Simple and Stable Dynamic Traffic Engineering for Provider Scale Ethernet

António Teixeira, José Legatheaux Martins

CRC 2010

INTRODUCTION

Motivation

- Ethernet's constant increasing transmission speeds and decreasing costs -> Provider Scale Ethernet
- Legacy Ethernet Routing (e.g. STP) is not viable in Provider Scale Networks
- Static Traffic Engineering: is penalized by sudden and unexpected changes of the network traffic

Problem Definition

- Dynamic TE is more suitable at handling traffic oscillations, but has several open problems

- Requirements for a successful Dynamic TE solution include:
    - Optimality
    - Stability
    - Implementation Compatibility
    - Scalability

Contributions

- A Simple and Stable Dynamic Traffic Engineering Framework (SSD-TE)
- Combines two different routing approaches and a dynamic algorithm that adjusts the traffic distribution
- Takes into account the goals of optimality, stability, low implementation complexity and scalability

Presentation Outline

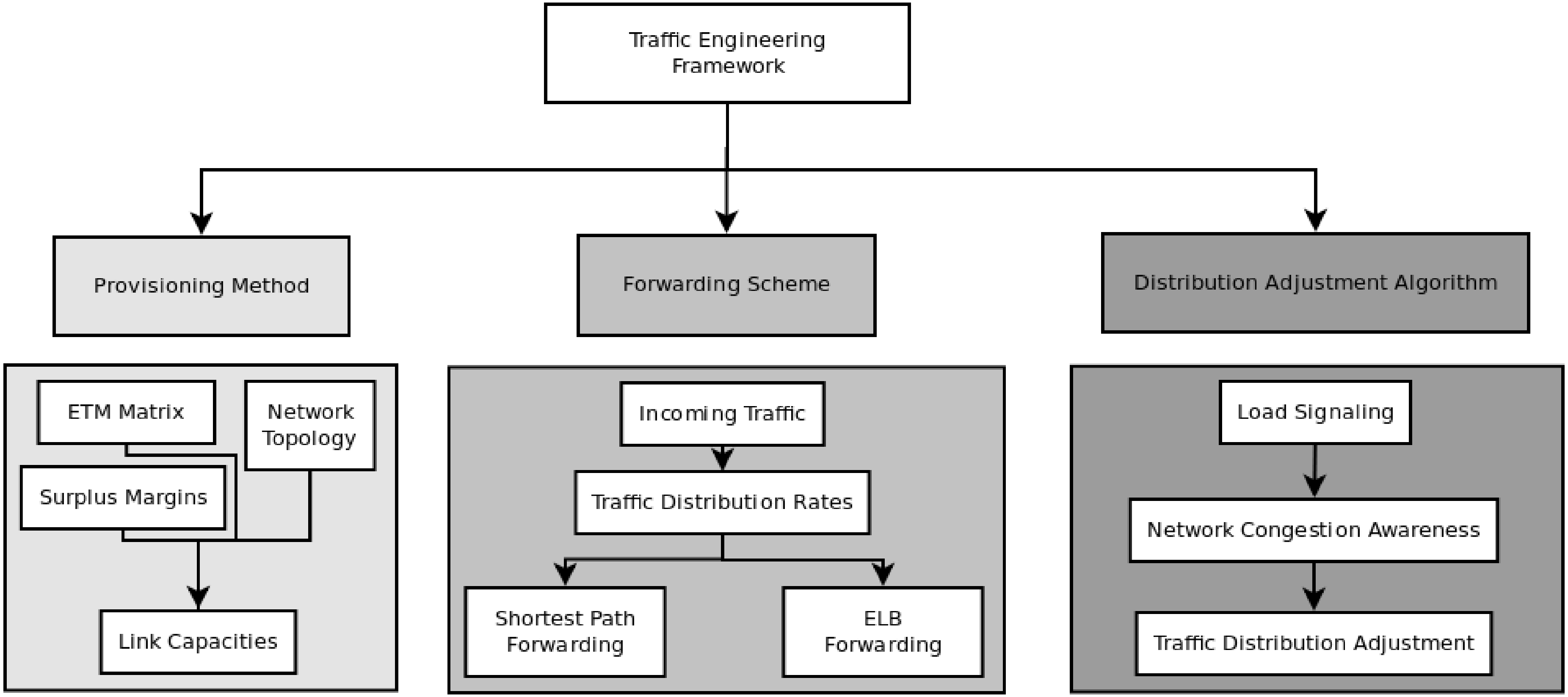
- Introduction
- Related Work
- SSD-TE Framework
- Validation
- Conclusions and Future Work

SSD-TE FRAMEWORK

Framework Introduction

- SSD-TE relies on a simple idea to operate a network:
  - Provisioning it in a way that shortest path routing of the expected load (ETM) does not cause congestion
  - Distributing the surplus load using pre-provisioned capacity and under-used network paths
- The distribution of traffic is adjusted by a dynamic monitoring algorithm

Framework Architecture



Provisioning Method

- Provisions enough capacity to:
  - Forward the ETM by the shortest path
  - Forward the surplus ingress rates using the ELB algorithm
- The ELB approach used by SSD-TE is based on the VLB Gravity Full Mesh scheme

Routing Scheme

- Two traffic rates per flow specifying the amount of traffic being forwarded either by the shortest path or ELB
- The rates are set statically or dynamically
- The static approach guarantees that all traffic within the ETM and surplus rates fits the network
- The dynamic approach adjusts rates as the network traffic varies

Dynamic Algorithm

- Distributed optimization of a convex cost function
  - flow-per-flow optimization
  - requires end-to-end signaling of load
- Convex cost function steers the traffic rate distribution towards shortest path routing, whilst avoiding congestion
- Formal analysis shows that the algorithm converges to optimality

Framework Implementation

- Provisioning Method
  - low complexity computation
  - requiring the definition of an ETM
- Routing Management
  - using the PBB-TE architecture to support shortest path routing and the ELB algorithm
- Traffic Monitoring and Rate Adjustment Algorithm
  - end-to-end signaling to gather link load data
  - end nodes running the dynamic algorithm

VALIDATION

Validation Model

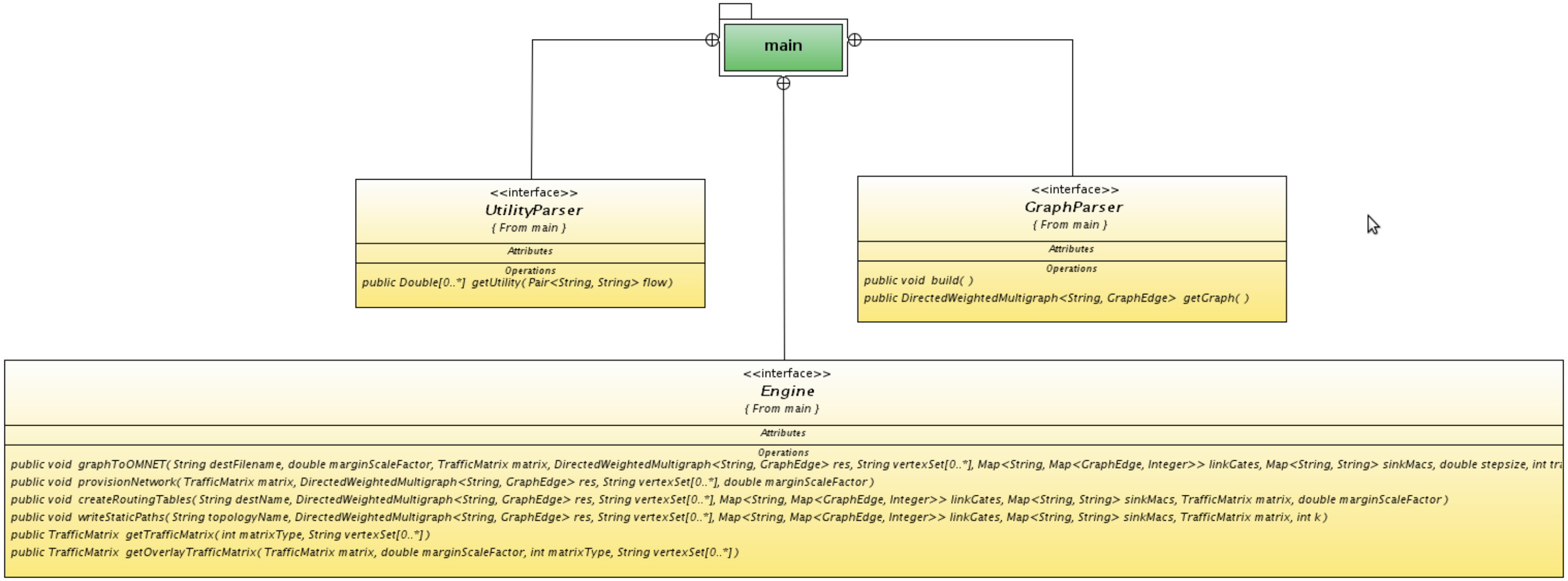
- Validation using network simulation
- Rocketfuel network topologies
- Traffic models: CBR and VBR
- Matrix modification model
- Performance metric: flow utility
- Optimality criteria: comparison with a static TE algorithm, using congestion costs as metric

Simulation Process

- OMNeT++ simulator: modular structure and open-source
- Simulation process
  - Protocol implementation
  - Network specification
  - Routing information generation
  - Traffic generation
  - Simulation execution
  - Statistics retrieval

SSD-TE Java Tool

- Implementation of a Java tool
  - Maps the provisioning method of SSD-TE
  - Executes the static traffic engineering algorithm
  - Generates OMNeT++ input from Rocketfuel files
  - Parses results of simulation runs to spreadsheets



```
public Double[0..*]  getUtility( Pair<String, String> flow)
```

```
public void  build( )
public DirectedWeightedMultigraph<String, GraphEdge>  getGraph( )
```

```
public void  graphToOMNET( String destFilename, double marginScaleFactor, TrafficMatrix matrix, DirectedWeightedMultigraph<String, GraphEdge> res, String vertexSet[0..*], Map<String, Map<GraphEdge, Integer>> linkGates, Map<String, String> sinkMacs, double stepsize, int tr
public void  provisionNetwork( TrafficMatrix matrix, DirectedWeightedMultigraph<String, GraphEdge> res, String vertexSet[0..*], double marginScaleFactor )
public void  createRoutingTables( String destName, DirectedWeightedMultigraph<String, GraphEdge> res, String vertexSet[0..*], Map<String, Map<GraphEdge, Integer>> linkGates, Map<String, String> sinkMacs, TrafficMatrix matrix, double marginScaleFactor )
public void  writeStaticPaths( String topologyName, DirectedWeightedMultigraph<String, GraphEdge> res, String vertexSet[0..*], Map<String, Map<GraphEdge, Integer>> linkGates, Map<String, String> sinkMacs, TrafficMatrix matrix, int k )
public TrafficMatrix  getTrafficMatrix( int matrixType, String vertexSet[0..*] )
public TrafficMatrix  getOverlayTrafficMatrix( TrafficMatrix matrix, double marginScaleFactor, int matrixType, String vertexSet[0..*] )
```

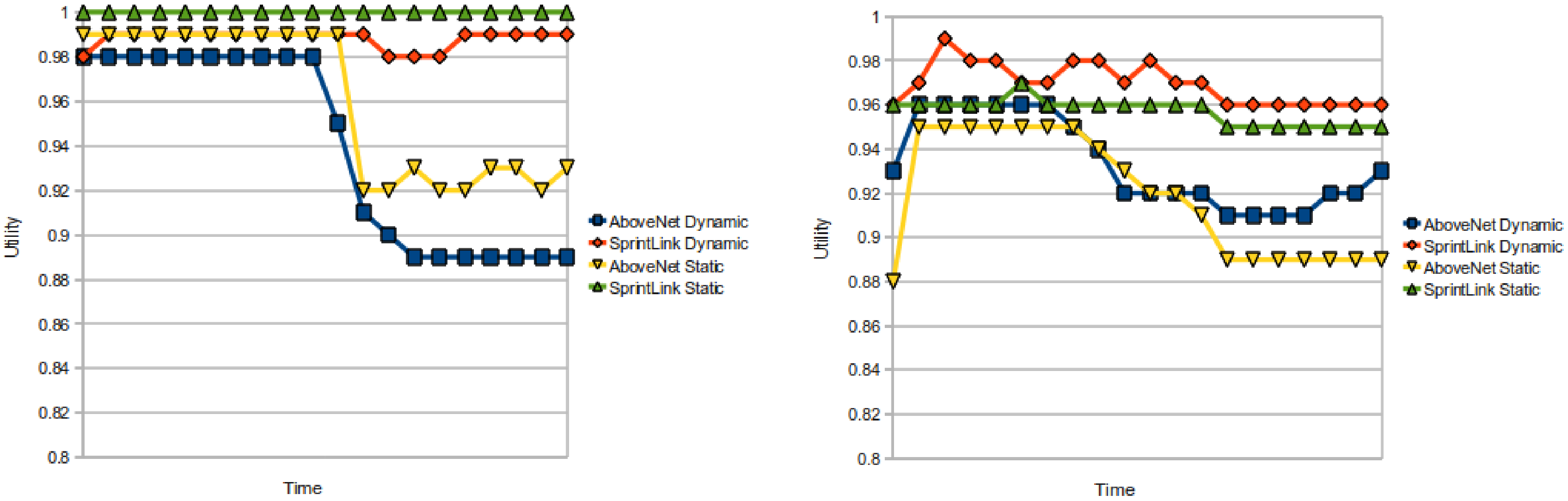
Test Cases

- Model variables
  - Network topology: AboveNet and SprintLink
  - Provisioning type: 0.2 and 0.5
  - Matrix type: symmetric, asymmetric and mixed
  - Traffic type: CBR and VBR
  - Stepsize: 10, 100 and 1000
- 24 optimality test cases
- 4 stability test cases

Results: Optimality

Symmetric, VBR, 0.2

Asymmetric, CBR, 0.5



Results: Optimality

- SSD-TE utility results identical to those of the static TE approach, close to maximum achievable utility
- SSD-TE often surpassed the static TE approach after flow modification occurs
- SSD-TE has better performance for asymmetric and mixed traffic matrices

Results: Stability

- Converges to a stable traffic distribution
- Rate oscillations are increasingly smaller after a period of time with the same traffic pattern
- The stepsize parameter manages the trade-off between speed of convergence and stability

CONCLUSIONS AND FUTURE WORK

Conclusions

- SSD-TE presents a novel dynamic TE framework combining two different routing approaches
- Validation results shown that it is nearly optimal and fairly stable
- SSD-TE was also shown to be compatible with current Ethernet standards and architectures
- Scalable enough for provider networks

Future Work

- Support for multicast traffic engineering
- Node and link fault tolerance
- Validation improvements