-nodes, and a set of links. Edge-nodes designate the network nodes where packets enter or exit the network. Core-nodes designate nodes which only function is to forward packets to other network nodes.

For validation purposes, a flow designates a sequence of packets with an ingress node and an egress node, both from the edge-nodes set. A flow can follow one or more paths in the network between the ingress and egress nodes. Each path comprises the ordered set of links which it traverses from the ingress to egress nodes. A path may also be specified by an ordered set of nodes, or the empty set of intermediate nodes if the two edge nodes are directly connected.

A traffic matrix specifies the average ingress rate for each network flow. This is to say that, for a given traffic matrix M, M_{ij} indicates the average ingress rate for the flow with ingress node i and egress node j.

It is assumed that these flows are comprised of only unicast traffic, i.e. a flow only has a ingress node and an egress node, not multiple ingress or egress nodes. Additionally, the model assumes that of failures in nodes or links are outside of scope and ignores them. These two assumptions are done in respect to the goals outlined in chapter 1.

4.1.2 Network Topologies and Link Capacities

The network topologies used for the framework validation were based in the Rocketfuel ISP maps. Rocketfuel is an ISP topology mapping engine of the University of Washington, dating

from 2002. Rocketfuel's database comprises maps with ISP Points of Presence (POPs), inferred weights and link latencies. These maps were built using traceroutes sourced from several traceroute servers, yielding information about routers, POPs and links. Rocketfuel topologies are frequently used in network research. For instance, the TeXCP protocol [32] is a dynamic traffic engineering approach which relied on Rocketfuel topologies for its validation.

For simulation purposes, the POPs of a given ISP were considered as the simulation nodes. Without loss of generality, all the simulation nodes are deemed to be edge-nodes. In addition, the Rocketfuel latencies between POPs were used as the respective link latencies. The link capacities were assigned by the means of the provisioning scheme described in chapter 3. Recall that, given a network model and a traffic matrix M, the capacity c_l of link l is assigned as follows:

$$c_l = \sum_{l \in shortestPath(i,j)} r_{ij} + \frac{2\,\eta_i\,\eta_j}{R}$$

Note that r_{ij} represents the traffic rate of the flow with ingress node i and egress node j, whereas η_i and η_j are the added traffic rate margins for node i and j, respectively. R is the sum of all nodes' added traffic rate margins.

4.1.3 Traffic Model

The traffic generation is driven by a traffic matrix and the traffic type. Two traffic types are defined as follows.

Constant Bit Rate Constant bit rate traffic, or CBR for short, maps a traffic pattern which ingress rate is constant. In practice, this traffic type is generated by setting the inter-arrival time of frames as a constant value (based on the desired ingress rate).

Variable Bit Rate Variable bit rate traffic, or VBR for short, represents a traffic pattern which ingress rate may oscillate over time whilst having a constant average rate. In practice, this traffic type is generated by setting the time between frame arrivals using an exponential distribution.

4.1.4 Matrix Modification Model

A model of dynamic matrix modification was used to test the response of traffic engineering approaches to unpredictable changes in the traffic matrix. This model implies the change from one traffic matrix M to a traffic matrix M' where the rate of one or more flows have been modified.

For each modified flow, the modification is defined by the increase fraction f and a time period t. The flow modification is done progressively, following a linear pattern over the time period.

For instance, consider that only one flow is modified in the same period. Then, for a flow with ingress node i and egress node j, at any instant t_k between the modification start time t_i and end time t_f (where $t = t_f - t_i$), the matrix is modified as follows:

$$M'_{ij} = M_{ij} \times \left(1 + f \times \frac{t_k - t_i}{t}\right)$$

4.1.5 Performance Metrics and Optimality Criteria

A performance metric is necessary to compare each tested traffic engineering approach. Let $u(f_{ij})$ designate the utility value for the flow f between ingress node i and egress node j. In addition, the previous section has stated that the traffic types that will be tested are inelastic.

As seen in [21], the utility value combines the packet loss probability p_f and queuing delay q_f for flow f, which assuming a best-effort QoS model (all packets have the same service class) can be calculated as:

$$u(f_{ij}) = 1 - \alpha_p * p_f + \alpha_q * \frac{q_f - q_ref}{q_ref}$$

q_ref is the reference queuing delay for flow f (calculated as the minimum queuing delay achievable for the correspondent network model) whereas $\alpha_p = 10$ and $\alpha_q = 1$. The value for α_p makes the value of the utility function become zero for a 10% packet loss and the value for α_q guarantees that the flow utility is equal to zero when the queuing delay doubles the reference value.

Accordingly, the average and standard deviation results for the queuing delay and packet loss for each flow in each test will be mapped to the q_f and p_f parameters.

In order to provide a comparison standard for the several dynamic traffic engineering approaches regarding the utility metric, the utility results for a static traffic engineering scheme are considered as the optimal solution for a given traffic matrix, presented below.

The static traffic engineering results will be calculated resorting to the optimization of the multipath path selection model presented in chapter 2: the objective function minimizes the network's congestion costs metric for each link, selecting a set of network paths for each flow in the process. This model assumes that each flow has a previously computed set of paths.

As seen in chapter 2, the congestion costs metric aims to capture the objective of maximizing the overall utility of the network's flows by weighing in the parameters of loss and delay. This is achieved by defining congestion costs for a given link by computing a pre-defined convex function that maps link utilisation ($u(l) = \frac{load(l)}{capacity(l)}$) to a congestion value. The path selection model using the congestion costs metric can be seen in table 4.1 (derived from [21]).

Indices	
$f = 1, \ldots, F$	Flow f
$p \in \rho_f$	Path p for flow f
$f = 1, \ldots, F$	Link l
$s = 1, \ldots, S$	Step s of the congestion costs function
Parameters	
r_f	Rate of flow f
ρ_f	Set of available paths for flow f
l_p	Length of path p
c_l	Capacity of link l
ϕ_p	Set of links belonging to path p
p_s^x	Additional congestion costs in step s
q_s	Lower threshold of step s
Variables	
ξ	Maximal link utilisation
u_l	Utilisation of link l
a_{fp}	Fraction of flow f being routed via path p
x_{sl}	Value of exceeded threshold of step s on link l
Objective Function	
Minimise $\sum_l \sum_s p_s^x x_{sl}$	
Constraints	
$\sum_{p \in \rho f} a_{fp} = 1$	$\forall f$
$\sum_f \sum_{p \mid l \in \phi_p} r_f a_{fp} = c_l u_l$	$\forall l$
$u_l \leq \xi$	$\forall l$
$\xi \geq 0$	
$0 \leq u_l \leq 1$	$\forall l$
$x_{sl} \geq u_l - q_s$	$\forall s \forall l$
$x_{sl} \geq 0$	$\forall s \forall l$
$0 \leq a_{fp} \leq 1$	$\forall f \forall p \in \rho_f$

Table 4.1 Static traffic engineering model

4.2 Simulators and Tools

The two main approaches used for network behaviour analysis are analytical modeling and computer simulation. The former can be described as a mathematical analysis of a networking abstraction – such as the network and traffic model presented in the previous section. However, this method cannot capture dynamic changes in the network behavior and is therefore inadequate to predict the real-life scenario where these changes are frequent [11]. Additionally, increasing model complexity, namely in what concerns specified protocols or topologies, may render this method unfeasible.

On the other hand, discrete-event simulations comprise a chronologically ordered series of events of a system which follows a specific simulation model and parameterization. As a result, computer simulation tools and frameworks (or network simulators) relying on discrete-event simulation are often used to test network models.

Several network simulators are in use today in both academic research and commercial applications, offering a highly variable set of features. Among these features are discrete-event traffic generators, network protocol implementations, graphical user interfaces and statistics generation. In the following sections we present three network simulators, not only describing their features but also discussing the advantages and disadvantages of using them.

4.2.1 ns-2

The *network simulator*, or ns for short, is an event-driven simulator which originated as a branch of the REAL network simulator which went on to be developed by the Virtual InterNetwork Testbed (VINT) project (with support from DARPA) [3]. The aim of ns was providing a simulation tool to study several protocols under varying conditions, offering abstraction and network emulation [7].

Abstraction is accomplished by splitting the programming model: packet processing implementation is done in a system level language (C++) whereas simulation setup is specified in a scripting language (Objective-TCL). Furthermore, ns allows for a running simulation to interact with real-world operating network nodes, thus providing emulation. The network animator tool (Nam) can be used to visualize and debug network protocols simulated with ns.

Ns-2 is the second version of this simulator and enjoys widespread use in academic research. The ns-2 simulation kernel source code is open, allowing for extensions of the simulation environment, creating new applications and protocols and modifying parameters at different layers [35]. Accordingly, ns-2 has a wealth of user and developer contributions available and several protocol implementations (which are not included in the base package) can be found in the community.

However, ns-2 has a steep learning curve, owing to its structure not being entirely modular. There are also limitations regarding the implementation of protocols such as TCP which can be read in the ns-2 website.

4.2.2 OPNET

The Optimized Network Engineering Tool (OPNET) is a commercial solution for the specification, simulation and performance analysis of communication networks [11]. OPNET provides a modeling and simulation cycle, which comprises the problem definition, validation (by building a model, simulating and analysing results) and decision making. The OPNET Modeler is responsible for the validation stage, relying on discrete-event simulation and incorporating a broad suite of network protocols, such as TCP, OSPFv3 or MPLS. In addition, a full-featured commercial version of OPNET offers implementations of almost any standardized protocol for wired or wireless networks.

The OPNET Modeler offers a series of editors to configure the network, node and process models as well as tools for analysing the simulation results. The latter include a probe editor, a filter tool and an animation viewer. These features are aggregated in a GUI (Graphical User Interface) alongside extensive documentation and case studies, which eases the learning curve and speeds up the simulation process [35].

Unfortunately, being a commercial product OPNET does not provide its source code. Even though the provided models are highly customizable, in what concerns our aims, this issue may pose unavoidable limitations on the extensibility of the simulation environment.

4.2.3 OMNeT++

OMNeT++ is a C++ based discrete event simulator for modeling communication networks, multiprocessors and other distributed or parallel systems [44]. The use of OMNeT++ is increasing, not only on academic research, but also by companies like IBM, Intel and Cisco.

Unlike the previous simulators, OMNeT++ does not provide simulation components directly: instead it offers a base framework for simulations. The components needed for each particular simulation model are implemented on top of this framework and are typically provided as an independent package. As an example, the INET framework [2] comprises several components and models for OMNeT++ which target the TCP/IP stack protocols.

OMNeT++ additionally provides an integrated development environment based on the Eclipse CDT framework, which alongside a modular structure and open source code, makes this simulator very easy to use and extend. Furthermore, network models and components are described in a special purpose language, *NED*, which allows for re-usability and extension, and a GUI is provided to run and debug simulation instances.

Nevertheless, features provided by OMNeT++ packages, such as traffic generation, are not yet as mature as in the previous simulators. This issue can be fixed by integrating other simulator's components (e.g. the ns-2 traffic generator) with OMNeT++, usually resorting to a linking library.

The decisions taken regarding the use of simulation tools in this thesis, as well as the resulting simulation process are presented in the following section.

4.3 Simulation Process

The validation of the model that was specified in section 4.1 is to be done resorting to network simulation. Thus the simulations tools and frameworks to be used must be defined and organized in a coherent simulation process.

OMNeT++ was the chosen simulator to implement the model and execute the required simulations and performance analysis. This decision relies on the fact that the model requires significant changes to existing protocol implementations and OMNeT++ provides the most sound alternative for extension, due to its modular structure and open source code.

As a consequence, the simulation process can be summarised in six distinct phases:

1. Protocol Implementation – Extension of the simulator to implement the traffic engineering framework.

2. Network Specification – Comprises the description of the network topology and protocol to be simulated.

3. Routing Information Generation – Generates the routing information for each network node based on the network specification.

4. Traffic Generation – Generates traffic of a certain type for a given network specification.

5. Simulation Execution – Executes a simulation instance, using the specification, traffic generation and routing tables from the earlier phases.

6. Statistics Retrieval – Computes several statistics based on the output of a simulation instance.

The extension of the OMNeT++ simulator is described in detail in the remainder of this section, whereas the SSD-TE Java tool that was built for phases 2,3,4 and 6 is presented in section 4.4.

4.3.1 OMNeT Extension

In order to simulate Ethernet switches and links, the INET framework was used on top of OMNeT++. INET contains implementations of several Ethernet standards. Among these implementations are the different Ethernet frame types and MAC protocols (i.e., CSMA/CD). What is more, Ethernet switches with the MAC learning algorithm are included in the framework code.

The INET modules to be used as simulation nodes were EtherSwitch2 and EtherHost2, which respectively implement an Ethernet backbone switch and an Ethernet node, working in full duplex mode. Specifically, the EtherHost2 module was used in two separate ways.

First, as a traffic generator for a given network flow. This type of operation was accomplished by attaching the module to the ingress backbone switch of the flow, while at the same time setting its outbound traffic destination to the egress backbone switch sink.

Secondly, as a backbone switch sink. An EtherHost2 module was connected to each edge backbone switch, in order to collect all traffic targeted at that switch.

On the other hand, an EtherSwitch2 module mapped each backbone switch being simulated. Figure 4.1 shows a sample of an OMNeT++ network, comprised of 4 switch nodes (Chicago, New York, Dallas and Atlanta), which follows the presented approach.

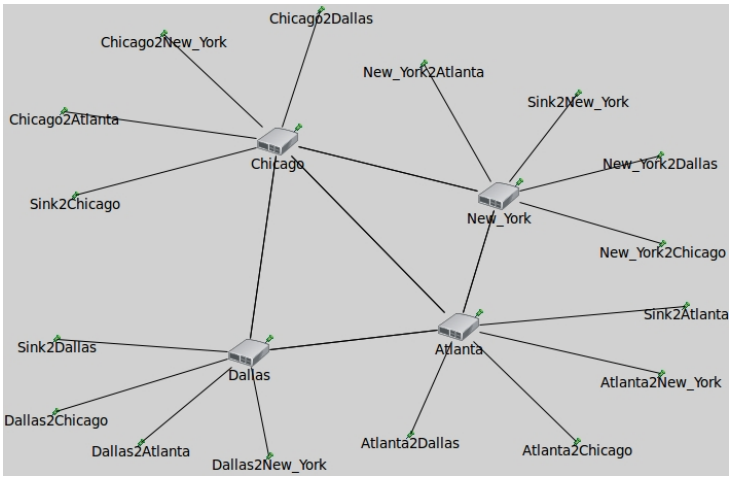

Figure 4.1 OMNeT++ Sample Network

The switching processes are handled in the MACRelayUnit submodule of EtherSwitch2. Thus, the MACRelayUnit source code was changed in order to implement the traffic engineering framework algorithms. Namely, support for shortest path routing and ELB was granted to MACRelayUnit, as well as explicit path routing required by the static traffic engineering model described in section 4.1.5. Moreover, the existing implementation of MAC learning functionality was disabled.

The EtherAppCli module, responsible for generating traffic at EtherHost2 modules, was also modified. These models were adapted to send the traffic types presented in subsection 4.1.3, CBR and VBR. Figure 4.2 shows a simplified model of these extensions.

Furthermore, the Ethernet base frames were extended by adding a treeID and a pathList, respectively allowing for ELB and explicit path routing. This was done to reduce the simulation complexity and avoid the implementation of standards such as PBB-TE or MPLS. However, an extra header size was added to simulation frames to compensate for this simplification. The added size was 20 bytes, which is close to the actual size of a PBB-TE header ($164bits = 20.5bytes$). This is motivated by section 3.7, which states that in a real network scenario the above routing types can be fully implemented with PBB-TE frames.

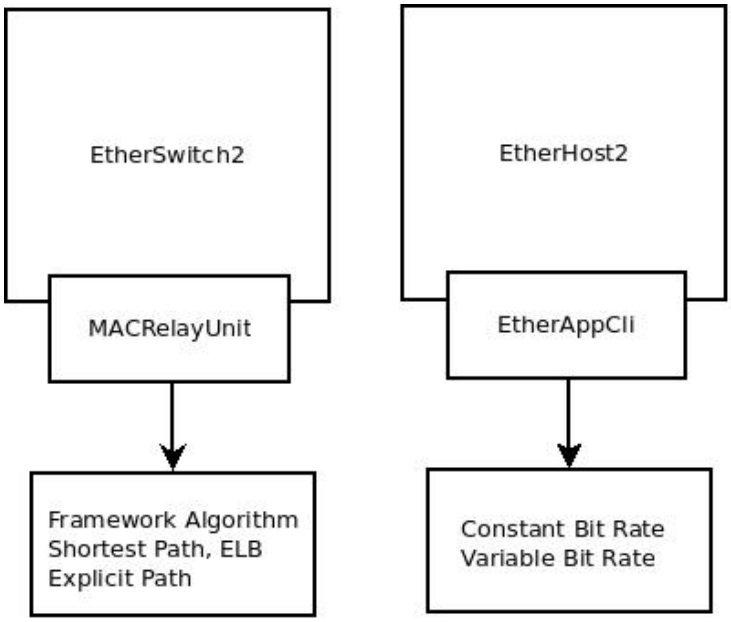

Figure 4.2 OMNeT++ Extension Model

Finally, existing signaling protocols in the Ethernet scenario were not simulated. However, the load signaling messages required by SSD-TE's distribution adaptation algorithm were transmitted in the payload of Ethernet frames between backbone switches.

4.4 SSD-TE Java Tool

Other than running simulations, OMNeT++ was just used to implement SSD-TE's and static traffic engineering components required by simulation nodes. To implement their off-line components, a software tool was developed in the Java language. In addition, this tool was meant to deal with processing test parametrisations (e.g. rocketfuel topologies, traffic matrices) into simulation input models, as well as parsing the results of simulation runs.

To this end, the architecture of the SSD-TE Java tool comprises three standalone components. First, a *Graph Parser* that transforms a Rocketfuel file into a java implemented graph. Second, an *Engine* that uses a topology graph and a test case parametrisation to yield OMNeT++ input files and graph modifications. Finally, the *Utility Parser* calculates the utility values of all flows from a simulation run, using as input an OMNeT++ simulation scalar results file.

Figure 4.3 depicts this architecture with a UML diagram of the SSD-TE Java component interfaces. The following sections describe each component in further detail.

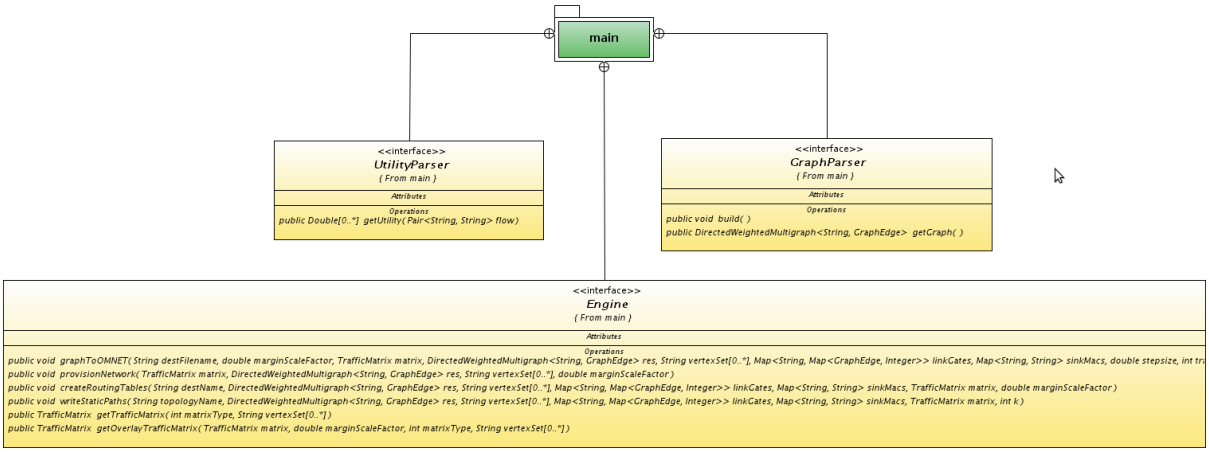

Figure 4.3 SSD-TE Java API Diagram

4.4.1 Graph Parser

This component relies on the JGraphT library to implement a network graph, which is built from Rocketfuel latency files. This library provides several graph related objects and algorithms, which allow for easy manipulation of the network topology. The resulting graph is weighted by the latency and each link has an attribute mapping its capacity (which before provisioning is zero).

Additionally, the graph parser generates graph information that is used by the engine component. This information includes maps of links to gate indexes and MAC addresses of sink simulation nodes.

4.4.2 Engine

The main component of SSD-TE Java is the Engine. Its methods are described as follows.

graphToOMNET A method that transforms an input graph into the .ned model file and .ini configuration file that OMNeT++ requires for a simulation run. The test case parameterisation is also needed to ensure well-formed files.

provisionNetwork An implementation of the provisioning scheme of the traffic engineering framework. This part uses a network topology and respective traffic matrix, yielding the link capacities assigned by SSD-TE's provisioning method.

createRoutingTables A forwarding table generator for shortest path and ELB routing algorithms. Each switch requires different forwarding tables, depending on the network model (with link capacities) and routing algorithm. Since the time complexity of this process is fairly large, a multi-thread implementation was developed in order to decrease its running time.

writeStaticPaths A static traffic engineering problem solver. This part uses the GNU Linear Programming Kit [1] in order to optimise a linear programming instance. The model described in section 4.1.5 is instantiated using a particular network topology and traffic matrix. The output is a series of flow network paths, with their respective traffic distribution ratio.

getTrafficMatrix/getOverlayTrafficMatrix A traffic matrix generator. This generator uses the network model and matrix parametrisation (i.e., type and average node ingress rate) to generate a random traffic matrix. In addition, it allows the scaling of an existing traffic matrix by increasing its rates.

4.4.3 Utility Parser

The utility parser of SSD-TE reads the values written in an OMNeT++ scalar results file, which comprises all simulation statistics. These statistics include the packets sent/dropped and end-to-end minimum/average/maximum latency, for each network flow. The values for these statistics exist for several simulation time intervals, which duration can be increased or decreased as a simulation parameter.

This component uses the said values to compute the utility values for each network flow at all accounted time periods. The minimum latency is used as the queuing reference delay for the flow. The utility results are output into a .csv (comma separated values) file which is suitable to be processed by spreadsheet software.

4.5 Test Cases

Several test cases were defined in order to create simulation instances to be run by OMNeT++. The test cases were built by assigning values to the variables of the validation model, namely network topology, traffic matrix, traffic type and framework parameters. These values are discussed in the following sections.

4.5.1 Network Topology

In order to test the behavior of the traffic engineering framework in different network scenarios, two different topologies were picked from the Rocketfuel database. The considered topologies belonged to United States ISPs, the average node degree and node size being used as criteria to choose among them. The AboveNet and SprintLink were the selected topologies, having distinct values of the said criteria. In addition, the two topologies are deemed to be representative of the Rocketfuel set of ISPs and have been used for validation of other dynamic traffic engineering approaches (e.g. TeXCP [32]). Note that both topologies were filtered to include only US-based PoPs. See table 4.2 for a summary of the topologies.

Name	AS Number	Node Size	Avg Node Degree	StdDev Node Degree
AboveNet	7018	13	2.596	4.039
SprintLink	1239	30	3.486	2.999

Table 4.2 Network Topology Summary

4.5.2 Provisioning Type

To adequately provision the network with the traffic engineering framework, a random ETM traffic matrix was generated for each test case.

In order to make the simulations run in feasible time, all ETM matrices obey the following constraint: the average node ingress rate is 100Mbps. Although this transmission rate is far below the current rates at backbone networks, scaling down the ingress rates does not imply a more favorable validation scenario. In fact, lower transmission rates are likely to penalise the test case optimality results. This is due to the fact the fraction of network load caused by frame headers and the signaling protocol increases when the traffic ingress rate decreases.

The distribution of node ingress rates r_i in the ETM matrix is random, albeit obeying the following constraint:

$$100Mbps \times (1-0.2) \leq r_i \leq 100Mbps \times (1+0.2) \tag{4.1}$$

In addition, the distribution of flow ingress rates r_{ij} per node i in the ETM matrix is random and obeys the following constraint (n is the number of nodes):

$$\frac{r_i}{n-1} \times (1-0.2) \leq r_{ij} \leq \frac{r_i}{n-1} \times (1+0.2) \tag{4.2}$$

The provisioning type is set by the traffic surplus factor γ used. This surplus factor is used to calculate the surplus traffic margin per node η_i used by the provisioning method of SSD-TE, such that:

$$\eta_i = ETM_i * \gamma \tag{4.3}$$

The tested γ values were 0.2 and 0.5, respectively corresponding to small and large degrees of capacity overprovisioning.

4.5.3 Traffic Matrix

The actual traffic matrix of the test run, M, is built from the ETM matrix by increasing the ingress rate per node of the original ETM.

$$M_i = ETM_i \times (1+\gamma) \tag{4.4}$$

Assuming that ETM_i is the sum of the rates of all flows ingressing the network at node i, then the rates of those flows must also be increased in order to meet the new node ingress rate M_i

$$M_i = \sum_{j \neq i} \left[ETM_{ij} + \varepsilon_{ij} \right] \tag{4.5}$$

The distribution of the ε_{ij} values define the traffic matrix type. If all ε_{ij} values are equal at node i, then the traffic matrix is said to be symmetric. If one ε_{ij} value is equal to $M_i - ETM_i$ and all other are zero at node i, then the traffic matrix is asymmetric. The mixed traffic matrix type describes intermediate ε_{ij} distributions between symmetric and asymmetric types. All three traffic matrix types were tested.

Furthermore, all traffic matrices were subject to matrix modification during test runs, as specified in section 4.1.3. Two flows were modified in the same period of time of duration $t = 30s$. The two traffic flows were selected at random and increased by a fraction $f = 2.0$.

4.5.4 Traffic Type

The tested traffic types were CBR and VBR, introduced in section 4.1.3. Only one type of traffic is generated in each test case.

4.5.5 SSD-TE Parameters

The stepsize parameter λ was tested at three different values in order to test the framework stability. Tested values were 10.0, 100.0 and 1000.0. As seen in section 3.6, λ regulates the dynamic algorithm speed of convergence by scaling the rate adjustments done at each iteration.

Furthermore, the parameter regulating the distortion of the SSD-TE's cost function towards a shortest-path distribution, α (see section 3.6.4), was set to a fixed value of 10.

4.5.6 Test Case Definition

Test cases were created with combinations of the previous parameters. The number of generated test cases depended not only on the available combinations of the previous variables, but also on the criteria being tested: optimality or stability. Therefore, 24 test cases were defined for optimality validation, whereas 6 were specified for stability testing.

In respect to the optimality test cases, the number represents the permutations of the 6 traffic matrices, the 2 provisioning types per matrix and the 2 traffic types. The stepsize was set at a constant value of $\lambda = 100.0$.

As to what concerns stability, the 4 test cases refer to 2 traffic types and 2 stepsize values. Mixed traffic matrices were used with the AboveNet topology, with a provisioning factor $\gamma =$ 0.5.

These test cases are summarised in tables 4.3 and 4.4. The test cases were run using the described simulation process. Their results are presented and discussed in the following section.

Matrix Type	Traffic Type	Provisioning Type
Symmetric	CBR	$\gamma = 0.2$
Symmetric	CBR	$\gamma = 0.5$
Symmetric	VBR	$\gamma = 0.2$
Symmetric	VBR	$\gamma = 0.5$
Asymmetric	CBR	$\gamma = 0.2$
Asymmetric	CBR	$\gamma = 0.5$
Asymmetric	VBR	$\gamma = 0.2$
Asymmetric	VBR	$\gamma = 0.5$
Mixed	CBR	$\gamma = 0.2$
Mixed	CBR	$\gamma = 0.5$
Mixed	VBR	$\gamma = 0.2$
Mixed	VBR	$\gamma = 0.5$

Table 4.3 Optimality test cases

Stepsize Parameter	Traffic Type
$\lambda = 10.0$	CBR
$\lambda = 10.0$	VBR
$\lambda = 1000.0$	CBR
$\lambda = 1000.0$	VBR

Table 4.4 Stability test cases

4.6 Results

This section presents and analyses the results obtained for the test cases specified in the previous section. These results were generated by OMNeT++ simulation runs. Each run spanned 90 seconds of simulation time, with flow modification occurring between $t_i = 30s$ and $t_f = 60s$.

The periodicity of the signaling probes used to measure link load, introduced in chapter 3, was set to $0.1s$. Furthermore, for each flow, only one ELB tree path is probed at each probing event.

The next two subsections separate the presentation of results by the criteria being validated. The first subsection targets the optimality criteria, which was formally defined earlier in section 4.1.5. The second subsection discusses the framework's stability, presenting further results of the simulation runs.

4.6.1 Optimality

This subsection presents the optimality results for 24 test cases. Specifically, the results of a given test case comprise the average and standard deviation of the flow utilities, over a 90 second period. Each test case was run twice, using the traffic engineering framework on one run and the static traffic engineering model on the other.

The average utility results throughout the time period are shown in line charts. The charts show 20 average utility results, each corresponding to a time period of 4.5 seconds. In order to allow for easier comparison, the results of test cases with the same network parameterization (matrix type, traffic type, provisioning type) are grouped together.

The results are first presented and discussed in respect to traffic matrix types. The section then goes on to analyse the effects of the other test case parameters, namely traffic type, provisioning type and network topology. Finally, the signaling overhead introduced by the SSD-TE framework is discussed in respect to the results.

4.6.1.1 Symmetric Traffic Matrices

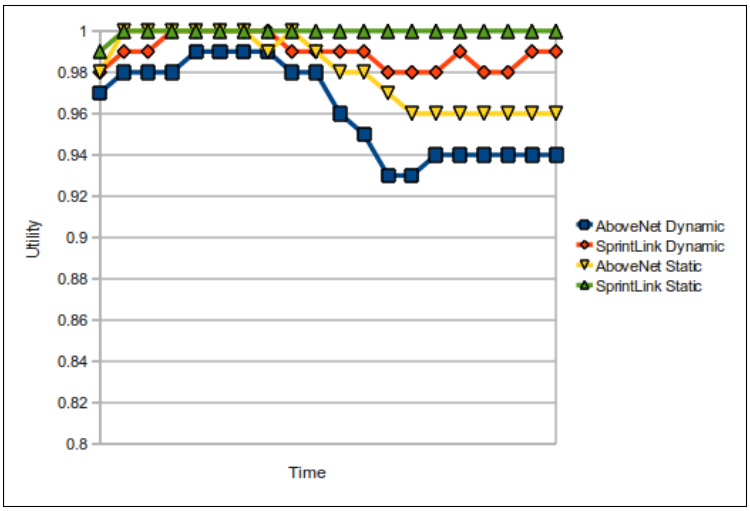

Figure 4.4 Average utility over time: symmetric matrix, CBR Traffic, $\gamma = 0.2$

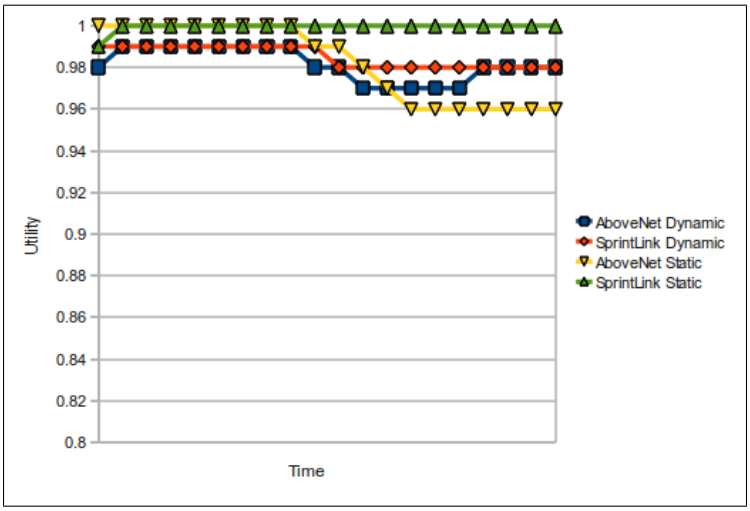

Figure 4.5 Average utility over time: symmetric matrix, CBR Traffic, $\gamma = 0.5$

Symmetric traffic matrices present the worst case results for the SSD-TE framework, specially until flow modification changes the network traffic pattern. The worst scenario is shown in figure 4.6, where the framework doesn't even show signs of recovery after the load change. However, this is an expected situation. Symmetric traffic matrices have an even distribution of traffic between flows and, as a result, the pre-calculated static traffic distributions remain suitable to the network load. Static approaches maintain congestion costs to a minimum, and therefore have room to accommodate the flow modifications.

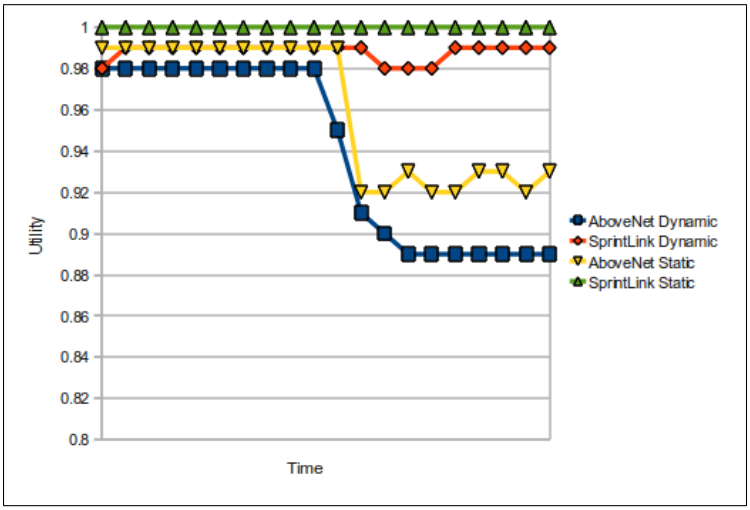

Figure 4.6 Average utility over time: symmetric matrix, VBR Traffic, $\gamma = 0.2$

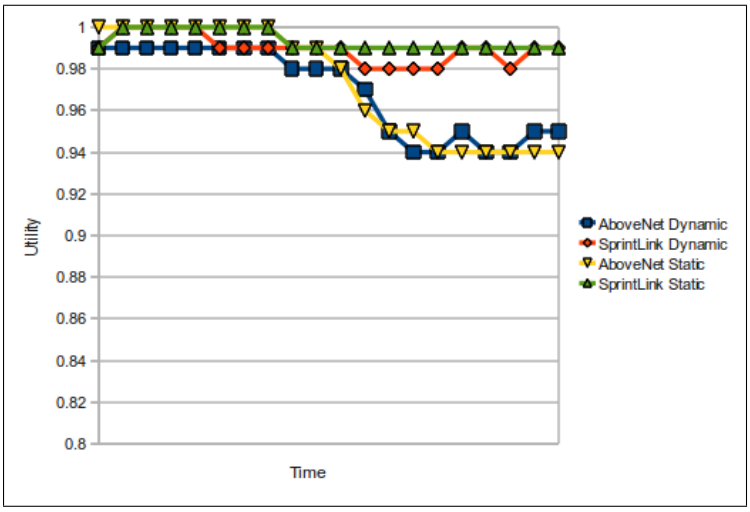

Figure 4.7 Average utility over time: symmetric matrix, VBR Traffic, $\gamma = 0.5$

However, the lower bound of SSD-TE utility results are above 0.9 in all of the symmetric test cases. At most, an utility result of 0.9 implies a packet loss rate of 1% and a 5% delay increase. These values are under acceptable loss and delay parameters for most types of traffic, and are close to the values obtained by the static traffic engineering approach. What is more, symmetric traffic matrices are not a common occurrence in practice.

4.6.1.2 Asymmetric Traffic Matrices

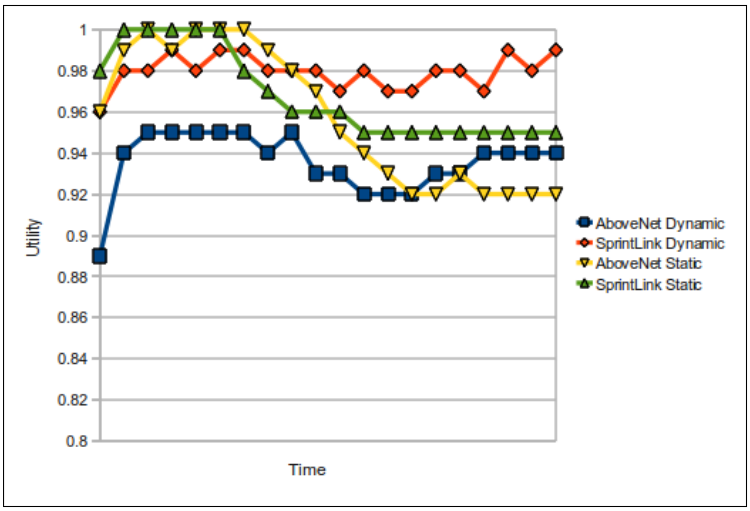

Figure 4.8 Average utility over time: asymmetric matrix, CBR Traffic, $\gamma = 0.2$

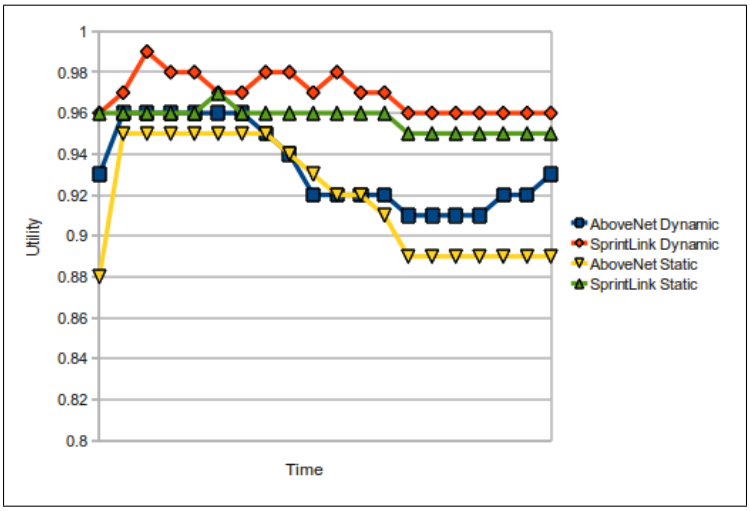

Figure 4.9 Average utility over time: asymmetric matrix, CBR Traffic, $\gamma = 0.5$

On the other hand, asymmetric traffic matrices show the best case scenario for the SSD-TE framework, particularly after flow modification occurs. The high variability in the distribution of the surplus traffic makes the fixed rate distribution of the static approach be untailored to the network load. As a result, some links are overly congested and there is little room to adjust to the new traffic matrix. Since SSD-TE is adept at collecting information about network load and changing distribution accordingly, the framework greatly outperforms static traffic engineering in all asymmetric test case results.

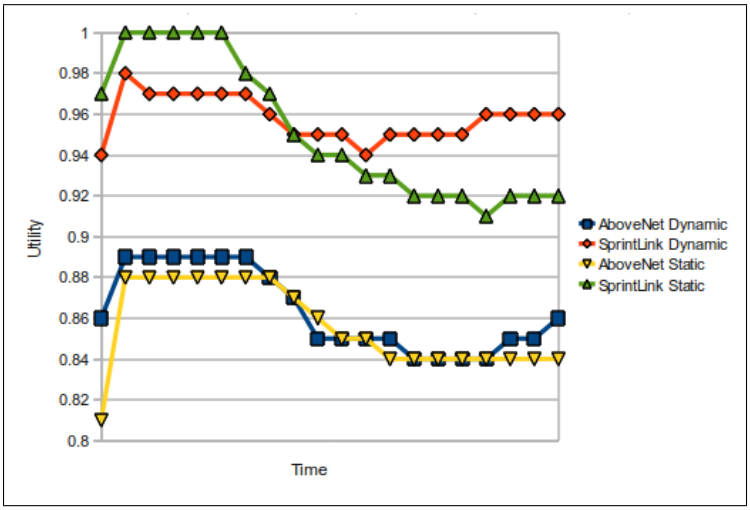

Figure 4.10 Average utility over time: asymmetric matrix, VBR Traffic, $\gamma = 0.2$

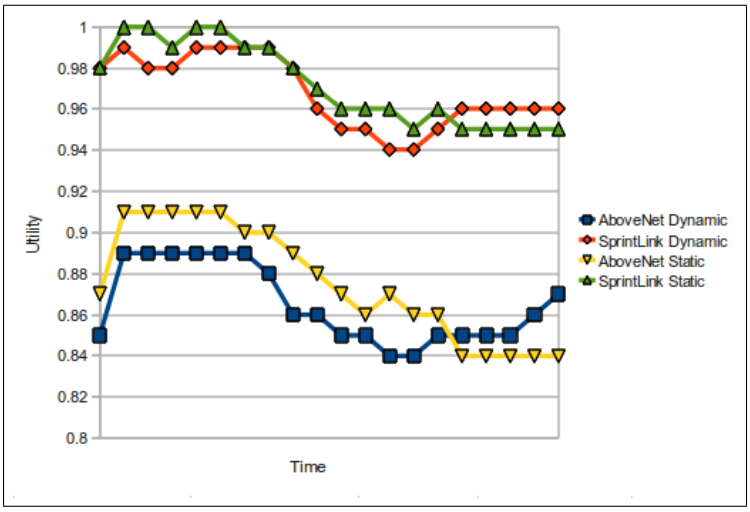

Figure 4.11 Average utility over time: asymmetric matrix, VBR Traffic, $\gamma = 0.5$

Moreover, the test case charts for asymmetric matrices show that utility results are in general lower than for other matrix types. This is because this matrix type makes link congestion more likely to occur, and such, any traffic bursts or variations in traffic (e.g. flow modification) are bound to increase packet loss and delay. This is specially noticeable for VBR traffic results in figures 4.10 and 4.11.

4.6.1.3 Mixed Traffic Matrices

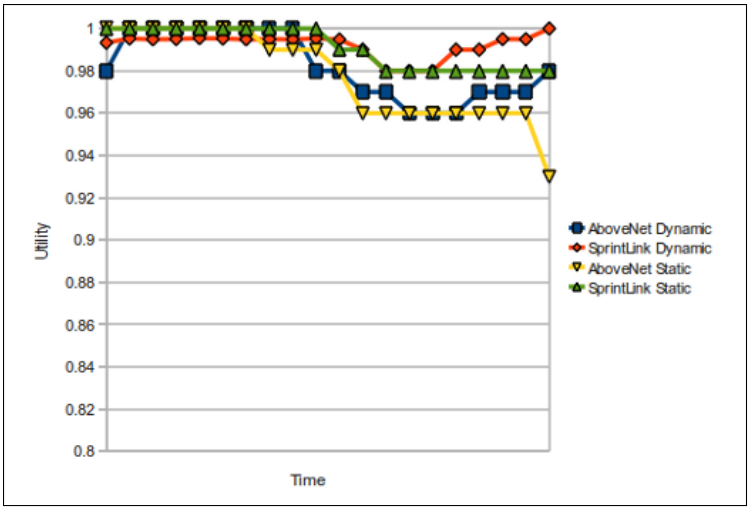

Figure 4.12 Average utility over time: mixed matrix, CBR Traffic, $\gamma = 0.2$

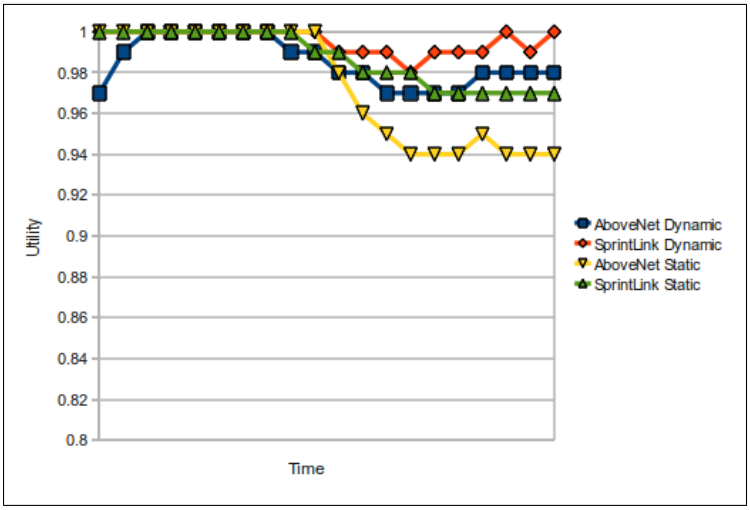

Figure 4.13 Average utility over time: mixed matrix, CBR Traffic, $\gamma = 0.5$

The mixed traffic matrices show an approximation between the previous types of matrices, asymmetric and symmetric. On the one hand, utility results are higher than those of asymmetric traffic matrices. This is explained by the existence of less network congestion due to the ingressing traffic at each node being evened out among its several flows.

On the other hand, the SSD-TE framework improves greatly on the results for symmetric traffic, specially for the AboveNet topology. Moreover, the performance difference between

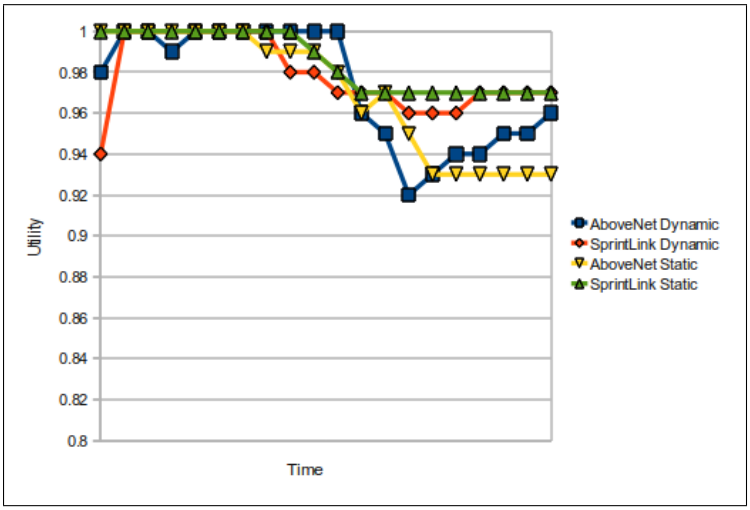

Figure 4.14 Average utility over time: mixed matrix, VBR Traffic, $\gamma = 0.2$

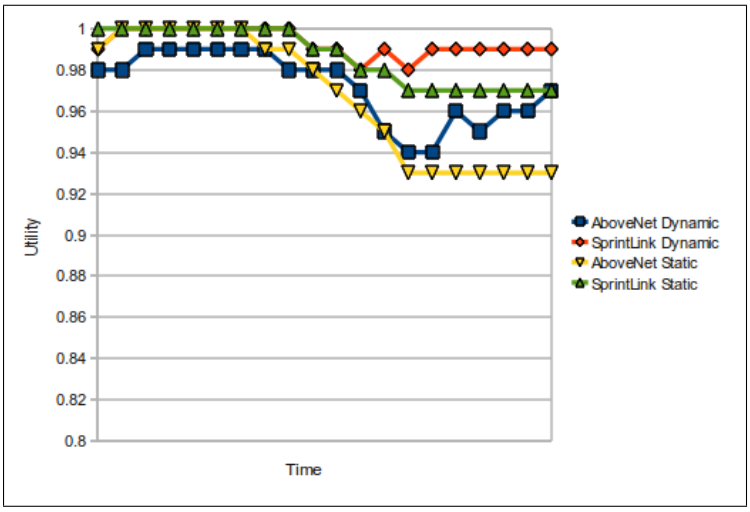

Figure 4.15 Average utility over time: mixed matrix, VBR Traffic, $\gamma = 0.5$

SSD-TE and the static approach peaks at 0.04 utility, in figure 4.15. The framework manages to recover from the flow modification in all test cases, with the exception of the test case in figure 4.14 for the SprintLink topology.

4.6.1.4 Provisioning Type

In general, higher γ values appear to increase the framework's results in respect to static traffic engineering, particularly after flow modification has occurred. Moreover, results do not oscillate as much with $\gamma = 0.5$. One reason for this is the existence of larger surplus rates with higher γ values, which in turn increase the potential for ELB load distribution when the network is congested, since ELB paths are provisioned with more capacity.

4.6.1.5 Traffic Type

The CBR traffic type is shown to produce higher and smoother utility values. Unlike VBR, this traffic type follows a constant frame inter-arrival time pattern, which results in the absence of traffic bursts. Therefore, CBR's stability avoids the occurrence of packet loss caused by bursts, increasing utility. This phenomena is more prone to happen when network load increases due to flow modification and, as a result, VBR penalises overall utility even more after flow modification.

Nevertheless, both CBR and VBR do not seem to give advantage to either the framework or the static approach. In fact, the charts of any given network parametrisation but different traffic type follow the same utility pattern. Figure 4.4 and figure 4.6 show this behaviour.

4.6.1.6 Network Topology

The results showed that the SprintLink network gives more stable results than the AboveNet topology. Furthermore, the results present higher utility values. This can be explained by the larger node size of the former network, and consequently, the larger number of flows. As such, the effects of flow modifications and, in case of VBR test cases, load bursts, are smoothed out when looking at average utility values.

However, when a drop in utility values occur after flow modification, the SSD-TE framework generally manages to recover from the event, independently of the topology. Examples for both topologies are figure 4.10 for SprintLink and figure 4.13 for AboveNet.

4.6.1.7 Signaling Overhead

Several factors are important to estimate the signaling overhead induced by the rate distribution adjustment algorithm of SSD-TE. First, the probe periodicity, set at $0.1s$. Second, the size of each probe, which was simulated as 512 bits (64 bits per link load data $\times$ 8 links). Third, the number of probes per probing event, 2. Finally, the average flow ingress rate, which depends on the topology: SprintLink – $\frac{100Mbps}{30nodes} = 3.33Mbps$, AboveNet – $\frac{100Mbps}{13nodes} = 7.69Mbps$.

Using these variables, we can calculate the signaling overhead percentage of flow traffic as follows:

$$overheadRate = \frac{probeSize \times numProbesEvent}{probePeriodicity} \tag{4.6}$$

$$overheadPercentage = \frac{overheadRate}{averageFlowRate} \times 100 \tag{4.7}$$

Thus, the estimated signaling overheads for SprintLink and AboveNet are, respectively, 0.3% and 0.1%. Albeit small, this overhead percentage may explain why in most of the presented results, the SSD-TE framework could not achieve 1.0 utility values. However, in practice the signaling overhead values would be much smaller and it would not affect the framework's performance. This is because the average flow rate in provider backbone networks may reach values up to tens of Gbps, which are much higher than those simulated.

4.6.2 Stability

This section presents results that aim to test SSD-TE's stability in respect to different traffic types and stepsizes. As such, four test cases were run in OMNeT++ and the framework's traffic rates at three instants were collected. Recall that the traffic rate for flow s comprises the (x_{sb}, x_{sb}) pair, comprising the rates respectively forwarded by the shortest path and by the ELB algorithm. The time instants selected were $t = 30s$, $t = 60s$ and $t = 90s$, respectively corresponding to flow modification start, flow modification end and simulation end.

Since the collected values are too numerous to include in the dissertation – in the order of 300 rates per test case – four flows were selected at random for each test case and their results are presented as follows. Note that an aggregation of the results (e.g. mean, standard deviation) would not capture any oscillation of individual flows, which is vital to check the framework's stability, and thus it is not presented.

FlowID	$t = 30s$	$t = 60s$	$t = 90s$
1	(12.268, 0.515)	(12.268, 0.120)	(12.268, 0.068)
2	(10.145, 0.180)	(10.145, 0.084)	(10.145, 0.054)
3	(10.717, 2.124)	(9.549, 3.292)	(9.317, 3.524)
4	(7.828, 1.863)	(9.621, 0.264)	(9.621, 0.098)

Table 4.5 (x_{sb}, x_{sb}) rates: CBR Traffic, stepsize = 00.0

It is deemed that a stable rate adjustment algorithm will eventually converge to a stable distribution. All the presented tables show that the distribution rates converge to a near-shortest path distribution. What is more, the adjustments in the second time interval ($t \in [60, 90]$) are, in general, smaller than in the first period, which is another proof of convergence to a stable distribution.

Tables 4.5 and 4.7 show that CBR traffic yields more stable results, whereas tables 4.6 and 4.8 denote the bursty nature of VBR traffic, inducing a small degree of variability in flow rates.

FlowID	$t = 30s$	$t = 60s$	$t = 90s$
1	(11.712, 1.721)	(11.036, 2.485)	(11.141, 2.118)
2	(11.446, 0.474)	(13.486, 0.494)	(12.581, 0.408)
3	(13.206, 1.271)	(13.348, 0.751)	(13.406, 0.585)
4	(11.294, 0.577)	(12.243, 0.503)	(11.840, 0.474)

Table 4.6 (x_{sb}, x_{sb}) rates: VBR Traffic, stepsize = 10.0

FlowID	$t = 30s$	$t = 60s$	$t = 90s$
1	(14.700, 0.240)	(14.700, 0.095)	(14.700, 0.059)
2	(11.621, 3.041)	(14.595, 0.323)	(14.595, 0.105)
3	(12.268, 0.007)	(12.268, 0.004)	(12.268, 0.002)
4	(8.840, 0.006)	(8.840, 0.003)	(8.840, 0.002)

Table 4.7 (x_{sb}, x_{sb}) rates: CBR Traffic, stepsize = 1000.0

FlowID	$t = 30s$	$t = 60s$	$t = 90s$
1	(13.173, 0.549)	(12.983, 0.580)	(11.857, 0.523)
2	(15.972, 0.716)	(14.241, 0.741)	(14.005, 0.657)
3	(15.372, 0.668)	(16.249, 0.873)	(14.538, 0.692)
4	(12.804, 0.718)	(13.168, 0.445)	(13.188, 0.497)

Table 4.8 (x_{sb}, x_{sb}) rates: VBR Traffic, stepsize = 1000.0

Moreover, the results for VBR traffic with stepsize 1000.0 have traces of flow instability, as it can be seen in flow 1 and 3 of table 4.8. Both flows present a significantly higher adjustment in the second time interval than in the first (the difference between the traffic rates of two time instants), which might indicate lack of algorithm convergence.

However, since the results of table 4.6 do not show instability in flow values, the cause for this instability is most likely the large value of stepsize chosen. High values of stepsize produce steeper rate adjustments, as seen in chapter 3. This parameter, as well as the periodicity of rate adjustments and signaling messages, are determinant to avoid instability. Therefore they tuned carefully by the operator in order to trade off faster rate adjustments (and thus convergence) for stability.

4.7 Summary

This chapter presented the work done to validate the Simple and Stable Dynamic Traffic Engineering framework. The validation model was introduced, as well as the process implementing it. The extensions to the OMNeT++ simulator and the SSD-TE Java tool were also detailed. Furthermore, the test cases and respective results presented the validation of two fundamental

design requirements of SSD-TE: *optimality* and *stability*.

The *optimality* results showed that the framework achieves its goals in the considered network parameterizations. This claim is supported by two observations.

First, the utility values obtained with the framework are identical to those obtained by the static traffic engineering approach, granted that the network is not congested. In fact, before flow modification occurs in any of the test runs, the average flow utility of the framework is rather close to the maximum utility achievable.

Second, the previous sections show that the framework often surpasses the static traffic engineering results after flow modification, that is to say, in periods of network congestion. In particular, the best framework performance is obtained for asymmetric matrices. This can be explained by the fact that for asymmetric matrices, the flow modification model increases total network traffic by a larger amount than for other matrix types. Thus, since there is superior congestion, the framework's adaptability to the network load gives it a significant advantage over the static approach.

On the other hand, the *stability* of the framework was demonstrated for different types of traffic and stepsize values. In practice, the stepsize parameter is responsible for managing the trade off between responsiveness and *stability*.

Following the description of the framework in chapter 3 and the validation of its design requirements, the next chapter finishes the dissertation, presenting its conclusions and the future work to be done on SSD-TE.

5 . Conclusions and Future Work

This chapter concludes the dissertation, outlining the achieved goals and contributions. It further summarises the developed traffic engineering framework and states its most important results. The chapter finishes with a description of the future work to be done on this subject, along with possible directions to be explored.

5.1 Conclusions

Despite the common drawbacks of dynamic traffic engineering (e.g. instability), it was discussed in the previous sections that this paradigm is well tailored for use in provider networks. This comes from the fact that dynamic approaches are adaptable "on-the-fly" to sudden changes in network conditions, potentially increasing performance. However, the design requirements of optimality, stability, fault tolerance, implementation compatibility and scalability must be taken into account in order to provide a suitable dynamic traffic engineering solution.

This dissertation presented a dynamic traffic engineering framework, the Simple and Stable Dynamic Traffic Engineering (SSD-TE), which targeted the design requirements of unicast traffic optimality, stability, implementation compatibility and scalability, leaving multicast and fault tolerance for future work. After a presentation of state-of-the-art traffic engineering approaches, the underlying philosophy and formal description of SSD-TE were described, finishing with the validation of the properties and design requirements of the framework in an Ethernet scenario.

SSD-TE is built upon the idea that the best method to operate a network is provisioning it in a way that shortest path routing of the expected load does not congest the network. If network congestion occurs, the surplus traffic is balanced in an appropriate way. SSD-TE implements this model and provides three components as follows: a provisioning method to assign network link capacities, a routing approach that relies on both shortest path routing and the ELB algorithm, and finally, a distribution adjustment algorithm that grants SSD-TE its dynamic adaptability to changing network conditions.

The previous chapters shown that SSD-TE meets the goals stated in the introduction. The framework not only achieved results similar to those of static traffic engineering solutions for provisioned traffic matrices, but also outperformed the latter when the network load dynamically changes. This claim is based on two validation results.

First, SSD-TE converged to a traffic distribution that yielded flow utility values similar to those obtained by the static traffic engineering approach (using link congestion costs as the optimisation metric). Second, SSD-TE surpassed static traffic engineering utility values after load modifications were induced, in the majority of the test case runs. Specifically, SSD-TE seems particularly adept at handling asymmetric traffic matrices. Similar results were already observed in earlier work [41].

Other than the optimality results, the validation chapter shown SSD-TE's capability to remain sufficiently stable in various load instability scenarios. In none of the test cases did SSD-TE induce significant oscillations to the traffic distribution, regardless of the chosen stepsize value. In fact, the stepsize parameter of SSD-TE proved to be effective in controlling the traffic distribution convergence speed, trading off responsiveness for stability.

These results were substantiated by the framework theoretical background, presented in chapter 3. The formal description of SSD-TE demonstrated that it was bound to converge to an optimal traffic distribution, assuming that network conditions remained constant for a period of time. Moreover, SSD-TE was shown to be fairly scalable in respect to space and signaling complexities.

Finally, the implementation of the framework on Ethernet scenarios was deemed possible using current standards. Section 3.7 shown that the Provider Backbone Bridges – Traffic Engineering (PBB-TE) architecture is capable of implementing the routing scheme of SSD-TE in the backbone network, whereas the distributed adaptation algorithm can be implemented at the edge of the network.

As specified in the introduction chapter, the concerns of multicast traffic optimality and tolerance to node and link failures were not considered in SSD-TE. Undertaking work on these design requirements was deemed to be unfeasible in the time frame assigned for this dissertation. However, there are possible directions to tackle these concerns and they are presented in the next section, which discusses the future work to be done on the SSD-TE framework.

5.2 Future Work

The primary future work to be done on SSD-TE is to ensure the framework meets the design goals of multicast traffic optimality and fault tolerance. On top of that, the existing SSD-TE framework should be subject an improved validation model. The next paragraphs discuss these topics and shed some light into possible paths that can be followed.

Multicast Traffic

The increasing amounts of video traffic in the Internet, due to video sharing applications/websites, IP-TV or video-on-demand, give renewed importance to multicast traffic. Multicast provides an efficient way to disseminate point-to-multipoint traffic to end users and, as a result, it cannot be disregarded from traffic engineering approaches.

Nevertheless, handling multicast traffic is not straightforward, since it poses several open challenges to developers of traffic engineering solutions. As seen in the introduction to this dissertation, the traffic that a multicast flow induces in the network cannot be directly estimated. For unicast traffic, the traffic caused by a network flow can be calculated resorting to the sender rate and the paths it traverses towards the destination. In multicast, the dynamic nature of the destination group renders this approach unfeasible. Moreover, the multicast routing approach

strongly depends on the desired optimality metric. The performance of this metric is highly dependant on the network condition, which makes it very hard to select.

For instance, assume that the chosen metric is ensuring that packets follow the shortest path from the source to each member of the destination group. If the network is not congested, this metric is well suited and leads to the expected optimal performance, assuming that scalability and implementation concerns are set aside. However, in congestion situations, multicast approaches that minimise the induced load on the network would likely outperform this metric.

Finally, an approach for multicast traffic engineering relies on integrating explicit path forwarding architectures, such as MPLS or PBB-TE, with multicast traffic engineering trees. The MPLS multicast tree (MMT) [10] protocol uses this approach, yielding interesting results in comparision with other proposals (e.g. optimising the weights of a modified PIM-SM protocol [46]). This type of solution might cope well with the SSD-TE framework, which is targeted at an explicit routing architecture and deals with forwarding trees.

Fault Tolerance

Although provider backbone networks are commonly well protected against node and link failures using resilient and redundant components, a suitable traffic engineering framework is expected to grant a reasonable degree of fault tolerance. In fact, the uptime and performance of these networks are expected to be very high, which makes this concern all the more important.

There are a range of techniques to handle path failures, most of which are already implemented in explicit routing protocols. For instance, MPLS specifies a backup path for each path (LSP) between nodes. Furthermore, it uses the fast reroute (FRR) mechanism, which switches traffic traversing the primary path to the backup path, allowing for recovery times.

This approach can be used in SSD-TE to achieve node and link fault tolerance for shortest paths. However, the choice of backup paths and the necessary changes to the provisioning method are unclear at this point.

On the other hand, this approach cannot be used to protect the ELB paths used by SSD-TE. Recall that each ELB path comprises a set of logical paths between the source and destination nodes. Thus, using the previous approach, a backup path would need to be provided for each of these logical paths, for each flow in the network. However, a more straightforward solution exists. The VLB network architecture can be easily adapted to deal with node or link failures, provided that extra capacity is provisioned. Specifically, in order to tolerate k node failures in a network of N nodes, with r being the maximum ingress rate per node, each VLB logical link requires a capacity of $\frac{2r}{N-k}$.

Further work is necessary to adapt this mechanism to a gravity full mesh ELB architecture, which is used in the SSD-TE framework.

Validation

Other than targeting the remaining design concerns which SSD-TE opted not to meet, the framework requires several improvements to its validation model. One of these improvements is adding traffic models which are likely to occur in practice, namely short-lived TCP flows and long-lived TCP flows (see section 4.1.3 for details). The traffic generation model should also consider the existence of micro-flows (e.g. TCP sessions) inside backbone flows. This consideration is particularly important because these micro-flows cannot be split among backbone paths without greatly impairing their performance.

Another validation improvement consists in fully simulating the real protocols and standards which the framework requires to function properly. In fact, testing a framework implementation on a physical network would be the best alternative to validate the implementation compatibility requirement. However, the costs of such approach are most likely unfeasible.

Bibliography

[1] Glpk (gnu linear programming kit). http://www.gnu.org/software/glpk/.

[2] Inet framework, open-source communication networks simulation package for the omnet++ simulation environment. http://inet.omnetpp.org/.

[3] The network simulator - ns-2. http://www.isi.edu/nsnam/ns/.

[4] Ieee standards for local and metropolitan area networks. virtual bridged local area networks. *IEEE Std 802.1Q, 2003 Edition*, 2003.

[5] Ieee standard for local and metropolitan area networks media access control (mac) bridges. *IEEE Std 802.1D-2004 (Revision of IEEE Std 802.1D-1998)*, pages 0_1–269, 2004.

[6] David Andersen, Hari Balakrishnan, Frans Kaashoek, and Robert Morris. Resilient overlay networks. *SIGCOMM Comput. Commun. Rev.*, 32(1):66–66, 2002.

[7] Sandeep Bajaj, Lee Breslau, Deborah Estrin, Kevin Fall, Sally Floyd, Padma Haldar, Mark Handley, Ahmed Helmy, John Heidemann, Polly Huang, Satish Kumar, Steven McCanne, Reza Rejaie, Puneet Sharma, Kannan Varadhan, Ya Xu, Haobo Yu, and Daniel Zapalla. Improving simulation for network research. Technical report, University of Southern California, 1999.

[8] Anindya Basu, Alvin Lin, and Sharad Ramanathan. Routing using potentials: a dynamic traffic-aware routing algorithm. In *SIGCOMM '03: Proceedings of the 2003 conference on Applications, technologies, architectures, and protocols for computer communications*, pages 37–48, New York, NY, USA, 2003. ACM.

[9] P. Bottorff and P. Saltsidis. Scaling provider ethernet. *Communications Magazine, IEEE*, 46(9):104–109, September 2008.

[10] Ali Boudani and Bernard Cousin. Mpls multicast traffic engineering. In *In IEEE ROCC*, 2003.

[11] Xinjie Chang. Network simulations with opnet. In *WSC '99: Proceedings of the 31st conference on Winter simulation*, pages 307–314, New York, NY, USA, 1999. ACM.

[12] R. Gallager D. Bertsekas. *Data Networks*. Prentice Hall, 1987.

[13] A. Elwalid, C. Jin, S. Low, and I. Widjaja. Mate: Mpls adaptive traffic engineering. In *INFOCOM 2001. Twentieth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 3, pages 1300–1309 vol.3, 2001.

[14] Simon Fischer, Nils Kammenhuber, and Anja Feldmann. Replex: dynamic traffic engineering based on wardrop routing policies. In *CoNEXT '06: Proceedings of the 2006 ACM CoNEXT conference*, pages 1–12, New York, NY, USA, 2006. ACM.

[15] B. Fortz, J. Rexford, and M. Thorup. Traffic engineering with traditional ip routing protocols. *Communications Magazine, IEEE*, 40(10):118–124, Oct 2002.

[16] B. Fortz and M. Thorup. Internet traffic engineering by optimizing ospf weights. In *INFOCOM 2000. Nineteenth Annual Joint Conference of the IEEE Computer and Communications Societies. Proceedings. IEEE*, volume 2, pages 519–528 vol.2, 2000.

[17] I. Gojmerac, T. Ziegler, F. Ricciato, and P. Reichl. Adaptive multipath routing for dynamic traffic engineering. In *Global Telecommunications Conference, 2003. GLOBECOM '03. IEEE*, volume 6, pages 3058–3062 vol.6, Dec. 2003.

[18] Huaizhong Han, Srinivas Shakkottai, C. V. Hollot, R. Srikant, and Don Towsley. Multipath tcp: a joint congestion control and routing scheme to exploit path diversity in the internet. *IEEE/ACM Trans. Netw.*, 14(6):1260–1271, 2006.

[19] J. He, M. Bresler, M. Chiang, and J. Rexford. Towards robust multi-layer traffic engineering: Optimization of congestion control and routing. *Selected Areas in Communications, IEEE Journal on*, 25(5):868–880, June 2007.

[20] O. Heckmann and R. Steinmetz. On the elasticity of traffic matrices and the impact on capacity expansion. In *Telecommunications Network Strategy and Planning Symposium. NETWORKS 2004, 11th International*, pages 223–229, June 2004.

[21] Oliver M. Heckmann. *The Competitive Internet Service Provider: Network Architecture, Interconnection, Traffic Engineering and Network Design (Wiley Series on Communications Networking & Distributed Systems)*. John Wiley & Sons, 2006.

[22] Network Working Group IETF. Request for comments: 1058, routing information protocol. http://www.ietf.org/rfc/rfc1058.txt, 1988.

[23] Network Working Group IETF. Request for comments: 1195, use of osi is-is for routing in tcp/ip and dual environments. http://www.ietf.org/rfc/rfc1195.txt, 1990.

[24] Network Working Group IETF. Request for comments: 2328, ospf version 2. http://www.ietf.org/rfc/rfc2328.txt, 1998.

[25] Network Working Group IETF. Request for comments: 3031, multiprotocol label switching architecture. http://www.ietf.org/rfc/rfc3031.txt, 2001.

[26] Network Working Group IETF. Request for comments: 3209, rsvp-te: Extensions to rsvp for lsp tunnels. http://www.ietf.org/rfc/rfc3209.txt, 2001.

[27] Network Working Group IETF. Request for comments: 3212, constraint-based lsp setup using ldp. http://www.ietf.org/rfc/rfc3212.txt, 2002.

[28] Network Working Group IETF. Request for comments: 3414, an architecture for describing simple network management protocol (snmp) management frameworks. http://www.ietf.org/rfc/rfc3414.txt, 2002.

[29] Network Working Group IETF. Request for comments: 3630, traffic engineering (te) extensions to ospf version 2. http://www.ietf.org/rfc/rfc3630.txt, 2003.

[30] Network Working Group IETF. Request for comments: 5036, ldp specification. http://www.ietf.org/rfc/rfc5036.txt, 2007.

[31] A. Iwata, Y. Hidaka, M. Umayabashi, N. Enomoto, and A. Arutaki. Global open ethernet (goe) system and its performance evaluation. *Selected Areas in Communications, IEEE Journal on*, 22(8):1432–1442, Oct. 2004.

[32] Srikanth Kandula, Dina Katabi, Bruce Davie, and Anna Charny. Walking the tightrope: Responsive yet stable traffic engineering. In *ACM SIGCOMM*, Philadelphia, PA, August 2005.

[33] A. Khanna and J. Zinky. The revised arpanet routing metric. *SIGCOMM Comput. Commun. Rev.*, 19(4):45–56, 1989.

[34] Ramana Rao Kompella, Alex C. Snoeren, and George Varghese. mplane: an architecture for scalable fault localization. In *ReArch '09: Proceedings of the 2009 workshop on Re-architecting the internet*, pages 31–36, New York, NY, USA, 2009. ACM.

[35] G. F. Lucio, M. Paredes-Farrera, E; F. Jammeh, and M. J. Reed. Opnet modeler and ns-2 - comparing the accuracy of network simulators for packet-level analysis using a network testbed. *WSEAS Transactions on Computers*, 2(3):700–707, July 2003.

[36] King-Shan Lui, Whay Chiou Lee, and Klara Nahrstedt. Star: a transparent spanning tree bridge protocol with alternate routing. *SIGCOMM Comput. Commun. Rev.*, 32(3):33–46, 2002.

[37] Ivan Pepelnjak. *EIGRP Network Design Solutions*. Cisco Press, 1999.

[38] Radia J. Perlman. Rbridges: Transparent routing. In *INFOCOM*, 2004.

[39] Thomas L. Rodeheffer, Chandramohan A. Thekkath, and Darrell C. Anderson. Smartbridge: A scalable bridge architecture, 2000.

[40] David Sontag, Yang Zhang, Amar Phanishayee, David G. Andersen, and David Karger. Scaling all-pairs overlay routing. In *CoNEXT '09: Proceedings of the 5th international conference on Emerging networking experiments and technologies*, pages 145–156, New York, NY, USA, 2009. ACM.

[41] A. Teixeira, J. Legatheaux Martins, and P. Mariano. Enhancing ethernet forwarding algorithms. In *Actas da Conferência de Redes de Computadores 2009 (CRC'2009)*. Instituto Superior Técnico, 10 2009.

[42] L. G. Valiant and G. J. Brebner. Universal schemes for parallel communication. In *STOC '81: Proceedings of the thirteenth annual ACM symposium on Theory of computing*, pages 263–277, New York, NY, USA, 1981. ACM.

[43] R. van Haalen, R. Malhotra, and A. de Heer. Optimized routing for providing ethernet lan services. *Communications Magazine, IEEE*, 43(11):158–164, Nov. 2005.

[44] András Varga and Rudolf Hornig. An overview of the omnet++ simulation environment. In *Simutools '08: Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*, pages 1–10, ICST, Brussels, Belgium, Belgium, 2008. ICST (Institute for Computer Sciences, Social-Informatics and Telecommunications Engineering).

[45] Jun Wang, Limin Sun, Xiu Jiang, and ZhiMei Wu. Igmp snooping: a vlan-based multicast protocol. In *High Speed Networks and Multimedia Communications 5th IEEE International Conference on*, pages 335–340, 2002.

[46] Ning Wang and George Pavlou. Traffic engineered multicast content delivery without mpls overlay. *IEEE Transactions on Multimedia*, 9:2007.

[47] Dahai Xu, Mung Chiang, and J. Rexford. Link-state routing with hop-by-hop forwarding can achieve optimal traffic engineering. In *INFOCOM 2008. The 27th Conference on Computer Communications. IEEE*, pages 466–474, April 2008.

[48] Rui Zhang-Shen. *Designing a predictable backbone network using valiant load-balancing*. PhD thesis, Stanford, CA, USA, 2007. Adviser-Mckeown, Nick.

[49] Dapeng Zhu, Mark Gritter, and David R. Cheriton. Feedback based routing. *SIGCOMM Comput. Commun. Rev.*, 33(1):71–76, 2003.