Resilience and Quality of Service Assurance Methods in Ethernet and UTRAN Networks

János Farkas

Ph.D. Dissertation

Supervisor:
Dr. László Győrfi
Department of Computer Science and Information Theory
Budapest University of Technology and Economics

Industrial supervisor:
Dr. Csaba Antal
Ericsson Hungary

Budapest
2011
Megbízhatóságot növelő és szolgáltatásminőség-biztosítást elősegítő módszerek Ethernet és UTRAN hálózatokban

Farkas János

Ph.D. disszertáció

Tudományos vezető:
Dr. Györfi László
Számítástudományi és Információelméleti Tanszék
Budapesti Műszaki és Gazdaságtudományi Egyetem

Ipari konzulens:
Dr. Antal Csaba
Ericsson Magyarország

Budapest
2011
## Contents

1 Introduction .............................................. 1

2 Resilient Ethernet ........................................ 5
   2.1 Introduction ......................................... 5
   2.2 Resilient Ethernet Architecture ..................... 7
   2.3 Fast Failure Handling ................................ 10
   2.4 Determining the VLAN-trees ......................... 17
   2.5 Physical Topology Discovery ......................... 28
   2.6 Conclusions ......................................... 33

3 Enhancements to Shortest Path Bridging .................. 34
   3.1 Introduction ......................................... 34
   3.2 Neighbour Synchronisation for Loop Prevention ...... 35
   3.3 Evaluation of Neighbour Synchronisation ............ 43
   3.4 Root Controlled Bridging ............................ 47
   3.5 Performance Analysis of SPB Control Protocols ...... 52
   3.6 Conclusions ......................................... 58

4 Connection Admission Control for UTRAN ............... 59
   4.1 Introduction ......................................... 59
   4.2 Traffic and QoS Model ............................... 63
   4.3 Model Based Connection Admission Control .......... 64
   4.4 Connection Admission Control for IP ............... 68
      UTRAN Implementing WFQ .............................. 68
   4.5 Connection Admission Control Algorithm for Flow Level QoS Guarantees ...... 80
   4.6 Conclusions ......................................... 90

5 Summary .................................................. 92
   5.1 Resilient Ethernet ................................... 92
   5.2 Enhancements to Shortest Path Bridging ............ 94
   5.3 Connection Admission Control for UTRAN .......... 95
Bibliography 96

Publications 102

Appendix 107

A SPB Examples 108
  A.1 SPB Forwarding .......................... 108
  A.2 Neighbour Synchronisation in a Generic Network .................. 109
  A.3 Neighbour Synchronisation Prevents Farkas Loop ................... 110
  A.4 Root Controlled Bridging Prevents Farkas Loop ...................... 112
  A.5 Simulation Parameter Settings for Control Protocol Evaluations .... 112

B UTRAN Examples 114
  B.1 An Example for Random Activity Factor ............................ 114
  B.2 Activity Factor Distribution for GSM Speech ....................... 115
List of Abbreviations

ATM  Asynchronous Transfer Mode
BPDU  Bridge Protocol Data Unit
CAC  Connection Admission Control
CB-WFQ  Class Based Weighted Fair Queueing
CN  Core Network
DCH  Dedicated Channel
DTX  Discontinuous Transmission
FDB  Filtering Database
FHP  Failure Handling Protocol
GPS  Generalized Processor Sharing
GSM  Global System for Mobile Telecommunication
HSPA  High Speed Packet Access
IIH  IS-IS Hello
IP  Internet Protocol
ISDN  Integrated Services Digital Network
IS-IS  Intermediate System to Intermediate System
LAN  Local Area Network
LSDB  Link State Database
LSP  Link State PDU
LTE  Long Term Evolution
MAC  Media Access Control
MAN  Metro Area Network
MSTI  Multiple Spanning Tree Instance
MSTP  Multiple Spanning Tree Protocol
NMS  Network Management System
OSPF  Open Shortest Path First
PDU  Protocol Data Unit
PGPS  Packet-by-packet Generalized Processor Sharing
PLMN  Public Land Mobile Network
PSTN  Public Switched Telephone Network
QoS  Quality of Service
RAB  Radio Access Bearer
RCB  Root Controlled Bridging
RLC  Radio Link Control
RNC  Radio Network Controller
RSTP  Rapid Spanning Tree Protocol
SPB  Shortest Path Bridging
SPF  Shortest Path First
SPT  Shortest Path Tree
TA  Tree Advertisement
TTI  Transmission Time Interval
TTL  Time To Live
UE  User Equipment
UMTS  Universal Mobile Telecommunication System
UTRAN  UMTS Terrestrial Radio Access Network
VLAN  Virtual LAN
VPN  Virtual Private Network
WAN  Wide Area Network
WFQ  Weighted Fair Queueing
Abstract

The use of packet switched technologies such as Ethernet or IP in telecommunications networks requires the application of quality assurance solutions in order to meet the telecom grade requirements. Ethernet is attractive to network providers due to the high bandwidth provided at low cost, therefore, Ethernet is evolving from the enterprise to the carrier. Nevertheless, as a LAN technology, Ethernet did not offer the features that are very important in the new carrier grade application areas, e.g. fast failure handling and sophisticated management.

One of the most important carrier grade requirements is resilience as it is the basis to provide Quality of Service (QoS) guarantees to the customers. Carriers have got used to the failover performance and robustness of the SONET/SDH networks, hence they expect similar performance from a packet network too. However, the control protocols of Ethernet networks did not provide a controllable bounded failover time or they were not applicable in arbitrary network topologies. Therefore, I proposed a scalable resilient architecture in order to meet the carrier grade requirements in Ethernet networks comprised of IEEE 802.1Q-2005 standard bridges. I evaluated the performance and the robustness of the architecture by means of measurements in a prototype implementation.

A recent development in Ethernet networks is the introduction of link state control as specified by the IEEE 802.1aq Shortest Path Bridging (SPB) standard, which aims to improve resilience among its goals. However, the introduction of link state control in Ethernet networks raises some issues among which loop prevention is the most crucial one. Therefore, I proposed loop prevention solutions that can be implemented as extensions to a standard link state protocol, e.g. IS-IS. I proved that the proposed algorithms prevent the occurrence of loops; furthermore, I evaluated their performance by means of simulations.

Internet Protocol is also widely used for transport. Radio Access Networks, e.g. UTRAN networks are often based on IP. Nevertheless, to meet the stringent QoS requirements of the real time traffic, specific solutions are needed, e.g. Connection Admission Control (CAC). I proposed analytical CAC algorithms that take into account the characteristics of UTRAN and allow the utilisation of network resources; furthermore, I evaluated the CAC algorithms by means of simulations.
Kivonat

A csomagkapcsolt technológiák – pl. az Ethernet ill. az IP – távközlő hálózatokban történő alkalmazása a távközlési minőség biztosításához szükséges megoldások alkalmazását igényli. Az Ethernet az alacsony költségen nyújtott nagy sávszélesség miatt vonzó a hálózati szolgáltatók számára, ezért a vállalati, egyetemi szegmensből a szolgáltatói hálózatok irányába fejlődik. Az Ethernet mint LAN technológia azonban nem rendelkezett a szolgáltatói minőségű alkalmazási területeken nagyon fontos tulajdonságokkal, pl. gyors hibakezelés és kifinomult menedzment.

A hibatúrés az egyik legfontosabb szolgáltatói követelmény, mivel ez az alapja a szolgáltatásminőség-biztosítási (QoS) garanciák nyújtásának. A szolgáltatók hozzászoktak a SONET/SDH hálózatok teljesítményéhez és robusztusságához, ezért a csomagkapcsolt hálózattól is hasonló teljesítményt várak el. Azonban az Ethernet vezérlő protokolljai nem tudtak behatolni, szabályozható hibakezelési időt nyújtanak, ill. nem voltak alkalmazhatók tetszőleges hálózatokban. Ezért javasoltam egy jól skálázódó hibatúréző architektúrát, amely az IEEE 802.1Q-2005 szabvány szerinti hidakból álló Ethernet hálózatokban teljesíti a szolgáltatói követelményeket. Egy prototípus implementáción végzett mérések segítségével kiértékelt az architektúra teljesítményét és robusztusságát.

Az Ethernet hálózatok területén egy új fejlemény a kapcsolatállapot alapú vezérlő protokollok alkalmazása, ahogy azt az IEEE 802.1aq Shortest Path Bridging (SPB) szabvány specifikálja, amelynek egyik célja a hibatúrés javítása. Azonban a kapcsolatállapot alapú vezérlés Ethernet hálózatokba történő bevezetése felvet néhány problémát, amelyek közül a hurokmelegítés a legkritikusabb. Ezért javasoltam hurokmelegítő algoritmusokat, amelyek szabványos kapcsolatállapot alapú protokollok, pl. az IS-IS kiterjesztéseként implementálhatók. Bebizonyítottam, hogy a javasolt algoritmusok megakadályozzák a hurkok kialakulását; továbbá szimulációs analízis segítségével kiértékelt a teljesítményüket.

Az Internet Protokoll is széles körben használt transzport technológia; a rádiós hozzáférési hálózatok, pl. az UTRAN gyakran alapulnak IP-n. Ahhoz, hogy teljesíteni tudjuk a valós idejű forgalmak szigorú QoS követelményeit, speciális megoldások szükségesek, mint pl. a kapcsolatengedélyezés (CAC). Javasoltam analítikus CAC algoritmusokat, amelyek figyelembe veszik az UTRAN jellegzetességeit, és lehetővé teszik a hálózati erőforrások kihasználását; továbbá szimulációk segítségével kiértékelt a CAC algoritmusokat.
I express my deepest gratitude to all people who have contributed to this work. First, I thank to my supervisors Dr. László Györfi and Dr. Csaba Antal for their help and support during my Ph. D. studies and research. I thank to all my colleagues at Traffic Lab for the inspirative atmosphere and for the joint work, especially to Dr. Csaba Antal. I am also grateful to all my co-authors for the joint research. I also thank Dr. András Császár, Dr. Sándor Rácz and Dr. Gábor Rétvári for their helpful suggestions and review. I thank for all the support I have got from Ericsson Traffic Lab and HSN Lab. Above all, I thank Beatrix and to my parents for their love and support.
Chapter 1

Introduction

Packet switched networks are deployed more and more widely as they are replacing circuit switched networks and they are also applied in new networking areas, e.g. in data centre networks. Furthermore, connectionless technologies such as the Internet Protocol (IP) and Ethernet gain ground in telecommunication networks as well, which raises new demands against them. They have to provide guarantees for Quality of Service (QoS) and have to be resilient against failures, i.e. provide the connectivity despite of failure events. Resilience is also required in order to be able to meet QoS requirements.

Ethernet has become an attractive network technology in various network deployments due to its simplicity and the high bandwidth provided at low cost. Since its invention in the 1970s, Ethernet has proven that it can adapt to evolving requirements. Ethernet was originally developed to provide connectivity in Local Area Networks (LAN) and has become the de facto standard for enterprise networking. Ethernet is evolving from the enterprise to the carrier, nevertheless as a LAN technology Ethernet did not offer the resilience that is required in carrier grade service networks to provide quality of service guarantees to customers. Ethernet is becoming widely applied in access and Metro Area Networks (MAN). An access network interconnects the customer network with the Internet or service provider networks as illustrated in Figure 1.1. Along the Fixed Mobile Convergence (FMC) initiative, the same access network may be used by broadband and mobile networks.

Nowadays, the size of an access network may reach a metropolitan size; which is larger than a LAN but smaller than a Wide Area Network (WAN) as illustrated in Figure 1.2. Ethernet is also applied in metro networks and the Metro Ethernet Forum (MEF) develops the corresponding carrier Ethernet technical specifications. Due to the challenging carrier grade demands, access networks are also developing, calling for solutions for provisioning reliable services that traditional Ethernet technology could not provide.

One of the most important carrier grade requirements is resilience. Carriers
have got used to the 50 milliseconds (ms) failover performance and the robustness of Synchronous Optical Networks (SONET) and Synchronous Digital Hierarchy (SDH) networks, hence they expect similar performance from a packet network too. Ethernet networks are traditionally controlled by the spanning tree protocols [1], which were mainly designed to provide a loop free topology. However, they are the fault handling protocol as well and they may not provide failover within 50 ms. There are other control protocols designed specifically for ring topology Ethernet networks [2, 3, 4], which cannot be used in an arbitrary topology.

A recent development in Ethernet networks is the introduction of a link state control protocol instead of the formerly used distance vector protocols. Thus network utilisation and transmission bandwidth can be increased compared to that of spanning tree protocols. The ISO/OSI Intermediate System to Intermediate System (IS-IS) [5] routing protocol is a suitable basis for defining the new control for Ethernet networks because it supports the handling of MAC addresses and it is defined based on Type, Length and Value (TLV) structures. There are two recent standards addressing this problem space: the IEEE 802.1aq Shortest Path Bridg-
(SPB) [6] and the IETF Transparent Interconnection of Lots of Links (TRILL) [7, 8], both rely on IS-IS. As SPB is specified by IEEE 802.1, it preserves the 802.1 architecture and it is compatible with all other 802.1 standards. Therefore, SPB uses standard 802 frame formats; it inherits for example Ethernet Operations, Administration and Maintenance (OAM) specified by 802.1ag Connectivity Fault Management [9] and the scalability provided by 802.1ah Provider Backbone Bridges (PBB) [10]. Furthermore, SPB is also compatible with the data centre bridging standards being specified by 802.1. As opposed to this, TRILL specifies a new frame format, i.e. introduces a new data plane which implies the need for new hardware. Furthermore, TRILL is limited to customer Ethernet services [7] because it is specified assuming IEEE 802.1Q-2005 [1] thus it is not compatible with any amendments to [1], i.e. with IEEE 802.1 standards published after 2005. Therefore, TRILL has no OAM, it has scalability issues and it is not compatible with recent data centre standards, which is summarised e.g. by Eastlake [11], who is the chair of the TRILL Working Group. They are about to start to develop an OAM solution [12], specify interworking with data centre bridging standards [13] and define a solution for scalability issues [11]. Due to the above differences, SPB is applicable in a much wider space of networking scenarios. SPB is able to take advantage of a dense topology, which is typical in data centres. In addition, using the shortest path also decreases latency, which is crucial for real time traffic, e.g. in access networks. Furthermore, the OAM provided by SPB is a key component for network operators, especially if they aim to provide carrier Ethernet services.

Providing connection oriented services by a connectionless packet network e.g. IP, which is often over Ethernet, may require additional functionality if QoS is to be assured. A Universal Mobile Telecommunications System (UMTS) network may be deployed using IP for terrestrial transport, then the UMTS Terrestrial Radio Access Network (UTRAN) is based on IP. Therefore, the latency and packet loss requirements of UTRAN have to be met by the IP transport.

The objective of this dissertation is to provide solutions for three resilience and QoS assurance problems: (1) providing carrier grade failover by an Ethernet network, (2) providing loop prevention for link state control, and (3) ensuring QoS for IP based UTRAN. These problems should be solved because:

(1) The restoration methods of former Ethernet standards do not provide 50 milliseconds failover time, which is required by carriers. In order to make Ethernet applicable in metro or access networks, predictable failover time is required and it is desired to be kept under 50 ms.

(2) The introduction of link state control in Ethernet requires extensions and adaptation. A link state protocol, as already standardised, cannot be applied
for the control of an Ethernet network because it does not avoid transient loops, which may cause network melt down.

(3) Connection Admission Control has to be applied in an IP UTRAN network in order to ensure that the QoS requirements of admitted connections are met.

The Outline of the Dissertation

Chapter 2 provides a distributed resilient Ethernet architecture. Providing 50 milliseconds failover is in the focus of the architecture, therefore, the proposed fast failure handling protocol is a key component of it. In addition, an algorithm is provided to determine the forwarding topologies that are able to survive single failures. Furthermore, a physical topology discovery algorithm is also specified, which provides the input for determining the forwarding topologies. The architecture was implemented in a prototype network, and the characteristics of its components were evaluated by means of measurements. The experimental results showed that the 50 milliseconds carrier grade recovery time is achieved and the architecture is robust.

Chapter 3 proposes two different approaches for the extension of link state protocols such as IS-IS in order to make them applicable for the control of Ethernet networks, e.g. an SPB network. Two extensions are described, which mainly address loop prevention. Indirect proofs show that transient loops are avoided by these extensions. Furthermore, the proposed algorithms were implemented in a packet level simulator and their effect on network convergence time was evaluated by means of simulations.

Chapter 4 describes Connection Admission Control algorithms for the Iub interface of UTRAN, which comprises the transport links between base stations and radio network controllers. These algorithms are applicable for the assurance of QoS requirements. The CAC algorithms are based on analytical grounds and they were evaluated and verified by means of simulations.
Chapter 2
Resilient Ethernet

2.1 Introduction

Ethernet networks are evolving from the enterprise to the carrier due to simplicity and high capacity provided at low-cost. The bandwidth provided by native Ethernet increased over the time. Gigabit Ethernet (GbE) and 10 GbE over fibre are commonly available today, which have preserved the frame structure and simplicity of lower speed Ethernet standards. As a LAN technology Ethernet had been optimized for fast data transfer but not for fast failure handling. Therefore, recent developments aim to construct carrier grade Ethernet networks, providing highly reliable transport based on Ethernet.

Carrier grade networks require fast failure handling. Spanning Tree Protocol (STP) [14], which was developed to ensure loop free topologies, is also responsible for failure handling. Therefore, the speed of STP determines the failover time, which is in the order of tens of seconds. Rapid Spanning Tree Protocol (RSTP) [15] was developed to reduce the convergence time and fix STP issues. The next step in the evolution of the STP family was the introduction of Multiple Spanning Tree Protocol (MSTP) [1], which does not improve further the failover time. The xSTP protocols are distance vector protocols, thus the count-to-infinity problem may appear, e.g. if the root bridge of a tree breaks down. If there remains a ring topology after the loss of connectivity to the root, then the bridges that were not directly connected to the root may advertise outdated information on the reachability of the root because they are not aware of its break-down. The outdated and the correct information on the root will then chase each other around the ring until the old one ages out as described in detail e.g. by Myers [16]. Note that count-to-infinity is only about control information, it does not cause data loop as RSTP ensures loop free operation. Network operators are typically not in favour of xSTP because of the inexact implementations and the relatively not easy configuration of the desired forwarding paths.
There are other mechanisms standardized for Ethernet networks including the Ethernet Ring Protection Switching [2], Resilient Packet Ring (RPR) [3] and Ethernet Automatic Protection Switching (EAPS) [4] for ring topologies, but they cannot be applied in arbitrary network topologies. Furthermore, RPR requires new Medium Access Control (MAC) protocol, which is not supported by most network devices. Failover time is reduced to sub-second range in the Viking architecture proposed by Sharama [17]. Each bridge is configured to send Simple Network Management Protocol (SNMP) [18] traps to the Central Manager in case of a failure event. The Central Manager is a central server, which is responsible for the overall operation of the network including fault handling. After failure notification, the central server figures out which paths are affected and informs the end-nodes about the necessary reconfiguration in order to use a backup path. Each of the end-nodes runs a client module that is responsible for the selection of the appropriate path. Viking approach requires a failure management centre, which decelerates the failover procedure. Furthermore, it has to be ensured that the Central Manager is not a single point of failure.

These solutions do not provide carrier grade and scalable in-band solution for failure handling in arbitrary network topologies. Therefore, I aimed at providing a framework that makes fast failover possible in a general Ethernet network in a robust and scalable manner, which maintains the tree based forwarding applied in Ethernet. That is multiple trees are maintained for frame forwarding, which have to be designed such that they survive the failure events.

The calculation of spanning trees is widely examined in the literature. Typically, edge disjoint spanning trees in dense topologies are in the focus of these algorithms, which were investigated by e.g. Nash-Williams [19], Miura [20] and Curran [21]. For instance, Gabow [22] developed a polynomial-time algorithm to find \( k \) edge disjoint spanning trees with the smallest cumulative weight in a directed graph. Roskind’s algorithm described in [23] can be used for the same purpose in an undirected graph. Nonetheless, the topology of telecommunication networks typically does not allow the construction of edge disjoint spanning trees. Furthermore, if only a part of the network is aimed to be protected, then edge disjoint spanning trees are not necessary. Werneck [24] proposed an algorithm to find \( k \) minimum spanning trees, meaning that the \( k \) spanning trees are not necessarily disjoint but have a minimal total cost. Young’s algorithm [25] also finds \( k \) minimum spanning trees. Nonetheless, even looser constraints may be formed for the design of forwarding trees than the ones assumed in the literature, which allows the construction of an algorithm providing fault protection using less trees.

The accurate knowledge of the physical topology is essential in order to determine the forwarding topologies. Link Layer Discovery Protocol (LLDP) [26] is a neighbour discovery protocol specified to simplify troubleshooting. LLDP defines a standard method for Ethernet bridges to advertise information about themselves.
to neighbour nodes and store locally the information they receive in an SNMP Management Information Base (MIB). LLDP can be used to assist the automatic discovery of the physical topology, nonetheless, it may be the case that LLDP is not implemented in all Ethernet bridges in a network. Topology discovery in Ethernet networks was also addressed by the research community, which is summarised by Bejerano in [27]. Almost all of these topology discovery methods rely on Filtering DataBase (FDB) information, which is collected via SNMP MIBs from the bridges. Therefore, they can only find spanning tree paths and are not able to find physical links inactivated by xSTP. Another approach is the measurement-based topology discovery proposed e.g. by Black [28], Caceres [29], Bu [30] and Rabbat [31]. These algorithms rely on end-to-end measurements in order to determine the topology of the network without collecting MIB information. They provide the logical topology that is used for forwarding but can neither discover inactive links nor map logical nodes to network elements. Furthermore, they require the deployment of a monitoring software at the end stations and generate a large number of probe messages. Son [32] proposed another approach to discover both active and blocked links, which relies solely on SNMP MIBs. However, besides standard MIBs, they also use optional MIBs, which are not necessarily implemented in all bridges (e.g. D-Link DES-3526). Therefore, their method fails to determine the network topology in Ethernet networks containing bridges that do not implement the adopted optional MIBs or MIB objects. That is all of former algorithms have their own limitations; they either can only discover the topology determined by xSTP or cannot be used in Ethernet networks with bridges from multiple vendors.

This chapter describes a resilient Ethernet architecture and its components in detail, which meets carrier grade resilience requirements and resolves the issues described above. Furthermore, the evaluation of the proposals is included. Section 2.2 gives an overview of the architecture. The failure handling mechanism applied in the architecture is described in Section 2.3 in detail. An algorithm for the computation of the necessary forwarding trees is given in Section 2.4. The physical topology for tree computation is provided by the discovery algorithm described in Section 2.5.

2.2 Resilient Ethernet Architecture

I proposed an Ethernet architecture that provides resilience in a distributed manner ensuring fast failover, which is described in this section in detail besides Farkas [J3] and Farkas [C8]. In the core of the network, the architecture consists of IEEE 802.1Q-2005 standard Ethernet bridges. The extra functionality needed for providing resilience is implemented in the edge nodes of the Ethernet network. The core nodes are not modified, at most some configuration is required on them.
Predefined trees are statically set up across the network to serve as either primary or alternative paths that can be used to route traffic in the network and to be able to handle possible failures. The trees may be spanning trees depending on which nodes participate in the connectivity service provided by the given trees. The trees have to be designed such that at least one tree survives the failure event against which the network is aimed to be protected, i.e. the trees have to be fault tolerant. To achieve protection against any single link or node failure, it has to be ensured that there remains at least one complete functional tree in the failure event of any single network element. The trees are set up before network start-up, and they can be calculated by the algorithm I proposed in Farkas [C8] and also described in Section 2.4. The trees remain unchanged during operation unless the network operator wants to adapt them to a new topology.

In the event of a failure, each edge node has to stop forwarding frames to the affected trees and redirect traffic to unharmed trees. Therefore, a protocol is needed for failure detection and for notifying all the edge nodes about the broken trees. Failover time mainly depends on the time elapsed between the failure event and its detection by the edge nodes because protection switching from a tree to another one is done without any re-configuration of the Ethernet bridges. I proposed a failure handling method in Farkas [C7], which is also described in Section 2.3.

The predefined trees are implemented using Virtual LANs (VLAN) [1]. A unique VLAN Identifier (VID) is assigned to each tree, therefore, traffic forwarding to a tree can be controlled by means of VIDs in the edge nodes. These trees are referred to as VLAN-trees in the following. Spanning tree protocols are disabled because they are not needed to provide loop free topology. That is the topology layering is changed a little bit as illustrated in Figure 2.1. In the standard [1], a spanning tree protocol forms the loop free active topology on top of the physical topology. The VLANs are then on top of the active topology. A higher topology is a subset of a lower one, not necessarily a real subset. In the resilient architecture, the VLAN and the active topologies are combined and implemented by the VLAN-trees, which are also loop free. As a consequence of using VLAN-tree topologies, protection switching becomes a simple VLAN switching in this architecture. That is the fault handling principle has been changed from restoration to protection.

Figure 2.2 shows an example of the proposed network architecture. The example network is comprised of four core bridges B1, B2, B3, B4, and four edge

![Figure 2.1: Topology layers](image-url)
bridges EB1, EB2, EB3 and EB4. The edge nodes may be bridges or a combined implementation of a router and a bridge. Three VLAN-trees are needed to protect the network against any single failure: T1, T2 and T3.

In Ethernet networks, Virtual Private Network (VPN) separation is also implemented by VLANs. If just a subset of the nodes takes part in a VPN, then protection should only be provided for the links and nodes that take a role in the interconnection of the VPN. That is, the number of required VLAN-trees for a given logical network may be less than that of the whole network. However, each VPN should have its set of VLAN-trees in order to provide protected connectivity in the proposed network architecture. Note that VPNs are not discussed in the following because they are a straightforward extension of the approach providing connectivity among all the nodes. Due to this simplification, the VLAN-trees are spanning trees in the following. In other words, a VLAN does not refer to a VPN in the following but only to a VLAN-tree.

Once the trees are set up, they can be used in either primary-backup or load sharing mode. In the former mode, a single tree is used as a primary tree and all the traffic is sent on the corresponding VLAN. If the primary tree breaks down, then one of the trees that remained complete is used for traffic forwarding. Note that VLAN IDs have to be also reserved for backup trees in order to provide fast protection switching and those VLAN-trees stay idle during normal operation. The VIDs are listed in the same priority order in each edge node; the primary VID has the highest priority. If a VID has to be selected for traffic forwarding, then the VID having the highest priority is chosen. Thus each edge node sends user traffic on the same VLAN after a failure event and after its restoration too. In the load-sharing mode, traffic is evenly distributed among all operational trees. In the event of a failure, traffic is redistributed among the trees not affected by the failure.
2.3 Fast Failure Handling

The most important design goals of a failure handling mechanism are fast failover, simplicity, robustness and small transport overhead. A further aim was to construct a protocol with a built-in synchronization mechanism, i.e. no other protocol is needed to synchronise the communication among the edge nodes. In addition, the Failure Handling Protocol (FHP) I proposed in Farkas [C7] is a simple and lightweight distributed protocol implemented in the edge nodes of the network.

Message Types and Node Roles

The FHP relies on a few broadcast messages to detect failures and to provide fast reaction to them. The FHP monitors the availability of each VLAN-tree and uses all of them in order to provide information on their status to the edge nodes, i.e. the FHP treats the VLAN-trees together. The operation of the FHP relies on three broadcast messages as follows:

- **Keep Alive (KA)** messages are broadcasted periodically by one or more edge nodes referred to as *emitter* over each VLAN-tree according to a predefined time interval $T_{KA}$. The received KA messages indicate that the VLAN-tree is alive and operational.

- **Failure** message is issued by an edge node named *notifier* when a KA message does not arrive over a VLAN-tree within a pre-defined detection interval $T_{DI}$ to inform all the other edge nodes of a failure in that VLAN-tree. The Failure message is sent on each VLAN-tree that the notifier perceives alive, i.e. on all VLAN-trees except for the ones whose KA message is missing. A notifier may wait for multiple, e.g. three, missing KA messages before issuing a Failure message.

- **Repaired** message is issued by the notifier that detected the failure when a KA message arrives over a previously failed VLAN-tree. Thus the notifier informs all the other edge nodes about the reparation of the failed VLAN-tree.

Two types of *notifiers* are distinguished based on their timer settings: *primary* and *secondary*. A couple of *notifiers* are configured as *primary*; all the other nodes that are neither *emitters* nor *primary-notifiers* are *secondary-notifiers*. Fundamentally, all three types of broadcast messages can be sent by all edge nodes in the network; it only depends on configuration which node sends which message. The roles of the edge nodes are illustrated in Figure 2.2.

Figure 2.3 shows a schematic time sequence chart of the protocol messages for the different edge node roles. Figure 2.4 shows the flowchart specifying the operation of the failure handling protocol at the two types of edge nodes. KA messages
are broadcasted periodically by the emitter over each VLAN at the beginning of the KA period. Thus, KA messages have to arrive over all VLANs at each edge node within the predefined $T_{DI}$. Each notifier edge node registers the arrival of KA messages and starts a timer to measure whether $T_{DI}$ has elapsed. There is no need for additional synchronisation besides the KA messages. If the arrival of KA messages is not registered within $T_{DI}$, then the corresponding VLANs are considered down. Note that as many KA messages have to arrive in a VLAN as many emitter nodes are in the network in order to consider that VLAN unharmed. For instance, if a notifier receives a single KA message over a VLAN in a two-emitter system, then either a link or node failure happened.

All edge nodes, except the emitter, supervise the reception of KA messages. However, to avoid broadcast storms after a failure, there are only a few primary-notifier edge nodes whose task is to notify each edge node about the failure. The

![Figure 2.3: FHP message time sequence](image)

![Figure 2.4: Operation of the FHP](image)
detection interval of the primary-notifiers is shorter than that of the secondary-notifiers, and it can be adjusted depending on the network size and other parameters. When a primary-notifier detects a failure, it broadcasts a failure message over each operating VLAN informing the IDs of the broken VLANs. As each edge node receives the failure messages, all of them become aware of the failed VLANs.

As the number of primary-notifiers is intentionally limited, some failures might be undetected by the primary-notifiers depending on the network topology. Therefore, if a secondary-notifier detects a failure based on the missing arrival of a keep-alive message, then this node broadcasts the failure message to notify all the other edge nodes on the failure.

The reparation of the broken element is also handled by the protocol. The emitter always broadcasts KA messages over all VLANs even if a failure has been detected before. If the failure is repaired then the same notifier that detected the failure detects its reparation as well because it again receives KA messages over the broken VLAN. Thus, it can notify the other edge nodes by broadcasting a repaired message to them on the repaired VLAN.

The failure handling protocol is also protected against the breakdown of edge nodes. As there are multiple notifier nodes, their breakdown does not prohibit fault detection because any notifier that recognises a failure informs the others if that failure has not been already reported. Nonetheless, the outage of an emitter edge node is a special case, which can be also recognised. If the emitter goes down, then no KA message arrives on either VLAN from the emitter. Therefore, if no KA message arrives within $T_{KA}$, then the emitter edge node is supposed to be broken (also assuming that a single failure can happen at a time). The so-called backup emitter then takes over the emitter’s role. If the emitter is repaired and comes back, then it again receives KA messages in each VLAN, thus knowing that there is already an emitter in the network; therefore, it becomes the backup emitter. Thus the Failure Handling Protocol has no central entity that is exclusively responsible for a task, instead, each role is located in a different part of the network, which results in a robust architecture.

Configuration of the Protocol

The emitter and primary-notifier nodes have to be selected among the edge nodes. The proposed default configuration is that each edge node should be set as secondary-notifier node. One of them has to be then configured as emitter and another one as backup-emitter. Depending on the size of the network, one or more edge nodes are configured as primary-notifiers among the rest of them.

In order to achieve the shortest failover time and minimise the number of broadcast messages, I proposed the following approach for the node selection in Farkas [J3]:
• **emitter**: the edge node that is the closest in average to all other edge nodes in each tree because the transmission delay is minimised this way between the emitter and notifier nodes.

• **backup-emitter**: the edge node which is closest to the emitter, since the backup has to take over the role of emitter in case of its breakdown. This choice ensures the smallest change in transmission delay compared to the original setup.

• **primary-notifier**: the minimal set of edge nodes whose connection path to the emitter covers each link of each tree. This definition also determines the number of necessary primary-notifiers. If the links are categorised as risky and non-risky links, then it is enough to detect the breakdown of risky links by the primary-notifiers, failures of non-risky links can be detected by the secondary-notifiers. It is then enough to configure as primary-notifiers the minimal set of edge nodes whose connection path to the emitter covers each risky link of each tree. This configuration assures that most of the failures are detected by the primary-notifier, which makes the failover time shorter.

• **secondary-notifier**: all the remaining edge nodes.

### Failover Time

Failover time is a key performance indicator of resilience approaches. The failure handling protocol described here is fast because it only depends on the end-to-end transmission time of messages and on the $T_{KA}$, which is determined from the transmission time. The upper bound of the failover time ($T_F$) is given by:

$$T_F \leq T_{KA} + T_{DI} + T_{tr} + T_{pr}, \tag{2.1}$$

where $T_{tr}$ and $T_{pr}$ are the worst case end-to-end transmission and packet processing delays in the network, respectively.

The reason for this is that in the worst case, a failure happens at the beginning of a KA period and it is only detected in the next KA period shortly before the end of the detection interval. In the worst case, a secondary-notifier detects the failure, thus its $T_{DI}$ has to be taken into account.

I proposed the following protocol parameter settings in Farkas [J3] for providing a predefined failover time. The $T_{DI}$ should be determined based on the largest Round Trip Time (RTT) measured or estimated on the different VLAN-trees between the emitter and the farthest primary-notifier. The transmission delay is approximately the half of RTT, however, to avoid the effects in the variance of the transmission delay $T_{DI}$ is proposed to set not smaller than the RTT.
\[ T_{DI} = RTT \] \hspace{1cm} (2.2)

Larger \( T_{DI} \) results in even more robust setting, nonetheless, \( T_{KA} \) gives an upper bound as \( T_{DI} \) cannot be larger than \( T_{KA} \) in order to avoid the overlapping of KA periods.

By assuming that \( T_{DI} \) of primary-notifiers is set according to Equation 2.2, \( T_{KA} \) then can be calculation based on Equation 2.1:

\[ T_{KA} = T_F - T_{DI} - \frac{RTT}{2} = T_F - 3 \cdot \frac{RTT}{2}. \] \hspace{1cm} (2.3)

The upper bound of \( T_{DI} \) at secondary-notifiers is also \( T_{KA} \). Nevertheless, it is better to set shorter interval in order to achieve better failover when a secondary-notifier has to react to a failure. As secondary-notifiers are distinguished in order to avoid broadcasting storms thus their \( T_{DI} \) has to be set larger than that of primary-notifiers. Thus the failover time is

\[ T_F \leq T_{KA} + 1.5 \cdot RTT. \] \hspace{1cm} (2.4)

That is, besides the network specific delays, the failover time can be controlled by \( T_{KA} \), of which smallest value is 3 ms in practice.

**Performance Analysis**

The performance of FHP was evaluated on a prototype network and the results were published in Farkas [C7]. FHP was implemented on Linux PCs, which were used as edge nodes of the resilient architecture. FHP was implemented using 68-byte Ethernet frames, providing room enough to accommodate all of the needed protocol messages and additional parameters. The core nodes are standard Ethernet bridges with VLAN support; no additional features are required to support FHP or to provide protection. Combinations of bridges from two different vendors were tested: bridges from D-link and Extreme Networks were applied.

A test network set-up is shown in Figure 2.5. A Tester PC transmitted and received the test traffic, while controlling the bridge in the middle of the link between B1 and B3 in order to generate failures. Besides the topology shown in Figure 2.5, FHP was tested on other network topologies, e.g. on a 3x4 grid.

Table 2.1 shows the measured failover time results collected over 1000 protection switching events for several emitter / primary-notifier node configurations in the network topology shown in Figure 2.5. The applied KA transmission period (\( T_{KA} \)) was 15 ms; the detection-interval \( T_{DI} \) was 5 ms in the primary-notifier and 10 ms in the secondary-notifiers. The test traffic enters the network through edge node EB1 and leaves the network through edge node EB2, as shown in Figure 2.5.
The results are consistent with the theoretical predictions. The average failover time equals to $0.5 \cdot T_{KA} + T_{DI}$, and $T_{KA} + T_{DI}$ is an upper bound for the maximum (worst case) time.

The results show that the third scenario provides the fastest failover time. The reason for this is that in this scenario the primary-notifier is able to detect the failure and initiates the notification process. In the first, second and fourth scenarios, one of the secondary-notifiers can only detect the network failure and initiates the failure handling process.

Figure 2.6 shows the failover time as a function of $T_{KA}$ in the topology shown in Figure 2.2. The KA period increased from 6 ms to 50 ms, the $T_{DI}$ of the primary-notifier was set to $T_{KA}/3$, the $T_{DI}$ of the secondary-notifiers was set to $2 \cdot T_{KA}/3$. The roles of the edge nodes was as follows: EB3 was the emitter, EB4 was the primary-notifier, EB1 and EB2 were secondary-notifiers. In this configuration, the failure was detected by a secondary-notifier node (specifically EB2).

It can be observed that the measured worst case failover is consistent with the failover expected from Equation 2.1. The results indicate that the maximum failover performance can be maintained below 50 ms by keeping the KA period below 25 ms in the evaluated network.

Table 2.1: Failover Time of the FHP

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Average</td>
<td>19.62</td>
<td>21.83</td>
<td>15.38</td>
<td>20.47</td>
</tr>
<tr>
<td>Maximum</td>
<td>29</td>
<td>29</td>
<td>24</td>
<td>28</td>
</tr>
<tr>
<td>Minimum</td>
<td>12</td>
<td>11</td>
<td>7</td>
<td>12</td>
</tr>
<tr>
<td>Std. Dev.</td>
<td>4.69</td>
<td>5.15</td>
<td>4.87</td>
<td>4.31</td>
</tr>
</tbody>
</table>
The traffic overhead is also an important characteristic of a control protocol. The traffic overhead of the FHP is

\[ L_{\text{FHP}} = 68 \cdot 8 \cdot \frac{N_{\text{VLAN}}}{T_{\text{KA}}}, \]  

(2.5)

where 68 is the Ethernet tagged frame size used in the implementation, \( N_{\text{VLAN}} \) is the number of VLANs in the network and the measure unit of \( L_{\text{FHP}} \) is kbps.

Figure 2.7 shows the protocol load normalised to 100 Mbit/s Ethernet as a function of \( T_{\text{KA}} \), with \( N_{\text{VLAN}} \) as a parameter. As the results show, the protocol overhead is below 1% of the capacity of the Fast Ethernet link, even if the KA period is 10 ms on 15 VLANs.

With regard to the repeated node failure tests, the results were in full agreement with Table 2.1, and all the failover time results were between the minimum
and maximum values of the table. The measurement results for the other tested topologies were also in-line with Table 2.1. Note that these results were achieved by a prototype implemented on a non-real time operating system, which could be further improved by the integration of the proposed protocol into a high-performance hardware.

An additional feature of FHP that has been experimentally verified in the prototype network is that no frames are lost at the edge nodes during the restoration phase, corresponding to either link or node repair. The reason for this hitless operation is that the VLAN-trees are fully operational after the reparation. Thus the redirection of the traffic from one tree to another does not cause frame loss.

2.4 Determining the VLAN-trees

Fault tolerant trees have to be designed in order to be able to protect the network against failures. The primary goal is to protect the network against any single failure, i.e. the failure of a link or a node. Furthermore, the number of the trees should be minimised due to the limited number of VIDs and in order to ease network management. In addition, large running time is not allowed for a tree computation algorithm as new trees might be required after a network reconfiguration. This section describes the algorithm I proposed in Farkas [C8] for the calculation of fault tolerant trees addressing the above requirements.

The aim is to have a spanning tree that remains complete in case of the breakdown of a single element. For this reason, the requirements for the spanning trees can be formulated in the following way for the two types of failures:

**R1 Link failure** – For each link, there has to be a spanning tree that does not involve that particular link.

**R2 Node failure** – For each node, there has to be a spanning tree where that particular node is a leaf, i.e. its degree (D) is one.

If these constraints are fulfilled, then there is at least one spanning tree that is not affected by a single failure, thus the traffic can be transmitted on that tree. The spanning trees in Figure 2.2 fulfil the above defined requirements, connectivity is provided despite of the breakdown of any link or node.

Note that the terminology applied in telecommunications networks is used in the following instead of the graph theory terminology in order to apply consistent terminology in the entire dissertation and to avoid confusion, e.g. between the terms edge and edge node. Therefore, node is used instead of the mathematical term vertex, and similarly link is applied instead of edge.

The algorithm proposed to compute the fault tolerant trees is split into two phases according to the failure types addressed, which gives flexibility for the
application of the algorithm. Phase 1 (Algorithm 2.1-1) is invoked first, which
determines spanning trees for handling a link failure. Phase 2 (Algorithm 2.1-2) is
only invoked if a node failure has to be also handled. Phase 2 determines trees in
addition to the result of Phase 1. The high level operation of the algorithm is as
follows:

**Algorithm 2.1. – Determining Fault Tolerant Trees**

**Phase 1 – Algorithm 2.1-1:** Determining spanning trees for link protection

**Step 1.** Select the Central node.

**Step 2.** Construct the first tree from the Central node such that include all possible
links but reserve a link for the second tree at each node if possible.

**Step 3.** Construct further trees until link failure handling criterion R1 is met.

**Phase 2 – Algorithm 2.1-2:** Determining spanning trees for node protection

**Step 4.** Construct further trees until node failure handling criterion R2 is met.

Algorithm 2.1-1 computes spanning trees on the input topology in order to be
able to provide the connectivity among the nodes of the topology such that at
least a tree survives a single failure. The sub-algorithms for the two phases are
described in the following in detail after determining a lower bound for Phase 1.

**A Lower Bound on the Number of Spanning Trees**

Determining the minimal number of spanning trees required to protect a topology
against a link failure is an NP-complete problem as proven by Čičić [33, 34]. Besides
addressing the problem with Algorithm 2.1-1, providing a lower bound for the
number of spanning trees required is very useful to see the characteristic of a
topology.

In order to protect a network against a link failure, its topology has to be 2-
connected and at least two spanning trees are needed. Two spanning trees are only
enough if the physical topology is comprised of enough links to construct two link
disjoint spanning trees. Nonetheless, this condition is typically not met in case of
real network topologies, hence generally more than two spanning trees are needed.

Let \( n \) denote the number of nodes and \( l \) denote the number of links comprising
topology \( G = (N, L) \), where \(|N| = n\) and \(|L| = l\).

**Theorem 2.1.** At least \( k = \lceil \frac{l}{l-n+1} \rceil \) spanning trees are needed in order to provide
protection against a single link failure in topology \( G = (N, L) \).
Proof

The trees are not disjoint if more than two spanning trees are needed to meet the link failure requirement R1. If each link is included in \( k - 1 \) spanning trees, then R1 is met and the links are best utilised in terms of the number of the spanning trees they are included to. Thus, the number of necessary links to form \( k \) spanning trees is:

\[
l = \left\lceil \frac{k}{k-1} (n-1) \right\rceil.
\] (2.6)

Therefore, the lower limit for the number of necessary spanning trees can be calculated as:

\[
k = \left\lceil \frac{l}{l-n+1} \right\rceil.
\] (2.7)

This is a lower bound for the number of spanning trees necessary for handling a link failure as I also described in Farkas [C8]. Typically more spanning trees are needed because the ideal tree construction, i.e. each link is included in \( k - 1 \) trees, is not possible. The minimum average degree, denoted by \( D \), can be then calculated as:

\[
D = \frac{2 \cdot l}{n} = \frac{2 \cdot k \cdot (n-1)}{(k-1) \cdot n}.
\] (2.8)

Note that if the average degree is \( D \) in a topology, then the constraint for fault protection is not necessarily met with \( k \) trees. The minimum of \( D \) is naturally 2 hence the topology has to be two-connected in order to assure any type of resilience. Based on Equation (2.8) the minimum average degree for providing protection against a single link failure by \( k = 2 \) spanning trees is

\[
D = 4 - \frac{4}{n}.
\] (2.9)

That is, the necessary average node degree converges to 4 as \( n \) increases if link protection is aimed to be provided by two disjoint trees.

Determining Spanning Trees for Link Protection

It might be required to determine the trees very quickly after a change in the physical topology, thus, the algorithm has to be fast. Therefore, a heuristic algorithm is provided in the following for Phase 1 of Algorithm 2.1, which determines the trees needed for handling a link failure.

Algorithm 2.1-1 determines the trees needed for handling link failures as specified by the flowchart depicted in Figure 2.8. The algorithm aims to be forward looking as much as possible in order to minimise the number of trees both for link
and node failures. Despite Algorithm 2.1-1 only addresses link failures, the trees are constructed in order to be able to handle node failures as well or at least do not deteriorate the computation of node protection trees if possible. Furthermore, in each step the algorithm aims to make a decision taking into account potential further steps.

The algorithm tries to fulfil the requirement with two spanning trees and only applies additional trees if needed. The construction of the first spanning tree is very important as it determines the possibilities when building the latter trees. Therefore, the first tree strongly influences the number of trees needed to fulfil the requirements. The algorithm first constructs a star-like tree beginning with the node having the largest degree referred to as the Central node (C). Further nodes are then connected to the sub-tree of C until the tree becomes spanning. It is very important that during the construction of the first spanning tree the algorithm tries to reserve a link of each node for the second tree. If the second spanning tree is a disjoint complement of the first one, then no more spanning trees are needed. Otherwise, further spanning trees are computed until each link is protected.

In order to implement the principles described above, the algorithm assigns attributes to the nodes and links of a topology. In step $i$ of the algorithm, the tree under construction is denoted by $T_i$. A weight ($w$) is assigned to each link in the graph, which shows the number of spanning trees where the link is included. Requirement R1 for handling link failures is met if $w < i$ for each link because if $w = i$ for a given link, then the breakdown of that particular link splits each tree. In the algorithm, the following attributes are assigned to each node:

- **degree**: the degree of the node
- **leaf**: shows if there is a tree where it is a leaf, i.e. its degree is 1
- **usage**: the ratio of the total number of links originating from the node in all trees to the degree of the node
- **MAC**: the smallest MAC address belonging to the node.

**Node Selection Process**

Nodes have to be selected based on the above attributes several times during the operation of the algorithm. During the node selection process, the nodes are first categorised as either leaf or non-leaf nodes. Thus, the algorithm for link failures is already able to aid tree construction for node failures if possible. The node selection during the construction of the $k$-th tree is then performed according to the following rules:

1. At the first place, the algorithm tries to choose a node among those that are leaf in any tree $T_j$, $j < i$, thus aiding tree computation for node protection.
As long as it is possible, there are trees that survive the split of the $i$-th tree due to a node failure.

2. If a leaf node cannot be chosen, then a non-leaf node is selected.

If a node has to be selected from either leaf or non-leaf nodes, then the following tie-breaking rules are applied. A node is superior to another:

- if it has higher degree
- if it has smaller usage in case of equal degrees
- or if it has smaller MAC in case of equal usages.

Preferring higher degree nodes gives more chance for constructing a complementary tree in the next step or being able to connect further nodes as leaves. Smaller usage nodes are preferred because the breakdown of a higher usage node affects more trees. Usage is also applied for balancing the number of trees that incorporate the links of a node. Usage also aids providing protection against node failure with the trees applied for link protection. MAC address is only used if the decision cannot be made based on any other attribute.

The operation of Algorithm 2.1-1 depicted in Figure 2.8 is described in the following in detail. First, a so-called Central node (C) is selected for the spanning tree calculations according to the node selection rules described above. C is the starting node for the tree constructions.

There is a special rule only taken into account during the construction of the first tree. If possible, then the algorithm reserves a link from each node for the second tree. If it is not possible then two trees are not enough.

The construction of the first tree begins with a star topology comprising node C and its neighbours except for one. The tree is then spread from this star until it becomes spanning.

During the computation of further trees, all links having weight ($w < i - 1$), i.e. included in less than the already determined trees, are added to the $i$-th tree in the very first step, such that the $i$-th topology is enforced to remain a tree. If the input topology is so dense that two disjoint trees can be formed, then the second tree provided is link disjoint to the first one and tree computation for link protection is ready. In real networks, it is more likely that two trees are not enough for link protection. Therefore, the trees of the forest resulted by the first step have to be connected in order to form the $i$-th spanning tree. The construction of the $i$-th spanning tree is then also started from node C, i.e. from the tree containing node C. Further trees or nodes of the current $i$-th forest are then added one by one. The algorithm selects a node of the tree of C and one of its neighbours belonging to another tree or being isolated, and then adds the link in between them. Nodes
are chosen according to the node selection process. Thus the tree of C spreads until it becomes spanning. Instead of adding isolated nodes, adding other trees to the tree of C is preferred because the links of those trees meet requirement R1. The links for connecting isolated trees or nodes are selected such that R1 is maintained, i.e. they have to meet the $w < i - 1$ criterion. If no new link can be added to the spanning tree under construction insisting on the weight criterion, then a link is added ignoring it. If the weight criterion is violated, then another tree is necessary to fulfil the requirement for protection against link failures. New trees are constructed until requirement R1 is met, i.e. there is no link for which $w = i$. 

Figure 2.8: Algorithm 2.1-1 – Spanning tree calculation for link protection
Determining Spanning Trees for Node Protection

Phase 2 of Algorithm 2.1 determines the additional spanning trees needed to handle a node failure is specified by its flowchart in Figure 2.9. First of all, the algorithm checks whether or not there is a need for another tree according to R2, i.e. whether there is any node that is not a leaf in any of the trees. If so, then further trees are constructed until R2 is met.

During the construction of a new tree, a maximum forest is first constructed covering the nodes that are already leaves in a former tree. This forest is a set of non-spanning trees in most of the cases. The spanning tree is then constructed from the tree of C such that other isolated trees are added first and isolated nodes are then connected. At the first place, an isolated tree or node is connected to a node that is already leaf in one of the former spanning trees. If it is not possible to connect an isolated tree or node to one of the former leaf nodes, then it is connected to a non-leaf node, which implies the need for another spanning tree. This is repeated until each node becomes leave in one tree at least.

There is an advantage of separating the calculation of trees for the two types of failures. If one does not want to protect its network against node failures but only for link failures, then it is enough to build up the VLAN-trees according to the output of the first algorithm. Thus, the number of VLANs can be reduced at the price of robustness.

Figure 2.9: Algorithm 2.1-2 – Spanning tree calculation for node protection
Evaluation of the Tree Computation Algorithm

The algorithms described above are applicable for arbitrary topologies and they result in near-optimal solution in terms of the number of necessary spanning trees for any network topology. A numerical evaluation of the algorithms is given in the following.

The algorithms were first evaluated on the quite common grid topology. Grid is a dense topology, an $A \times A$ grid is comprised of $n = A^2$ nodes and $l = 2A(A - 1)$ links. Based on Equation (2.7), the minimal number of trees needed to be protected against a link failure is

$$k = \left\lceil \frac{2A(A - 1)}{2A(A - 1) - A^2 + 1} \right\rceil = \left\lceil \frac{2A}{2A - (A + 1)} \right\rceil = \left\lceil 2 + \frac{2}{A - 1} \right\rceil = 3 \text{ if } A \geq 3. \quad (2.10)$$

That is, at least 3 spanning trees are needed to make 3x3 or larger grids resilient to a link failure.

Algorithm 2.1 was evaluated on various size grid networks, 50 times on each. Figure 2.10 shows an example output of Algorithm 2.1 for 4x4 grid, where four trees are enough for both link and node failure protection. The numbers in the nodes show the order of their $MAC$ parameter.

![Resilient spanning trees for 4x4 grid](image)

Figure 2.10: Resilient spanning trees for 4x4 grid

The evaluations have shown that Algorithm 2.1-1 always results in three spanning trees for handling link failures in case of 3x3 or larger grids. Algorithm 2.1-2 adds at most two more spanning trees for avoiding the effect of node failures independently of the size of the grid. Thus, Algorithm 2.1 provides resilience for grid
networks by using at most five spanning trees. That is the traffic between any source and destination pair within the grid can be carried using one of the five VIDs. Note that the algorithm proposed by Sharama [17] requires 38 VIDs in an 8x8 grid for 500 source destination pairs.

Algorithm 2.1 was also evaluated on random graphs, of which topology is randomly generated at given number of nodes and average node degree. The algorithm was evaluated with several node and average degree settings, 50 simulations for each setting. The number of nodes in the random topologies varied between 5 and 50; the average degree increased from 2.5 to 5. Figures 2.11-2.13 show the number of necessary spanning trees as a function of number of nodes and their average degree. Figure 2.11 shows the lower bound in order to be able handle link failures, which is calculated according to Equation (2.7) but not performing the ceil operation. Figure 2.12 shows the average of the 50 results calculated by Algorithm 2.1-1 for handling link failures. Figure 2.13 shows the average number of trees calculated by Algorithm 2.1 for handling both link and node failures.

The figures show that the results provided by Algorithm 2.1-1 are near to the lower bound. For example, in case of network topologies consisting of 30 nodes whose average degree is 3.5, the lower bound for handling link failure is 2.85 in Figure 2.11. Algorithm 2.1 provided 3.08 for handling link failures and 4.92 for handling both link and node failures as shown in Figure 2.12 and in Figure 2.13, respectively. As shown by the figures, the algorithm provided a near-optimal solution in the number of necessary spanning trees. For topologies having at least 2.8 average degree, Algorithm 2.1-1 provided as many spanning trees for link protection as the lower bound or only a single additional one. Note, that a two-connected topology with the average degree of 2 is a ring topology.

Similarly to the Ethernet architecture described above and published in Farkas [C8], the idea of providing resilience at IP layer using fault tolerant trees was raised at the same time by Menth [35]. The IP resilience scheme relies on multi-topology routing extensions to link state routing protocols [36, 37]. An upper bound on the number of trees needed in order to provide tolerance against a single link failure was given later by Čičić in [38], where results of a heuristic algorithm were also published. Table 2.2 provides a comparison of the results published in [38] for 16-node networks and the results provided by Algorithm 2.1-1 and the lower bound provided by Equation (2.1). The results provided for the algorithms are not integer because they are the average of several runs of the algorithms. Algorithm 2.1-1 always provided either the ceil or the floor of the average value. The upper bound provided in [38] is in fact the Largest Minimal Cycle (LMC) in the topology, which was determined for each random topology of the multiple runs. The table provides the average of the LMCs in the different random topologies.
Necessary Condition

Figure 2.11: Lower bound of the number of spanning trees necessary for link protection

Handling Link Failure

Figure 2.12: Average number of spanning trees of Algorithm 2.1-1
As the table shows, Algorithm 2.1-1 slightly outperforms the algorithm of Čičić [38]. Both algorithms provide results closer to the lower bound than to the upper bound, i.e. both tend to minimise the number of trees for handling link failures. Further results published by Čičić in [38] show even larger gap between the upper bound and heuristic results. As a consequence, the lower bound provides more accurate picture on the characteristics of the topology than the upper bound.

Table 2.2: Number of trees required for handling a single link failure in 16-node networks

<table>
<thead>
<tr>
<th>Average node degree</th>
<th>4</th>
<th>4.4</th>
<th>4.8</th>
</tr>
</thead>
<tbody>
<tr>
<td>Upper bound of Čičić [38]</td>
<td>4.6</td>
<td>4.5</td>
<td>4.2</td>
</tr>
<tr>
<td>Average of heuristic algorithm of Čičić [38]</td>
<td>2.6</td>
<td>2.5</td>
<td>2.2</td>
</tr>
<tr>
<td>Average of Algorithm 2.1-1</td>
<td>2.4</td>
<td>2.18</td>
<td>2.16</td>
</tr>
<tr>
<td>Lower bound: Equation (2.1)</td>
<td>2</td>
<td>2</td>
<td>2</td>
</tr>
</tbody>
</table>
2.5 Physical Topology Discovery

The information on the exact physical topology of the network is needed in order to be able to determine the fault tolerant VLAN-trees. Furthermore, automatic discovery of the physical topology is essential for many OAM tasks including traffic engineering, resource management and fault management. The topology discovery has to be accurate even in heterogeneous multi-vendor networks.

I proposed a physical topology discovery algorithm in Farkas [C3], which accommodates the bridge network such that it takes into account the protocols implemented by the bridges comprising the network. The high level description of the algorithm is as follows.

Algorithm 2.2. – Physical Topology Discovery

\underline{Step 1 – LLDP discovery:} Discovery of the topology segment that implements the Link Layer Discovery Protocol (LLDP) [26], which is the standardized support for topology discovery.

\underline{Step 2 – Node discovery:} The nodes that do not support standard topology discovery, i.e. non-LLDP bridges, are determined by the replies to a broadcast ping issued by the Network Management System (NMS) implementing the topology discovery algorithm.

\underline{Step 3 – Spanning tree discovery:} Based on the ping messages and replies, the links comprising the spanning tree that carries management traffic among non-LLDP bridges are determined by the NMS.

\underline{Step 4 – Inactive link discovery:} The links span among non-LLDP bridges and not included in the spanning tree are then finally determined.

The algorithm is able to discover the entire physical topology including bridges that do not support any type of already standardised topology discovery protocol. The algorithm requires at least STP or RSTP, VLAN tagging, SNMP, Bridge MIB [39], MIB-II [40] and Interface MIB [41] to be implemented in the bridges. Note that, when the bridges are plugged at network start-up, RSTP is enabled on them in order to provide a basic connectivity and prevent loops, which is the default configuration of an Ethernet bridge.

The steps of the algorithm are described in detail in the following.
LLDP-based Discovery

The algorithm first discovers the physical topology among the bridges implementing LLDP. In order to achieve this, the Network Management System (NMS) implementing the topology discovery algorithm queries the LLDP MIB by means of SNMP in a sequential order. That is, the NMS queries its neighbour first, includes the retrieved data into the topology database (DB), then the NMS continues retrieving the MIBs from the neighbours of those bridges that are included in the topology DB and have LLDP capable neighbour.

The difficult part comes after the LLDP discovery, i.e. to determine the connectivity among the nodes yet hidden.

Node Discovery

The NMS issues a broadcast ping message to the sub-network broadcast address and waits for the replies in order to discover all other bridges that do not implement LLDP hence not yet included in the topology database. The replies provide the information on those nodes that do not implement LLDP. In order to avoid the effect of lost ping messages the node discovery may be repeated a number of times, e.g. three times.

Having each node included in the topology database, the next step is to discover the links among them.

Spanning Tree Discovery

Due to the broadcast ping and corresponding reply messages, some MAC addresses are learnt and stored in the Filtering Database (FDB) of the bridges, which can be queried by means of SNMP. The NMS queries the FDB of bridges that are not yet included in the topology database. The spanning tree is not obvious from the FDBs retrieved as they are incomplete. The flowchart of the spanning tree discovery algorithm is depicted in Figure 2.14, which determines the spanning tree set-up in the network to provide the basic connectivity, e.g. for management traffic.

The FDBs are automatically filled in with MAC addresses during the node discovery as a result of address learning. In this reverse path learning process, a bridge associates the source MAC address of an incoming frame to the reception port. Entries are added to the FDB due to the broadcast ping sent out by the NMS and the replies to it. The set of the FDBs of all bridges provides the information for the calculation of the spanning tree.

The algorithm assigns a weight \((w)\) parameter to each node, which helps determining the spanning tree. The weight of a node equals to the number of entries contained in the FDB. Thus, the more destination addresses are added into the FDB, the heavier the node. Therefore, the weight of a bridge correlates to the
distance between that bridge and the NMS in the spanning tree because bridges closer to the NMS receive replies to the broadcast ping from more bridges hence their FDB contains more entries. Thus, the heavier the bridge, the closer to the NMS. Note that the node directly connected to the NMS is always the heaviest since all the ping replies go through it before reaching the NMS. Note that the algorithm disregards any MAC address in the FDBs that is not included in the topology database as a MAC address of a node, thus background user traffic cannot disturb the discovery procedure.

The final step of the spanning tree discovery determines how each bridge is connected to the others in the tree. Thus eventually, the spanning tree is determined. Among the nodes reachable from an investigated bridge port according to the FDB, the algorithm adds the heaviest bridge as the one directly connected to the given bridge port. In other words, the algorithm determines the physical neighbour.

To determine which port of the physical neighbour is connected to the investigated bridge port, the algorithm looks up for the port on the physical neighbour’s FDB that points to the NMS. Eventually, this iteration process results in the discovery of the complete spanning tree.

Nevertheless, there might be still undiscovered links at the end of this stage, which are the target of the next step.

**Inactive Link Discovery**

As spanning tree protocols may block a link to avoid loops, the topology database is still not complete. There might be physical connections among non-LLDP bridges
not taking part in the spanning tree. Turning off a bridge port triggers an SNMP trap message, which is sent by both ports connected to the link that went down even if the link is not part of the spanning tree. Taking advantage of this feature, the NMS logs in the non-LLDP bridges and one by one turns down the ports not yet included in the topology database. The flowchart of the inactive link discovery algorithm is shown in Figure 2.15. Note that turning down and up a link that is not part of the spanning tree neither causes re-convergence in the spanning tree protocol nor disturbs user traffic.

By the end of this phase, the entire physical topology is discovered including all nodes and links comprising it.

**Physical Topology Monitoring**

The physical topology discovery algorithm is extended for updating the topology database in case of a change in the physical topology. Note that the update of the topology database does not have to be fast because FHP does not rely on it. In fact, the topology database has to be only updated if the change is considered to be permanent and the trees have to be recomputed. Nevertheless, it is better to avoid of rerunning the entire discovery mechanism but discover changes only. The flowchart of the physical topology update algorithm is depicted in Figure 2.16.

The first step is to determine whether already detected nodes have been removed from the network. To do so, the algorithm issues a unicast ping request...
message to each of the detected nodes for a pre-defined number of times (three times in the figure), with a pre-defined time interval between them. Failure in receiving an expected ping replies leads to the removal of the corresponding node and its connections from the database.

The second step verifies the status of each interface of every detected bridge via SNMP. If the status of an interface changed from up to down, then that interface and the corresponding link are simply removed from the database. If an interface’s status changed from down to up, the algorithm adds the corresponding port to the database and turns this port down and then up. Note that the corresponding link is new and not yet used as it is not yet added to any VLAN-tree. If the NMS gets the link down traps from both nodes at the end of that link, then it means that the peer node is already known. Therefore, the algorithm determines, from the received traps, that a new connection between two already detected nodes has been created and adds both ports to the database. If the NMS does not get a link down trap from the peer node, then the algorithm concludes that a new node or segment has been added to the network.

The algorithm adds the port that leads to this new node to the management VLAN, thus the node becomes reachable by the NMS. After that, the algorithm runs the node, the spanning tree and the inactive links discovery phases again, with the exception that they are now limited to the newly discovered nodes and the network segments they form.
Evaluation of the Topology Discovery Algorithm

The physical topology discovery algorithm was tested in a prototype implementation. The NMS was implemented on a Linux PC and the network comprised of Cisco Catalyst 2950, D-Link DES-3526, Extreme Networks Summit 200-24e, HP ProCurve, 3400cl, Nortel BES 110-24T Ethernet bridges.

Various physical network scenarios involving up to 12 nodes with mesh topologies have been set up and tested. The collected results showed that the topology discovery mechanism is capable of accurately determining the entire physical topology of the network. The evaluation showed, the algorithm is able to discover the entire physical topology including bridges that do not support any type of topology discovery protocol. Details on the evaluation were published in Farkas [C3].

2.6 Conclusions

This chapter presented and experimentally validated a lightweight and efficient protection technique providing a robust Ethernet architecture. The network is comprised of standard Ethernet bridges in its core and implements the distributed fault handling features in its edge nodes.

An automatic physical topology discovery algorithm is proposed for heterogeneous multi-vendor Ethernet networks that is able to determine the entire physical topology of the network, including the links inactivated by a spanning tree protocol, and keep the topological view updated as links and nodes fail, are added or removed. The topology discovery method provides the input that is necessary for determining fault tolerant forwarding paths.

The forwarding is implemented by means of VLAN-trees in the proposed architecture. An algorithm has been proposed for the construction of the VLAN-trees necessary to survive any single failure. The algorithm aims to determine as few trees as possible on arbitrary topologies. Minimising the number of trees is advantageous due to the limited number of VLAN IDs, in order to keep the monitoring traffic of VLAN-trees low and to ease network maintenance. The application of the algorithm is not limited to Ethernet, it can be applied if trees are used for the forwarding topology.

A distributed fast failure handling protocol was also defined, which monitors the availability of the VLAN-trees, detects failure and reparation events, furthermore, controls traffic redirection from one VLAN-tree to another.

The proposed high availability architecture was implemented in a prototype network, and its components were evaluated. The protocol performance and robustness were validated by means of extensive testing. Experimental results showed that worst case failover can be maintained below 50 ms with low protocol overhead, i.e. carrier grade failover is provided.
Chapter 3
Enhancements to Shortest Path Bridging

3.1 Introduction

Ethernet networks deployed before 2008 typically employ Rapid Spanning Tree Protocol (RSTP) [15] or Multiple Spanning Tree Protocol (MSTP) [1] for the control of data paths. Thus, the forwarding from the source to the destination may not be on the shortest path, which results in waste of bandwidth. Furthermore, making use of multiple equal cost paths is not supported.

Therefore, the IEEE 802.1aq Shortest Path Bridging (SPB) standard [6] specifies a link state control protocol for bridges. SPB uses the ISO standard Intermediate System to Intermediate System (IS-IS) routing protocol [5], as it is ideally suited to provide Layer 2 control. The introduction of a link state protocol allows the exploitation of all the possibilities provided by a network topology, e.g. improved bandwidth efficiency by shortest path forwarding. Furthermore, multipoint services, such as E-LAN and E-TREE [42], can be more easily set up by using the auto-discovery of topology and interest of service provided by a link state protocol.

SPB provides frame forwarding on the shortest path within a region of a network. SPB sets up and maintains at least one Shortest Path Tree (SPT) for each SPT Bridge, which connects to every other bridge of an SPT Region. Each bridge roots at least one SPT and uses its own SPT for transmission of frames received from outside of the Region. Thus, SPB implements source rooted trees, which is ideal for multicast forwarding and applications using it, e.g. IPTV. An SPT is either identified by a VLAN ID (SPBV) or by the MAC address of the bridge rooting the SPT (SPBM). In addition, SPB changes the multicast forwarding principle from $(*,G)$ to $(S,G)$ for proper frame delivery, where $G$ denotes a multicast group and $S$ is the source. That is, source specific multicast forwarding is applied, which is described more in detail in Appendix A.1.
The application of a link state protocol in a bridge network adds new constraints and raises issues to be solved. For instance, it is essential to avoid loops in an Ethernet network, so as to prevent the multiplication of flooded unknown destination, broadcast and multicast frames. If a link state protocol is used for the control of the network, then transient loops may appear because of network nodes having different views on the physical topology after a topology change.

There is a need for a loop prevention mechanism in case of the link state controlled SPB as loop free operation at all times is an absolute requirement in Ethernet networks. Therefore, SPB has to incorporate a mechanism that prevents loops irrespective of the number of link state updates in progress and the order of their arrival and inclusion in the link state computation at each bridge. Note that some loop prevention mechanisms have been proposed for IP Fast Re-Route (IPFRR). However, as summarised by Shand [43] none of these is a pure control protocol approach being able to handle multiple failures in a reasonable time. Thus they are not applicable in SPB.

This chapter describes two extension alternatives to the standard link state operation in order to provide control for SPB networks in Section 3.2 and in Section 3.4. The two proposals are then evaluated and compared in Section 3.3 and in Section 3.5.

### 3.2 Neighbour Synchronisation for Loop Prevention

All stable forwarding topologies are loop free both in Ethernet and IP networks, however, loops may occur during topology transients. The application of a loop mitigation mechanism was proposed for SPB in order to treat transient loops. Reverse Path Forwarding Check (RPFC) can be applied to audit the port of arrival of a frame in order to ensure that it arrives on the port lying on the shortest path from the source. Note that RPFC is referred to as ingress check in the SPB specification [6].

I have shown that RPFC does not ensure the avoidance of loops. Figure 3.1 shows an example for a loop occurring despite of RPFC, which was published in Farkas [S7] and later referred to as Farkas loop e.g. by Allan et al. [44].

The figure shows only a part of a topology, there might be further nodes connected to B, C, D or E. Nevertheless, the figure shows all the links connecting the nodes depicted in the figure and also indicates the cost of each link. The solid line links are active in the SPT of A but the dash dot line links are inactivated, e.g. by frame discarding implemented by the ingress check. The little arrows show the direction of frame forwarding on the SPT of node A.

There is a topology change in the initial stage of the example: the physical
connection between A and B is cut and a new physical connection appears between B and E at the same time. The initial and the final topologies are loop free as illustrated in the figure. The link between A and D is not used in the initial topology; and the link between C and D is unused in the final one. Nonetheless, a loop is formed during the transient if nodes A, B and E are aware of the change thus have an updated view on the topology but nodes C and D have an outdated view. The loop appears even if ingress checking is applied. As a consequence of the loop, multiple copies of a multicast or broadcast frame may be spread towards other nodes connected to B, C, D or E as shown by the little arrows. Note that the Time To Live (TTL) field applied in IP packets is a weaker loop handling technique than RPFC. It is not possible to set exact TTL for each destination in case of multicast and broadcast packets. Therefore, if TTL is the only loop treatment mechanism applied, then multicast and broadcast packets are circulated and multiplied in a loop thus several copies are spread out from the loop, which are then received by destinations close enough to the loop. That is TTL does not prevent the appearance of loops, but provides means to live with them.

**Topology Database Synchronisation between Neighbours**

I have proposed a generic and simple mechanism for loop prevention in Farkas [S6], which can be implemented as an extension to any link state protocol thus applicable either in link state controlled Ethernet or IP networks. The basic idea is to avoid trespassing between forwarding topologies set up based on different views on the physical topology, which I also described in Farkas [J2, P3]. That is, neighbour nodes only exchange data packets if they have the same view on the physical topology, i.e. their topology database is synchronised. In order to make sure that they have the same topology view, neighbour nodes need to exchange some additional information very quickly right after being notified about a topology change. The exchange mechanism may be in a form of a handshake. Note that, there is no global synchronisation or ordering of topology updates, this is only
a local neighbour to neighbour synchronisation independently of the rest of the network. If the two neighbours have the same topology view, then the link between them cannot be part of a loop.

Basically, there are two ways to ensure that the two topology databases are the same. After the reception of a Link State PDU (LSP) describing a topology change, the neighbours may check that both of them have got the same LSP by means of exchanging its LSPID. Alternatively, the neighbours may exchange a digest of their topology database after being notified on a topology change. The topology digest has to be produced by a well defined algorithm on a pre-determined subtract of the Link State Database (LSDB) or on an agreed description of the physical topology. Both type of information may be exchanged in Hello PDUs. The two approaches provide the same result; the topology digest based approach is applied in the rest of the dissertation.

As topology database synchronisation is performed with the neighbours independently of each other, different ports may have different synchronisation states. The state of a port of a node can be described by the two-state state machine illustrated in Figure 3.2. The names of the states reflect whether or not the node is synchronised with its neighbour connected through the given port.

The port is in Sync if the LSDB of the two nodes is the same, otherwise it is in Non-Sync. The port remains in Sync until the node is not notified about any topology change or the neighbour node does not send a topology digest differing from the locally stored one. If any of these two happens, then the state of the port changes to Non-Sync. The port remains in Non-Sync as long as the two neighbours have mismatching digests. As soon as they have matching digests again, the port moves to the Sync state. If the state of the port is Non-Sync, then it blocks data communication to its neighbour connected through the given port. That is, the frames or packets aimed to be sent to the neighbour or received from the neighbour are either dropped or buffered. Data communication only operates if the port is in Sync state.

Let us assume that a two-way handshake mechanism is used to implement the topology database exchange. Synchronisation Request (Syn Req) messages are used to notify a neighbour on an update in the topology. The neighbour replies a Synchronisation Acknowledge (Syn Ack) message if it has the same topology. Covered.
database. Both Syn Req and Syn Ack messages contain the topology digest and can be implemented in Hello PDUs. There are three events that require action from the Neighbour Synchronisation algorithm: a change in the topology, the reception of either a Synchronisation Request or a Synchronisation Acknowledgement message. The flowcharts specifying the operation for these events are depicted in Figure 3.3.

If a node receives an LSP containing information on a topology change, then it first determines the new topology digest as shown in Figure 3.3(a). Non-Sync state is then set on all ports having different remote digest than the recently computed local one. The new topology digest is then sent out in Synchronisation Request messages on each port. There is a single process for the algorithm depicted in Figure 3.3(a) in each node.

Figure 3.3(b) shows the operation at the reception of a Synchronisation Request, which is a per port process. The remote topology digest carried in the Syn Req is first stored by the port. It is then checked whether or not the local and the remote topology digests are the same. If not, then the state of the port becomes Non-Sync. If they are the same, then the port state is set to Sync. The port then checks whether or not it had already sent out a Synchronisation Request with the same topology digest received from the neighbour. If yes, then nothing else is to be done as the neighbour is also aware of having the same topology digest. Otherwise, a Synchronisation Acknowledgement carrying the local topology digest is sent to the neighbour in order to notify it on having the same topology database.

The operation at the reception of a Synchronisation Acknowledgement is depicted in Figure 3.3(c), which is also a per port process. Acknowledgement can

![Figure 3.3: Operation of Neighbour Synchronisation](image-url)
only be received by a port if it formerly issued a Request carrying the same topology digest. Nevertheless, it is double checked that the remote and the local digests are the same before the port state is set to Sync. If they happened to be different for some reason, then the remote digest is stored as the most recent one and the port state becomes Non-Sync.

The operation described above ensures that data packets are not exchanged between neighbours if their topology view is not synchronised. The operation in case of applying a three-way or "one-way" handshake is the same in principle as described above.

The operation of Neighbour Synchronisation for a generic example assuming random LSP updates is explained in Appendix A.2. Neighbour Synchronisation prevents the appearance of Farkas loop as shown in Appendix A.3.

Analysis

The Neighbour Synchronisation mechanism ensures that there is no trespassing between different topologies, which is illustrated in Figure 3.4. Node A belongs to topology k as it only received the LSPs describing topology k. As opposed to this, node B belongs to topology k+1 as it has received one or more LSP on topology information which differs from topology k. Note that it may happen that multiple LSPs describe the same topology change. As A and B have different topology views, the link between them is blocked by the Neighbour Synchronisation.

![Figure 3.4: Topology separation](image)

If a packet to be forwarded to B received by A on topology k, then A may perform two actions on the packet. A either drops the packet or stores it in a buffer and forwards it to B when they belong to the same topology again. If buffering is applied, then a packet may pass through a topology update, i.e. from an older topology to a newer one.

**Theorem 3.1.** *The Neighbour Synchronisation algorithm prevents the appearance of loops in the forwarding topology.*

**Proof**

Let us assume that there is a forwarding loop in a 4-node ring despite of the Neighbour Synchronisation algorithm. Figure 3.5 shows a 4-node ring topology where each node is connected to its neighbour by a single physical link.
This 4-node ring is an appropriate model for the looping behaviour because the operation is the same for larger rings, i.e. potential loops. Nodes A and B have topology view $\alpha$ and nodes C and D have topology view $\beta$. The forwarding behaviour is shown by the solid lines for topology $\alpha$ and by the dashed lines for topology $\beta$. That is, both A-C and B-D links are incorporated in both forwarding topologies. Loop does not exist within any topology, i.e. within $\alpha$ or $\beta$ or if $\alpha = \beta$, because the forwarding is always along a tree within a consistent topology. A loop only exists if frames or packets are sent around by the 4 nodes. That is, Node A sends a frame to Node B on topology $\beta$ and Node B forwards it to Node D on either topology $\alpha$ or $\beta$. Node D and C then send the frame back to Node A according to topology $\beta$ thus Node A sends it to Node B again. However, this leads to a contradiction with the initial assumption because all nodes block their port 2 as it connects them to a node having a different topology view as $\alpha \neq \beta$. That is, the Neighbour Synchronisation algorithm prevents the appearance of forwarding loops.

The Neighbour Synchronisation algorithm ensures that packets do not trespass different forwarding topologies. Nevertheless, if instead of dropping, packets are buffered meanwhile the neighbours are not in synch and forwarded as soon as they have the same topology view, then the same packet may be transmitted multiple times by the same node in case of unlucky timing of topology change events as illustrated in Figure 3.6.

Source node S sends packets to destination node D. The C cost of each link is also indicated in the figure, where the ID of the link is the same as the exponent for links having larger cost than $2^0$. The ID of those links also gives the ID of the topology because in the i-th topology the i-th link goes down, of which cost is $2^i$. In the initial $T_0$ topology, there are $k$ paths between nodes S and D. S sends packet f to P and P sends it to Q in order to reach D on the shortest path. The actual forwarding is indicated by the blue arrow line. Nevertheless, link 1 goes down at the same time Q has received the packet, thus the topology changes to $T_1$. Q may buffer packet f until its topology view is not synchronised with R and send packet f to R instead of D as that is the actual shortest path towards D. Nonetheless, link
2 goes down as soon as R receives packet f, hence the topology becomes $T_2$. Thus, R sends buffered packet f back to node P. This unlucky sequence of failures may keep going, i.e. links go down in the order of their cost exponent as illustrated by Figure 3.6. The final topology is $T_k$, where P is finally able to send packet f to D on link k. Nevertheless, before that, packet f was circulated by nodes P, Q and R, therefore, packet f passed node P multiple times before it could reach its destination.

**Definition 3.1.** The state of a packet towards its destination is defined by

$$\langle X, D \rangle, \quad (3.1)$$

where
- $X$ is the number of topology changes in the network minus the number of topology updates the packet has passed through and
- $D$ is the distance to the destination in the number of hops remaining to the destination within the current topology.

Lexicographic order can be applied for the states:

$$\langle X_1, D_1 \rangle > \langle X_2, D_2 \rangle \equiv \langle X_1 > X_2 \rangle \lor \langle X_1 = X_2 \land D_1 > D_2 \rangle, \quad (3.2)$$
That is, state 1 is greater than state 2 if the packet in state 2 has been passed through more topology updates than the packet in state 1 or if the packets in both states are within the same topology and the distance in state 2 is smaller than the distance in state 1.

The state of a packet is (X,0) when it reaches its destination. If X=0 too, then the packet had not been passed from a topology to another before it reached the destination, i.e. it did not pass any topology update. If X=k, then the packet has been affected by k topology changes as it had to go through k topology updates, which means that the packet might been forwarded along k+1 different topologies.

An optimal loop prevention algorithm ensures that a particular packet is not forwarded more than k+1 times by any nodes if there were no more than k topology changes in the network.

**Theorem 3.2.** The Neighbour Synchronisation algorithm provides optimal loop prevention even if packet buffering is applied instead of packet dropping because it ensures that each packet gets to its destination such that it is transmitted by a node at most k+1 times if there are no more than k topology changes in the network.

**Proof**

Neighbour Synchronisation ensures that the state of a packet is always strictly decreasing, i.e. either X or D decreases in the state of a packet following a former one, otherwise the two states are the same.

It has been shown in Figure 3.4 that no packet can be exchanged between nodes being in different topologies. Nodes only exchange data packets if they have the same topology view, i.e. they are part of the same topology.

As described above in detail, each packet is always forwarded along a tree and transmitted by a node only once within a given topology.

Due to buffering, a packet may move from a topology to a more recent one if the node storing the packet is updated to a more recent topology.

If there are k topology changes in the network, then a packet may be passed to an updated forwarding topology at most k times. Thus, a packet may be forwarded at most along k+1 topologies.

Therefore, the Neighbour Synchronisation method ensures that a packet is forwarded at most k+1 times by a node, thus it provides optimal loop prevention.

I proposed the Neighbour Synchronisation algorithm in Farkas [S6] to IEEE 802.1 for loop prevention in SPB. The loop prevention algorithm specified in the standard is referred to as Agreement Protocol and it relies on the basic idea of Neighbour Synchronisation, i.e. on topology database synchronisation between neighbour nodes (see e.g. Clause 13.17 of [6]).
3.3 Evaluation of Neighbour Synchronisation

Suspending packet exchange between neighbours adds some delay to network convergence time after a change in the topology. The question is how much the Neighbour Synchronisation algorithm increases the convergence time. Note that convergence time is also referred to as recovery time if the topology change is a failure event. In order to investigate this issue, the operation of the Neighbour Synchronisation algorithm was evaluated by means of measurements and simulations and I published the results in Farkas [S1].

Simulation Based Analysis

The operation of the Neighbour Synchronisation algorithm was analysed in a packet simulator developed in OMNeT++ 4.0 [45], which is a discrete event object oriented C++ simulator environment. The bridge architecture was implemented according to the IEEE 802.1Q [1] specification in the INET framework of OMNeT++. IS-IS was then implemented as a Higher Layer Entity of the bridge architecture along the ISO specification [5]. The Neighbour Synchronisation algorithm was then implemented as described in Section 3.2. Table 3.1 summarises the parameter settings applied during the simulations. The default values specified by the IS-IS standard are also given.

The `helloInterval` determines the interval between the regularly issued two consecutive IS-IS Hello (IIH) PDUs. Note that if an IIH PDU has to be issued because of a change, then there is no need to wait the end of the hello interval. If a node does not receive the IIH PDU from its neighbour within the `holdingTime`, then the node considers the connectivity broken. The LSPs are stored in the LSDB for `maxAge` time. If the LSP is not refreshed within `maxAge`, then it expires and it is removed from the LSDB. The time elapsed before re-sending an LSP in order to refresh it

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Default value</th>
<th>Simulation value</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>helloInterval</code></td>
<td>1000 ms</td>
<td>1000 ms</td>
</tr>
<tr>
<td><code>holdingTime</code></td>
<td>2000 ms</td>
<td>2000 ms</td>
</tr>
<tr>
<td><code>maxAge</code></td>
<td>1200 s</td>
<td>1200 s</td>
</tr>
<tr>
<td><code>minLSPGenerationInterval</code></td>
<td>30 s</td>
<td>30 s</td>
</tr>
<tr>
<td><code>maxLSPGenerationInterval</code></td>
<td>900 s</td>
<td>900 s</td>
</tr>
<tr>
<td><code>minLSPTransmissionInterval</code></td>
<td>1000 ms</td>
<td>0 - 1000 ms</td>
</tr>
<tr>
<td><code>lspThrottle</code></td>
<td>33 ms</td>
<td>33 ms</td>
</tr>
<tr>
<td><code>spfDelay</code></td>
<td>1000 ms</td>
<td>0 - 1000 ms</td>
</tr>
<tr>
<td><code>C</code></td>
<td>1.2 · 10^{-6} s</td>
<td>1.2 · 10^{-6} s</td>
</tr>
<tr>
<td><code>linkDelay</code></td>
<td>1 ms</td>
<td>1 ms</td>
</tr>
<tr>
<td><code>jitter</code></td>
<td>25 %</td>
<td>25 %</td>
</tr>
</tbody>
</table>
is between the \( \text{minLSPGenerationInterval} \) and the \( \text{maxLSPGenerationInterval} \). In general, the \( \text{minLSPTransmissionInterval} \) has to be elapsed between two consecutive LSPs issued by the same node. Furthermore, the \( \text{lspThrottle} \) interval has to elapse between two consecutive LSPs sent out on the same interface. After finishing a Shortest Path First (SPF) computation a node waits the \( \text{spfDelay} \) interval before starting another SPF computation. These are the parameters defined by the standard. The last three parameters are only used by the simulator. The time required for the SPF computation is estimated according to Equation 3.3.

\[
\text{SPF Computation} = C \cdot (L + N \cdot \log N),
\]

where \( N \) and \( L \) denote the number of nodes and links in the network, respectively. The value of the \( C \) parameter was set based on the results of Francois [46]. Fast-Ethernet links were assumed during the simulations with a propagation delay of 1ms, which stands for the message sending and receiving. The propagation and message processing delay are determined by the \( \text{linkDelay} \) and the \( \text{jitter} \) parameters in the simulator. As it was pointed out e.g. by Francois [46] the convergence can be accelerated by using smaller interval values than the default ones specified by the standard. The IS-IS convergence had to be accelerated in order to be able to determine the additional delay caused by the loop prevention mechanism. Therefore, IS-IS convergence was fine tuned by setting the \( \text{minLSPTransmissionInterval} \) and the \( \text{spfDelay} \) to zero in most of the simulations.

The simulation analysis was performed on six topologies: the 22-node AT&T [47], the 37-node COST266 [48] reference topology widely used in European projects, a 50-node German backbone network [47] and three artificially constructed networks. One of them is a 71-node network referred to as Rings because it comprises a 11-node inner ring and the rest of the 60 nodes form sub-rings connected to the inner ring as illustrated in Figure 3.13(a). In addition, 100-node and 150-node random topologies referred to as R100 and R150 were used, which were generated by the BRITE tool [49] based on the Waxman model. The diameter of the applied topologies is summarised in Table 3.2.

Besides verifying the prevention of loops, the additional delay caused by the Neighbour Synchronisation handshake is in the focus of the following evaluation. The network convergence time after a single link or node failure was investigated, which is the time elapsed from a failure event realised by IS-IS until the forwarding has been recovered, i.e. all LSPs affecting the forwarding have been processed and the new forwarding topology has been set. That is, a physical layer upcall was

<table>
<thead>
<tr>
<th></th>
<th>AT&amp;T</th>
<th>COST266</th>
<th>Germany50</th>
<th>Rings</th>
<th>R100</th>
<th>R150</th>
</tr>
</thead>
<tbody>
<tr>
<td>Diameter</td>
<td>5</td>
<td>8</td>
<td>9</td>
<td>3</td>
<td>7</td>
<td>7</td>
</tr>
</tbody>
</table>

Table 3.2: Network diameters

44
Table 3.3: Convergence time after a link failure [ms]

<table>
<thead>
<tr>
<th></th>
<th>AT&amp;T</th>
<th>COST266</th>
<th>Germany50</th>
<th>Rings</th>
<th>R100</th>
<th>R150</th>
</tr>
</thead>
<tbody>
<tr>
<td>min</td>
<td>2667</td>
<td>5305</td>
<td>4941</td>
<td>2320</td>
<td>3898</td>
<td>4921</td>
</tr>
<tr>
<td>max</td>
<td>4281</td>
<td>6050</td>
<td>5899</td>
<td>3517</td>
<td>5321</td>
<td>6092</td>
</tr>
<tr>
<td>avg</td>
<td>3641</td>
<td>5767</td>
<td>5472</td>
<td>3042</td>
<td>4593</td>
<td>5499</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>AT&amp;T</th>
<th>COST266</th>
<th>Germany50</th>
<th>Rings</th>
<th>R100</th>
<th>R150</th>
</tr>
</thead>
<tbody>
<tr>
<td>min</td>
<td>7.376</td>
<td>17.982</td>
<td>34.659</td>
<td>77.176</td>
<td>162.687</td>
<td>394.084</td>
</tr>
<tr>
<td>max</td>
<td>8.387</td>
<td>19.103</td>
<td>35.668</td>
<td>77.434</td>
<td>164.091</td>
<td>395.740</td>
</tr>
<tr>
<td>avg</td>
<td>7.980</td>
<td>18.632</td>
<td>35.432</td>
<td>77.331</td>
<td>163.648</td>
<td>394.811</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th></th>
<th>AT&amp;T</th>
<th>COST266</th>
<th>Germany50</th>
<th>Rings</th>
<th>R100</th>
<th>R150</th>
</tr>
</thead>
<tbody>
<tr>
<td>min</td>
<td>8.379</td>
<td>18.990</td>
<td>35.496</td>
<td>78.182</td>
<td>164.091</td>
<td>395.093</td>
</tr>
<tr>
<td>max</td>
<td>9.444</td>
<td>20.109</td>
<td>36.674</td>
<td>78.439</td>
<td>165.096</td>
<td>396.747</td>
</tr>
<tr>
<td>avg</td>
<td>9.008</td>
<td>19.615</td>
<td>36.019</td>
<td>78.336</td>
<td>164.734</td>
<td>395.758</td>
</tr>
</tbody>
</table>

considered where the IS-IS entity gets immediate information on the change in the status of a physical link. Thus, IS-IS operation is immediately triggered by the change. Operation based on the loss of Hello messages is not taken into account since those scenarios do not provide correct picture on the protocol performance.

The evaluation was performed for two settings without the loop prevention and for one setting with loop prevention. The standard default IS-IS parameters were used during the first setting. IS-IS without loop prevention was then investigated with the fine tuned settings eliminating the artificial delays embedded in the default parameters, i.e. only messaging and computation times are taken into account. Finally IS-IS with the Neighbour Synchronisation was evaluated such that the fine tuned parameters were used. Each scenario was simulated hundred times. Tables 3.3 and 3.4 summarise the results for link failures and for node failures, respectively.

The comparison of the default and the fine tuned IS-IS results show that the convergence time decreases at least by an order of magnitude due to the change in the IS-IS parameters. Comparing the Neighbour Synchronisation results to the pure IS-IS results, it can be seen that the additional delay is roughly 1 ms in case of a link failure. The increase varies more in case of a node failure, the additional delay to the average value is between 1 ms and 10 ms. That is the Neighbour Synchronisation algorithm does not deteriorate network convergence.
Table 3.4: Convergence time after a node failure [ms]

<table>
<thead>
<tr>
<th></th>
<th>AT&amp;T</th>
<th>COST266</th>
<th>Germany50</th>
<th>Rings</th>
<th>R100</th>
<th>R150</th>
</tr>
</thead>
<tbody>
<tr>
<td>Default IS-IS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>min</td>
<td>3890</td>
<td>6217</td>
<td>6356</td>
<td>3134</td>
<td>4864</td>
<td>6146</td>
</tr>
<tr>
<td>max</td>
<td>6032</td>
<td>8184</td>
<td>8705</td>
<td>4394</td>
<td>6311</td>
<td>7441</td>
</tr>
<tr>
<td>avg</td>
<td>5168</td>
<td>6975</td>
<td>7294</td>
<td>3792</td>
<td>5645</td>
<td>6518</td>
</tr>
<tr>
<td>Fine tuned IS-IS</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>min</td>
<td>8.437</td>
<td>19.206</td>
<td>37.102</td>
<td>77.433</td>
<td>164.085</td>
<td>394.734</td>
</tr>
<tr>
<td>max</td>
<td>10.332</td>
<td>26.143</td>
<td>38.344</td>
<td>101.758</td>
<td>165.089</td>
<td>395.740</td>
</tr>
<tr>
<td>avg</td>
<td>9.339</td>
<td>21.044</td>
<td>37.629</td>
<td>86.904</td>
<td>164.490</td>
<td>395.140</td>
</tr>
<tr>
<td>Fine tuned IS-IS with Neighbour Synchronisation</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>min</td>
<td>8.441</td>
<td>22.004</td>
<td>37.676</td>
<td>78.439</td>
<td>164.692</td>
<td>395.740</td>
</tr>
<tr>
<td>max</td>
<td>11.396</td>
<td>43.168</td>
<td>50.789</td>
<td>128.502</td>
<td>166.094</td>
<td>397.752</td>
</tr>
<tr>
<td>avg</td>
<td>10.202</td>
<td>29.319</td>
<td>44.809</td>
<td>95.342</td>
<td>165.333</td>
<td>396.746</td>
</tr>
</tbody>
</table>

Measurement Based Analysis

The operation of the Neighbour Synchronisation algorithm was also evaluated by means of measurements in a prototype network comprised of Debian GNU/Linux PCs. The prototype uses the ISIS routing daemon called *isisd* from the *quagga* open source routing suite [50]. The Neighbour Synchronisation algorithm was implemented in the *quagga* daemon. The test network is shown in Figure 3.7.

The figure shows the node IDs and the cost of the links between them. The link between N1 and N2 is asymmetric, i.e. it has different costs in the two directions in order to produce looping traffic after a link failure. The effect of the N2-D link break-down was investigated during the measurements. If the N2-D link goes down, then the packets sent from S to D are sent around in the N1-N2-N3 loop.

The *helloInterval*, *minLSPGenerationInterval* and *spfDelay* IS-IS parameters

![Figure 3.7: Test network used for measurements](image-url)
Table 3.5: Measured convergence time [s]

<table>
<thead>
<tr>
<th></th>
<th>IS-IS</th>
<th>IS-IS with Neighbour Synchronisation</th>
</tr>
</thead>
<tbody>
<tr>
<td>min</td>
<td>1.41</td>
<td>1.43</td>
</tr>
<tr>
<td>avg</td>
<td>2.03</td>
<td>2.10</td>
</tr>
<tr>
<td>max</td>
<td>2.50</td>
<td>2.56</td>
</tr>
</tbody>
</table>

were set to 1 s, which is their minimal value possible to set in the *quagga* implementation. All other IS-IS parameters were set to their smallest value allowed by the standard [5], which is shown in Table 3.1. That is the implementation was tuned to be as fast as possible within the timer intervals allowed by the standard. That is, the fine tuning used during the simulation was not applied mainly because the *quagga* implementation does not allow such deviation from the standard IS-IS values. Thus the values used are supported by any existing IS-IS implementation.

The convergence time was then investigated with and without the loop prevention algorithm. The convergence time was measured from the reception of the first notification on the link failure until the last FIB update associated with the topology change has been completed in the network. Table 3.5 summarises the results of twenty measurements.

The comparison of the convergence time values measured with and without loop prevention shows that loop prevention does not increase the convergence time significantly in case of standard IS-IS parameter settings, i.e. the difference is two orders smaller than the convergence time itself.

Both the simulation and measurement evaluations showed that the loops occurring without loop prevention are eliminated by the Neighbour Synchronisation algorithm. Furthermore, the Neighbour Synchronisation does not add significant delay to network convergence.

3.4 Root Controlled Bridging

If standard IS-IS is used for the control of SPB, then an All Pairs Shortest Path (APSP) algorithm has to be applied for the computation of the forwarding paths because each bridge has to determine the SPTs of all bridges in order to set up the source rooted SPTs for multicast traffic. That is the computation complexity is increased compared to other application areas of IS-IS where Dijkstra algorithm is used for path computation. IS-IS implementing the Neighbour Synchronisation and an APSP algorithm is applicable for the control of SPB and is referred to as Basic IS-IS in the rest of this dissertation.

I proposed a new approach for the control of SPB networks referred to as Root Controlled Bridging (RCB), which is described in Farkas [C2] in detail besides...
this section. RCB is also an extension to IS-IS but changes the operation a bit more than Basic IS-IS. RCB decreases the computational complexity compared to Basic IS-IS and provides an embedded loop prevention mechanism that applies less blocking than the Neighbour Synchronisation algorithm.

RCB takes advantage of the SPB feature that each bridge has its own SPT rooted by the owner bridge. In RCB, the Root Bridge of a tree controls its own tree in a link state manner. As each bridge is aware of the entire network topology and each bridge has its own tree, each tree owner is able to control its tree. That is the Root Bridge computes its tree and controls the set-up or update of its tree. Thus we get a distributed architecture as each bridge controls a different tree. With respect to a single tree the control is centralised, which has significant advantages for reducing computation complexity. Note that if a bridge breaks down, then the tree where it was the Root Bridge becomes unnecessary.

**Applied Messages**

The operation of the RCB control protocol relies on IS-IS. That is adjacency is discovered by standard IS-IS Hello operation and the neighbour information is advertised in standard IS-IS LSPs. That is, each bridge maintains its own topology database as specified in the IS-IS standard. RCB only differs from standard IS-IS in the set-up and updating of forwarding paths. The Root Bridge calculates its tree based on the LSDB and advertises its tree to the rest of the bridges in LSPs. These LSPs are referred to as Tree Advertisement (TA) messages, which implement tree description in a new TLV (Type, Length, Value) defined according to RFC 3784 [51]. An important feature of a TA message is that it is only forwarded along the tree that it describes from the root towards the leaves.

I propose a compressed description of SPTs in order to minimise the size of control messages. This description is illustrated for an example tree depicted in Figure 3.8. The description in the TA message that arrives to Bridge 1 is shown in Figure 3.9. It contains the description of the advertised tree similar to a preorder listing. The first ID, i.e. Bridge 1 in this example, is the addressee of the TA message. Each sub-tree is described in braces according to the number of branches in the tree. There are two branches going out from Bridge 1 and three

![Figure 3.8: An example for a sub-tree](image-url)
branches from Bridge 2. During the processing of the TA, Bridge 1 can remove the information that was addressed to Bridge 1 and can forward corresponding information to its neighbours downwards on the tree. Thus the tree description for Bridge 2 is shown in Figure 3.10(a) and Bridge 3 receives a TA message containing the tree description shown in Figure 3.10(b).

![Figure 3.9: Sub-tree description for Bridge 1](image)

As shown in Figure 3.10(a), the TA message describes that Bridge 2 is connected to Bridge 4, Bridge 5 and Bridge 6. The TA message illustrated in Figure 3.10(b) shows that Bridge 3 is connected to Bridge 7. As leaf bridges always have a single link and no further connection, there is nowhere to forward the TA, thus the tree description for leaf bridges is NULL. Note that the size of the TA message decreases as it goes towards the leaves thus decreasing the load of the control protocol.

**Detailed Operation**

The operation only differs from standard IS-IS when a forwarding tree is computed and set up in the network. Therefore, the operation of these processes is described in detail the following. Figure 3.11(a) shows the procedure of the computation of the owned tree in a Root Bridge.

The Root Bridge computes its new SPT if there is a change in the topology. If there is a change in the tree as well, then the new tree has to be set up in the network. The Root Bridge then sets its discarding ports, i.e. the blocked ports. After that the forwarding ports are set too, and the SPT is distributed in TA messages by the Root Bridge.

The operation when a TA is received is shown in Figure 3.11(b). It is very important during the operation that the TA messages follow the tree under configuration. That is the TA message is only forwarded on the links that are part of the new tree being set up but not forwarded on the links that are not included in the tree. Furthermore, discarding ports are always set before the forwarding ports.
Two important features of RCB are investigated in the following.

**Loop Prevention**

RCB provides a specific order for tree updates: from the Root Bridge towards leaf bridges. Furthermore, the Tree Advertisements are forwarded on the tree under configuration and link blocking is always set before link activation. These features assure that accidental loops do not appear in an RCB controlled SPB network.

**Theorem 3.3.** *Root Control Bridging prevents the appearance of loops in the forwarding topology of an SPB network.*

**Proof**

Let us assume that a loop appears in a given scenario. Note that a single scenario is depicted here but the operation and the statements are the same for any potential loop of any size. Furthermore, bridges outside of a potential loop cannot be involved in the given potential loop because that would be another potential loop. The initial topology of the examined network is shown in Figure 3.12(a), where a new link is added to the network. The tree rooted by Bridge 1 is observed during the topology transient. A port of a bridge is either in Forwarding (F) or in Discarding (D) state as it is indicated in the figure. Note that a link is blocked if the port at either end of the link is in Discarding state.

The topology formed at the end of the convergence is depicted in Figure 3.12(c). The stable topologies, i.e. the one before the topology change and the other one at the end of network convergence are loop free.
Figure 3.12: Topology transient controlled by RCB

Let us assume that there is a loop during the transient formed by Bridges 1, 2, 5, 8, 6 and 3 as shown in Figure 3.12(b). Let us assume that the loop appears because nodes are in different stages in their topology update as the figure shows.

However, there is a contradiction with the initial assumption. Namely, one end of the link between Bridge 6 and Bridge 8 is in Discarding state in the transient stage because:

- either Bridge 8 has not received the TA message thus has not been updated, therefore, its port towards Bridge 6 is Discarding
- or Bridge 8 has processed the TA message, hence, its port towards Bridge 5 is Discarding

That is, the proposed RCB protocol provides a loop prevention mechanism.

The operation of RCB for the Farkas loop example is explained in Appendix A.4.

Scalable Tree Computation

If RCB is applied for the control of an SPB network, then the computation complexity decreases compared to that of Basic IS-IS. As only the Root Bridge calculates a single tree in RCB the computational complexity has been decreased significantly. Note that in the Basic IS-IS approach each bridge has to calculate each tree in order to be able to set up all SPTs appropriately. The reason for this is that each bridge has to determine whether or not it is on the shortest path between any other bridge pairs, i.e. the bridge has to figure out how to set up the forwarding for each SPT in the network.

In link state routing protocols, the basis of the route calculation is the Dijkstra algorithm, which is also known as Shortest Path First (SPF) algorithm. On a
topology $G(N, L)$ comprised of nodes $N$ and links $L$ the complexity of the Dijkstra
algorithm can be expressed as a function of $|N|$ and $|L|$, which was investigated
e.g. by Barbehenn [52]. According to Francois [46], commercial routers use binary
heap implementation. The computation complexity of a binary heap Dijkstra is

$$O(|L| + |N| \cdot \log |N|).$$  

(3.4)

In Basic IS-IS, the computational complexity increases to that of an All Pairs
Shortest Path (APSP) algorithm evaluated e.g. by Moffat [53], which is

$$O(|N| (|L| + |N| \cdot \log |N|)).$$  

(3.5)

As each node computes at most a single SPT in RCB, its computational com-
plexity is given by Equation (3.4). Thus the path computation performed by each
node is much simpler in RCB than in Basic IS-IS.

### 3.5 Performance Analysis of SPB Control Protocols

Two control protocol proposals for SPB is described above, nevertheless, the SPB
project was initiated at IEEE 802.1 with a third approach. MSTP can be applied
for the control of SPB if a Multiple Spanning Tree Instance (MSTI) is assigned to
each bridge on which the owner bridge is the Root Bridge, thus each bridge has
its own SPT, as proposed by Finn [54]. The MSTP based SPB control protocol is
referred to as MSTP-SPB in the following. Count-to-infinity caused by a bridge
break-down is not a problem for MSTP-SPB because the MSTI of the broken bridge
then becomes unused. Furthermore, MSTP implements a very fast switching to a
safe alternate path after a failure event if such a path exists. That is, if the Root
Bridge becomes unreachable for another bridge due to a failure, then a formerly
blocked alternate path is activated, which is a local decision at the bridge, thus
it can be very fast. The denser the network topology, there is higher chance for
the existence of a safe alternate path. Having three alternatives, the question on
their relationship arises. Analysing these three control protocols, we also get a
comparison of distance vector and link state protocols.

A performance analysis of the three control protocol alternatives proposed for
SPB is provided in the following, which I also published in Farkas [C1]. The
evaluation was performed through simulation analysis. The OMNeT++ simulator
described in Section 3.3 was used for the evaluations. MSTP-SPB, Basic IS-IS
and RCB were implemented as higher layer entities in the bridge architecture
[1]. That is, all three protocols were implemented on the same platform thus
helping objective comparison by eliminating implementation differences as much
as possible.
The convergence time was investigated, which is the time elapsed from a failure event realised by the control protocol until the forwarding has been recovered, i.e. all control protocol messages affecting the forwarding have been processed and the new forwarding topology has been set. That is, the end of processing of the last Bridge Protocol Data Unit (BPDU) causing any change in an MSTI is considered as the end of the convergence time for MSTP-SPB. The end of convergence is the end of the processing of the last LSP causing any change in an SPT for IS-IS based approaches. The performance of the protocol is in the focus of the analysis, therefore, the time needed for failure detection is not taken into account. That is, physical layer upcall is assumed for the immediate notification of the MSTP or the IS-IS entity if a link goes down.

The convergence time depends on network topology, therefore, three type of topologies were used for the evaluation of protocol performance. The size of a network also affects the convergence time especially in larger networks; hence the different topologies had various network sizes from 50 to 280 bridges.

Ring topologies are widely deployed in real networks, e.g. in access networks. Therefore, one of the investigated topologies, which is referred to as Rings, consists of an inner and multiple outer rings as depicted in Figure 3.13(a). More redundant topologies are used in core networks, therefore, two types of mesh were also investigated. The Heavy-mesh, which is illustrated in Figure 3.13(c), comprises an almost full-mesh inner part and a bunch of outer bridges connected to the same inner bridge pair in a dual-homing manner. Figure 3.13(b) shows the Light-mesh topology, which consists of a denser inner part than that of the Rings topology but sparser than that of the Heavy-mesh topology. Light-mesh and Heavy-mesh also differ in the number of outer bridges dual-homing to the inner part. Convergence time has been investigated for link and node failure events occurring in the inner part of the network topologies.

![Figure 3.13: Simulation topologies](image)

53
Parameter Settings

The core of the protocols was aimed to be compared, therefore, the protocol parameters affecting convergence time were set such that they do not put any limit on it. For the same reason, immediate messaging is assumed in reaction to an event without any random delay before message transmission. The effects of IS-IS timers were eliminated as proposed by Francois [46]. The protocol parameters were set for the simulations as shown in Table 3.1, i.e. such that they do not affect convergence time. Thus, the simulation results only show computation and messaging times.

The messaging and processing delays were modelled and implemented in the simulator based on the measurement results published in Farkas [C5]. The transmission delay of a message on a link is proportional to the linkrate and the message size. Fast-Ethernet links were assumed during the simulations and the size of the control messages was determined based on the standards. Furthermore, control traffic had high priority thus it does not suffer queuing delay due to background traffic.

The message processing time is another delay factor appearing in all approaches, nonetheless, heavily implementation dependent. Therefore, the convergence time was evaluated for various control message processing time values from 0.1 ms to 10 ms, for which the results are presented in the following. The same 1 ms message processing time value was then applied during further evaluations for all the three protocols in order to provide comparable results. Therefore, the results presented here are deterministic, which is useful for unbiased comparison of protocol performance in such a large parameter space. The convergence time of the three control protocols have been analysed with various parameter settings for the topologies described above. The effects of different parameter settings were also evaluated and described in detail in Appendix A.5 besides [C1]. The dependency on network topology and size was in the focus of the investigations afterwards, for which the results are presented below after the investigations on message processing.

Message Processing Delay

The processing time of control messages is a key component in protocol convergence time, therefore, its effect on the performance of the three protocols was investigated too. The processing of Tree Advertisement messages introduces an extra message processing delay in the RCB approach. The TA messages suffer the same delays as any other LSP, but the parsing of the tree description that they carry takes additional time, which is implemented in the simulator as

\[ ParseDelay = P \cdot n, \]

where n nodes comprise the sub-tree carried in the TA. Measurement results
showed that $P = 0.001 ms$ is realistic, nonetheless, $P = 0.01 ms$ was also evaluated in order to provide a wider picture on the performance of RCB. The effect of message processing delay on convergence time for a bridge failure in 50-node and 200-node Rings topologies are shown in Figure 3.14(a) and Figure 3.14(b), respectively. All approaches are very sensitive to control message processing time, especially RCB, which is the most intense in messaging. Nonetheless, the computational intensity of Basic IS-IS increases with the size of the network, therefore, the difference between Basic IS-IS and RCB with the most realistic parsing delay ($P = 0.001 ms$) decreases.

BPDU processing time is in the order of milliseconds according to Farkas [C5], nevertheless, the measurement results published by Chiabaut [55] suggest LSP processing time in the order of 0.1 ms. The processing time was set to 1 ms both for BPDUs and LSPs in the following simulations in order to provide comparable results, i.e. $\text{linkDelay} = 1 ms$ was set in the simulator.

**Network Topology and Size**

The convergence time of every control protocol largely depends on the network topology and on the size of the network. Therefore, the convergence time of the SPB protocols was evaluated for various network sizes of the topology types described above. Figures 3.15(a), 3.15(b) and 3.15(c) show the convergence time as a function of the network size in case of a single link failure. As the figures show the size of the network has different impact on the protocols for different network types. In the sparse Rings topology, the convergence time of the different protocols changes similar to each other if we take into account the realistic $P = 0.001 ms$ for RCB. This topology is not favourable for control messaging as the control information has to travel long paths, thus distance vectors of MSTP-SPB are also propagated on long paths. Furthermore, it is not likely in a sparse
topology that MSTP-SPB is able to switch to a safe alternate path after a failure event. Nonetheless, MSTP-SPB is the less affected by the increase in network size in denser topologies as the results for the mesh topologies show, because it is more likely that a safe alternate path exists. Switching to a safe alternate path can be invoked very fast, which explains the slight increase in the convergence time of MSTP-SPB despite of the increasing network size. As opposed to this, Basic IS-IS is considerably affected by the network size as both the number of nodes and links influence its computational complexity. Therefore, RCB with realistic parsing speed outperforms Basic IS-IS over a certain network size.

The performance of the protocols was also investigated for a bridge break-down event and the results are presented in Figure 3.15(d), 3.15(e) and 3.15(f). A bridge failure initiates a major reconfiguration of SPTs. The sparser the topology, the more SPTs have to be updated, which mostly affects MSTP-SPB and RCB as they involve more messaging for SPT reconfiguration. A bridge failure decreases the chance for having a safe alternate path, therefore, the convergence time of MSTP-SPB increases compared to the link failure case. Furthermore, MSTP-SPB becomes more sensitive to network size even in mesh topologies for the same reasons. Basic IS-IS is the least affected by the different types of failure as its computational complexity remains the same and the processing time of additional LSPs is not significant. Nonetheless, RCB uses TA messages for the reconfiguration of SPTs hence its convergence time increases compared to the link failure scenario as a bridge failure affects more SPTs. Therefore, the intersection point of RCB and Basic IS-IS curves moves towards larger network sizes in case of mesh topologies compared to that of the link failure scenarios.

The performance analysis showed that the convergence time of the Basic IS-IS approach is sensitive to the size of the network in case of mesh topologies due to its computational complexity. RCB, which reduces the computational complexity but applies more control messages instead, converges faster than Basic IS-IS as the size of mesh topologies increases. Nevertheless, RCB performs slightly worse than Basic IS-IS in sparse topologies as messages are transmitted on longer paths in these topologies. The MSTP based SPB approach performs well in mesh topologies because MSTP is able to quickly switch to a safe alternate path, which likely exists in a dense topology. The performance of MSTP based SPB is worse in sparse topologies, e.g. in rings, where alternate paths are typically not available.
Figure 3.15: Simulation results for convergence time
3.6 Conclusions

This chapter investigated and provided solutions to crucial issues when a link state control such as SPB is to be applied in the IEEE 802.1 bridge architecture. Two solutions were proposed for loop prevention which is a major concern in Ethernet networks. One of them is the Neighbour Synchronisation, which can be used as an add-on to any link state protocol. The loop prevention algorithm specified in the SPB standard relies on the basic idea of the Neighbour Synchronisation. The other one is the Root Controlled Bridging, which is a novel approach based on link state principles for the control of an SPB network. RCB also addresses computational complexity issues besides loop avoidance.

The analytic, measurement and simulation based analysis verified the loop prevention feature of both RCB and Neighbour Synchronisation. Furthermore, the performance analysis provided the characteristics of the control protocols proposed for SPB in terms of network convergence time. The evaluations showed that the Neighbour Synchronisation algorithm does not deteriorate network convergence. The performance analysis of the control protocols showed that Root Controlled Bridging converges faster than Basic IS-IS as the size of mesh topologies increases. The more meshed and the larger the topology, RCB converges faster than the alternative Basic IS-IS approach.
Chapter 4

Connection Admission Control for UTRAN

4.1 Introduction

Universal Mobile Telecommunication Systems (UMTS) put stringent requirements on the transport networks carrying UMTS traffic, which has to be taken into account during network operation. It is especially important at lower traffic aggregation levels, where the link capacities may be small. It has to be ensured at each network segment that the QoS requirements are met in order to guarantee the QoS end-to-end. Connection Admission Control (CAC) is one of the tools used for providing QoS, e.g. in UMTS networks. The functional architecture of UMTS is illustrated in Figure 4.1.

A number of logical network elements are grouped into the UMTS Terrestrial Radio Access Networks (UTRAN) and the Core Network (CN). The third element of the UMTS architecture is the User Equipment (UE). The CN is responsible for switching and routing voice and data connections to external networks in addition to managing session and mobility information. A Node B of UTRAN terminates the air interface (Uu) and forwards the traffic to the Radio Network Controller (RNC). The main task of UTRAN is to create and maintain Radio Access Bearers

Figure 4.1: UMTS network architecture
(RABs) for communication between the UE and the CN. The UTRAN handles all radio related functionalities, e.g. radio resource management and handover control. The Iub interface of UTRAN, which is between the RNC and the Node B, is typically provided by an ATM or an IP transport network. The connectivity for IP may be provided by an Ethernet network. Along the fixed mobile convergence, a broadband access network may provide the Iub transport. In order to avoid congestion on the Iub interface, it has to implement a Connection Admission Control algorithm. The links of UTRAN typically have limited capacity and high cost, especially the links of Node Bs, which are often referred to as first or last mile. Therefore, high bandwidth efficiency is a primary target, which necessitates the application of QoS solutions.

3GPP summarised the QoS concept of UMTS in [56], which distinguishes four QoS classes. Conversational and streaming classes, which serve voice and video applications, have inherent real time requirements, for example the total UTRAN delay for voice is 5-7 ms as specified by 3GPP in [57]. If carried in a Dedicated Channel (DCH), then the interactive and background classes used for Internet applications such as web browsing, file transfer and e-mail also have real time requirement due to the control protocols of the Iub interface. The main reason for the stringent delay requirements for these classes is that the RLC/MAC protocol, which controls radio resources of the user equipment (UE), is terminated in the RNC. Therefore, an excessive delay (more than 20-40ms) would make the synchronization of soft-handover legs and the power control inefficient. Packet switched services, i.e. interactive and background classes have less stringent loss requirements due to the retransmission capability of the RLC protocol. A further real time traffic is the synchronization of Node Bs, which has the most stringent delay requirement (~ 2ms). Another special property of UTRAN traffic over Iub interface is periodicity. That is, PDUs containing user traffic are emitted periodically as they have to be sent over the air interface during a Transmission Time Interval (TTI). The independent periodic streams are modulated with an ON-OFF process due to Discontinuous Transmission (DTX) for voice sources and higher layer protocols for mobile applications. TTI of voice traffic is 20 ms, TTI for DCH packet switched services can have four values: 10 ms, 20 ms, 40 ms or 80ms.

Note that Long Term Evolution (LTE) specified by 3GPP Release 9 defines a different architecture than that of UMTS. Evolved UTRAN (E-UTRAN) of LTE differs significantly from UTRAN because E-UTRAN does not involve RNC but it is only comprised of eNodeBs, which are directly connected to the Core Network. Therefore, LTE does not involve any interface similar to the Iub of UTRAN.

3GPP Release 99 and Release 4 only allowed the use of ATM as the transport layer in the various interfaces. 3GPP Release 5 [58] introduced the possibility to use IP at the transport layer in the Iub, Iur, Iu-Ps and Iu-Cs interfaces, as an alternative to ATM.
The Weighted Fair Queuing (WFQ) packet scheduling technique provides guaranteed bandwidth for services and allows the utilisation of unused reserved bandwidth, which is also referred to as Packet-by-packet Generalized Processor Sharing (PGPS) as described by Parekh [59]. Therefore, Class Based WFQ (CB-WFQ) is an appropriate scheduling mechanism for buffers of an IP UTRAN network distinguishing different QoS classes. WFQ schedulers have been analysed extensively in the past. Nevertheless, performance analyses assume deterministic delay guarantees, which is a too conservative assumption for UTRAN. Paschalidis [60] and Zhang [61] provided analytic results for CB-WFQ in the stochastic regime using large deviation theory to obtain an exponential approximation for the delay distribution. These results, however, are based on burst level models, which do not suffice when the tight delay requirements necessitate the incorporation of packet scale delays in the queuing models. As shown by Malomsoy [62, 63], packet scale delays, i.e. delays that occur even if the total input rate is below the link capacity, are in the order of the delay requirement. In UTRAN, therefore, it must be also considered in the CAC algorithms. Malomsoy [62, 63] provided CAC algorithms for FIFO and strict priority systems. Besides considering packet scale delays, the CAC applied in a CB-WFQ system should be designed such that it takes advantage of the scheduling which allows traffic classes to exceed their allocated rate in the scheduler if the buffers of other traffic classes are underutilised.

CAC algorithms used in access networks for multiplexed traffic sources are typically based on aggregated system characteristics, such as aggregate loss probability. The relation of these aggregate performance measures to the performance of specific flows is not trivial. The most important real time application for operators is undoubtedly voice, therefore, meeting QoS requirements at flow level might be of importance at least for voice traffic. As mentioned above, a generally accepted traffic model for voice sources is the periodic ON-OFF source model. The distribution of the duration of ON and OFF periods can be arbitrary. A common descriptor to characterise ON-OFF sources is the activity factor, which is also applied by the 3GPP [64]. Activity factor is defined as the ratio of the total duration of ON periods and the total lifetime of the flow. In case of voice connections, the activity factor means the portion of time when the speaker is busy talking. The activity factor of a flow depends on several effects and it can vary flow by flow. Nonetheless, a constant activity factor is often used for the characterisation of voice sources. According to Tran-Gia [65] and Arvedson [66] voice activity factor typically takes values from 0.4–0.6.

If admission control is designed for aggregate loss requirement, then the loss rate of individual flows may violate their requirement even if they are admitted by the admission control. A specific example for such violation is due to the varying activity factor of voice sources, which was shown by Ni [67] and Westholm [68]. There may be time intervals when the activity factor of many ongoing flows is
larger than the average activity factor. Flows experience high loss during these periods.

Measurements and analysis of ON-OFF parameters of voice sources have been the subject of research, especially in relation to mobile communication systems. Westholm [68] determined the distribution of voice activity factor by means of measurements performed in a GSM network. Jiang [69] measured and analysed the distribution of ON and OFF duration of voice flows in a VoIP system. The activity factor of a flow depends on the user behaviour and also on the applied transport protocol. As shown by Jiang [69] and Beritelli [70], it also depends on the voice activity detection algorithm and on the operation of voice coders. Regarding user behaviour the sex of the speakers, their language and personality also affect the activity factor according to Arvedson [66] and Beritelli [71]. Sun [72] studied the effect of these differences on the perceived voice quality in IP networks.

QoS enabled packet networks are extensively studied in the literature, e.g. by Kelly [73] and Le Boudec [74], by using deterministic methods, which is a worst case analysis method as it gives upper bounds for packet delay. By introducing stochastic bounds for packet delay, which slightly weakens the QoS requirement, better network utilisation can be achieved. Burchard [75] and Liebeherr [76] proposed stochastic network calculus to solve this problem, where they determined stochastic bounds for the service experienced by a single flow when resources are managed for aggregates of flows. The above techniques were developed for the analysis of a wide variety of systems. Nevertheless, they do not exploit all information assumed to be available in a multi-service system where voice service has differentiated treatment. That is, efficient utilisation of network resources meanwhile providing even flow level assurances for voice traffic should take into account system and traffic properties including the random nature of activity factor of voice flows.

The rest of this chapter is structured as follows. Section 4.2 gives a model for the traffic of UTRAN Iub and its QoS requirements. Section 4.3 then describes the principles of the Connection Admission Control algorithm proposed to be applied for UTRAN Iub. After that, Section 4.4 provides the details of the CAC algorithm proposed for IP UTRAN networks implementing CB-WFQ scheduling. A refined CAC algorithm is then described in Section 4.5, which is applicable if flow level QoS requirements to be guaranteed instead of the formerly used guarantees on aggregates.
4.2 Traffic and QoS Model

The user traffic transmitted on the UTRAN Iub interface and their QoS requirements can be modelled and described as follows.

Traffic flows of the UTRAN Iub system can be categorised into traffic classes, which can be modelled by flows of independent ON-OFF modulated periodic sources.

**Traffic Class**

A traffic class refers to the set of traffic sources that have the same QoS and traffic parameters. The QoS requirement describes the acceptable packet loss rate and the acceptable packet delay. The QoS requirement is described statistically.

The traffic parameters of traffic class $i$ are $\{T_i, b_i, \alpha_i\}$, where

- $T_i$ is the source period. That is, $T_i$ denotes the Transmission Time Interval (TTI) of the traffic class, which is the deterministic inter-arrival time if the source was always in ON state.
- $b_i$ is the packet size. That is, a source sends $b_i$ size packets in state ON and it does not generate any packets in state OFF.
- $\alpha_i$ is the activity factor, which describes the ON-OFF behaviour. The activity factor of a flow is the ratio of the total duration of ON periods and the total lifetime of the flow. If the activity factor is described by a single value for a traffic class, then it is the average of the activity factor values of the individual traffic flows of that traffic class.

The QoS parameters for traffic class $i$ are $\{d_i, \varepsilon_i\}$, where

- $d_i$ is the delay requirement of class $i$.
- $\varepsilon_i$ is the tolerance level for the QoS violation.

The QoS requirement is met if the packet drop rate is smaller than $\varepsilon_i$.

**Traffic Mix**

The number of flows in a traffic class may vary during the time.

$N_i$ denotes the number of ongoing flows in traffic class $i$.

$\mathbf{N}$ is the traffic mix, which is a vector $\mathbf{N} = N_1, N_2, \ldots, N_K$ that gives the number of ongoing flows of each traffic class in a $K$-class system. Note that a traffic mix is a point in the space span by the number of flows of different traffic classes.
The ON-OFF behaviour is illustrated in Figure 4.2 for voice flows served at the guaranteed rate of the voice traffic class, which can be modelled by a *FIFO* system having capacity $C$ and buffer size $B$. Only the voice traffic class is shown in the figure. The call arrivals and departures are marked by vertical lines and packet arrivals are marked by small bars in Figure 4.2. The initial phase of the periodic packet transmission is random, which is assumed to be distributed uniformly over the interval $[0, T_i]$. Hence the traffic in a buffer can be modelled as a superposition of independent periodic streams with a random initial offset as illustrated in the figure. In the system shown in Figure 4.2, a connection admission control algorithm limits the maximum number of parallel flows to $N_{\text{max}} = 4$. The actual value of the activity factor can be different for each flow. For example, the activity factor of flow $j$ is close to 0.6. Nonetheless, the activity factor of flow $k$ is close to 0.9, meaning that the source almost always sends packets.

### 4.3 Model Based Connection Admission Control

Admission control algorithms are used during connection set-up to ensure that the QoS requirements of ongoing connections and the newly arriving connection are met while allowing for statistical multiplexing gain. If the resources of the communication network are not enough to fulfil the QoS requirements of either the new or the ongoing connections, then the new connection is not admitted. As the admission decision has to be performed during the set-up of each connection, the CAC algorithm has to make a rapid decision. Furthermore, the CAC has to ensure that the QoS requirements are met such that high system utilisation is also
possible, i.e. resources are not wasted.

CAC methods are often based on measurements, especially if there is no enough information on the properties of the incoming traffic and the communications system. Nevertheless, a CAC algorithm may take advantage of the knowledge on the system and traffic characteristics if they are available. Traffic flows of the UTRAN Iub system can be modelled as described in Section 4.2. The $n \cdot D/D/1$ queuing system described in detail by Roberts [77] can be extended in order to describe the operation of UTRAN Iub. Therefore, a model based CAC algorithm can be applied for the Iub interface of UTRAN, which is explained in the following.

The ON-OFF traffic behaviour causes a burst level fluctuation due to the burst of sessions being in ON state. QoS requirements may be violated due to buffer overload caused by the burst of traffic sources in ON state, which is often referred to as overload or burst level QoS violation. Even if there is no overload, a QoS requirement may be violated due to temporary packet congestion in a buffer, which causes larger packet delay than allowed thus the receiver cannot use the packet, i.e. it is dropped. Therefore, this second type of violation is referred to as delay or packet scale QoS violation. As Malomosky [62, 63] described, these two type of QoS violations can be treated independently of each other. That is, the packet drop QoS requirement can be split into two:

$$\varepsilon_i = \varepsilon_i^{\text{burst}} + \varepsilon_i^{\text{packet}}. \quad (4.1)$$

Thus burst scale and packet scale effects can be analysed separately. Note that the probability of buffer overload is the same for traffic classes directed to the same buffer $k$. Therefore, a common $\varepsilon_k^{\text{burst}}$ value can be taken into account for traffic classes sharing the same buffer $k$, where

$$\varepsilon_k^{\text{burst}} = \min\{\varepsilon_i^{\text{burst}}\} \quad \forall i \in k. \quad (4.2)$$

The packet level and the burst level constraints are illustrated in Figure 4.3, which shows the surfaces determined by these constraints in the space of the number of flows of different traffic classes for a two-class system where the traffic classes have their own buffer. Three example traffic mixes are pointed in the figure. Traffic mix $A$ causes burst level violation. Traffic mix $B$ causes packet level violation despite of not causing burst level violation. Traffic mix $C$ does not cause either QoS violation, therefore, it can be admitted. A traffic mix can only be admitted if it is below each constraint surface, i.e. it is within the Admissible Region where no QoS requirement is violated. The Admissible Region is the green filled region in the example shown in Figure 4.3.

The burst level operation can be modelled by bufferless multiplexing because the delay requirements are small compared to burst level dynamics in UTRAN. That is, all packets are assumed to be lost in a buffer within a TTI if the input rate of the buffer is larger than its service rate. Presti [78] also showed that bufferless
multiplexing is applicable when the buffers are small, which is the case in UTRAN, so queuing at ON-OFF time scale can be neglected.

As opposed to this, it can be assumed that the buffers do not overflow when the delay requirement is violated due to a packet scale burst. Therefore, the system can be modelled as if the buffer was infinite when evaluating packet scale QoS violations.

As depicted in Figure 4.4 the model based CAC algorithm is comprised of two main components:

- **Off-line initialisation**: All the parameters required for the model applied by the CAC are determined, e.g. the number of traffic classes, their descriptors, QoS requirements etc. Having the parameters, the CAC algorithm also performs all the off-line calculations during the initialisation phase, e.g. related to the packet scale constraint surfaces.

- **On-the-fly admission decision**: The admission decision is performed on-the-fly at every new connection request. The CAC algorithm has to decide whether or not the QoS requirements can be met if the new connection was admitted, i.e. having a new traffic mix in the system. As Figure 4.4 shows, the admission decision is split into two according to the two type of possible QoS violations:
  - **Decision 1** checks whether the new traffic mix caused overload.
  - **Decision 2** evaluates the new traffic mix with respect to delay violation.

A new connection is only admitted if neither of the evaluations detects QoS violation.

That is, the on-the-fly admission decision performs the following checks when there is an admission request for a new flow of class $i$ such that all the parameters
\{T_i, b_i, \alpha_i, d_i, \varepsilon_i^{\text{burst}}, \varepsilon_i^{\text{packet}}\} are known for \( i = 1 \ldots K\) in an K-class system. Given that the old traffic mix was \( N_{\text{old}} = N_1, \ldots, N_i, \ldots, N_K\), the new traffic mix will be \( N_{\text{new}} = N_1, \ldots, N_i+1, \ldots, N_K\) if the new class \( i\) request gets admitted. Therefore, the CAC performs its checks on \( N_{\text{new}}\).

That is, the CAC decisions are performed as follows. A new connection of class \( i\) is only admitted by Decision 1 if the following inequality is valid for buffer \( k\):

\[
P[\text{input rate of buffer } k > \text{service rate of buffer } k] \leq \varepsilon_k^{\text{burst}}. \quad (4.3)
\]

Furthermore, the new connection is admitted by Decision 2 if

\[
P[\text{delay for class } i \text{ packets } > d_i] \leq \varepsilon_i^{\text{packet}} \quad \forall i \in K. \quad (4.4)
\]

In other words, Decision 2 only admits a new connection request if

\[
P[N_{\text{new}} < \text{delay constraint surface of class } i] = 1 \quad \forall i. \quad (4.5)
\]

The admission decision has to be fast in order to support short connection set-up time. I proposed a CAC algorithm in Farkas [C10] that takes into account the packet scale operation of WFQ scheduling when checking the delay violation, which is also described in Section 4.4. I proposed another CAC algorithm for the evaluation of overload when the ON-OFF behaviour is of special importance, i.e. QoS is aimed to be guaranteed for the flows of a traffic class instead of only having guarantee on their aggregate, which is described in Farkas [C9] and in Section 4.5.
4.4 Connection Admission Control for IP UTRAN Implementing WFQ

The Connection Admission Control (CAC) only admits such connection mixes for which the required QoS can be guaranteed. An IP transport network is considered between Node Bs and RNCs. Transport network nodes implement a limited number of buffers and a scheduling algorithm which is able to guarantee a minimum bandwidth for each buffer and bounded delay for real time (RT) traffic classes. These buffers are assumed to be served according to class-based weighted fair queuing (CB-WFQ) scheduling. Real time and non-real time (NRT) traffic classes are assigned to different buffers of CB-WFQ, thus RT and NRT buffers are distinguished. Note that multiple RT traffic classes can be assigned to a RT buffer, similarly multiple NRT traffic classes can be assigned to the same NRT buffer. The CAC algorithm I proposed in Farkas [C10] for this IP UTRAN Iub interface is described in the following. The proposed algorithm aims to allow for traffic sources to exceed their allocated rate in the scheduler if other buffers are underutilised.

Admission Decisions

The admission decisions proposed to be applied by the CAC are described here. After that, the analysis leading to these proposals are provided.

Overload Check

The ratio of packets lost due to a buffer overload has to be smaller than $\varepsilon_{k}^{burst}$ as expressed by Equation (4.3). Direct calculation of the loss ratio is a computationally complex task if the number of sessions is large, hence an approximation is proposed here.

Applying the central limit theorem, the distribution of the number of active sessions in a buffer can be approximated by a normal distribution if the number of sources grows. Then, the overload check of Equation (4.3) can be performed using the following closed form approximation:

$$1 - \Phi\left(C, \tilde{R}_k, \tilde{V}_k\right) + \frac{\tilde{V}_k}{\tilde{R}_k} \cdot \varphi\left(C, \tilde{R}_k, \tilde{V}_k\right) \leq \varepsilon_{k}^{burst},$$

where $\varphi(x, \mu, \sigma^2)$ and $\Phi(x, \mu, \sigma^2)$ are the density and cumulative distribution functions of the normal distribution with mean $\mu$ and variance $\sigma^2$, respectively. Furthermore, $\tilde{R}_k$ and $\tilde{V}_k$ are the mean and variance of the input rate in buffer $k$, which are corrected according to the CB-WFQ operation. The details for obtaining Equation (4.6) are described below.
Packet level analysis of the operation of the CB-WFQ system has shown that the delay constraint surface is complex. Therefore, I proposed to approximate the operation of the CB-WFQ system with a set of Strict Priority (SP) systems that operate separate from each other. Thus, I proposed to approximate the delay constraint surface of a traffic class with a set of hyperplanes obtained from the SP operation. As it was explained related to Figure 4.3, a traffic mix does not violate the delay criterion if it is below the delay constraint surface.

The delay constraint surface of class $i$ is approximated by a set of hyperplanes. Traffic mix $N_{\text{new}}$ does not cause delay violation for class $i$ if $N_{\text{new}}$ is below any of the hyperplanes, which is tested by checking

$$\max_Y \left( F_{ii}^{Y} + 1 - \sum_{j \in Y} F_{jj}^{Y} \frac{F_{ji}^{Y}}{F_{jj}^{Y}} \cdot N_j \right) > 0,$$

where $Y$ denotes the index set of RT buffers; $F_{ji}^{Y}$, $F_{ii}^{Y}$ and $F_{jj}^{Y}$ are used to determine a hyperplane.

If traffic mix $N_{\text{new}}$ passes the check of Equation (4.7) for each traffic class, then admitting the new request resulting in $N_{\text{new}}$ does not cause delay violation.

Let us investigate in detail how the above results can be obtained. Note that Table 4.1 summaries the notations applied in this chapter.

**Burst Scale QoS**

The ON-OFF behaviour of traffic sources causes a burst level fluctuation due to the burst of sessions being in ON state. QoS requirements may be violated due to buffer overload caused by the burst of traffic sources in ON state.

To capture the burst level fluctuation of input traffic, the heterogeneity in TTIs can be excluded as proposed by Malomsoky [62, 63]. It is performed by replacing a source by another equivalent source such that the queuing behaviour remains approximately the same. Replacing the original TTI by the largest TTI in the system ($T_{\text{max}}$) for each source in any class $i$: $T_{\text{new}} = T_{\text{max}}$; and also replacing the activity factor of class $i$ sources to $\alpha_{i}^{\text{new}} = \alpha_{i}^{\text{old}} \cdot T_{i}/T_{\text{max}}$ gives an upper approximation for the delay distribution. The burst level operation is then concerned by the input and output rates over a $T_{\text{max}}$ period. The burst level model of sources is an ON-OFF source, which either transmits with its $\rho_{i} = b_{i}/T_{i}$ peak rate or does not transmit at all during $T_{\text{max}}$.

The number of active sessions (i.e. sources in ON state) in a given $T_{\text{max}}$ period is denoted by $\mathcal{A} = (A_1, A_2, \ldots, A_K)$, where $K$ is the number of traffic classes. The number of ongoing sessions (i.e. sources in either ON or OFF state) is given by the traffic mix $N = (N_1, N_2, \ldots, N_K)$. The distribution of the number of active sessions
<table>
<thead>
<tr>
<th>Notation</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$C$</td>
<td>capacity</td>
</tr>
<tr>
<td>$B_i$</td>
<td>size of the buffer of class $i$</td>
</tr>
<tr>
<td>$d_i$</td>
<td>packet delay requirement for class $i$</td>
</tr>
<tr>
<td>$\varepsilon_i$</td>
<td>maximum allowed packet drop rate for class $i$</td>
</tr>
<tr>
<td>$\rho_i$</td>
<td>peak rate of class $i$</td>
</tr>
<tr>
<td>$\alpha_i$</td>
<td>activity factor of class $i$</td>
</tr>
<tr>
<td>$b_i$</td>
<td>packet size of class $i$</td>
</tr>
<tr>
<td>$T_i$</td>
<td>packet inter-arrival time (source period) of class $i$</td>
</tr>
<tr>
<td>$N_i$</td>
<td>number of admitted ON-OFF sources of class $i$</td>
</tr>
<tr>
<td>$A_i$</td>
<td>number of sources in ON state in class $i$</td>
</tr>
<tr>
<td>$D_i$</td>
<td>queueing delay for class $i$</td>
</tr>
<tr>
<td>$K$</td>
<td>number of traffic classes in a system</td>
</tr>
<tr>
<td>$K_{SP}$</td>
<td>number of traffic classes in a Strict Priority (SP) system</td>
</tr>
<tr>
<td>$Q_k$</td>
<td>index set of traffic classes forwarded to queue $k$</td>
</tr>
<tr>
<td>$Y$</td>
<td>index set of queues in a Strict Priority system</td>
</tr>
<tr>
<td>$R_k$</td>
<td>input rate for queue $k$</td>
</tr>
<tr>
<td>$S_k$</td>
<td>service rate for queue $k$</td>
</tr>
<tr>
<td>$c_k$</td>
<td>guaranteed minimum rate for queue $k$</td>
</tr>
<tr>
<td>$M_k$</td>
<td>rate multiplier factor for queue $k$</td>
</tr>
<tr>
<td>$SAT$</td>
<td>saturated</td>
</tr>
<tr>
<td>$NSAT$</td>
<td>non-saturated</td>
</tr>
<tr>
<td>$RT$</td>
<td>real time</td>
</tr>
<tr>
<td>$NRT$</td>
<td>non-real time</td>
</tr>
<tr>
<td>$L$</td>
<td>number of RT buffers in the system</td>
</tr>
<tr>
<td>${k} \cup A \cup B$</td>
<td>a partitioning of system queues to queue $k$ and two disjoint set of queues</td>
</tr>
<tr>
<td>$F_{ji}$</td>
<td>the maximal number of class $j$ sessions in the SP system if the delay requirement of a single class $i$ session should be kept and all other classes are empty</td>
</tr>
<tr>
<td>$\varphi(x, \mu, \sigma^2)$</td>
<td>density function of the normal distribution with mean $\mu$ and variance $\sigma^2$</td>
</tr>
<tr>
<td>$\Phi(x, \mu, \sigma^2)$</td>
<td>cumulative distribution function of the normal distribution</td>
</tr>
<tr>
<td>$\Phi(x)$</td>
<td>standard normal distribution function</td>
</tr>
<tr>
<td>$\Phi^{-1}(x)$</td>
<td>inverse of the standard normal distribution function</td>
</tr>
</tbody>
</table>

The ON-OFF behaviour causes a burst level fluctuation due to the burst of sessions being in ON state. A burst level input rate $R_k$ can be defined for a buffer $k$ based on the number of sources in ON state in traffic classes $j \in Q_k$ during $T_{\text{max}}$ such that $R_k(n) = \sum_{j \in Q_k} n_j \rho_j$, where $Q_k$ denotes the index set of traffic classes that are forwarded into buffer $k$. Similarly, a burst level service rate $S_k(n)$ can be also defined, which expresses the available transmission resources for buffer $k$ in a

\[
\Pi(n) = \mathbb{P} \{ \forall i \in K : A_i = n_i \} = \prod_{i \in K} \binom{N_i}{n_i} a_i^{n_i} (1 - a_i)^{N_i - n_i}. \tag{4.8}
\]
given $T_{max}$ period.

Burst scale QoS violation occurs if a buffer is overloaded, i.e. its input rate exceeds its service rate ($R_k > S_k$). The probability of such QoS violation in buffer $k$ can be calculated as

$$P_{burst}^k = \frac{\sum_{n_i > S_k(n)} n_i \cdot \Pi(n)}{\sum_{n_i} n_i \cdot \Pi(n)},$$

(4.9)

which is the ratio of packets lost due to burst scale QoS violation to the number of all packets. Burst scale operation, i.e. overload, does not cause QoS violation for buffer $k$ if

$$P_{burst}^k \leq \varepsilon_{burst}^k.$$  

(4.10)

The operation of the CB-WFQ system having RT and NRT queues has to be taken into account. For the analysis, real time (RT) queues having QoS requirement are first separated from non-real time (NRT) queues not having QoS requirement.

A lower bound for the service rate $S_k$ for queue $k$ in a WFQ system can be determined as

$$S_k \geq \frac{c_k}{c_k + \sum_{j \in B} c_j} \left( C - \sum_{j \in A} R_j \right),$$

(4.11)

where $\{k\} \cup A \cup B$ is any partitioning of queue indices as explained by Zhang [61]. The partitioning of the index sets applied in the following such that $\{k\} \cup A \equiv RT$ and $B \equiv NRT$. Applying this partitioning we get

$$S_k'(n) := M_k \left( C - \sum_{j \in A} R_j(n) \right) \leq S_k(n),$$

(4.12)

where $M_k = \frac{c_k}{c_k + \sum_{j \in NRT} c_j}$. This yields an upper bound for the loss ratio in buffer $k$. This bound is a good approximation for cases when the probability is small for the event that the system is overloaded but queue $k$ is not. If the system is overloaded and queue $k$ is not, then the overload is due to other queue(s). That is, the loss ratio of other overloaded queue(s) is high. If that queue is also a real time queue with similar tolerance level for QoS violation, then its QoS is also violated. Therefore, if all queues have similar QoS tolerance, then the bound is a good approximation for the loss ratio, and it is conservative in all cases.
Applying the central limit theorem, the distribution of the number of active sessions $A_i$ in a class can be approximated by a normal distribution if the number of sources $N_i$ grows. The distribution of the input rate in queue $k$ can be then approximated as

$$\mathbb{P}\left[\sum_{j\in Q_k} A_j \rho_j < x\right] \approx \Phi(x, r_k, v_k),$$  \hspace{1cm} (4.13)

where $r_k = \sum_{j\in Q_k} N_j \rho_j a_j$ is the mean and $v_k = \sum_{j\in Q_k} N_j \rho_j^2 a_j (1 - a_j)$ is the variance of the input rate in queue $k$. Using this approximation, the loss ratio bound can be written as

$$\int_{-\infty}^{\infty} \cdots \int_{-\infty}^{\infty} e^{-\sum_{j\in Q_k} N_j \rho_j a_j (1 - a_j)} dx_1 dx_2 \cdots dx_L$$

from which the following closed form approximation can be expressed for the burst level QoS violation probability:

$$P_{\text{burst}}^k = 1 - \Phi\left(C, \tilde{R}_k, \tilde{V}_k\right) + \tilde{V}_k \tilde{R}_k \cdot \varphi\left(C, \tilde{R}_k, \tilde{V}_k\right),$$  \hspace{1cm} (4.15)

where $\tilde{R}_k = M_k^{-1} r_k + \sum_{j\neq k} r_j$ and $\tilde{V}_k = M_k^{-2} v_k + \sum_{j\neq k} v_j$. The burst scale criterion of class $i$ is fulfilled if $P_{\text{burst}}^k$ of Equation (4.15) is smaller than $\epsilon_i^{\text{burst}}$ if $i \in k$.

Packet Scale QoS

The delay requirement is crucial in UTRAN because in many cases the receiver cannot use a packet if it has too large delay, i.e. the packet is dropped. Thus a QoS requirement may also be violated due to a temporary packet congestion even if the input rate of a buffer is smaller than its service rate ($R_k < S_k$). The probability of QoS violation due to large delay can be calculated as

$$P_{\text{packet}}^i = \frac{\sum_{n_i \cdot \Pr\left(D_i > d_i \mid A = n\right) \cdot \Pi(n)}}{\sum_{n_i \cdot \Pi(n)}}.$$  \hspace{1cm} (4.16)

which is the ratio of packets with excessive delay due to packet scale QoS violation in traffic class $i$ to the number of all packets sent in the same traffic class $i$. Packet scale operation does not cause QoS violation for class $i$ if
The packet scale constraint region of a traffic class is the set of traffic mixes that fulfil the packet scale QoS criteria of that class. The packet scale constraint surface of a given traffic class $i$ is the maximal number of admissible sessions in class $i$ with respect to class $i$ (class $i$ only) packet scale QoS criteria as the function of the number of sessions in other traffic classes. An important result of Malomsoky [62, 63] is that the packet scale constraint surface of a FIFO queue fed by heterogeneous periodic sources modulated by ON-OFF processes can be efficiently approximated by a hyperplane.

A packet level constraint surface provided by packet level simulations of a CB-WFQ system is shown in Figure 4.5. It is the packet level constraint surface of class 1 in a three-class system where each class has its own buffer in the scheduler. Each traffic class has its own packet level constraint surface. It can be observed that the surface is non-linear and decreasing. It reaches its maximum when none of the classes have session except for the observed one. Furthermore, it never falls below a certain level, which corresponds to the guaranteed minimum rate, even if other classes generate high load.

Decision 2 of the CAC algorithm has to be able to decide whether or not a traffic mix is under the packet scale constraint surface for each traffic class. There is no analytical formula for the description of the complex packet scale constraint surface of WFQ systems, therefore, the CAC has to apply approximations in order to be able to make a rapid decision. Two approximation approaches are described in the following.

![Figure 4.5: Simulation results for packet level constraint surface of class 1](image-url)
**Separated FIFO Model**

In this simple CAC algorithm, which can be applied for any rate-controlled scheduling method, each buffer $k$ is modelled as a stand-alone FIFO system with a capacity that equals to the guaranteed minimum rate provided by the scheduler, denoted by $c_k$: $S_k = c_k$. Therefore, the Separated FIFO model approximates the packet scale constraint surface of a given class by its absolute minimum level. The approximation for the packet scale constraint surface applied by the Separated FIFO approximation is illustrated in Figure 4.6 for a two-class system where the traffic classes have their own buffer.

Note that this model is used in the following as the simplest reference. The model ignores the fact that bandwidth not used by a queue can be redistributed among other queues. Therefore, it is a good approximation when other buffers are fully utilised. If large packets are segmented to sufficiently small segments in the node, then the operation of real time queues approximates a stand-alone FIFO system at shorter time scales, too, thus the resulting delay approximation is also conservative. Note that the influence of other classes can be considered by a worst case delay factor that is subtracted from the delay budget.

An admission control of FIFO systems applicable to UTRAN networks is described by Malomsoky [62]. It extends $n \cdot D/D/1$ queuing results of Roberts [77] for describing packet scale QoS violation and applies a specific method for checking burst scale QoS violation. Both packet scale and burst scale results can be directly applied in the CAC algorithm for any buffer $k$ by restricting the total set of traffic classes to $Q_k$, where $Q_k$ denotes the index set of traffic classes that are forwarded into buffer $k$.

In order to be able to achieve better system utilisation than provided by the Separated FIFO model, a better approximation is required for the packet scale operation of WFQ.

![Figure 4.6: Separated FIFO approximation of packet scale constraint](image)

Figure 4.6: Separated FIFO approximation of packet scale constraint
Separated Strict Priority Model

Service rate $S_k$ of a buffer $k$ could be substantially higher than its guaranteed minimum rate $c_k$ if some of the other buffers have an input rate smaller than their guaranteed rate. Therefore, the Separated FIFO model may result in a too pessimistic approximation for the queuing delay, which is the typical case if multiple real time classes are present in the system. Therefore, the aim in the following is to give a more accurate but still conservative approximation for the packet scale constraint regions of the traffic classes.

Separation of Saturated Queues

A lower bound for the service rate $S_k$ for queue $k$ in a WFQ system can be determined according to Equation 4.11, where $\{k\} \cup A \cup B$ is any partitioning of queue indices. The bound is conservative for each partitioning and it gives a good approximation if $A \equiv NSAT$ and $B \equiv SAT$, i.e. $A$ involves the \textit{lightly loaded} queues and $B$ involves the \textit{saturated} queues. The aim is to propose such equivalent systems to the original CB-WFQ system that allows to express the packet scale constraint region of class $i$ served in queue $k$.

In order to provide an equivalent system, the saturated buffers (with index set $B$) are first separated from the scheduler. Thus we get a reduced system that includes buffers in index set $A$ and buffer $k$. The service rate of the reduced system is

$$C' = \frac{c_k}{c_k + \sum_{j \in B} c_j} C,$$

which is an upper bound to the service rate of queue $k$.

The service order of packets depends on the actual value of the minimum bandwidth assignments ($c_i$). To give a worst case approximation for the queueing delay in buffer $k$, packets in buffers $A$ are assumed to be served as if they had priority over packets in buffer $k$.

The packet size has to be also adjusted for buffers in $A$ in order to achieve proper operation in the reduced system. These packets are served at the linkrate of the original system, which appears for a buffer $k$ packet in the reduced system as if the size of the packets in any buffer in $A$ were reduced to

$$b'_i = \frac{c_k}{c_k + \sum_{j \in B} c_j} b_i; \quad \forall i \in A.$$

As a result, the reduced system operates as a Strict Priority scheduler with parameters $C'$ and $b'$, therefore, this model is referred to as \textit{Separated Strict Priority} model. Depending on which buffers are considered to be saturated, $2^{L-1}$ different reduced systems can be distinguished, where $L$ is the number of real time
buffers. The Separated SP model gives a good approximation of the service rate of queue $k$ if the utilisation in buffers $A$ is low. Different reduced systems give good approximations under different traffic conditions, and a combination of them gives a conservative approximation for the service rate at any traffic mixes. Therefore, queueing delay of buffer $k$ packets in the model is an upper bound for their queueing delay in the original system.

Figure 4.7 illustrates the model in case of a three-queue system ($L = 3$), where the observed queue is queue 2 ($k = 2$) and queue 1 is saturated ($B = \{1\}$).

Figure 4.7: Separated SP model

Note that the Separated FIFO model is a special case of the Separated SP model, where each buffer except buffer $k$ is considered to be saturated. As traffic in NRT buffers is not controlled, they are always treated as saturated ones.

Multi-class Strict Priority System

By using the Separated SP model, the approximation problem of the packet scale constraint surface in the CB-WFQ system is reduced to the problem of approximating the packet scale constraint surface of low priority (LP) classes in multi-class strict priority systems. As shown by Malomsoky [63], the delay constraint surface of a LP class can be approximated by a single hyperplane and this approximation is conservative. The equation of the approximating hyperplane can be determined e.g. as follows. Denote $Y$ the index set of queues and $K_{SP}$ the number of classes in the SP system. An approximation hyperplane is given, if $K_{SP}$ points of the hyperplane are known. Therefore, the points of the approximation hyperplane of class $i$ are determined where only a single class $i$ session is in the system and all other classes but $j$ is empty, i.e. no ongoing sessions are present from other classes. Accordingly, $F_{ji}^Y$ is the maximal number of class $j$ sessions in the SP system if the delay requirement of a single class $i$ session should be met and all other classes are empty. Formally,

$$F_{ji}^Y = \max\left\{ N_j \sum_{n_j=0}^{N_j} \Pr[D_{n_j}^{ji} > d_i] \cdot \Pi(n_j) \leq \varepsilon_i^{delay} \right\}, \quad (4.20)$$
where \( P[D_i^n > d] \) is the delay distribution of a single class \( i \) session if the number of class \( j \) sessions in the system is \( n_j \). Two cases can be distinguished depending on whether class \( i \) and \( j \) are served in the same queue or not. It corresponds to a two class FIFO system if class \( i \) and class \( j \) are served in the same queue, thus \( n \cdot D/D/1 \) queueing results of Roberts [77] can be used. If class \( i \) and class \( j \) are served in different queues, this corresponds to a two class SP system where class \( j \) packets have priority over class \( i \) packets, and closed-form formulae presented by Tóth [79] can be applied.

The equation of an approximating hyperplane of the delay constraint of class \( i \) can be then written as

\[
\sum_{j \in Y} \frac{F_{ji}^Y}{F_{ji}^Y} \cdot N_j = F_{ii}^Y + 1.
\]

### Admission Control Based on Separated SP

Let us investigate the detailed operation of the CAC algorithm depicted in Figure 4.4. Admission Decision 1, i.e. QoS violation due to overload is checked based on Equation (4.15). The new traffic mix can only be admitted if \( P_{kB}^{burst} < \varepsilon_{kB}^{burst} \) for \( \forall k \in RT \).

Admission Decision 2 checks the new traffic mix against the packet constraint surface, i.e. to its approximation according to the Separated Strict Priority model. It has been shown above that \( 2^{L-1} \) different reduced systems can be constructed for a CB-WFQ system, which conservatively approximate the service rate of class \( i \). Accordingly, \( 2^{L-1} \) different hyperplanes can be determined, and all of them conservatively approximate the packet scale constraint surface of class \( i \) in the original system. Figure 4.8 demonstrates the linear approximation of the packet scale constraint surface of class 1 in a three-class system where each class has its own queue. Plane 1 belongs to the case when none of the queues is saturated. Plane 2 and Plane 3 correspond to the case when one of the queues is saturated and finally at Plane 4 both queue 2 and queue 3 are saturated. The approximation of the constraint region of class 1 can be obtained as the union of the regions bordered by the approximating surfaces.

If traffic mix \( N \) is below at least one of the approximating hyperplanes in the space of the number of sessions, then the specific packet scale constraint is fulfilled for class \( i \). This is tested by checking

\[
\max_Y \left( F_{ii}^Y + 1 - \sum_{j \in Y} \frac{F_{ji}^Y}{F_{ji}^Y} \cdot N_j \right) > 0.
\]

This test is performed for each traffic class. Note that the coefficients of the
approximating hyperplanes are computed and stored during the initialization phase of the CAC algorithm.

The packet scale constraint region, i.e. the admissible region for the entire system, is the intersection of the packet scale constraint regions of the individual traffic classes, which is illustrated in Figure 4.9 for a two-class system.

Summarising the admission decision: a traffic mix can be admitted if packet scale criterion of Equation (4.22) is fulfilled for each traffic class and burst scale criterion of Equation (4.15) is fulfilled for each RT buffer.

Figure 4.9: Separated Strict Priority approximation of packet scale constraint
Evaluation of the CAC Algorithm

The Separated FIFO and Separated Strict Priority CAC algorithms were implemented in a call level simulator to compare their bandwidth efficiency and the results were published in Farkas [C10]. In order to have a reference providing optimal performance, an ideal CAC algorithm was also implemented, which incorporates packet level simulations for the evaluation of QoS characteristics. The ideal CAC admits all combination of sessions for which the QoS characteristics meet the QoS requirements.

A Poisson process with exponential holding time distribution is assumed for connection arrival. The maximal blocking probability for each traffic class is 0.01. Each $T=20$ ms, QoS violation probability is 0.01, $b_1=40$ bytes, $b_2=200$ bytes and $b_3=1000$ bytes, the segmentation size is 100 bytes. The activity factors are $a_1=0.5$ and $a_2=a_3=1.0$. The linkrate, the delay requirements and the CB-WFQ weight settings are changed during the simulations. The linkrate is denoted by $C$, $w = (w_1, w_2, w_3)$ is the weight vector and $d = (d_1, d_2, d_3)$ is the delay requirement vector where the delay values are in ms. Each class had its own queue.

The first round of simulations was performed using the ideal CAC in each scenario, where the linkrate was fixed and the traffic load was increased to its maximum value at which the QoS requirements were met. The ratio of the traffic volume generated by different classes was set proportionally to the weight setting in the CB-WFQ scheduler. For the load obtained in the previous step, the CAC algorithm of the Separated FIFO and the Separated Strict Priority models were applied and their capacity need to meet the QoS requirements was determined. The capacity need of the traffic classes was determined independently of each other for the Separated FIFO system in order to reflect its operation; its total capacity need is the sum of the capacity need of individual classes. The ratio of the individual capacities gives the optimal weight setting if FIFO separation is applied to approximate the WFQ system.

Table 4.2 shows the results as a function of the linkrate when $d = (2, 4, 8)$ and $w = (4, 2, 1)$ is the weight setting at the ideal CAC simulations and at the Separated Strict Priority simulations. The last column in Table 4.2 shows the optimal weight setting for FIFO Separation.

We can see that the larger the linkrate the smaller the difference between the

<table>
<thead>
<tr>
<th>Offered load [Erlang]</th>
<th>Ideal</th>
<th>Sep. Prio</th>
<th>FIFO</th>
<th>$w_{FIFO}$</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\lambda = (94.8, 4.7, 0.47)$</td>
<td>3840</td>
<td>3898</td>
<td>5687</td>
<td>(0.41, 0.51, 1)</td>
</tr>
<tr>
<td>$\lambda = (340, 17.0, 1.7)$</td>
<td>7680</td>
<td>8211</td>
<td>9739</td>
<td>(0.91, 0.73, 1)</td>
</tr>
<tr>
<td>$\lambda = (476, 23.8, 2.4)$</td>
<td>9600</td>
<td>10548</td>
<td>11883(1.12, 0.8, 1)</td>
<td></td>
</tr>
<tr>
<td>$\lambda = (610, 30.5, 3.0)$</td>
<td>11520</td>
<td>12802</td>
<td>13804</td>
<td>(1.3, 0.86, 1)</td>
</tr>
</tbody>
</table>
bandwidth efficiency of the CAC methods. The reason for this is that if the linkrate exceeds a given limit, then the packet scale operation does not dominate any more. As the CAC methods mainly differ in packet scale behaviour, the effect of difference diminishes at large links. The application of FIFO Separation instead of Strict Priority Separation may result in 46% increase in the capacity need.

That is, the proposed CAC algorithm based on the Separated Strict Priority model is able to take advantage of the flexibility provided by the CB-WFQ scheduler in an IP UTRAN network.

4.5 Connection Admission Control Algorithm for Flow Level QoS Guarantees

The CAC algorithm has to evaluate whether a newly arriving connection would cause overload taking into account the ON-OFF behaviour of traffic sources, which is Decision 1 in Figure 4.4. This decision is generally performed on aggregates of traffic flows, e.g. based on Equation (4.15). Nevertheless, if the admission control is designed for only taking into account aggregate loss requirement, then the loss rate of individual flows may violate their requirement even if they are admitted by the admission control. Therefore, this section goes beyond QoS for aggregates.

If QoS is aimed to be guaranteed at flow level not only for aggregates, then the QoS definition of Farkas [C9] should be applied. Due to the importance of the voice traffic, ensuring quality for voice flows might be of special consideration of network operators. The QoS requirement for aggregates, e.g. for a traffic class is typically determined by the delay requirement $d$ and packet drop requirement $\varepsilon$. However, besides the number of admitted flows, the fraction of dropped packets also depends on the activity factor of ongoing flows. The activity factor can take any value in $(0, 1)$ interval in practical cases according to Westholm [68]. Therefore, if packet loss rate was required to be less than $\varepsilon$ for all flows, then we would need to calculate with an activity factor of 1 for each flow in the admission control. Nevertheless, this would inherently exclude any gain from the multiplexing of voice sources. As a reasonable compromise proposed by Farkas [C9], it might be required instead that the $\delta$ probability of the violation of the $\varepsilon$ packet drop rate requirement is small.

The flows of traffic sources belonging to the same traffic class, e.g. voice, are fed into the same queue as illustrated in Figure 4.2. This single class system is in the focus of the following analysis. Bufferless multiplexing is applied to evaluate the burst level operation, which is also applied when taking into account the flow level QoS requirement. The multiplexer serves $Z = C \cdot T/b$ packets in a source period, and multiplexes $N$ ON-OFF sources with random activity factor. If more packets arrive in a source period $T$ than the system capacity $Z$, then all packets are considered lost, which is a worst case assumption that is valid when the buffer
Table 4.3: Parameters of the model

<table>
<thead>
<tr>
<th>Parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>$Z$</td>
<td>$CT/b$ server capacity</td>
</tr>
<tr>
<td>$N$</td>
<td>number of admitted ON-OFF sources</td>
</tr>
<tr>
<td>$\varepsilon$</td>
<td>max allowed packet drop rate</td>
</tr>
<tr>
<td>$\delta$</td>
<td>probability of not fulfilling drop requirement</td>
</tr>
<tr>
<td>$\alpha_j$</td>
<td>random activity factor of flow $j$; $\alpha_j \sim F(x)$</td>
</tr>
</tbody>
</table>

size $B$ is larger than the delay requirement $d$. The activity factor distribution can be arbitrary. The parameters used in the model are summarised in Table 4.3.

The QoS function specified in Farkas [C9] indicates whether the fraction of dropped packets is below $\varepsilon$. Assuming fixed $\alpha_1 = a_1, \ldots, \alpha_N = a_N$ values for the activity factors of the ongoing flows, the QoS function is

$$Q(a_1, \ldots, a_N) = \begin{cases} 
0, & \text{fraction of dropped packets is below } \varepsilon; \\
1, & \text{otherwise.} 
\end{cases}$$

(4.23)

That is, QoS violation is indicated by $Q = 1$.

Denote $f(a_1, \ldots, a_N)$ the joint density function of the random activity factors. Applying the law of total probability, the probability that the system does not meet the flow level QoS requirement for any of the ongoing flows can be calculated as follows:

$$\delta = \int_{a_1, \ldots, a_N} Q(a_1, \ldots, a_N) \, f(a_1, \ldots, a_N) \, da_1 \ldots da_N.$$ 

(4.24)

I proposed an efficient method for the evaluation of $\delta$ of Equation (4.24) in Farkas [C9], which is also described in the following.

The CAC Evaluates the QoS Requirement

The probability of drop rate violation has to be determined for a new connection request. Let us assume that the admission request is for the $N$-th flow, i.e. $N - 1$ flows are already in the system.

The admission control algorithm should calculate the probability of larger drop rate than the requirement at $N$ number of flows. If it is larger than its allowed maximum, then the new connection is rejected, otherwise it is admitted.

I propose to determine the probability of drop rate violation $\delta_N$ for $N$ flows as follows:

$$\delta_N = \Phi \left( \Phi^{-1} (1 - \varepsilon) ; \mu = \mathbb{E} [Y], \sigma^2 = \mathbb{E} [Y^2] - \mathbb{E} [Y]^2 \right),$$

(4.25)

where $Y = \left( Z - \sum_{i=1}^N \alpha_i \right) / \left( \sqrt{\sum_{i=1}^N \alpha_i (1 - \alpha_i)} \right)$ and the mean and variance of $Y$ can be determined as described later in detail.
The new $N$-th flow can be admitted if $\delta_N < \delta$.

The analysis for determining $\delta$ is provided in the following. An accurate QoS function based on Gaussian approximation is first presented. The integral in Equation (4.24) is then evaluated in an efficient manner.

**Evaluating the QoS Function for Fixed Activity Factors**

The random nature of the activity factor is not yet taken into account; the activity factors first considered to have a single value. According to the bufferless model, the QoS requirement on the fraction of dropped packets is fulfilled if:

$$\mathbb{P}[\text{the number of sources in state ON } \geq Z] \leq \varepsilon, \quad (4.26)$$

which is the QoS defined on aggregation of flows.

The distribution of the number of sources in state ON depends on the values of the activity factors. Denote $a_i$ the constant activity factor of source $i$. The distribution of the number of sources in state ON follows the sum of $N$ independent Bernoulli random variables with parameter $a_i$, i.e.

$$\mathbb{P}[\text{the number of sources in state ON } \leq Z] = \mathbb{P} \left[ \sum_{i=1}^{N} X_{a_i} \leq Z \right], \quad (4.27)$$

where $X_p$ is a Bernoulli random variable with parameter $p$.

The calculation of the distribution of the sum of Bernoulli random variables was extensively studied in the literature. Two well-known approximations of the sum in Equation (4.27) are the Chernoff bound and the Gaussian model explained by Kelly [73]. The Gaussian approximation is used here, which applies the central limit theorem and approximates the sum of the Bernoulli random variables with the normal distribution. This means that

$$\mathbb{P} \left[ \sum_{i=1}^{N} X_{a_i} \leq Z \right] \approx \Phi \left( \frac{Z - \sum_{i=1}^{N} a_i}{\sqrt{\sum_{i=1}^{N} a_i (1 - a_i)}} \right). \quad (4.28)$$

Finally, the QoS function $Q$, which indicates whether the fraction of dropped packets is below $\varepsilon$, is as follows:

$$Q(a_1, \ldots, a_N) = \begin{cases} 1 & \frac{Z - \sum_{i=1}^{N} a_i}{\sqrt{\sum_{i=1}^{N} a_i (1 - a_i)}} \leq \Phi^{-1}(1 - \varepsilon) \\ 0 & \text{otherwise} \end{cases} \quad (4.29)$$
where the $\Phi(x)$ and $\Phi^{-1}(x)$ denote the standard normal distribution function and its inverse. The $\Phi(x; \mu, \sigma^2)$ denotes the distribution function of a normally distributed random variable with mean $\mu$ and variance $\sigma^2$.

**Evaluating the QoS Function for Random Activity Factors**

The randomness of the activity factor is taken into account here. The random activity factors of $N$ sources $\alpha_1, \ldots, \alpha_N$ are independent and identically distributed random variables. The effect of random activity factors on the QoS is illustrated in Appendix B.1. The value of the flow level QoS measure $\delta$ can be obtained by substituting Equation (4.29) into Equation (4.24) and using the joint density function of the activity factors:

$$
\delta = \mathbb{P} \left[ \frac{Z - \sum_{i=1}^{N} \alpha_i}{\sqrt{\sum_{i=1}^{N} \alpha_i(1 - \alpha_i)}} \leq \Phi^{-1}(1 - \varepsilon) \right].
$$

The direct calculation of $\delta$ is very complicated even for small values of $N$. This motivates the application of an accurate and fast approximation of $\delta$ for a CAC algorithm. For the sake of easier calculation of $\delta$ let us introduce

$$
\mathcal{Y} = \frac{Z - \sum_{i=1}^{N} \alpha_i}{\sqrt{\sum_{i=1}^{N} \alpha_i(1 - \alpha_i)}}.
$$

The following steps are applied to derive $\delta$:

- show that $\mathcal{Y}$ is normally distributed independently of the distribution of activity factors;
- determine the mean and the variance of $\mathcal{Y}$ using the first moments of the activity factor;
- and calculate $\delta = \mathbb{P}[\mathcal{Y} \geq \Phi^{-1}(1 - \varepsilon)]$.

The following two theorems give the formal description of the solution method explained above.

**Theorem 4.1.** The distribution of $\mathcal{Y} = \frac{Z - \sum_{i=1}^{N} \alpha_i}{\sqrt{\sum_{i=1}^{N} \alpha_i(1 - \alpha_i)}}$ follows a normal distribution with negligible error.

**Proof**

The derivation has three steps:
1. The random variables $\sum_{i=1}^{N} \alpha_i$ and $\sum_{i=1}^{N} \alpha_i(1 - \alpha_i)$ are normally distributed for large $N$;

2. The random variable $\sqrt{\sum_{i=1}^{N} \alpha_i(1 - \alpha_i)}$ is also normally distributed;

3. The ratio of two normal random variables approximately follows a normal distribution too.

The first step directly follows from the central limit theorem.

For the proof of the second step, assume an $X_N$ normally distributed random variable with mean $\mu_N$ and variance $\sigma_N^2$. We are interested in the distribution of $\sqrt{X_N}$ if $\mu_N \to \infty$. The Taylor expansion of the square root function at $\mu_N$ is

$$\sqrt{x} = \sqrt{\mu_N} + \frac{1}{2} \frac{x - \mu_N}{\sqrt{\mu_N}} + \frac{1}{8} \frac{(x - \mu_N)^2}{\mu_N \sqrt{\mu_N}} + O \left( (x - \mu_N)^3 \right).$$

Substituting $x$ with the random variable $X_N$ in the series we get

$$\sqrt{X_N} = \sqrt{\mu_N} + \frac{1}{2} \frac{X_N - \mu_N}{\sqrt{\mu_N}} + \frac{1}{8} \frac{(X_N - \mu_N)^2}{\mu_N \sqrt{\mu_N}} + O \left( (X_N - \mu_N)^3 \right) \sim \sqrt{\mu_N} + \frac{\sigma_N^2}{2\sqrt{\mu_N}} \Phi(x) + \frac{\sigma_N^2}{8\mu_N \sqrt{\mu_N}} \chi_2^2,$$

where the $\chi_2^2$ denotes the chi-square distribution with two degrees of freedom. In this case $X_N$ has to be substituted with $\sum_{i=1}^{N} \alpha_i(1 - \alpha_i)$. The coefficients of $\chi_2^2$ in Equation (4.32) tend to zero if $N \to \infty$:

$$\frac{\sigma_N^2}{\mu_N \sqrt{\mu_N}} = \frac{N (\mu_2 - 2\mu_3 + \mu_4 - \mu_2^2 + 2\mu_1 \mu_2 - \mu_2^2)}{N^{3/2} (\mu_1 - \mu_2)^{3/2}} \to 0,$$

where $\mu_j$ denotes the $j$-th moment of the activity factor. Finally this means that for a large $N$ the square root transformation of $\sum_{i=1}^{N} \alpha_i(1 - \alpha_i)$ tends to a normal distribution with mean $\sqrt{\mu_N}$ and variance $\frac{\sigma_N^2}{\sum_{i=1}^{N} \alpha_i}$.

For the proof of the third step, the ratio of two correlated normal random variables has to be evaluated, which is the numerator and the denominator of $Y$. Hinkley [80] studied this ratio and examined the exact distribution. He showed that a normal approximation can be applied when the denominator is greater than zero even if the numerator and denominator are correlated. Applying these results we obtain the theorem. \qed
Theorem 4.2. The first and the second moments of $Y$ can be calculated as infinite Taylor:

$$
\begin{align*}
\mathbb{E}[Y] &= A(\mu_1) + C(\mu_1) N \sigma^2 + E(\mu_1) N \mathbb{E}[(\alpha - \mu_1)^3] + \ldots \\
\mathbb{E}[Y^2] &= A^2(\mu_1) + N \sigma^2 \times \\
&\quad \left[ B^2(\mu_1) + 2 A(\mu_1) C(\mu_1) \right] + N \mathbb{E}[(\alpha - \mu_1)^3] \times \\
&\quad [2A(\mu_1)E(\mu_1) + 2B(\mu_1)C(\mu_1)] + \ldots
\end{align*}
$$

where $\mu_i$ is the $i$-th moment, $\sigma^2$ is the variance of the activity factor and

$$
\begin{align*}
A(a) &= \frac{Z - Na}{\sqrt{Na(1-a)}} \\
B(a) &= -\frac{1}{2} \frac{(Z - Na)(1 - 2a)}{[Na(1-a)]^{3/2}} - \frac{1}{\sqrt{Na(1-a)}} \\
C(a) &= \frac{1}{2} \frac{Z - Na}{[Na(1-a)]^{3/2}} + \frac{3}{8} \frac{(Z - Na)(1 - 2a)^2}{[Na(1-a)]^{5/2}} + \\
&\quad \frac{1}{2} \frac{1}{[Na(1-a)]^{3/2}} - \frac{9}{4} \frac{(Z - Na)(1 - 2a)^2}{[Na(1-a)]^{5/2}} - \\
&\quad \frac{15}{8} \frac{(Z - Na)(1 - 2a)^3}{[Na(1-a)]^{7/2}} - \\
E(a) &= \frac{3}{4} \frac{(Z - Na)(1 - 2a)^2}{[Na(1-a)]^{5/2}} - \frac{9}{2} \frac{(Z - Na)(1 - 2a)}{[Na(1-a)]^{5/2}}.
\end{align*}
$$

Proof

The Taylor expansion of the function $f(x_1, \ldots, x_N)$ is first determined, which is defined as follows:

$$
f(x_1, \ldots, x_N) = \frac{Z - \sum_{i=1}^{N} x_i}{\sqrt{\sum_{i=1}^{N} x_i (1 - x_i)}}.
$$

This function is symmetric in $x_1, \ldots, x_N$ and takes $\infty$ if at least one of the $x_i$ is 0 or 1. The symmetric property ease the Taylor expansion. The structure of the Taylor expansion of $f(x_1, \ldots, x_N)$ around $x_1 = a, \ldots, x_N = a$ is the following:

$$
\begin{align*}
&f(x_1, \ldots, x_N) = A(a) + B(a) \sum_{i=1}^{N} (x_i - a) + \\
&C(a) \sum_{i=1}^{N} (x_i - a)^2 + D(a) \sum_{i \neq j} (x_i - a) (x_j - a) + \\
&\quad E(a) \sum_{i=1}^{N} (x_i - a)^3 + F(a) \sum_{i \neq j} (x_i - a)^2 (x_j - a) + \ldots
\end{align*}
$$

85
and the value of the coefficients is as follows:

\[
\begin{align*}
A(a) &= f(a, \ldots, a) \\
B(a) &= \frac{\partial f(x_1, \ldots, x_N)}{\partial x_1}
\bigg|_{x_1=a, \ldots, x_N=a} \\
C(a) &= \frac{1}{2!} \left( \frac{\partial^2 f(x_1, \ldots, x_N)}{\partial x_1^2} \right)
\bigg|_{x_1=a, \ldots, x_N=a} \\
D(a) &= \frac{\partial^2 f(x_1, \ldots, x_N)}{\partial x_1 \partial x_2}
\bigg|_{x_1=a, \ldots, x_N=a} \\
E(a) &= \frac{1}{3!} \left( \frac{\partial^3 f(x_1, \ldots, x_N)}{\partial x_1^3} \right)
\bigg|_{x_1=a, \ldots, x_N=a} \\
F(a) &= \frac{1}{2!} \left( \frac{\partial^3 f(x_1, \ldots, x_N)}{\partial x_1^2 \partial x_2^2} \right)
\bigg|_{x_1=a, \ldots, x_N=a}.
\end{align*}
\]

Evaluating this we get the formulae of the statement. The Taylor expansion of the function \(f^2(x_1, \ldots, x_N)\) around \(x_1 = a, \ldots, x_N = a\) is then provided. The first terms of the Taylor expansion are the following:

\[
\begin{align*}
f^2(x_1, \ldots, x_N) &= A^2(a) + 2A(a)B(a) \sum_{i=1}^N (x_i - a) + \\
&\quad \left[ B^2(a) + 2A(a)C(a) \right] \sum_{i=1}^N (x_i - a)^2 + \\
&\quad \left[ 2A(a)D(a) + B^2(a) \right] \sum_{i=1}^N \sum_{i \not= j} (x_i - a)(x_j - a) + \\
&\quad \left[ 2A(a)E(a) + 2B(a)C(a) \right] \sum_{i=1}^N (x_i - a)^3 + \\
&\quad \left[ 2A(a)F(a) + 2B(a)C(a) + 4B(a)D(a) \right] \sum_{i=1}^N \sum_{i \not= j} (x_i - a)^2 (x_j - a) + \ldots
\end{align*}
\]

The function \(f(x_1, \ldots, x_N)\) is then applied to the random activity factors. Denote \(\mu_1\) the mean and \(\sigma^2 = \mu_2 - \mu_1^2\) the variance of the random activity factors \(\alpha_i, i = 1, \ldots, N\). \(\mathbb{E}[f(\alpha_1, \ldots, \alpha_N)]\) is calculated using the Taylor expansion of the function \(f(x_1, \ldots, x_N)\):

\[
\mathbb{E}\left[ \frac{Z - \sum_{i=1}^N \alpha_i}{\sqrt{\sum_{i=1}^N \alpha_i (1 - \alpha_i)}} \right] = \mathbb{E}[f(\alpha_1, \ldots, \alpha_N)] = A(\mu_1) + C(\mu_1)N\sigma^2 + E(\mu_1)N \mathbb{E}\left[ (\alpha - \mu_1)^3 \right] + \ldots
\]
Similarly, the second moment of \( f(\alpha_1, \ldots, \alpha_N) \) can be calculated as:

\[
\mathbb{E} \left[ \left( \frac{Z - \sum_{i=1}^{N} \alpha_i}{\sqrt{\sum_{i=1}^{N} \alpha_i (1 - \alpha_i)}} \right)^2 \right] = \mathbb{E} \left[ f^2(\alpha_1, \ldots, \alpha_N) \right] = A^2(\mu_1) + \left[ B^2(\mu_1) + 2A(\mu_1)C(\mu_1) \right] N\sigma^2 + [2A(\mu_1)E(\mu_1) + 2B(\mu_1)C(\mu_1)] N\mathbb{E} \left[ (\alpha - \mu_1)^3 \right] + \ldots
\]

Two example distributions of the activity factor are used to illustrate the numerical properties of the proposed method. In Figure 4.10(a), the distribution of \( \alpha \) is according to the GSM speech measurements of Westholm [68], which is depicted in Figure B.1 in Appendix B.2. The distribution of \( \alpha \) in Figure 4.10(b) is uniform over the interval [0.48, 0.68]. Figure 4.10(a) and 4.10(b) illustrate that the distribution of \( Y \) is close to a normal distribution even for a small value of \( N \) (\( N = 10 \)).

The proposed approximation for the distribution of \( Y \) is also shown in the figures. The fitted distribution is a normal distribution with the mean and the variance got from the exact distribution. It is seen in Figure 4.10(b) that the proposed and the fitted distribution run together very close to the exact distribution. The GSM speech activity factor distribution is spread over a much wider interval than the examined uniform distribution, which causes the small difference between the proposed and the exact distribution in Figure 4.10(a). Nonetheless, the exact and the fitted distributions run together in this figure too, which illustrates that \( Y \) has normal distribution. Note that Figure 4.10(a) and 4.10(b) illustrate the accuracy of the proposed method for \( N = 10 \). The error is even smaller for a larger \( N \).

Figure 4.10: Distribution of \( Y \): \( N = 10, Z = 8, \varepsilon = 0.001 \)
Application in a Connection Admission Control Algorithm

The above results can be applied in an admission control algorithm that takes into account flow level characteristics as proposed in Farkas [C9]. Firstly, the input parameters of the admission control algorithm should be determined in the initialisation phase of Figure 4.4. As Table 4.4 shows, the activity factor should be characterized by its mean, variance and the third moment of the distribution, i.e. \( \mu_1, \sigma^2 \) and \( \mu_3 \). These parameters can be determined off-line by means of measurements or e.g. the GSM distribution published by Westholm [68] can be used. The remaining input parameters are the drop requirement \( \varepsilon \), its allowed violation probability \( \delta \) and the system capacity \( Z \), which all are fixed system parameters. The only variable input of the admission control is the number of actual connections \( N \).

The on-the-fly evaluation part of the admission control algorithm can be performed if the input parameters for the flow level admission control are determined during the initialisation phase as shown in Figure 4.4. The admission control algorithm calculates the probability of having larger drop rate than the requirement if the admitted number of flows became \( N \). If it is larger than its allowed maximum, then the new connection is rejected, otherwise it is admitted. The \( \delta_N \) probability of drop rate violation for \( N \) flows, can be calculated according to Equation (4.30) and using the results proposed in Theorem 4.1 as follows:

\[
\delta_N = \Phi \left( \Phi^{-1} (1 - \varepsilon); \mu = \mathbb{E} [Y], \sigma^2 = \mathbb{E} [Y^2] - \mathbb{E} [Y]^2 \right).
\]

(4.33)

The mean and variance of \( Y \) should be determined based on Theorem 4.2. If \( \delta_N < \delta \), then the result of the decision is ”Admit”, otherwise it is ”Reject”.

This admission control has fast execution time because the obtained formula includes only basic operations.

Table 4.4 also shows the parameters of the usual aggregated QoS models in addition to the proposed flow level model. The admission criterion for fulfilling

<table>
<thead>
<tr>
<th>Aggregation level model</th>
<th>Flow level model</th>
</tr>
</thead>
<tbody>
<tr>
<td>QoS measure</td>
<td>The probability that a packet suffers higher delay than a predefined target delay value should be less than ( \varepsilon )</td>
</tr>
<tr>
<td>Model parameters</td>
<td>( N, Z )</td>
</tr>
<tr>
<td>Activity descriptor</td>
<td>( \mu_1 )</td>
</tr>
<tr>
<td>Output</td>
<td>Admit or Reject</td>
</tr>
</tbody>
</table>
the aggregate QoS requirement using Gaussian approximation is expressed by Kelly [73] (see (3.13)). It has the following form with the notations applied here:

\[\varepsilon \geq \Phi \left( \frac{Z - N\mu_1}{\sqrt{N\mu_1(1 - \mu_1)}} \right).\] (4.34)

This expression reinforces the former statement that activity factor is considered only with its mean value in the aggregate QoS model.

It is interesting to see the \(\delta_{\text{aggr}}\) probability of violating the packet loss requirement \(\varepsilon\) if the aggregate admission control algorithm is used by the same \(\varepsilon\) setting. The probability \(\delta_{\text{aggr}}\) can be obtained from Equation (4.33) by substituting \(\varepsilon\) from Equation (4.34) assuming equality:

\[\delta_{\text{aggr}} = \Phi \left( 0; \mu = \mathbb{E}[Y] - A(\mu_1), \sigma^2 = \mathbb{E}[Y^2] - \mathbb{E}[Y]^2 \right),\]

where \(A(\mu_1)\) is defined in Theorem 4.2. If \(\mathbb{E}[Y]\) equals to the first term of its Taylor series \(A(\mu_1)\), then \(\delta_{\text{aggr}} = 50\%\), which means that flows violate the drop requirement with 50\% probability.

The capacity requirement as a function of the number of multiplexed sources and the value of \(\delta\) is evaluated for two sample activity factor distributions in the following. Table 4.5 shows the capacity requirements if the system has to carry \(N\) multiplexed voice flows. Uniform activity factor distribution over (0.2–0.8) and the GSM speech activity factor distribution were used. The packet drop requirement \(\varepsilon = 1\%\) and the \(\delta\) probability of its violation runs from 0.001\% to 50\%.

We can see that the results for the two activity factor distributions are similar. The small difference can be attributed to the different mean values, which is 0.5 for the uniform distribution and 0.58 in case of the GSM speech.

Table 4.5: Capacity requirement \(Z\) for \(N\) sources with \(\varepsilon = 1\%\) and different activity factor distributions

<table>
<thead>
<tr>
<th>(\delta)</th>
<th>Uniform distribution over (0.2-0.8)</th>
<th>GSM speech distribution</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>50%</td>
<td>1%</td>
</tr>
<tr>
<td>N=20</td>
<td>15</td>
<td>17</td>
</tr>
<tr>
<td>N=40</td>
<td>27</td>
<td>30</td>
</tr>
<tr>
<td>N=60</td>
<td>39</td>
<td>42</td>
</tr>
<tr>
<td>N=80</td>
<td>50</td>
<td>53</td>
</tr>
<tr>
<td>N=100</td>
<td>61</td>
<td>65</td>
</tr>
<tr>
<td>N=120</td>
<td>73</td>
<td>76</td>
</tr>
<tr>
<td>N=140</td>
<td>83</td>
<td>87</td>
</tr>
<tr>
<td>N=160</td>
<td>94</td>
<td>99</td>
</tr>
<tr>
<td>N=180</td>
<td>105</td>
<td>110</td>
</tr>
<tr>
<td>N=200</td>
<td>116</td>
<td>121</td>
</tr>
</tbody>
</table>
The capacity requirement significantly changes if small \( \delta \) values (< 1%) are used instead of the \( \delta = 50\% \) setting. For example, if \( N = 200 \) and \( \delta \) changes from 50\% to 0.001\%, then the difference is 8 units at both activity factor distributions.

Figure 4.11 shows the plot of \( \delta \) against the variance of the activity factor at three capacity values \((Z = 40; 41; 42)\). The activity factors in different cases have the same mean \((0.5)\) and they have uniform distribution in different ranges. The figure shows that higher variance of the activity factor causes larger \( \delta \) value, i.e. more capacity is needed to achieve the same QoS. For example, if the variance grows from 0.003 to 0.03 then \( \delta \) increases from 0.8\% to 9.3\% at \( Z = 40 \).

That is, the proposed Gaussian approximation for the bufferless multiplexing model and the approximation of the packet loss violation probability with the quantile of a normal distribution obtained with the Taylor expansion of the random activity factors can be applied in a connection admission control algorithm ensuring flow level QoS guarantees.

### 4.6 Conclusions

This chapter investigated QoS assurance on the Iub interface of UTRAN. A Connection Admission Control was specified for IP UTRAN networks that apply CB-WFQ scheduling for the service of their buffers. The CAC algorithm takes into account the packet scale operation of CB-WFQ, which is crucial due to the stringent delay requirements of UTRAN. Furthermore, the CAC utilises the WFQ feature that unused bandwidth reserved for a traffic class may be used by other traffic. The packet scale part of the CAC algorithm models CB-WFQ by a set of strict priority systems with reduced capacity. The evaluation of the capacity need of the proposed CAC algorithm by means of simulations showed its efficient bandwidth.
utilisation at small link capacities.

A refinement for admission control algorithms was then described in order to guarantee per flow QoS, which may be especially important for voice traffic. The proposed method is based on Gaussian approximation applied for the bufferless multiplexing model. The packet loss violation probability is approximated as the quantile of a normal distribution. The parameters of the normal distribution are obtained with the Taylor expansion of a random variable, which is the function of the random activity factors. The obtained admission control formula includes the moments of the activity factor distribution.
Chapter 5

Summary

The evolution of access and metro area networks brought about a number of challenges that motivated the work presented in this dissertation. The quality requirements of services carried over these packet networks demand for new algorithms and protocols for the control of these networks. Providing resilience is a basic component for being able to meet QoS requirements. Ethernet is recently playing an important role in access and metro networks, which requires new techniques for the adaptation of this long existing technology to new networking fields. Ethernet often provides connectivity for IP, which is being deployed in UTRAN access networks. The QoS requirements of UMTS traffic necessitate the application of special algorithms, e.g. connection admission control in the IP network.

5.1 Resilient Ethernet

In Chapter 2, I provided a resilient architecture for Ethernet networks and I experimentally validated it. I have proposed a lightweight and efficient protection technique for a robust Ethernet architecture. The network is comprised of IEEE 802.1Q-2005 standard Ethernet bridges in its core and new functionality is only used in the edge nodes. The proposed high availability architecture was implemented in a prototype network.

- I have given an in-band fast failure handling protocol that operates in the Ethernet layer. The protocol is implemented in the edge nodes of the network and relies on three types of broadcast messages. The performance and robustness of the protocol were validated by means of extensive testing. Experimental results showed that the worst case failover can be maintained
below 50 ms with 1% protocol overhead of the link capacity in a Fast Ethernet network.

• I have provided a lower bound for the number of spanning trees that are needed to provide protection against a single link failure as a function of the number of nodes and links comprising the physical topology.

• I have given an algorithm to determine the fault tolerant trees for the resilient architecture, which aims at minimising the number of trees. I have evaluated the algorithm on random topologies and shown that the results approach the lower bound provided in this dissertation.

• I have proposed an automatic physical topology discovery algorithm for heterogeneous multi-vendor Ethernet networks that is able to determine the entire physical topology of the network and keep the topological database updated as links or nodes fail, are added or removed. The proposed algorithm provides input to the fault tolerant tree computation algorithm. The experimental results performed under various topology configurations in multiple vendor networks have shown that the mechanism is accurate and robust.

The results of Chapter 2 were published in [J3, C3, C4, C6, C7, C8]. Patent application [P16] was submitted on the failure handling protocol. The fault tolerant tree computation algorithm is also described in patent application [P15]. Besides Ethernet applications, the algorithm can be also applied within the multi-topology routing framework proposed by Menth [35] and Čičić [33] for IP resilience. Patent application [P11] was submitted on the topology discovery algorithm and [P12] provides a fault localisation method. Furthermore, the prototype implementation of the resilient Ethernet architecture described in Chapter 2 was the winner of an Ericsson-wide prototype competition in 2006.

The results can be applied, e.g. in the field of carrier Ethernet networks. Protection for point-to-point and point-to-multipoint Ethernet services can be provided by the Provider Backbone Bridge Traffic Engineering (PBB-TE) [81] standard, which applies similar principles to the resilient architecture described in this dissertation. Furthermore, protection for multipoint services in an SPB controlled Ethernet network can be implemented along the principles of the resilient architecture as described in detail in [P1].
5.2 Enhancements to Shortest Path Bridging

Chapter 3 provides link state protocol extensions for the control of an SPB network. Loop prevention is a crucial extension to a link state protocol for its application in Ethernet because a loop may cause serious problems for bridged networks.

- I have defined the Neighbour Synchronisation algorithm for preventing transient loops in link state networks. This loop prevention algorithm applies a synchronisation mechanism for the link state database of neighbour nodes before they exchange data packets. The Neighbour Synchronisation can be implemented as an add-on to a standard link state protocol.

- I have evaluated the effect of the Neighbour Synchronisation on network convergence time. The measurement and simulation results showed that the Neighbour Synchronisation loop prevention does not deteriorate the convergence time.

- I have developed Root Control Bridging for the control of SPB networks. RCB prevents loops and reduces the computational complexity compared to alternative solutions. RCB can be implemented as an extension to a standard link state protocol.

- I have shown by means of simulations over various topologies and parameter settings that RCB may provide faster network convergence than alternative link state solutions in mesh topologies.

SPB is able to natively enable and carry IP and IPVPN services for both unicast and multicast traffic on native Ethernet links as described by Unbehagen [82], thus it is well applicable in radio access networks as shown in Farkas [J2]. Furthermore, SPB is able to take advantage of the dense physical topology of data centre networks.

The results of Chapter 3 were published in [J2, C1, C2, S1, S6, S7]. The Neighbour Synchronisation algorithm [S6] was proposed to IEEE 802.1 for loop prevention in SPB. It has then become the loop prevention principle that should be applied at least for multicast traffic as described by Allan [44]. The basic idea has been finally incorporated in the standard loop prevention solution referred to as agreement protocol (see e.g. Clause 13.17 of [6]). Furthermore, the Neighbour Synchronisation can be also applied as a loop prevention method in the IETF IP Fast Re-Route (IPFRR) framework. The Neighbour Synchronisation was also submitted as patent application [P7]. Patent applications [P9] and [P10] describe the operation of Root Controlled Bridging in detail; [P5] and [P8] are also related to RCB.
5.3 Connection Admission Control for UTRAN

In Chapter 4, I have defined connection admission algorithms for the Iub interface of UTRAN using IP transport. Connection Admission Control is used to ensure that the stringent QoS requirements of admitted connections are met. On the other hand, the CAC should allow the utilisation of the available network resources.

- I have defined a Connection Admission Control algorithm for IP UTRAN networks implementing Weighed Fair Queuing. The WFQ scheduler is approximated by multiple strict priority schedulers by the CAC algorithm in order to allow the utilisation of reserved but unused bandwidth. I have shown by means of simulations that the proposed CAC algorithm improves bandwidth utilisation of small capacity links compared to the former algorithm.

- I have given a Connection Admission Control algorithm that guarantees QoS at the flow level such that Gaussian approximation is applied for the bufferless multiplexing model and the packet loss violation probability is approximated as the quantile of a normal distribution.

The results of Chapter 4 were published in [C9, C10]. The CAC algorithm specified in Section 4.4 was submitted in patent application [P18] and it can be applied in IP UTRAN networks. A dimensioning algorithm, e.g. the one specified in [P13], may also take into account the CAC algorithm implemented in transport network nodes.
Bibliography


98


[45] OMNeT++
   http://www.omnetpp.org


[47] Survivable fixed telecommunication Network Design library (SNDlib),
   http://sndlib.zib.de


   http://www.cs.bu.edu/brite
[50] GNU Quagga routing software
http://www.quagga.net


[56] 3GPP, ”QoS concept and architecture,” TS 23.107, V.9.1.0, July 2010.


Publications

Journal Papers


[J5] B. P. Gerő, J. Farkas, S. Kini, P. Saltsidis and A. Takács, ”Upgrading the metro Ethernet network,” *submitted for review to IEEE Communications Magazine*

Conference Papers


**Patent Applications**


Standardisation Contributions


Appendix A

SPB Examples

This appendix provides further details on the operation of SPB and the algorithms proposed for SPB in Chapter 3.

A.1 SPB Forwarding

As a consequence of applying multiple source routed trees, the multicast forwarding principle has to be changed and all nodes have to be aware of the SPT of all other nodes in SPB. The traditional multicast forwarding in Ethernet is only based on group membership, which can be described as \((*,G)\) for multicast group \(G\). Figure A.1 illustrates the need for the differentiation of the source in SBP.

![Figure A.1: Source specific multicast forwarding](image)

The figure shows two SPTs, which are rooted by different sources and directed towards the other nodes. Note that the rest of the SPTs are not illustrated for the simplicity of the figure. The red dotted \(SPT_A\) is rooted by node A and the blue dashed \(SPT_B\) is rooted by node B. Having a multicast group \(G\) comprising each node, node C has to forward the frames taking into account the source of the frame. If the source is node A, then C must not forward the frame to D. Therefore, source specific multicast forwarding is needed, i.e. \((S,G)\) is applied instead of \((*,G)\),
where S is the source and G is the group. Furthermore, for the implementation of the proper forwarding C needs to know whether or not it is on the shortest path between A and D. In general, each node has to be aware of the SPTs of all other nodes, which may have implications on the computation complexity of an SPB implementation.

Standard link state protocols apply periodic Hello messages to discover the neighbours of a node and to monitor the connectivity. Each node then advertises its information on its neighbours by flooding Link State Protocol Data Units (LSP). After a while, each node receives the same set of LSPs thus able to construct the very same Link State Database (LSDB). Each node is then able to compute the forwarding paths and set its forwarding tables accordingly. The forwarding path computation algorithm used in SPB has to provide congruency, i.e. the forward and the backward paths have to be the same between any source and destination pairs. Congruency is provided if the applied shortest path computation algorithm implements a tie-breaking providing a unique result for multiple equal cost paths.

\section*{A.2 Neighbour Synchronisation in a Generic Network}

The operation of the Neighbour Synchronisation mechanism is demonstrated in the following in an example network depicted in Figure A.2. Note that the physical topology is only presented in the figure, there can be various forwarding topologies on top this physical topology.

Let us assume that a change in the topology happened. It can be any kind of change, it does not matter for the algorithm, therefore, the change itself is not shown in the figure. The nodes are notified about the change in an unspecified order, which can be modelled as a random order. The order according to which the nodes become aware of the change does not matter for the operation of the Neighbour Synchronisation algorithm. An example order is selected and demonstrated in the following.

Let us assume that nodes B and F become aware of the change as illustrated in the first stage in Figure A.2. In the second stage, nodes B and F block their ports as soon as they realise the topology change and send out the Synchronisation Request because they are not in sync with any of their neighbour. After that, nodes A and H are also notified on the change as shown in Figure A.2-3). They send back the Synchronisation Acknowledge to the neighbours on the port they received the Synchronisation Request. They block all the rest of their ports and send out the Synchronisation Request. As soon as the acknowledgements arrive, A-B and F-H links become unblocked as depicted in Figure A.2-4). Nodes C, D, E, I and J become aware of the change in the next stage so they acknowledge former
requests, block all the rest of their ports and send out the Synchronisation Request on the ports where they had not received Synchronisation Request before. Note that nodes I and J send the request at the same time, which holds for nodes C and E too. The algorithm shown in Figure 3.3(b) is applied, i.e. if both neighbours send a request on the same topology digest on the same link, then they activate the link after the reception of the request from the peering node. Figure A.2-5) shows the stage when the previous messages are processed. Figure A.2-6) shows the final stage where all nodes are updated and the last one has sent back the acknowledgement thus there is no blocked link. As Figure A.2 shows, there are independent cuts in the forwarding topology during convergence, which are not synchronised to each other but only depend on the order the nodes that are notified on a topology change. Furthermore, it is very important that the network is not shut down during convergence.

A.3 Neighbour Synchronisation Prevents Farkas Loop

Farkas loop, which is shown in Figure 3.1, is also eliminated by the Neighbour Synchronisation algorithm. Figure A.3 shows the operation of the loop prevention
in this specific case. Nodes A, B and E become aware of the change in the topology as they are directly connected to the links changed. Thus, A, B and E issue Synchronisation Request messages. After exchanging the Synchronisation Request messages containing the new topology digest, B and E activate the link between each other. Nodes C and D have outdated view on the topology. A-D, B-C and E-D links are blocked at both ports they are connected to because the remote topology digest received in the Synchronisation Request messages differs from the locally stored one. The C-D link is active as nodes C and D have matching topology digest, but it is isolated from the rest of the nodes. Let us assume that D is then notified on the topology change before C. D then blocks its port towards C and issues a Synchronisation Request on this port. In addition, D replies a Synchronisation Acknowledgement to A and E thus A-D and E-D links become active. C is the last node becoming aware of the new topology, it issues a Synchronisation Acknowledgement to B and D when being updated on the topology changes. The B-C link is then activated as both B and C activates their corresponding ports. Nonetheless the C-D link remains blocked as it is not part of the final forwarding topology.

Figure A.3: Farkas loop is eliminated by Neighbour Synchronisation

As Figure A.3 shows the loop illustrated in Figure 3.1 cannot appear because the link between neighbours having different topology view is always cut.
A.4 Root Controlled Bridging Prevents Farkas Loop

Figure A.4 illustrates the step by step operation of Root Controlled Bridging for the Farkas loop topology change example. A recomputes its tree after the topology change and advertises its tree to the other bridges. As the figure shows, the Tree Advertisement is forwarded along the tree under update and the discarding ports are set before the forwarding ports.

As the figure illustrates, there is no loop at any time during the topology transient.

A.5 Simulation Parameter Settings for Control Protocol Evaluations

The investigation of simulator parameters for the performance evaluation presented in Section 3.5 is described in the following.

MSTP Parameter Settings

The convergence time of MSTP may be influenced by the upper limit set for the BPDU transmission rate on a port. The Transmit Hold Count (TxHoldCount) pa-
rameter limits the number of BPDUs that can be transmitted in a second, which can be set in the range of 1-10 and its default value is 6. That is, TxHoldCount may increase the convergence time if more BPDUs need to be exchanged for the recovery than the limit. In order to show the performance of the protocol itself without this limitation, TxHoldCount was set to 100 in the simulations.

**Dijkstra Computation Time**

The computation of the SPTs plays a key role in the convergence time of IS-IS based SPB protocols hence the corresponding parameter settings were analysed in detail. The computational time of a binary heap implementation of the Dijkstra algorithm, which is common in commercial routers according to Francois [46], is

\[ DijkstraDelay = C \cdot (L + N \cdot \log N), \quad (A.1) \]

where \( N \) and \( L \) denote the number of nodes and links in the network, respectively. Note that the computational complexity of the RCB approach is the same as that of the Dijkstra algorithm. As opposed to this a multiplier of \( N \) has to be applied on Dijkstra in case of Basic IS-IS because each bridge computes the trees of all bridges. The parameter \( C \) of Equation A.1 has to be carefully chosen in order to scale the Dijkstra computation realistically in the simulator tool. According to the measurement results published by Francois [46] \( C = 1.2 \cdot 10^{-6}s \). However, \( C = 2.2 \cdot 10^{-8}s \) were measure on a 2.2 GHz Intel Core2 Duo processor. Furthermore, network convergence time simulation results for Basic IS-IS after a single link failure were compared to measurement results of Chiabaut [55] when \( C = 2.2 \cdot 10^{-8}s \) and \( linkDelay = 0.1ms \) parameter settings were applied in the simulator. As the results summarised in Table A.1 show, a value in the order of \( 10^{-8} \) seconds is more appropriate today. Therefore, \( C = 2.2 \cdot 10^{-8}s \) was used in the simulations described in Section 3.5.

<table>
<thead>
<tr>
<th>N</th>
<th>Heavy mesh Measured</th>
<th>Light mesh Measured</th>
<th>Rings Measured</th>
<th>Heavy mesh Simulated</th>
<th>Light mesh Simulated</th>
<th>Rings Simulated</th>
</tr>
</thead>
<tbody>
<tr>
<td>50</td>
<td>0.0007</td>
<td>0.0008</td>
<td>0.0004</td>
<td>0.001285</td>
<td>0.001284</td>
<td>0.0004</td>
</tr>
<tr>
<td>100</td>
<td>0.0013</td>
<td>0.003478</td>
<td>0.0041</td>
<td>0.00412</td>
<td>0.00292</td>
<td>0.0016</td>
</tr>
<tr>
<td>200</td>
<td>0.0136</td>
<td>0.014727</td>
<td>0.0245</td>
<td>0.016581</td>
<td>0.009375</td>
<td>0.0068</td>
</tr>
</tbody>
</table>
Appendix B

UTRAN Examples

This appendix explains examples for the algorithms described in Chapter 4.

B.1 An Example for Random Activity Factor

An illustrative example is shown in the following on the effect of random activity factors. Let us consider an example system multiplexing $N = 50$ connections and having capacity $Z = 40$. Let \( \varepsilon \) be 0.1%. The activity factors of the connections are independent and identically distributed Bernoulli variables: \( X_{\alpha_i} \). Let the \( \alpha_i \) parameters have the following distribution:

\[
P[\alpha_i = 0.4] = P[\alpha_i = 0.6] = 0.5
\]

Applying Equation (4.27) and Equation (4.28) for the case when the activity factor of all connections equals to 0.4 we get

\[
\text{Packet drop probability} = 1 - \mathbb{P} \left[ \sum_{i=1}^{50} X_{0.4} \leq 40 \right] \approx 1 - \Phi \left( \frac{40 - 50 \times 0.4}{\sqrt{50 \times 0.4 \times 0.6}} \right) \approx 10^{-4}.
\]

If 29 connections have an activity factor of 0.6, then flows still fulfil the drop requirement as \( \mathbb{P} \left[ (\sum_{i=1}^{29} X_{0.6} + \sum_{i=30}^{50} X_{0.4}) \geq 40 \right] \approx 0.10\% \). However, if 30 connections have the larger activity factor, then flows violate the drop requirement as \( \mathbb{P} \left[ (\sum_{i=1}^{30} X_{0.6} + \sum_{i=31}^{50} X_{0.4}) \geq 40 \right] \approx 0.12\% \). If even more connections have an activity factor of 0.6 then the drop requirement cannot be fulfilled. For example, if all connections have the larger activity factor (0.6), then \( \mathbb{P} \left[ \sum_{i=1}^{50} X_{0.6} \geq 40 \right] \approx 2.96\% \).
Calculating Equation (4.29) we get that

\[ Q(\alpha_1, \ldots, \alpha_{50}) = 1, \text{ if } \#\{\alpha_i = 0.6\} \geq 30. \]

That is, the packet loss requirement is not met if more than 29 sources have an activity factor of 0.6 (and the rest have activity factor of 0.4).

Now \( \delta \) can be calculated based on Equation (4.30) and Equation (4.24). The probability of each particular realisation in case of our example is \( \frac{1}{2^{50}} \), e.g.

\[ P[\alpha_1 = 0.4, \ldots, \alpha_{50} = 0.4] = \frac{1}{2^{50}}. \] Thus

\[ \delta = \sum_{\alpha_1,\ldots,\alpha_{50}} P[\alpha_1 = a_1, \alpha_2 = a_2, \ldots, \alpha_{50} = a_{50}] \times Q(a_1, a_2, \ldots, a_{50}) = \frac{1}{2^{50}} \binom{50}{30} = 4.2\%. \]

Thus, the probability of the violation of the 0.1% drop requirement is 4.2%.

**B.2 Activity Factor Distribution for GSM Speech**

Figure B.1 shows the activity factor distribution for GSM speech published by Westholm [68].

![Activity Factor Distribution for GSM Speech](image)

Figure B.1: Density function of activity factors of GSM speech based on measurements