Verifying and optimising disjoint paths in ISP networks

Post, B.

Award date:
2014

Technische Universiteit Eindhoven University of Technology
Department of Mathematics and Computer Science

Verifying and optimising disjoint paths in ISP networks

Master Thesis

Bart Post BSc.

Supervisors:
prof. dr. Nikhil Bansal (TU/e)
dr. Piotr Zuraniewski (TNO)
ir. Bart Gijsen (TNO)

In cooperation with:

TNO innovation for life

Performance of Networks and Systems

Eindhoven, 10 October 2014

Abstract

Consider a company with two remote office locations that uses an Internet connection from an Internet Service Provider to connect the two offices. In addition, that same company purchases a second Internet connection hoping that when their primary connection fails, the secondary connection can take over. In order to assure that this is the case it needs to be verified that the pair of connections does not have a Single Point of Failure, causing both connections to fail simultaneously.

The challenge is to detect when two connections share common resources. We propose to check this by means of topology discovery. A lot of work has been done on Network layer (ISO layer 3) topology discovery, but the Network layer is a virtual layer that does not have a one-to-one relation with the physical network resources. Therefore we present a method of topology discovery on the Data Link layer (OSI layer 2) using Ethernet OAM technology, that has a correlation to the physical network. Furthermore, We generalize topology discovery and present two strategies which recover the topology of several simple network types. We present upper and lower bounds in terms of number of probes needed by these strategies applied to the simple network types and we also show average performance.

In addition to testing if connections are disjoint we investigate how to create a pair of disjoint connections given a discovered network topology. We transform problems of selecting a pair of disjoint connections with optimal reliability to well studied problems of selecting a pair of disjoint paths in a network with Min-Sum or Min-Max objectives. Finding a pair of disjoint paths with a Min-Max objective is a NP-hard problem and a Min-Sum solution can be used as approximation to a Min-Max solution. We present an algorithm which performs a smart search using the Min-Sum solution to find an improved Min-Max solution.

Acknowledgements

I want to take the opportunity to thank all people who have made this thesis possible. First of all I wish to thank Nikhil Bansal for being my supervisor at the TU/e and giving me guidance when I needed it. I also wish to thank my supervisors at TNO: Bart Gijsen and Piotr Zuraniewski for their wonderful advices, feedback and (sometimes too long) discussions. Also, I wish to thank Piotr for the time he spent on helping me with configuring the switches in the TNO lab, which took only three times as long as we initially expected. I thank all three of my supervisors along with Jacques Resing and Cor Hurkens for being on my graduation committee.

I feel that the PoNS department at TNO deserves a special mention. You people have welcomed me into the department, treating me as your fellow researcher. I thank you for all your time you spent listening to me and helping me. In particular I wish to thank Dick van Smirren for his personal attention to not only me but all the people of his department. Thank you for the educative Bila's and the many helpful, hilarious and sometimes awkward moments at TNO. I also wish to thank Rick for the times we enthusiastically spent on many problems and for all our wrestling sessions with LaTeX.

Thank you Fiona and Jasmijn for taking their time to read through my thesis and providing feedback. I also wish to thank Fiona, my parents, roommates and JC Capi for their mental support and not growing tired of my babbling about maths.

Contents

Lists

List of Figures

List of Tables

List of Algorithms

Programming Codes

Chapter 1

Introduction

The title of this thesis is *Verifying and optimizing disjoint paths in ISP networks*. As such we will dive into two different problems: verifying that two connections are disjoint by means of topology discovery and choosing an optimal pair of disjoint connections.

1.1 Motivation

Many large companies today heavily depend on data connections which they buy from Internet Service Providers (ISP's). Most of the time these data connections form a link between two remote points and may be used to create a connection between two geographically separated buildings. For example, two remote office buildings can be linked to the same Virtual Local Area Network (VLAN) via such a connection. Another possibility is that a local branch office may connect to the central database which is situated in another city. If such a connection fails, the local branch office cannot serve their costumers anymore which results in a loss of profit.

To protect against connection failure a bank buys a backup connection such that in case the primary connection fails, the backup connection takes over. However, the ISP's do not guarantee for 100% that the backup connection is operational when the primary connection fails. It is possible that whatever causes the primary connection to fail also causes the backup connection to fail. An example is when the primary and backup connection share some resources, thus creating a Single Point of Failure: if such a resource fails then both connections fail at the same time.

Our objective is to check if the primary and the backup connection "meet" somewhere other than at the two points they connect. We propose to do this via topology discovery: if we know the topology of the network and also how the connections run through the network, then we can see if the connections share resources. A lot of research and work has been done to recover the Network layer (ISO layer 3) topology of connections via the use of the Internet Protocol, e.g. the RocketFuel project [Spr04]. However, since the Network layer topology consists of routers and servers it does not provide a good view on the physical topology of the network. For instance; several virtual servers with different IP addresses may run on the same physical device. For a visualisation of different abstraction layers in computer networks, see Figure 1.1, which is taken from [Wik14].

Two connections that seem disjoint on the Network layer can physically still share resources. Switches

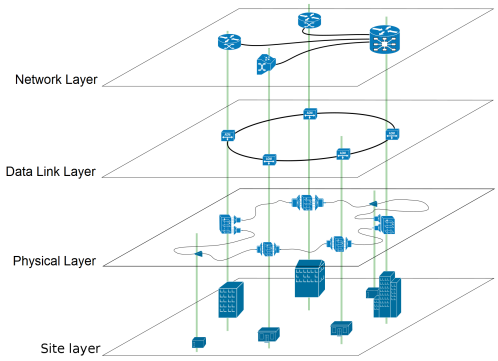


Figure 1.1: Different abstraction layers in a computer network.

and bridges, which are normally ISO layer 2 devices (Data Link layer), are not recognised by the Network layer. As a consequence, most Network layer topology discovery algorithms which are based on IP traceroute will not reveal switches and thus create an incomplete picture of the network topology.

In this thesis we will descend to the Data Link layer (ISO layer 2) and present an algorithm that uses layer 2 protocols to recover the layer 2 topology of the network. We will show how this algorithm performs and what its limitations are. We will extend the problem of topology discovery into a more general mathematical setting and analyse some classes of networks that are commonly used by ISP's.

In addition, we will have a look at the selection process of the two disjoint connections. Given the network topology (for instance found by using a topology discovery algorithm) we might choose our connections to optimize robustness. For that purpose we consider different types of robustness and figure out how "hard" it is to select the two best connections in terms of computational complexity.

1.2 Problem description

We will change the problem of testing disjointness of connections to a problem of topology discovery and address this problem at the Data Link layer. The idea is that once we know the topology of the network (on the Data Link layer) and we also know how the connections run through this topology, then we can easily verify if the connections share common resources (on the Data Link layer). Alternatively, we can view the connection as a network and recover the topology of the connection itself. In both cases the topology includes an identifier, a MAC address, for switch ports. As a result we can check if two connections contain the same identifier, which means they are not disjoint.

One of the problems we will be looking at is discovering the network topology at the Data Link layer using protocols of Connectivity Fault Management (CFM) (see Section 2.4). Can we discover the full network topology and under what circumstances?

The second problem that we will consider is the performance of (a generalised version of) topology discovery. Assuming that it is possible to find the topology using a probing operation, we wish to find the topology with the least number of probing operations. We want to find upper and lower bounds on the number of probes and also the expected number of probes needed to find the topology.

The third problem we consider concerns the selection process of the disjoint connections where we assume to know the network topology together with values on the links representing failure probabilities of that link. How can we select the best pair of disjoint connections given the network topology and failure probabilities? When is a pair of disjoint connections considered the best pair? What is the complexity of finding these optimal pairs?

1.3 Outline of this thesis

First we will have a quick glance at Ethernet, followed by a description of the Connectivity Fault Management (CFM) protocols in Chapter 2. Connectivity Fault Management contains a very useful protocol, known as the Linktrace protocol, comparable to the IP traceroute function. Chapter 2 will give a general description of Connectivity Fault Management and will also provide a technical description of the Linktrace protocol. In addition, we will cover another CFM protocol: the Continuity Check Message (CCM) protocol. The CCM protocol allows us to learn how many network elements are on the boundary of the measured network. These elements can be used to initiate a Linktrace or be used as a target for a Linktrace.

A theoretic description of a protocol is necessary but for a full understanding we have experimented and tested the behaviour of the protocol. In Chapter 3 we provide a set of different situations and the results of initiating both the Linktrace protocol and the Continuity Check Messages.

Once we obtain a good understanding of how the protocols work and what information we can extract by applying them, we will make the step to topology discovery. In Chapter 4 we will discuss the advantages and disadvantages of using protocols on the Ethernet layer. Moreover, we propose our algorithm which, under some constraints, can discover the Ethernet topology of the active Virtual Local Area Network (VLAN).

Continuing on the grounds of topology discovery, we will ascend to a more abstract version of topology discovery in Chapter 5 in which all points in the network can be used as probing points. For a tree and star network we will present our analysis of how many probing operations are needed in order to recover the full network topology.

In Chapter 6 we will focus on choosing disjoint connections that maximise robustness in the discovered network. For that purpose we introduce two important and well-studied 2-Disjoint Path problems and formulate a new observation which links the solutions of these two problems. We use this observation to come up with a heuristic which performs a smart search for a better solution than the solution of an already existing approximation algorithm. We will also show that there is no guarantee that the heuristic actually finds an improvement.

Finally, in Chapter 7 we will summarize our conclusions and shortly propose some directions for further research concerning the topology discovery using CFM.

Chapter 2

Ethernet Operations Administration and Maintenance

This chapter will give a short introduction on Ethernet, Carrier Ethernet and a special set of functionalities known as Operations, Administration and Maintenance (OAM). In order to provide these functionalities in an Ethernet setting various standardization organisations have developed a set of protocols known as Connectivity Fault Management (CFM). The main goal of this chapter is to clarify and understand the protocols of Connectivity Fault Management as they are described in the standards.

2.1 A short introduction to Ethernet

Ethernet is a technology designed for Local Area Networks (LANs) that works on the Physical and Data Link layer of the OSI model. Ethernet wraps data in variable sized frames and transports these frames over the network. Each frame has an Ethernet header containing a source address, a destination address and error checking data that allows damaged data to be detected.

In an Ethernet network every station (which can be a computer, printer, router, et cetera) listens to the network and will only start transmitting frames when no other station is using the network at the moment. When two or more stations start transmitting at the same time, their transmissions will result in a collision. In case of such a collision, the stations will stop transmitting and wait for a random period of time before retrying to transmit their frames. In reality there are no collisions anymore due to micro-segmentation and full-duplex. Micro-segmentation is segmenting the network in transmission domains by placing switches in the network. Full duplex allows for two directional traffic, e.g. Ethernet cables use separate pairs of cables to allow data transmission in both directions.

Similar to the IP address in the IP layer, Ethernet uses MAC addresses. Contrary to IP addresses, MAC addresses are configured at the factory and remain mostly unchanged where IP addresses may change every day, hour or even minute.

2.2 Carrier Ethernet and Ethernet OAM

Due to its simplicity, easy implementation and success, Ethernet became a popular candidate for Wide Area Network (WAN) technology. Telecommunications network providers started providing Ethernet services to customers and utilize Ethernet technology in their networks resulting in Carrier Ethernet. However, Ethernet was not designed as a carrier-class technology and as a consequence was missing OAM functionalities. OAM, shorthand for Operations, Administration and Maintenance, is a term referring to management functionalities such as the ones listed in Table 2.1. In particular, OAM for Ethernet (Ethernet OAM or E-OAM) refers to standards and their protocols involved with operating, administering, managing and maintaining an Ethernet network. The Institute of Electrical and Electronics Engineers (IEEE) similarly describes OAM in one of their standards, IEEE 802.1Q [IEE11, p. 19], as follows:

> ***Operations, Administration, and Maintenance (OAM):*** *A group of network management functions that provide network fault indication, performance information, and data and diagnosis functions. (adapted from ATM Forum Glossary).*

Observe at this point that Ethernet OAM is not a standard itself, but merely a term for anything that provides OAM functionalities in an Ethernet network, either carrier-class or local. Table 2.1 shows a

description of the three OAM categories and what functionalities they contain.

Category	Functionalities
Operations	• Automatically and pro-actively monitors environment. • Quickly detects and isolates faults and recovers from them. • Alerts administrators.
Administration	• Monitors performance. • Facilitates capacity planning.
Maintenance	• Performs upgrades. • Deploys new features. • Monitors network health.

Table 2.1: Ethernet OAM functionalities by category, as described in [JN10].

Several standardization bodies like IEEE, International Telecommunication Union (ITU) and Metro Ethernet Forum (MEF) have developed standards which add carrier grade OAM functionalities to Ethernet, enabling the use of Ethernet in carrier networks[1]. Although these standards for Ethernet OAM originated from requirements of the telecommunication network providers, they can be used for standard switched Ethernet networks as well. These standards imply injecting extra frames in the network, such that devices can communicate and react to each other by means of these special OAM frames.

2.3 E-OAM layers and standards

In the setting of a carrier network, three different parties arise naturally: the operator, the service provider and the customer. The operator is the party owning and managing network equipment like switches, routers, servers and the cables/wires connecting all the equipment. The service provider is the party providing services towards customers over the networks of the operators and the customer is the party that pays for a service provided by a service provider. An example of such a service is a data connection with a guaranteed bandwidth between two geographically separated office buildings. This allows the two offices to operate on the same VLAN. An operator can also directly provide a service to a customer, such that the operator is also the service provider. This means that an organization can fulfill more than one role.

Continuing with the idea of three different roles, the level on which Ethernet OAM functionalities work can be split in three categories. Each of these categories roughly corresponds to one of the roles; operator, service provider or customer.

Transport Layer: The transport layer is responsible for detecting "link down" failures and notifying higher layers. It operates between two directly connected peers (port to port) and ensures that they maintain bidirectional communication. This layer also monitors the link quality to ensure that the performance meets an acceptable level. The transport layer roughly corresponds to the operator party.

Connectivity Layer: This layer is also known as the **Network Layer**. It monitors the path between two non-adjacent devices and roughly corresponds to the service provider party.

[1] Note that E-OAM is not sufficient to have carrier grade Ethernet. E.g. one needs a Quality of Service before Ethernet is of carrier grade.

Service Layer: This layer measures and represents the status of the services as seen by the customer. It produces metrics that need to be monitored in order to confirm that the Service Level Agreements, contracted between the service provider and a customer, are satisfied. The service layer roughly corresponds to the customer party.

Section 2.2 already stated that three standardization bodies have developed several standards for Ethernet OAM. We will provide a description of a selection (inspired by technical reports and presentation [JN10; PM11; Das09]) of a few standards relevant to our cause.

MEF 16 is a standard developed by the Metro Ethernet Forum. The common name for MEF 16 is E-LMI: Ethernet Local Management Interface. MEF 16 provides protocols and mechanisms used for notification of adding, deleting or status changes of Ethernet Virtual Connections (EVC). Such an Ethernet Virtual Connection can be either active, not active or partially active. The abstract of MEF 16 [MEF06, p. 1] provides the following description:

> *The E-LMI procedures and protocol are used for enabling auto configuration of the customer edge to support Metro Ethernet services. The E-LMI protocol also provides a user-network interface and Ethernet Virtual Connection status information to the customer edge. The user-network interface and Ethernet Virtual Connection information enables automatic configuration of customer edge operation based upon the Metro Ethernet Network configuration.*

The abstract mainly describes automatic configuration of customer edge devices, devices that are on the boundary between the customer and the service provider. Anything on the customers side of the customer edge is maintained and operated by the customer himself.

IEEE 802.3ah [IEE12, Clause 57] is a standard developed by the Institute of Electrical and Electronics Engineers and is also known as Link Layer OAM or Ethernet in the First Mile. This standard provides mechanisms useful for monitoring link operation. More specific, this standard only focuses on the data link layer and is therefore only applicable to a single link at the time between two devices. It does not monitor multiple links at the same time. That means that this standards resides in the Ethernet OAM Transport layer.

The standard IEEE 802.1Q is also a standard developed by the Institute of Electrical and Electronics Engineers and is more commonly known as Connectivity Fault Management. This standard provides three protocols that can detect, verify and isolate connectivity faults: Continuity Check protocol, Loopback protocol and Linktrace protocol. The concepts, elements and protocols defined by this standard can be applied at all Ethernet OAM layers. Therefore this standard can also be used to monitor a single link. However, when this standard is restricted to only one link it is not the same as IEEE 802.3ah. For one, frame loss measurement is described in IEEE 802.3ah and not in IEEE 802.1Q.

The International Telecommunication Union (ITU) developed a standard, ITU-T Y.1731 [ITU11], which also describes Connectivity Fault Management, and even adds some more functionalities that are not covered by IEEE 802.1Q like performance management. For the remainder of this thesis, the part which is covered by both the standards ITU-T Y.1731 and IEEE 802.1Q is referred to as Connectivity Fault Management (mainly the three protocols mentioned above and the respective concepts on which these protocols are based). Connectivity Fault management is covered in more detail in Section 2.4. The other standards are not covered in detail since they will not be used in this thesis.

Figure 2.1 shows the four standards mentioned above and on what Ethernet OAM layers they work.

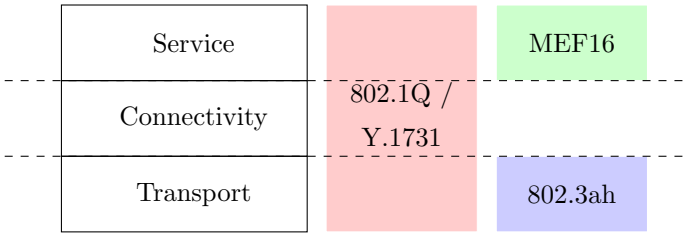


Figure 2.1: Four main standards operating on the E-OAM layers.

Although almost all literature considered in this thesis refer to the standard IEEE 802.1ag, the Institute of Electrical and Electronics Engineers (IEEE) lists 802.1ag as "superseded" on their website. The newer standard IEEE 802.1Q [IEE11] has replaced IEEE 802.1ag as of 2011 as active standard. The protocol descriptions in section 2.4 will follow the directions of the most recent and active standard IEEE 802.1Q.

2.4 Connectivity Fault Management (CFM)

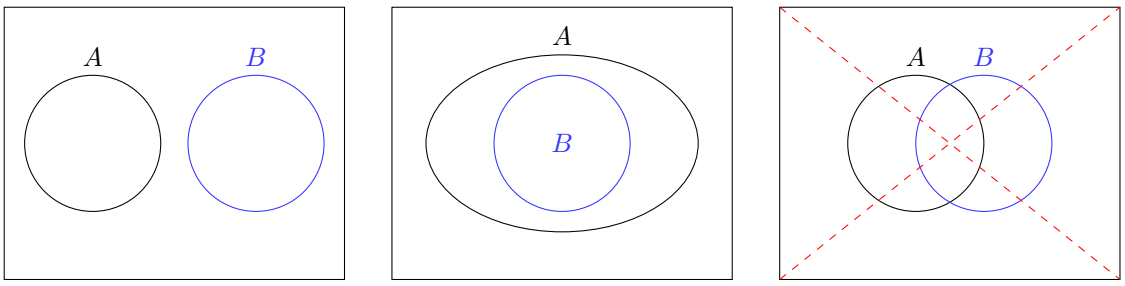
Connectivity Fault Management is described in IEEE standard 802.1Q, ITU-T Y.1731 and in MEF 30.1 and is sometimes referred to as Service Layer OAM. In contrast to what the name Service Layer OAM suggests, Connectivity Fault Management does not primarily operate at the service layer of Ethernet OAM. It can be applied at every layer, transport, connectivity and service.

Connectivity Fault Management introduces the concepts of Maintenance Domains, Maintenance End Points and Maintenance Intermediate Points. These concepts will be explained in Section 2.4.1. In addition, CFM also provides three protocols that can be used to detect and isolate connectivity faults. These protocols are described in Section 2.5.

2.4.1 Maintenance Domains, End Points and Intermediate Points

Both the standards IEEE 802.1Q and ITU-T Y.1731 define the concepts of Maintenance Domains, Maintenance End Points and Maintenance Intermediate Points, but the naming differs between the two. We will adapt the naming of IEEE 802.1Q and the corresponding naming of ITU-T Y.1731 is listed in Table 2.2.

Connectivity Fault Management divides the network in several hierarchical domains called Maintenance Domains. Different Maintenance Domains may be nested within each other but they may never overlap, see Figure 2.2. Each Maintenance Domain (MD) is the part of a network that is controlled by



(a) Allowed: disjoint MDs. (b) Allowed: nested MDs. (c) Not allowed: overlapping MDs.

Figure 2.2: Different configurations of two Maintenance Domains A and B.

a single administrator and in which the Connectivity Fault Management is enabled. Each Maintenance Domain is assigned a level number ranging from 0 to 7 (meaning there are at most 8 levels). This number is called the MD level and corresponds to the defined hierarchy. Although 8 levels have been provided, not all of them need to be used. For example in Figure 2.3 MD A may have level 7, B level 4 and C level 0. The higher a MD's level, the broader its reach. That is, a level 5 MD might contain several level 4 MDs and the level 5 MD spans at least as many devices as the level 4 MD (although not all devices need to respond to the level 5 MD). The Metro Ethernet Forum proposes an allocation of MD levels, described in [MEF12], such that:

- MDs operating on the E-OAM Service layer have MD level 5, 6 or 7,
- MDs belonging to service providers operating on the E-OAM Connectivity layer have MD level 3 or 4,

- MDs belonging to operators operating on the E-OAM Connectivity layer have MD level 0, 1 or 2,
- MDs operating on the E-OAM Transport layer have default MD level 0.

An operator defines CFM reference points that initiate and react to CFM messages. These reference points can be of two types:

MEP: Maintenance Domain End Point, a reference point that can initiate and react to CFM protocols. A MEP is a reference point on the boundary of the network. MEPs are configured on device ports. In a diagram they are displayed using the symbols ◀ and ▶. The triangle points towards the direction in which the MEP communicates.

MIP: Maintenance Domain Intermediate Point, a reference point that can only react to CFM protocols. It does not initiate CFM protocols and lives inside the network, not on the boundary. In this thesis a MIP will be represented by the symbol ● in diagrams.

When an operator has configured MEPs and MIPs inside a Maintenance Domain, IEEE 802.1Q states that the collection of these MIPs and MEPs forms a Maintenance Association (MA) which receives a unique identifier; a Maintenance Association Identifier (MAID). A MEP is uniquely bound to a Maintenance Association. It will only react to frames targeted at or originating from its own Maintenance Association. ITU-T Y.1731 does not differentiate between a Maintenance Domain and a Maintenance Association and uses the name MEG (Maintenance Entity Group) for a Maintenance Domain.

Reference points should only process protocol messages that belong to the associated Maintenance Domain and thus only the messages that belong to the same MD Level and contain the same MAID. A reference point should discard all messages that belong to a lower MD level and it should forward all messages belonging to a higher MD level.

This does not apply to MIPs. The standard IEEE 802.1Q provides an overview of frame routes through a MIP, provided in [IEE11, p. 860, Figure 19.3], in which it clearly shows that if the MD level is not equal to the MIPs MD level, the frame should be forwarded without changing it.

Discarding lower level CFM frames ensures that messages do not leave their MD. As a consequence, a MD is only aware of the MDs that are nested directly within itself. Consider Figure 2.3. Protocol messages belonging to MD C are discarded by any MEPs belonging to B. These protocol messages will never reach A and therefore A is not aware of the existence of C (if the Maintenance Domain is configured properly).

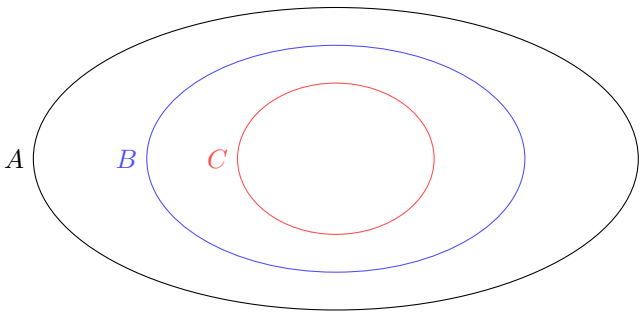


Figure 2.3: Three nested MDs: A, B and C. A is only aware of B and cannot see C.

MEPs come in two varieties: up and down. An *up* or *inward-facing* MEP is pointed away from the wire connected to the port on which it is configured, and is pointed towards the switch relay. A *down* or *outward-facing* MEP is pointed towards the wire connected to the port it is configured on and away from the switch relay. Typically, up MEPs are only used in switches while down MEPs can be used in both switches and routers. The difference between an up and a down MEP is visualized in Figure 2.4.

Since the MEPs and MIPs are configured at ports, one should take in account what happens when a spanning tree protocol (STP) is in operation. The (Rapid) Spanning Tree Protocol [IEE04, Clause 17] ensures that a switched network has a loop-free communication flow. For that purpose it constructs a

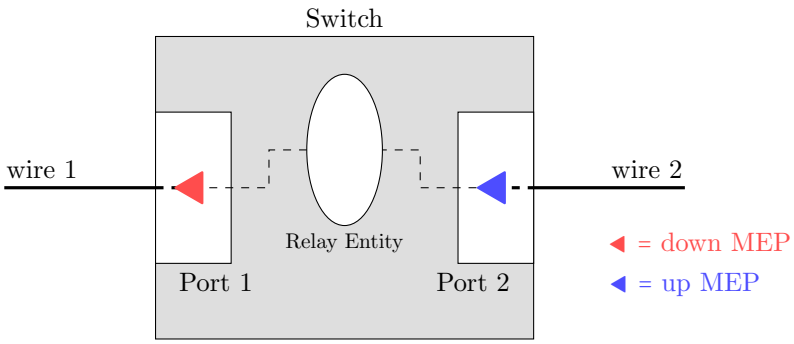


Figure 2.4: The difference between an up and a down MEP.

spanning tree by blocking certain links that are part of a loop. If those loops would be allowed, frames can circulate through these loops, being multiplied each time they arrive at a switch and thus causing so called "broadcast storms".

The spanning tree protocol can block a certain port on which a MEP or a MIP is configured such that the MEP or MIP might be disconnected from the domain it belongs to. This thesis assumes that MIPs and MEPs react to the Spanning Tree protocol as described in both Juniper ([JN14]) and Cisco ([Cis]) manuals for configuring Ethernet OAM. They state the following:

- If a down MEP is configured on a port which is blocked by the spanning tree protocol, then the MEP can still transmit and receive CFM packages using the wire to which the port is connected. Consider Figure 2.4 and suppose port 1 is in a STP blocking state. Then the red MEP at port 1 can still transmit and receive CFM packages using wire 1.
- If an up MEP is configured on a port which is blocked by the spanning tree protocol, then the MEP can no longer transmit or receive CFM messages. Consider Figure 2.4 and suppose port 2 is in a STP blocking state. Then the blue MEP at port 2 can no longer transmit or receive CFM messages.
- If a MIP is configured on a port which is blocked by the spanning tree protocol, then the MIP can still receive and respond to CFM messages on the wire connected to the port, but it can no longer receive or forward CFM messages to the relay side.

IEEE 802.1ag		ITU-T Y.1731	
ME	Maintenance Entity	**ME**	Maintenance Entity
MA	Maintenance Association	**MEG**	ME Group
MAID	MA Identifier	**MEGID**	MEG Identifier
MD	Maintenance Domain	**-**	
MD Level	MD level	**MEG Level**	MEG level
MEP	MA End Point	**MEP**	MEG End Point
MIP	MA Intermediate Point	**MIP**	MEG Intermediate Point

Table 2.2: Difference in naming between the two CFM standards (according to [Fin04]).

2.5 Connectivity Fault Management protocols

Both IEEE 802.1ag and ITU-T Y.1731 provide three protocols that can be used to detect and isolate connectivity faults. The three protocols are:

- **Continuity Check:** a protocol that allows detection of connectivity faults,

- **Loopback:** a layer 2 (ISO layer) ping protocol,
- **Linktrace:** a layer 2 (ISO layer) traceroute protocol.

The data packets or frames used in these protocols are called CFM packets. Each of these packets contains a source address belonging to the reference point that created the packet. The packets also contain a MD identification and a MD level. Moreover, each packet contains some protocol dependent information and some more information which is not of importance for the scope of this research.

For completeness of information we will cover all three protocols, although we will not use the Loopback protocol. The most important protocol that we use is the Linktrace protocol. We will also use the Continuity Check Message protocol but only to make an inventory of the MEPs in the network.

2.5.1 Continuity Check Message Protocol

The Continuity Check Message protocol is used to detect connectivity faults within a Maintenance Association (MA). In order to use the Continuity Check Message (CCM) protocol, each MEP must be configured with the following:

- A Maintenance Association Identifier (MAID),
- A Maintenance Domain level (MD level),
- A MEP Identifier (MEP ID),
- List of all MEPs in the MA (a list of MEP IDs).

Each MEP periodically sends a Continuity Check Message (CCM) into the network. This message is a multicast message and is received by all other reference points in the same MA. The transmission period t_{ccm} of CCMs, the period between two consecutive CCM messages of a MEP, can be configured to be one of 3.33ms, 10ms, 100ms, 1s, 10s, 1min or 10min. Once a Continuity Check Message has reached a MEP, its path ends. The MEPs do not send a reply to the CCMs.

MIPs do not generate Continuity Check messages and only react to them in the sense that they forward the CCMs without altering them. Therefore the MIPs are called transparent with respect to the Continuity Check protocol. This is consistent with the idea that MIPs do not initiate CFM protocols but only react to them.

The Continuity Check Protocol can detect the following connectivity faults (listed in IEEE 802.1Q, [IEE11]):

- Inability of a MEP to receive three consecutive CCMs from any one of the other MEPs in its MA, indicating either a MEP failure or a network failure;
- Reception by a MEP of a CCM with an incorrect transmission interval, indicating a configuration error;
- Reception by a MEP of a CCM with an incorrect MEPID or MAID, indicating a configuration error or a cross connect error;
- Reception by a MEP of a CCM with a MD Level lower than that of the MEP, indicating a configuration error or a cross connect error;
- Reception by a MEP of a CCM containing a Port Status TLV or Interface Status TLV indicating a failed Switch Port or aggregated port.

Figure 2.5 shows how the Continuity Check messages travel through a network. The Continuity Check messages are carried within an Ethernet frame that has a destination MAC address 01-80-C2-00-00-3y, where y corresponds to the MD level of the CCM.

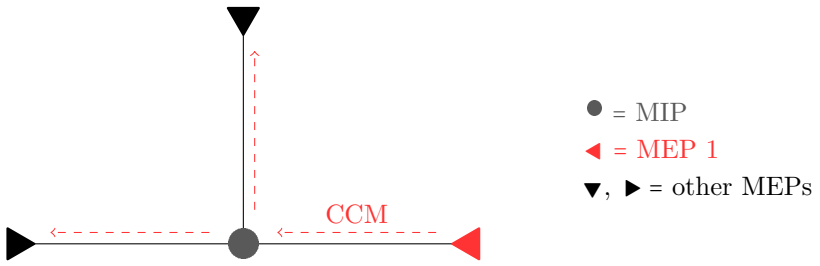


Figure 2.5: Continuity Check Message sent by the red MEP 1.

2.5.2 Loopback Protocol

The loopback protocol is an Ethernet ping protocol and is used to check the connectivity between a MEP and another reference point (MIP or MEP). The loopback protocol defines two types of messages: a loopback message (LBM) and a loopback reply (LBR). A loopback message contains a destination address and the destination point (target reference point) is the only reference point that replies to the loopback message. The source of a loopback message is always a MEP (MIPs can not initiate protocols). The target of a loopback message can be either a MIP or a MEP. When a loopback message has reached its destination, the target reference point will reply to the source MEP with a loopback reply. Figure 2.6 shows a MEP sending a loopback message to a MIP.

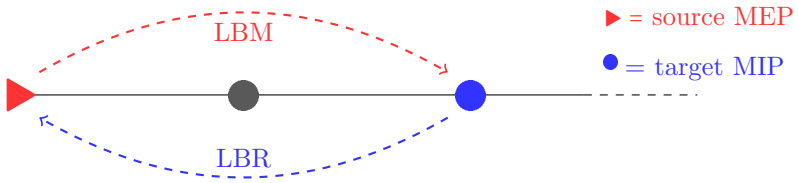


Figure 2.6: A loopback message (LBM) with the red MEP as source and the blue MIP as target. The MIP sends an answer in the form of a loopback reply (LBR) towards the source MEP.

A loopback message requires a destination address. MIPs do not broadcast their addresses by the Continuity Check Protocol and therefore MEPs need to learn the existence and addresses of MIPs by means of the Linktrace Protocol before they can send a Loopback Message towards a MIP.

2.5.3 Linktrace Protocol

The Linktrace protocol is used to recover the path between a MEP and another reference point in a Maintenance Domain and is initiated by a message called a Linktrace Message (LTM). This means that the target reference point can be either a MIP or a MEP. The Linktrace Message is carried in a Ethernet frame targeted at the multicast address 01-80-C2-00-00-3y, where y is 8 plus the corresponding MD level of the LTM. The Linktrace Message itself carries the destination address of the target MEP or MIP.

A network element with CFM capabilities has a Linktrace responder. This responder is responsible for dealing with Linktrace Messages. The Linktrace Message only reaches the Linktrace responder if it is deflected by a MIP or MEP at the right level. That means that if a Linktrace Message is sent at level 1, only level 1 MIPs and level 1 MEPs will deflect this Linktrace Message to the Linktrace responder. The Linktrace responder will reply to the LTM if one of the two following is true:

1. The network element where the MIP or MEP resides is aware of the target MAC address in the Linktrace request information and associates it to a single egress port, where the egress port is not the same as the port on which the frame with ETH-LT request information was received; or,

2. The target MAC address is the same as the MIP's or MEP's own MAC address.

A reply message consists of a Linktrace Reply inside an Ethernet frame. However, in this case the Ethernet frame will not contain a multicast address but the unicast MAC address of the source MEP.

In addition, if the LTM is received by a MIP which is not the target of the LTM, then the Linktrace responder also forwards the LTM and additionally decrements the LTM Time To Live (TTL) value by one. This value is given to the LTM message to identify the amount of hops taken to reach the target. This TTL value is then copied in the Linktrace reply message such the source MEP knows which hop belongs to which MIP or MEP and can thus sort the Linktrace replies by the traveling order. Figure 2.7 shows the message flow of a Linktrace protocol, issued by the red MEP.

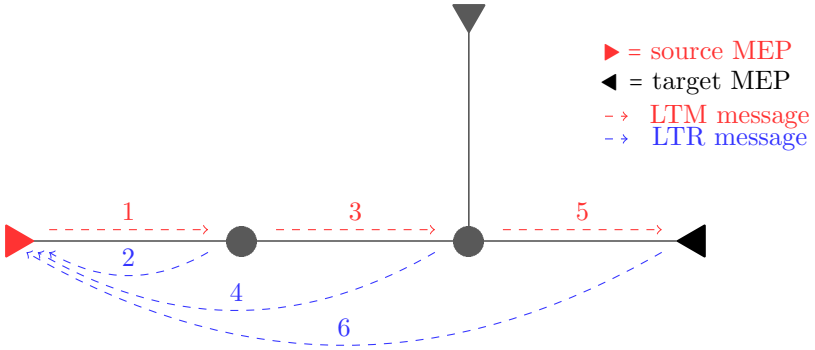


Figure 2.7: A Linktrace Protocol. The red dashed arrows represent LTMs send by the red MEP and the blue dashed arrows represent LTRs.

It is not always the case that every MIP and MEP sends a reply to the Linktrace Message, even though they are at the right level. How and when the Linktrace responder reacts to a Linktrace Message is described in [IEE11, Section 20.47.1], the section which describes the function *ProcessLTM()* of the Linktrace responder. Although this function seems to have quite a lot of distinctions between situations, the following abstract description of the rules suffices for the purpose of this research:

1. A Linktrace responder will only send one reply message, even though the path towards the target passes through two maintenance points on the switch on which the Linktrace responder resides.

2. A Linktrace responder will not respond if the path through the switch does not encounter any maintenance points on the same level as the source MEP of the Linktrace Message. In other words, if a LTM passes through two ports on a switch of which neither of them has a Maintenance Point configured at the right level, the LTM is never deflected to the Linktrace responder and the Linktrace responder will therefore never respond.

How these rules influence the Linktrace Replies will become clear in Chapter 3: CFM Protocols In Action.

Chapter 3

CFM Protocols In Action

Reading descriptions of standards is one way of getting familiar with them, yet it is as informative to experiment with the standards and their protocols and get some hands on experience. This chapter covers the setup and findings of these kind of experiments to further understand the protocols of Connectivity Fault Management. The focus of the tests is on the three protocols of CFM: to initiate them in different scenario's or setups and study the reply messages and the information learned and stored by the switches.

The Continuity Check Messages sent by MEPs indicate how many MEPs are configured in the maintenance domain. MEPs are always on the boundary of a maintenance domain. The Continuity Check protocol can therefore help in determining the amount of boundary points and that gives partial information on the topology of the network. The setups Alfa through Echo deal with extracting the CCM information stored by the MEPs and MIPs. These five configurations have only one Maintenance domain configured. The setups Foxtrot and Golf deal with multiple Maintenance Domains.

Recall that the Loopback protocol is meant to check the connectivity between a MEP and another maintenance point. It is not designed to be used for topology recovery. In the Alfa configuration the Loopback protocol is initiated once in order to confirm that indeed there is no additional information to be learned from the Loopback protocol.

The description of the Linktrace protocol by standard IEEE 802.1Q [IEE11, Section 20.3] requires that every MIP or MEP that is on the path towards the target maintenance point replies with a Linktrace Reply message. However, an earlier section of that very same standard [IEE11, Section 19.6] states that every switch entity has only one Linktrace Responder which serves all the maintenance points configured on that switch. This raises some questions on the reception of LTMs, their processing and the construction of LTRs. In particular we check the implementation of CFM of a given vendor in a given Operating System of the switches.

For the purpose of testing and observing network protocols, TNO has a special computer laboratory with many types of computer hardware. This includes four service delivery and aggregation switches; switches which are typically used by telecommunication providers. These four switches were used in experiments and tests for this research. In this thesis, the reference to the switches remains constant by the use of the notation S_i, $1 \leq i \leq 4$. In other words, S_i always refers to the same switch for every i, and in particular: S_1 is always the same switch. That explains why in some figures (e.g. Figure 3.1) the naming of the switches is not in the traditional clockwise (or anti-clockwise) direction, but consistent with the actual set-up at the time of the tests. Also we have obfuscated the MAC addresses of the switches in the output.

3.1 Alfa configuration

The Alfa test configuration consists of four switches: S_1, S_2, S_3 and S_4. They constitute a star topology with S_3 as center; S_3 is connected to all other switches while all switches S_1, S_2 and S_4 are only connected to S_3. The switches S_1, S_2 and S_4 have down MEPs configured on the ports that connect to S_3, and S_3 has MIPs configured on each port that is connected to one of the other switches. See Figure 3.1 for a graphical layout of the Alfa test configuration.

3.1.1 Remote MEP Information

In contrary to what IEEE 802.1Q and ITU-T Y.1731 demand, the vendor specific implementation of CFM does not require MEPs on switches S_1, S_2, S_3 and S_4 to be configured with all remote MEP IDs.

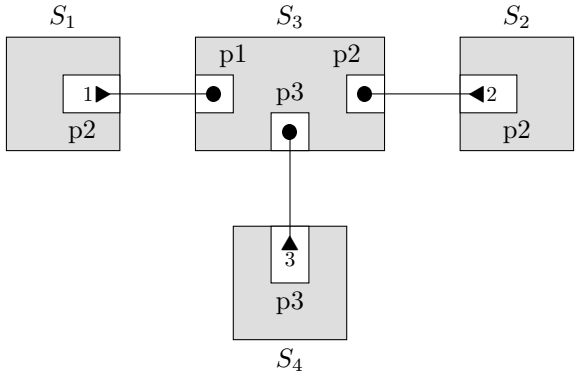


Figure 3.1: The switch configuration of the Alfa experiment. S_i is switch i, pj stands for port j and the number next to each MEP is the configured MEP ID.

The switches provide the functionality to automatically detect remote MEPs by listening to Continuity Check messages. In order to see what information the MEPs get from the Continuity Check messages, the MEPs were not configured with remote MEP IDs and the auto discovery was enabled. Requesting the remote MEP info from switch S_1 resulted in the output given in Table 3.1. For an explanation of the relevant parameters, see Table A.1 in Appendix A.1.

```
+------------------------------ CFM REMOTE MEPS -------------------------------+
|                |     |                 |State|Total    |Seq  |Last     |Fault|
|Service         |Mepid|Mac Address      |Ad|Op|Rx CCM   |Error|Seq Num  |F|P|R|
+----------------+-----+-----------------+--+--+---------+-----+---------+-+-+-+
|alfa            |2    |AA:BB:CC:DD:EE:03|en|en|1006     |0    |1006     | | | |
+----------------+-----+-----------------+--+--+---------+-----+---------+-+-+-+
|alfa            |3    |00:11:22:33:44:C4|en|en|997      |0    |997      | | | |
+----------------+-----+-----------------+--+--+---------+-----+---------+-+-+-+
```

Table 3.1: Output of remote MEP information request.

Clearly, the CCMs (Continuity Check Messages) contain the MEP ID and MAC address of the remote MEPs. Other than that, the CFM Remote MEPs overview does not provide any more information useful for topology recovery. Additional to viewing all remote MEPs it is also possible to focus on a single remote MEP. Remember that also this information is only recovered from the CCMs, and not configured beforehand. It appears that the only information relevant for the topology of the network learned by listening to the CCMs, is the MAC address and the MEP ID. For an overview of the information on a single MEP, see Appendix A.2, Table A.2.

MEPs are not the only maintenance points that can listen to the CCMs. MIPs can also listen to the CCMs, and switches S_1 to S_4 provide a configuration such that MIPs store CCM information. They create a list of MEPs in the Maintenance Domain which can be accessed and viewed. See Table 3.2 for the output of an MIP CCM database request.

Interestingly, the vendor specific implementation of CMF is such that the MIP CCM database contains information which is useful for recovering the topology of the network: the port number on which the queried switch finds the MEPs. The MIPs keep track of the port numbers on which the MEPs can be found. Consider Table 3.2: it shows that the MEP with MEP ID 1 can be found through port 1. Note that this information is not carried in the Continuity Check messages but it is determined by the switch or MIP itself. The Bravo 3 setup shows what the MIP CCM database looks like if not all ports have MIPs configured. In addition, the Charlie 2 setup shows what the MIP CCM database looks like if MIPs and an up MEP are configured on the same switch.

MIP CCM Database										
			Total	Last CCM Information						
VLAN	MAC Address	Port	CCM Rx	Seq Num	Time	Lv	Mepid	PS	RDI	
1	AA:00:BB:11:CC:93	1	859236	1128	0	2	1	Up		
1	AA:BB:CC:DD:EE:03	2	2724	1159	0	2	2	Up		
1	00:11:22:33:44:C4	3	2716	1121	0	2	3	Up		

Table 3.2: Output of local MIP CCM database request on switch S_3.

3.1.2 Initiating the Loopback Protocol

In this configuration, the Loopback protocol was initiated with source MEP 1 and target MEP 2. The resulting output is in Table 3.3. The output does not provide any information on the topology of the

MEP LOOPBACK MESSAGE INFORMATION										
	Local	Remote	Rem	Next	LBM		Rx LBR			Loss
Service	Mepid	Mac Address	Mepid	Seq Number	Tx	ToTx	Io	Ooo	Con	
alfa	1	AA:BB:CC:DD:EE:03	2	2	1	0	1	0	0	0

Table 3.3: Output of a Loopback protocol with source MEP 1 and target MEP 2.

network other than the remote MAC address which can also be found by querying the MEPs or MIP CCM databases. The Loopback protocol is therefore not considered in remainder of this thesis.

3.1.3 Initiating the Linktrace Protocol

In this configuration, the Linktrace protocol was initiated three times since there are $\binom{3}{2} = 3$ possible pairs of source and target MEPs in the Alfa setup. The results were symmetric in the MEP IDs and therefore only one Linktrace result is presented. The resulting output consists of two parts: the Linktrace Message in Table 3.4, and a list of Linktrace replies in Table 3.5. The relevant parameters are explained in Appendix A: Connectivity Fault Management information requests, Table A.3.

Linktrace Message								
				Target				
Service	Port	Vlan	Mep	Mac Address	Mepid	Trans Id	Ttl	FDB
alfa	2	1	1	AA:BB:CC:DD:EE:03	2	1	64	No

Table 3.4: Output of a Linktrace protocol with source MEP 1 and target MEP 2: Linktrace Message.

The presented Linktrace had source MEP 1 on switch s_1 and target MEP 2 on switch S_2. This can

Linktrace Responses									
	Ttl		Remote MP	Relay	Flags				
Ttl	Idx	Trans Id	Mac Address	Action	FY	TM	Ingress TLV	Egress TLV	
62	1	1	AA:BB:CC:DD:EE:03	Hit		X	MAC: AA:BB:CC:DD:EE:03 Port: 2 Action: Ok	MAC: Port: Action: undef	
63	1	1	A1:B2:C3:D4:E5:92	MFDB	X		MAC: A1:B2:C3:D4:E5:92 Port: 1 Action: Ok	MAC: A1:B2:C3:D4:E5:93 Port: 2 Action: Ok	

Table 3.5: Output of a Linktrace protocol with source MEP 1 and target MEP 2: Linktrace Replies.

also be seen in the Linktrace Message shown in Table 3.4. Moreover, it shows an initial TTL value of 64, which is the default value configured on switches S_1 to S_4.

First consider the amount of Linktrace replies. Based on the topology of this setup the expected amount of Linktrace replies would be two: one LTR from switch S_3 and one LTR from the target MEP. Remember from Section 2.5.3 that a Linktrace responder will only send one reply, even though two MIPs are configured on the same switch. Indeed, Table 3.5 shows two LTRs, and the path from MEP 1 to MEP 2 passes two MIPs which are on the same switch S_3.

Secondly, observe that the TTL values are indeed decreasing among the replies as the Linktrace message comes closer to the target. The first entry in Table 3.5 shows a *Hit*, meaning that the LTM reached its target. That reply message has TTL 62. Combining this with the initial value of 64, it means that the target MEP is $64 - 62 = 2$ hops away from the source MEP. In addition, these hops are known to be physical hops (e.g. switches) since a device sends at most one reply.

Finally, notice that the MAC addresses of the Ingress and Egress port of the second entry in Table 3.5 are distinct. Also, these MAC addresses are not the same as the chassis MAC of S_3, which is A1:B2:C3:D4:E5:90.

3.2 Bravo configurations

The Bravo configurations are almost the same as the Alfa configuration, but they differ in the amount of MIPs configured on S_3. Four variations on the Alfa configuration have been used to initiate and observe the Linktrace Protocol, and only one has been used to query the MIP CCM database (Bravo 3). The four configurations are:

- **Bravo 1:** S_3 only has MIPs configured on ports 1 and 2.
- **Bravo 2:** S_3 only has MIPs configured on ports 1 and 3.
- **Bravo 3:** S_3 only has a MIP configured on port 1.
- **Bravo 4:** S_3 has no MIPs configured.

3.2.1 Bravo 1 and 2

The Bravo 1 and 2 configurations are very much alike and can be considered symmetric. For the configuration of Bravo 1, see Figure 3.2 and for Bravo 2 see Figure 3.3.

In each of the two configurations a Linktrace was initiated with source MEP 2 and target MEP 3. In that way, the LTM should pass through two ports of S_3, ports 2 and 3, on which only one of those

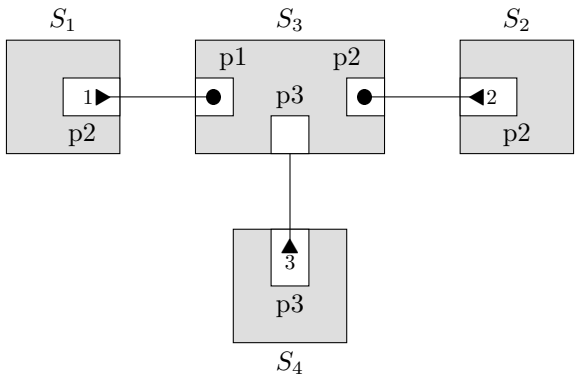


Figure 3.2: The switch configuration of the Bravo 1 experiment. S_i is switch i, pj stands for port j and the number next to each MEP is the configured MEP ID.

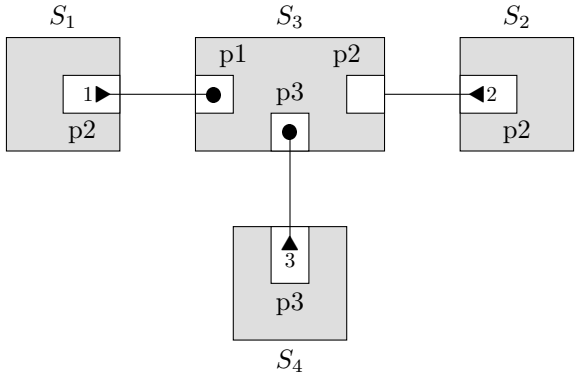


Figure 3.3: The switch configuration of the Bravo 2 experiment. S_i is switch i, pj stands for port j and the number next to each MEP is the configured MEP ID.

two ports has a MIP configured. The Linktrace replies of Bravo 1 are listed in Table 3.6 and the replies of Bravo 2 are listed in Table 3.7.

The Linktrace replies confirm that the Linktrace responder does not have any information on ports that have no MIP (or MEP) configured. In the Bravo 1 configuration, switch S_3 has no MIP configured on port 3. The path of the LTM goes through S_3; it enters at port 2 and leaves through port 3, meaning that port 3 is the egress port. However, since port 3 has no MIP configured, the Linktrace responder does not know about port 3 and therefore the egress values are unknown in Table 3.6, second row (with TTL of 63). This row corresponds to the LTR from the Linktrace responder of S_3. Similarly, the ingress values of the second row in Table 3.7 are unknown because in the Bravo 2 setup the ingress port on the path of the LTM, port 2, has no MIP configured.

3.2.2 Bravo 3

The Bravo 3 configuration is different from the Alfa configuration in that it only has a MIP configured on port 1 of switch S_3. The configuration can be seen in Figure 3.4.

The Bravo 1 and 2 configurations showed that the Linktrace responder is not aware of any ports that have no MIPs configured on them. The Alfa setup showed that the MIP CCM database knows which MEP can be reached from what port. So what knowledge does the MIP CCM database have if not all ports have a MIP configured? In this configuration (Bravo 3) the MIP CCM database was queried, and the result is in Table 3.8.

Linktrace Responses									
	Ttl		Remote MP	Relay	Flags				
Ttl	Idx	Trans Id	Mac Address	Action	FY	TM	Ingress TLV	Egress TLV	
62	1	2	00:11:22:33:44:C4	Hit		X	MAC: 00:11:22:33:44:C4 Port: 3 Action: Ok	MAC: Port: Action: undef	
63	1	2	A1:B2:C3:D4:E5:93	MFDB	X		MAC: A1:B2:C3:D4:E5:93 Port: 2 Action: Ok	MAC: Port: Action: undef	

Table 3.6: Linktrace replies of the Linktrace protocol in Bravo 1, with source MEP 2 and target MEP 3.

Linktrace Responses									
	Ttl		Remote MP	Relay	Flags				
Ttl	Idx	Trans Id	Mac Address	Action	FY	TM	Ingress TLV	Egress TLV	
62	1	3	00:11:22:33:44:C4	Hit		X	MAC: 00:11:22:33:44:C4 Port: 3 Action: Ok	MAC: Port: Action: undef	
63	1	3	A1:B2:C3:D4:E5:94	MFDB	X		MAC: Port: Action: undef	MAC: A1:B2:C3:D4:E5:94 Port: 3 Action: Ok	

Table 3.7: Linktrace replies of the Linktrace protocol in Bravo 2, with source MEP 2 and target MEP 3.

Clearly, the MIP CCM database still knows on what port number the CCMs of the MEPs arrive, even though not all of these ports have a MIP configured. That gives a lot of information concerning the topology; if the switch on which the MIP resides is considered as a vertex in a graph then the MIP CCM database indicates the degree of this vertex and also what MEPs can be reached through which "edge" (or port).

Concerning the Linktrace protocol, Section 2.5.3 Linktrace Protocol presented an abstract rule stating that *A Linktrace responder will not respond if the path through the switch does not encounter any maintenance points on the same level as the source MEP of the Linktrace message.* To confirm this, a Linktrace was initiated with source MEP 2 and target MEP 3. The LTM will therefore not pass through port 1 which is the only port on S_3 with a MIP configured. The output of the Linktrace is presented in Table 3.9. It might come as no surprise that indeed only one Linktrace reply is received, since the LTM did not encounter any MIPs in switch S_3. This Bravo 3 configuration shows that this also holds even when the switch has a MIP configured on other ports (port 1 in this case). This means that, even though the Linktrace protocol is seen as a layer 2 traceroute, the Linktrace protocol is not able to see a switch if the path of the LTM does not encounter at least one port with a MIP or MEP configured on that switch.

3.2.3 Bravo 4

This configuration deals with the situation when no MIPs are configured. The setup is shown in Figure 3.5. It follows naturally that the MIP CCM database of switch S_3 is empty in this case: there are no

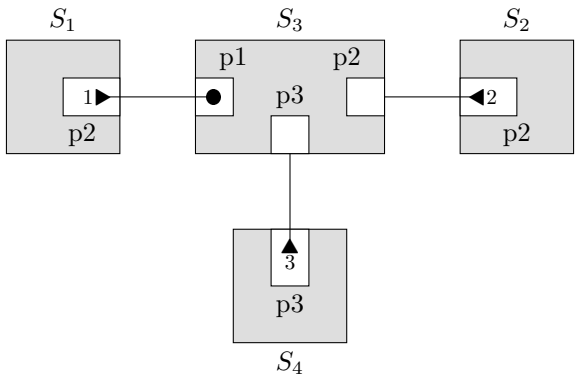


Figure 3.4: The switch configuration of the Bravo 3 experiment. S_i is switch i, pj stands for port j and the number next to each MEP is the configured MEP ID.

MIP CCM Database

VLAN	MAC Address	Port	Total CCM Rx	Last CCM Information: Seq Num	Time	Lv	Mepid	PS	RDI
1	AA:00:BB:11:CC:93	1	146	3151	0	3	1	Up	
1	AA:BB:CC:DD:EE:03	2	5	3222	0	3	2	Up	
1	00:11:22:33:44:C4	3	16	2852	0	3	3	Up	

Table 3.8: The MIP CCM database of switch S_3 in the Bravo 3 setup.

MIPs that report incoming CCMs. Also, triggering any Linktrace in this setup has the same type of output as the Linktrace in the Bravo 3 setup: there is only one Linktrace reply which comes from the target MEP.

				Linktrace Responses					
	Ttl		Remote MP	Relay	Flags				
Ttl	Idx	Trans Id	Mac Address	Action	FY	TM	Ingress TLV	Egress TLV	
63	1	1	00:11:22:33:44:C4	Hit		X	MAC: 00:11:22:33:44:C4 Port: 3 Action: Ok	MAC: Port: Action: undef	

Table 3.9: The Linktrace replies of the Linktrace protocol initiation in Bravo 3 with source MEP 2 and Target MEP 3.

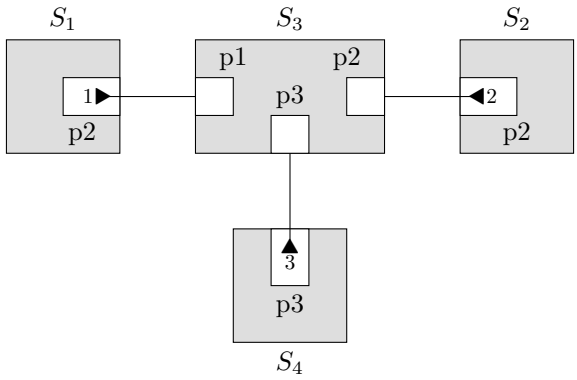


Figure 3.5: The switch configuration of the Bravo 4 experiment. S_i is switch i, pj stands for port j and the number next to each MEP is the configured MEP ID.

3.3 Charlie configurations

The Charlie configurations deal with situations where switch S_3 has a MEP configured on port 3, while the switches remain connected to each other in the same way as in the Alfa and Bravo setups. There are two Charlie setups:

- **Charlie 1:** only has a MEP configured on switch S_3, port 3.
- **Charlie 2:** has a MEP configured on port 3, and a MIP on both port 1 and 2 on switch S_3.

The purpose of these two configurations is to confirm if the Linktrace protocol indeed follows the rules stated in Section 2.5.3, especially when there is an up MEP configured on a switch (S_3). Secondly, what does the MIP CCM database look like when a MEP is configured on the same switch as the MIP (note that this must be an up MEP).

3.3.1 Charlie 1

In the Charlie 1 configuration, LTMs can pass through S_3 without having MEP 3 as a target and without passing a MIP. See Figure 3.6 for the Charlie 1 configuration.

Indeed when a Linktrace is initiated with source MEP 1 and target MEP 2, the only LTR that is received is from MEP 2 (thus actually the Linktrace responder of switch S_2). Clearly the second rule holds; *A Linktrace responder will not respond if the path through the switch does not encounter any maintenance points on the same level as the source MEP of the Linktrace message.*

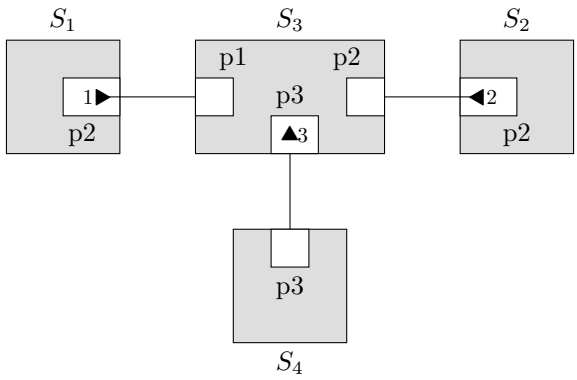


Figure 3.6: The switch configuration of the Charlie 1 experiment. S_i is switch i, pj stands for port j and the number next to each MEP is the configured MEP ID.

3.3.2 Charlie 2

In this configuration, switch S_3 contains both MIPs and a MEP, see Figure 3.7. The MIP CCM database

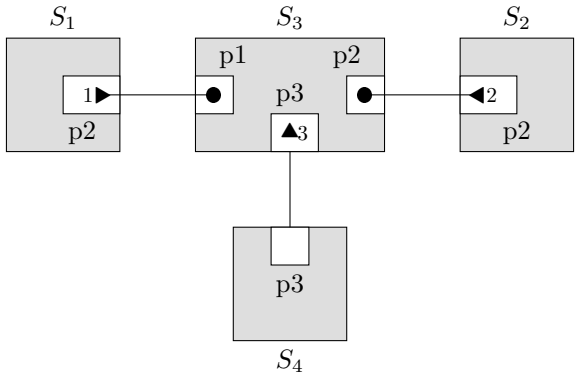


Figure 3.7: The switch configuration of the Charlie 2 experiment. S_i is switch i, pj stands for port j and the number next to each MEP is the configured MEP ID.

of switch S_3 in this configuration is presented in Table 3.10. Remarkably, the MIP CCM database does not contain information on MEP 3, which also resides on switch S_3. Apparently, MIPs do not receive Continuity Check messages from up MEPs configured on the same switch. That means that if the MIP CCM databases are used to recover all MEPs, at least two switches should be queried for the MIP CCM databases. Otherwise the MEPs that are configured on the switch are left out.

The Linktrace protocol, with source MEP 1 and target MEP 2, outputs the expected two LTRs in this configuration. Thus it behaves similarly to the Alfa configuration and according to rule 1 stated Section 2.5.3.

In addition, when the Linktrace protocol is initiated with source MEP 1 and target MEP 3 the output consists of only one LTR. This is indeed how it should behave according to rule 2 stated in Section 2.5.3.

MIP CCM Database									
VLAN	MAC Address	Port	Total CCM Rx	Last CCM Information: Seq Num	Time	Lv	Mepid	PS	RDI
1	AA:00:BB:11:CC:93	1	1046240	1836	0	1	1	Up	
1	AA:BB:CC:DD:EE:03	2	4297	1825	0	1	2	Up	

Table 3.10: The MIP CCM database of switch S_3 in the Charlie 2 setup.

3.4 Delta configuration

The Delta configuration deals with the case in which more than one up MEPs are configured on the same switch. For the setup, see Figure 3.8. When querying the MIP CCM database of switch S_3, the resulting

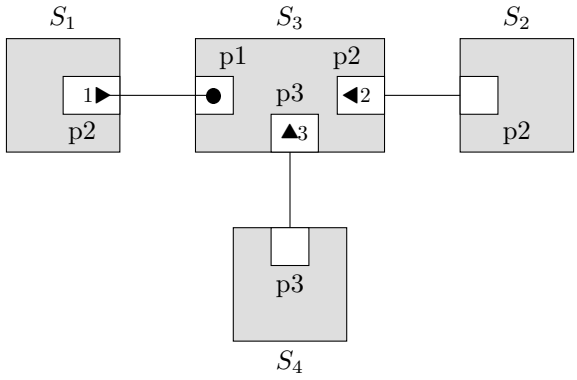


Figure 3.8: The switch configuration of the Delta experiment. S_i is switch i, pj stands for port j and the number next to each MEP is the configured MEP ID.

table lists only one remote MEP: MEP 1. This is similar to the case of the Charlie 2 configuration in which the MEP on the same switch was not listed either.

Initiating the Linktrace protocol with source MEP 2 and target MEP 3 results in an error: *Remote MEP not found*. That means that the Linktrace protocol can only be initiated between maintenance points which are on physically disjoint devices. This makes sense since the Linktrace responder is bound to the switch, and not to the maintenance points.

3.5 Echo configuration

This configuration is only for testing the MIP CCM database when CCMs of multiple MEPs arrive at the same port of the switch on which the MIP is configured. For the configuration, see Figure 3.8.

Switch S_1 was queried for the MIP CCM database and the output is in Table 3.11.

Observe that indeed MEPs 2 and 3 are both listed on port 2, which is the port of switch S_1 which is connected to MEPs 2 and 3. Thus, multiple MEPs can be listed on the same port in the MIP CCM database.

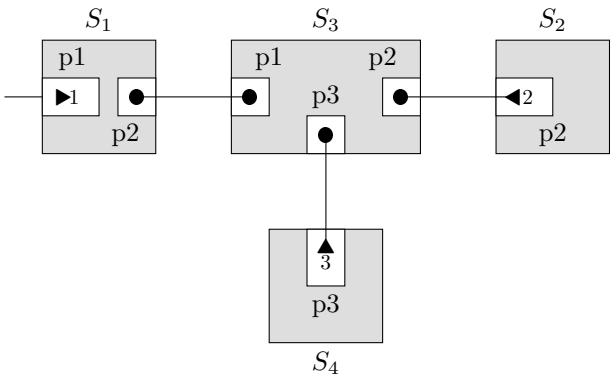


Figure 3.9: The switch configuration of the Echo experiment. S_i is switch i, pj stands for port j and the number next to each MEP is the configured MEP ID.

MIP CCM Database

VLAN	MAC Address	Port	Total CCM Rx	Last CCM Information: Seq Num	Time	Lv	Mepid	PS	RDI
1	AA:BB:CC:DD:EE:03	2	65	65	0	4	2	Up	
1	00:11:22:33:44:C4	2	51	51	0	4	3	Up	

Table 3.11: The MIP CCM database of switch S_1 in the Echo setup.

3.6 Foxtrot configuration

The Foxtrot configurations test the behaviour of MIPs and MEPs when different configurations are present at the same time. Note that IEEE 802.1Q requires that maintenance points discard CFM traffic of lower levels. The Foxtrot configuration has the Alfa and Echo configurations enabled at the same time. For the Alfa configuration, see Figure 3.1. For the Echo configuration, see Figure 3.9 .

When the Alfa configuration is active at level 2 and the Echo configuration is active at level 3 they will operate at the same time without interfering with each other. However, when the Alfa configuration is elevated to level 5, the Echo configuration indicates a remote defect at MEP 1. It does not receive CCMs of (Echo) MEPs 2 and 3 anymore. That means that these CCMs are blocked by the Alfa-MEP 1, as required by the standard.

Note that when Alfa was at level 2 it did not indicate any connectivity faults. That means that the MIPs on level 3 do not block the CFM frames of level 2.

3.7 Golf configuration

The configurations Alfa through Foxtrot are all configured in the same VLAN. This raises the question: what happens when CFM services are configured in different VLANs? The Golf configuration has the Alfa configuration enabled at level 2 in VLAN 1. Moreover, it has a Charlie 2 configuration enabled at level 5 on VLAN 2. For the Alfa configuration see Figure 3.1. For the Charlie 2 configuration see Figure 3.7.

This configuration does not trigger any connectivity faults. This shows that the Maintenance Domain nesting only applies to maintenance domains in the same VLAN. This allows different Maintenance Domains to intersect as long as they are not in the same VLAN.

3.8 Summary

Assuming we can initiate CFM protocols on a switch with a MEP configured we can query that switch in order to obtain the number of MEPs in the network together with their MEP IDs. Thus we can assume we have a list of MEP IDs. Moreover, we can learn on what ports that switch can reach the remote MEPs, even when not all ports that are active in the network have a MIP configured. However, the port information originates from the MIP CCM database, meaning that at least one MIP is required on the switch to learn this information.

Regarding the Linktrace protocol there is a very simple but important observation we make on how it behaves. A switch will send a LTR in response to a LTM if and only if the LTM passes through a port of that switch which has a MIP or MEP configured. In addition, the LTR will only contain information on the ports which have a MIP or MEP configured. This is important when dealing with switches that do not have Maintenance Points configured on all ports or not at all.

Chapter 4

Topology Discovery

Why go through all the trouble of finding Ethernet (ISO layer 2) topology discovery methods when a lot of research has been done already on Network Layer (ISO layer 3) topology discovery? The answer is fairly simple: two different Network Layer routes might seem disjoint while on the physical level they may meet several times. That is because the Network Layer does not recognise and is not aware of Ethernet switches. Switches are normally layer 2 entities and do not support Network Layer technology. As a consequence, they will not react to Network Layer protocols like Ping. That means that the IP traceroute function, which is the most commonly used function for Network Layer topology discovery, will not reveal switches. Descending to the Data Link layer (ISO layer 2) is therefore necessary to reveal these switches.

It is important to keep in mind that the Data Link layer is still not the physical layer. On the Data Link layer we still have the same sort of issues as we had on the IP layer; connections that appear to be disjoint can still share resources on the Physical layer. For example, optical switches can be used to redirect data flows and they work independently of the Data Link Layer. That means that the optical switch does not recognize Ethernet frames and does not respond to them. As such the Data Link layer will not reveal these types of switches. On the other hand, going to the Data Link layer is more precise than staying on the IP layer and is therefore a big step forward already.

In this chapter we will look at topology discovery on the Data Link Layer and we consider two sets of tools which we could use for topology discovery. The first is the Link Layer Discovery Protocol (LLDP) and the second is Ethernet OAM.

4.1 The Link Layer Discovery Protocol

The Link Layer Discovery Protocol [IEE09] is a protocol that allows a device to learn all its direct neighbours, but nothing beyond. A device will broadcast a LLDP frame to all of its neighbours at a preset interval. The LLDP frame contains the following mandatory information:

- Chassis ID of the source; to identify the source device,
- Port ID of the source; to identify the port of the source device,
- Time To Live; indicates how long the receiver can keep the information contained in this LLDP frame.

When a device receives a LLDP frame, it puts the information in a local database and then discards the frame. It does not reply to the frame, nor does it forward the frame. It follows that a LLDP supporting device will only learn of its direct neighbours.

4.1.1 Pros and Cons of LLDP

One option to use the LLDP for topology discovery would be to crawl through the network, visiting every device and ask for all its neighbours. One advantage of this method is that it only needs one starting point. As the algorithm crawls along the devices (or nodes) it learns of new nodes and edges. As long as the network is connected, the algorithm will find all devices that react to the LLDP. This crawling method could even make use of the Simple Network Management Protocol [Cas+90] minimizing manual actions on each device.

Secondly, this method would not require additional traffic on the network except for the SNMP messages on the management network of the devices.

Thirdly, since the LLDP is unaffected by blocking states of ports, this method will reveal all devices and links between devices. In other words: this method gives a complete physical topology of the network.

However, the LLDP does not require to send port status information. Therefore it might be impossible to actually distinguish the active topology from the complete topology. In the scope of this research it is more important to have information on the active topology: the links that are actually operational.

In addition, the party executing the algorithm should have access to all devices. This probably means that only operators will be able to use this method since they will be reluctant to give external parties access to their equipment. Also, the measurements are then bounded by devices that the executing party has no access to.

A short description of an algorithm using the LLDP is stated in Appendix B.

4.2 Choosing Ethernet OAM

The CFM protocols of Ethernet OAM have a few important advantages which could make the use of CFM very attractive for an operator. These advantages are the reason that we chose to research topology discovery based on EOAM. The advantages are as follows.

For one, it would allow an external party to measure the network without giving that external party any access to devices. As such, the operator of the network could provide the ability to measure the network as a service to a client.

Secondly, the operator can limit what parts of the network an external party can measure. This would allow the operator to only enable this measurement on the part of the network that is actually used by the external party. As such, the complete topology is not revealed to the external party.

Thirdly, the CFM protocols are designed to follow the same paths as service frames. As a result, the topology returned by algorithms using CFM is a topology of the active network and represents the data flow of the clients service and its normal traffic flow.

On the other hand, EOAM also has a few disadvantages. It is relatively new and based on Ethernet. There is no guarantee that the complete network will actually support CFM measurements. When there are devices that do not support CFM, it is very difficult to locate these devices and impossible to tell how many of them there are. On the other hand, devices that do not support CFM should not disrupt measurements since the EOAM frames are standard Ethernet frames and these devices should treat them like regular traffic.

Also, using CFM, and especially the Linktrace Protocol, additional Ethernet frames are injected in the network. This creates extra traffic loads on the switches and is especially dangerous when CFM is implemented in the software of the switch and not in the hardware.

We think that the advantages of using EOAM, compared to using the Link Layer Discovery Protocol, outweigh the disadvantages and therefore we have not further researched topology discovery using the LLDP.

4.3 Algorithms using Ethernet OAM

Depending on the situation and assumptions, different algorithms are possible with the use of CFM and Linktrace. We will present two algorithms dealing with a network which has CFM enabled on all devices and elaborate on the strategy to follow when there are devices in the network that do not respond to CFM and Linktraces.

The algorithms we propose build on the working of the Linktrace protocol and what information the LTRs provide. We assume that we know how many MEPs there are in the network and that we know their MEP IDs. This is a fair assumption since we can ask for this information at any switch with a MEP configured. Assuming we have access to at least one such switch (otherwise we cannot do anything

at all) we can query that switch for the remote MEP information as described in Chapter 3.

We will also assume that a switch has at most one MEP configured for we cannot initiate a Linktrace between two MEPs on the same switch. Consequently, each MEP is identified with a switch. Finding switches without MEPs is done by analysing the LTRs. Suppose we send a LTM from MEP a to MEP b and we receive two LTRs. One of them lists that it hits the target MEP while the other lists at least one port of a third switch. From a set of different LTRs we can identify different switches based on the MAC addresses of their ports. If a LTR contains a MAC address that we have observed before, then we know that it passes through the same port and thus the same switch. This allows us to group ports together to represent a switch when LTRs list both the ingress and egress ports MAC address.

4.3.1 A Simple Algorithm for topology discovery

Observe that, in order to gain a complete (or as complete as possible) topology, every MEP must be involved in at least one Linktrace. If there is a MEP which is not involved in a Linktrace then it is unknown how this MEP is connected to the rest of the network, resulting in an incomplete topology. This gives a lower bound on the amount of Linktraces needed to gain a complete topology. Given that there are n MEPs, the amount of needed Linktraces λ is bounded by

$$\frac{n}{2} \leq \lambda \leq \binom{n}{2} = \frac{n^2 - n}{2}.$$

The upper bound comes from the fact that there are $\binom{n}{2}$ possible different Linktraces, assuming that a Linktrace from MEP a to MEP b reveals the same path as the Linktrace from b to a (which is true in a tree).

Assuming an ideal situation it is possible to reveal the complete topology of a VLAN. The following assumptions are needed for an ideal situation (a motivation follows below):

1. All ports of switches in the VLAN are configured with MIPs or MEPs,
2. The CFM configuration is a valid configuration,
3. The VLAN forms a tree (no communication loops within the VLAN),
4. The number of MEPs and their ID and MAC address is known,
5. It is **not** possible to connect to MIPs,
6. It is possible to connect to at least one switch that has a MEP configured.

The first assumption makes sure that every switch reacts to the Linktrace protocol. For example, if there is a switch such that two (active) ports do not have any MIP or MEP configured, then it is possible that a Linktrace path goes through precisely those two ports and then the switch will not be noted in that Linktrace. Requiring that all ports have a MIP or MEP configured takes care of this problem.

Secondly, for the Linktrace to work properly; Connectivity Fault Management should be configured in a valid way. That means that there should be no conflicting Maintenance Domains or MEPs pointing in the wrong direction (up when they should be down).

The third assumption makes sure that the VLAN is a tree. If there would be a loop somewhere in the VLAN, then for sure broadcast storms will occur. In addition, if the VLAN is a tree then there is only one unique path from one MEP to another, making it easier to reconstruct the network topology. For example, consider three MEPs a, b and c. In a tree, if it is known how a is connected to b and how a is connected to c, then we know how b is connected to c. So only a maximum of $\binom{n}{2}$ Linktraces is needed instead of $n!$ to reveal the complete topology.

The fourth assumption makes it possible to determine source and target MEPs for the Linktraces. Also note that this assumption will be valid in practice since Chapter 3 showed us we can query a switch for all remote MEPs.

The fifth assumption basically prevents the use of MIP CCM databases. Also, this assumption makes sure that the algorithms based on these assumptions could be used by external parties instead of only

the operator. That is, there is no need to access switches that have MIPs configured (although the operator should still configure the MIPs). Note that the algorithms will not fail if the access to MIP CCM databases is enabled.

Lastly, in order to initiate a Linktrace it should be possible to connect to at least one switch that has a MEP configured. Switches that can be accessed can be used to initiate Linktraces.

Based on these assumptions the most trivial algorithm would initiate a Linktrace between every possible pair of MEPs. This would not exploit the assumption that the VLAN is a tree and it would even require that it is possible to connect to at least half the switches that have a MEP configured. A slightly better algorithm that exploits the assumption of a tree will initiate Linktraces from a single source MEP to all other MEPs. A description of this algorithm is given in Algorithm 1. Suppose

input : A source MEP s and a set of target MEPs M
output : A Tree T representing the topology of the VLAN

foreach $t \in M$ **do**
    Initiate a Linktrace from s to t;
    Process the output of the Linktrace such that it forms a path from s to t. Every node on the path is represented by an ingress and egress port;
end
Combine all the paths based on the ingress and egress port info into a tree T;
return *Tree T*

Algorithm 1: A Naive algorithm using Linktraces.

there are n MEPs, then this algorithm initiates $n-1 = \mathcal{O}(n)$ Linktraces. That means that $n-1$ LTMs (Linktrace Messages) are injected in the network, and also $n-1$ LTRs (Linktrace Replies) are injected in the network. This only accounts for the MEPs. The analysis for the MIPs is more difficult, since it depends on the actual topology of the network.

Algorithm 1 combines all the paths based on the ingress and egress port info. Basically we exploit that ports have a unique identifier known as a MAC (Media Access Control) address. It is very common that ports on the same device have an almost equal MAC address. Typically only the last elements of the MAC address are different for ports on the same device. Therefore we combine ports into a switch if the MAC address only differs my a small amount.

4.3.2 Connecting-Parts Algorithm

A more sophisticated approach makes even stronger use of the assumption that the VLAN is a tree. Every Linktrace is a path, which is also a tree. Because the VLAN forms a tree, combining the paths (representing the Linktraces) will also form a tree. Algorithm 1 uses the same MEP as source for every Linktrace but this is not necessary. Suppose there are n MEPs. Then, in a lucky situation the complete topology might be revealed using $\lceil n/2 \rceil$ Linktraces by using $\lfloor n/2 \rfloor$ different source MEPs and $\lceil n/2 \rceil$ different target MEPs. Every target MEP is used exactly once and every source MEP is used at least once, resulting in $\lceil n/2 \rceil$ Linktraces. In particular, for any star topology (where all MEPs are connected via only one MIP) it will only take $\lceil n/2 \rceil$ Linktraces to reveal the complete topology in this way.

For such an approach to work, a stronger assumption is required: *It must be possible to initiate Linktraces from at least half the switches that have a MEP configured.* The remaining assumptions 1 to 5 stated in Subsection 4.3.1 must remain valid.

Observe that when using different source MEPs and different target MEPs the resulting output might be a forest rather than a tree (i.e. there are disconnected components). These components still have to be glued together by initiating more Linktraces and this can be done in several ways. One way of doing this is to choose one MEP as source, and a target MEP in every other component and initiate Linktraces from the source to each target. However, this can result in doing Linktraces that do not add information, for example when three components are suddenly glued together by one Linktrace. Then we need only one Linktrace rather than two.

A cheaper method (concerning network traffic) is to iteratively choose one source MEP and one target MEP which lay in different components and initiate a Linktrace between them. The output is then used to glue components together. Then, if there are still disconnected components, repeat this process until the entire graph (tree) is connected. This method is shown in Algorithm 2.

Note that when choosing source MEPs, it should be taken into account which MEPs can be used to initiate Linktraces and which MEPs can not. It is not possible to choose a MEP as source when we can not initiate a Linktrace on the switch of that MEP.

input : A set of MEPs M, and a subset $M_S \subset M$ of MEPs from which Linktraces can be initiated, satisfying $|M_S| \geq \lfloor |M|/2 \rfloor$.
output : A Tree (connected forest) F representing the topology of the VLAN
initialize: Generate set M_{pairs} of pairs $(s,t) \in S \times M$ of source (s) and target (t) MEPs as described in Section 4.3.2, $F = (V,E) = (\varnothing, \varnothing)$
foreach $(s,t) \in M_{pairs}$ **do**
  Linktrace(s,t);
  Process Linktrace output: turn into Tree $T = (V_T, E_T)$;
  $F \leftarrow F \cup T$;
end
if F *is disconnected* **then**
  repeat
    Choose $s \in S$ and $t \in M$ such that s and t are not in the same connected component;
    Linktrace(s,t);
    Process Linktrace output: turn into Tree $T = (V_T, E_T)$;
    $F \leftarrow F \cup T$;
  until F *is connected*;
end
return F

Algorithm 2: Description of the Connecting-Parts algorithm.

Algorithm 2 needs $\lambda = n - 1$ Linktraces in the worst case to finish. After the first $\lceil n/2 \rceil$ Linktraces all MEPs have been used once and in a very unlucky case we have obtained $\lceil n/2 \rceil$ disconnected parts. We continue to be very unlucky when, for every extra Linktrace we initiate, we only manage to connect two parts every time. That means we have $\lceil n/2 \rceil$ initial Linktraces and need $\lceil n/2 \rceil - 1$ extra Linktraces. Adding them together results in $n - 1$ Linktraces. Figure 4.1 is an example of a network in which we might get unlucky. To obtain $n - 1 = 5$ Linktraces the random pairs could be: (a,b), (c,d), (e,f), (a,c),

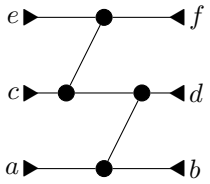


Figure 4.1: A network which attains a worst-case number of Linktraces.

(c,f). Then the first three pairs will induce three disconnected parts. The fourth pair, (a,c) will only connect two parts and thus a fifth pair, (c,f), is needed to connect everything.

4.4 Non-responding switches

A next natural step is to see what we can do when we do not have a network that satisfies the strengthened assumptions of Section 4.3.2. In particular, suppose there are some switches that do not respond to the

Linktrace protocol or ports that do not have MIPs configured. Can we still use Algorithm 2 to identify the network?

The answer is that we can partially discover the network topology. We will argue that the Linktrace protocol enables us to determine if (and where) there is a switch that does not respond to the Linktrace protocol. However, this is only possible when that switch has at least three ports connected to the network. The technique to identify a non-responding switch is to recognise when the Linktrace output encounters an unexpected diversion from an already known route.

Consider the simple situation in Figure 4.2b in which the switch S_2 does not respond to the Linktrace protocol (has no MIPs configured). If we perform a Linktrace from MEP a to MEP b and from MEP a

(a) S_1 is responsive switch.

(b) S_2 is a non-responsive switch.

Figure 4.2: Two simple situations involving responsive switches and a non-responsive switch.

to MEP c, the output will suggest that the topology is the one depicted in Figure 4.2a. When we also perform a Linktrace from MEP b to MEP c we observe that the path from b to c does not go via switch S_1. The corresponding output restricted to b and c would suggest $b \blacktriangleright\!\!\!-\!\!\!\blacktriangleleft c$. In other words, switch S_1 is not part of the route from b to c. Therefore, there must be another "junction" between S_1 and c to which b is connected. For the purpose of discovering S_2 it doesn't matter what is between a and S_1. In Figure 4.2 the part between a and S_1 consists of only a single link, but if that part of the network was bigger we would still observe that b and c are not connected via S_1. That means we can repeat this process for every switch which acts as a junction between *at least three* parts of the network, each part containing at least one MEP.

However, it is not possible to find all non-responsive switches; only those that connected at least three MEPs. In Figure 4.3 we see two situations which can not be distinguished from each other by use of the algorithm. Switch S_3 in Figure 4.3b divides the network in two parts and can therefore not be

(a) S_2 is non-responsive switch.

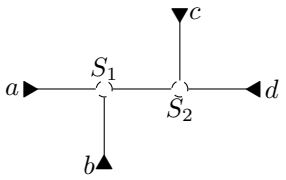
(b) S_2 and S_3 are a non-responsive switches.

Figure 4.3: These two situations can not be distinguished.

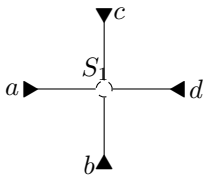
found by analysing Linktrace outputs.

Another situation in which the correct physical topology cannot be observed is shown in Figure 4.4a. The observed topology will correspond to the topology shown in Figure 4.4b. A fair question is if we can even find the correct topology in Figure 4.4b? To answer that we must zoom to port level and include port information, in particular the type of MEP: up or down. If all MEPs are down MEPs then we indeed obtain the correct topology. If one of the MEPs is an up MEP, there is no way in which we can detect the difference between the situations of Figure 4.5 since there will be no information on the ingress/egress ports on the switch of MEP c. We can however learn that MEP c is an up MEP when MEP c is a target MEP of a Linktrace. The corresponding LTR will show an empty ingress port and MEP c will be listed as egress port, which means c is an up MEP.

We can also learn if a source MEP is an up or down MEP by just asking the switch on which the source MEP is configured. This should be possible by using the same connection as the one used for

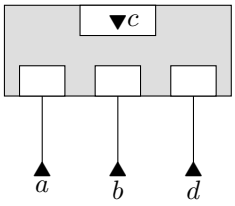


(a) S_1 and S_2 are non-responsive switches.

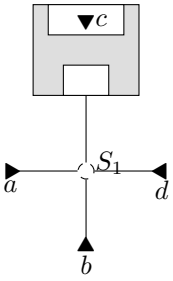


(b) S_1 is a non-responsive switch.

Figure 4.4: These two situations can not be distinguished.



(a) MEP c is an up MEP. MEPs a, b and d are connected via the switch of c.



(b) MEP c is an up MEP and S_1 is a non-responsive switch.

Figure 4.5: Zooming in on port level of the switch on which MEP c is configured. Empty ports have no MIPs configured.

initiating the Linktrace. Note that if we require that when a switch has an up MEP configured also all outgoing ports have MIPs configured, then the two situations of Figure 4.5 become distinguishable.

In short we have the following observations:

1. A non-responsive switch can only be identified if its deletion cuts the network into *at least* three disjoint parts, each containing at least one MEP,
2. It is impossible to separate two neighbouring non-responsive switches (as shown in Figure 4.4a),
3. A switch with an up MEP and no MIPs may imply a wrong topology.

The above mentioned problems can be overcome by setting some restrictions on the network:

1. Every non-responsive switch must connect at least three disjoint parts of the network,
2. Every non-responsive switch has no other non-responsive switch as a neighbour,
3. If a switch has a MIP or up MEP configured, then all ports have a MIP configured (except the port with an up MEP).

However, from the perspective of measuring the network it might of course be unknown if these restrictions are satisfied. Hence, if no information is available about non-responsive switches the resulting topology can contain untraceable errors.

If we know there are non-responding switches in the network, we need to initiate a Linktrace between all $\binom{n}{2}$ possible pairs of MEPs. Every pair of MEPs between which no Linktrace has been initiated may hide a non-responsive switch. For example, consider Figure 4.2b in which the pairs (a, b) and (a, c) have been used to initiate Linktraces. The non-responding switch S_2 remains unobserved as long as we do not initiate a Linktrace between b and c. For every non-used pair such a situation may exist. That means that we need to initiate Linktraces between all possible pairs of MEPs when we are dealing with non-responding switches.

4.4.1 The importance of non-responding switches

Besides inducing an incorrect topology, non-responding switches can cause another problem. Let us go back to the original problem: testing if two connections are disjoint. Most of the time these connections are set up over (different) tunnels or VLANs. Consider Figure 4.6a with two different VLANs; one for each connection. Since switch S_2 is not responding we cannot see that the red and blue VLAN collide

(a) Physical red and blue VLAN with a collision at non-responding switch S_2.

(b) Observed disjoint red and blue VLAN.

Figure 4.6: A non-responding switch can connect two different VLANs such that the red and blue VLAN seem disjoint but in reality are not.

at that switch. From measurements it will seem as if the two VLANs are disjoint and thus very robust while in reality they have a single point of failure: S_2. Similarly, a link can also be an unobserved single point of failure when it connects two non-responding switches.

4.5 Practical use of the algorithm

The algorithms and ideas presented in the previous sections are all based on theoretic descriptions, standards and laboratory experiments. We implemented only the naive algorithm in Java and tested it on a small network (Alfa configuration) and the output was indeed the physical topology. Of course the conditions were nearly perfect: all switches were responding and we had access to all switches. The question is if the algorithms are useful in real ISP networks, i.e. are our assumptions not too strong in order to be used in a real ISP network?

In order to answer this question we consulted SURFnet and presented our algorithm and ideas. Our first concern was the proportion of devices that could handle CFM. As it turns out, almost all equipment of SURFnet can handle CFM, meaning that there is almost no need to scan for non-responding switches. A disadvantage is that most devices have the CFM functionality implemented in software, which might cause a major inconvenience when many CFM frames have to be processed. That means that keeping the amount of Linktraces down is very important in order to prevent the switches from overloading their queues.

In addition, many switches might be able to handle CFM but most of the time MIPs are not configured on the switches. If someone triggers many Linktraces then the number of LTRs can be very high which effectively might be observed as a DDoS attack. To protect the network from overloading with CFM frames MIPs are not commonly configured. In addition, it is currently impossible to configure MIPs inside a tunnel.

The second concern regards the type of connections used to provide customers with Ethernet connections. SURFnet told us that in almost all cases they provide Ethernet connectivity based on a service over a tunnel using PBB-TE: Provider Backbone Bridge - Traffic Engineering. The idea is that a Network Management System (NMS) is responsible for configuring this tunnel such that it does not have loops (and thus will not cause broadcast storms). Data traffic enters the tunnel at one point and is only evaluated again at the end of the tunnel. Entities between the two end points of the tunnel only forward the data packets without looking at them. Once these tunnels have been set up they will preferably not change too much, meaning that the active topology of such a connection is pretty static.

It is clear that in theory our algorithm can work but in practice it is not yet possible when taking into account the above mentioned limitations. To overcome almost all these limitations we propose that the algorithm can be run as a service. When a customer of an ISP wants to measure the topology of their connections they can buy this as a service. The ISP in question can then configure a VLAN along

the path of the already existing tunnel and configure MIPs and MEPs on that VLAN. After that, the company can perform a measurement and when they are done the ISP will shut down the access to the VLAN. This prevents (accidentally) overloading the network and also allows to configure MIPs among all switches. SURFnet made clear that this is possible as long as the vendor of the Network Management Software implements this functionality.

4.6 Summary

We presented an algorithm which used the Linktrace protocol to recover the network topology. Under the assumption that all network elements (of the Data Link Layer and higher) respond to the Linktrace Messages our algorithm finds the correct topology when the network is a tree. Moreover, we have shown that it is possible to deal with switches that do not respond to the Linktrace protocol as long as there are no two of such switches directly adjacent to each other. Finally we have received positive feedback from SURFnet, basically stating that if a Network Management System vendor is willing to implement our algorithm it is technically possible to use that algorithm in practice.

Chapter 5

Generalised Topology Discovery

In this chapter we present a more abstract approach to topology discovery. In this approach we are interested in the number of probes that we need to discover the network topology. In a data network, initiating probes causes an increase in network traffic, and we want to keep the load on the network as small as possible.

Suppose we are given a set of vertices V and some probing operation ρ. Rather than limiting the vertices which can be used as probing points as in the situation of MEPs, we can now use all $|V| = n$ vertices as probing points. A probe $\rho(u, v)$ between vertices u and v will reveal the shortest path from u to v (in terms of the number of edges). So after the probe $\rho(u, v)$ we know all nodes and edges and their order on the shortest path between u and v. Assuming that we are dealing with undirected graphs, the shortest path from u to v is the reversed shortest path from v to u. Therefore we obtain $\rho(u, v) = \rho(v, u)$. We will present two strategies that use ρ to determine the network topology and analyse these strategies for some different types of tree graphs.

Similar to the goal of Chapter 4, we wish to recover the topology of the network (or graph) $G = (V, E)$. In other words, we wish to find the edges E of the network. The first natural question is if we can recover the topology by using ρ? Before we say something about this we set a limitation on the network. For the purpose of this chapter, we suppose that the network topology is a tree; a connected graph without cycles. This type of network is very interesting in the setting of VLANs or LANs, where loops are forbidden in order to prevent broadcast storms. Given a tree, we can discover the complete topology in the following way. Choose any vertex $s \in V$. Then, for every vertex $v \in V \setminus \{s\}$ retrieve $\rho(s, v)$. The combined outputs of ρ will form a tree. In this way we have determined the connection of all vertices relative to the vertex s. Since the network is a tree this ensures that we have indeed found all edges using $|V| - 1 = n - 1$ probes.

In Section 5.1 we will consider other probing strategies and we will analyse the performances for a line and a star graph. The analysis of the presented probing strategies is very hard for general trees and remains as an open problem.

5.1 Probing strategies

We already know that using ρ we can determine the topology of the network using $n-1$ probes. However, the procedure we introduced above seems pretty inefficient since some probes will reveal a path which was already known. Is there perhaps a better strategy? Can we recover the topology in less than $n - 1$ probes? A first simple improvement is to only probe to nodes which have not yet been observed. A node u is considered observed when it has either been used as a probing node or when the result of a probe includes u in the path output. The simple improvement results in $\Lambda \leq n - 1$, where Λ is the number of probes needed to recover the network topology. We will argue that any probing strategy will need $n - 1$ probes in the worst case.

First we define what a probing strategy actually is: given the current knowledge of the topology, a probing strategy should choose a pair of vertices, u and v, and increase the knowledge by adding $\rho(u, v)$. A good strategy will not choose u and v in the same already observed connected component since that does not add any new knowledge. Suppose we have a good probing strategy, i.e. one which does not choose u and v in the same already observed connected component. Construct a tree network as follows. Start with an empty network F on n vertices. Then, given the current situation, the strategy randomly selects two vertices u and v. To F, add the edge (u, v). Repeat this process until F is connected. If we now use the probing strategy on F, we need exactly $n - 1$ vertices. Also note that because the probing

strategy does not select a pair of vertices which lie in the same observed connected component, the resulting connected graph F has no loops and thus forms a tree.

We see that the worst-case bound for the number Λ is tight for any good probing strategy. Hence we try to make up a probing strategy which does well on average. We will now choose our pairs of vertices somewhat random.

Strategy I (Shrinking Random Pairs)**.** *From the set of vertices available to choose, randomly select two vertices u and v and reveal $\rho(u, v)$. Then, shrink $\rho(u, v)$ into a single "super vertex" which is treated as an unobserved vertex, but remember the underlying line $\rho(u, v)$. Continue until there is only one "super vertex" left.*

Alternatively one might come up with the strategy which only chooses vertices that have not yet been observed at all and where an already discovered connected part is left as it is. Then we would only choose already observed nodes if all (or all but one) nodes have already been observed and the discovered structure is still disconnected. The reasoning behind this strategy is that in the first probes one would expect to discover more edges.

Strategy II (Unused Nodes First)**.** *From the set of vertices available to choose, randomly select two vertices u and v and reveal $\rho(u, v)$. Then, mark u, v and all other vertices discovered by $\rho(u, v)$ as unavailable nodes. When the number of unavailable nodes is zero or one, shrink every observed connected component into a single node which is treated as an unobserved node (and thus available to choose). Remember the underlying structure of the shrunken connected component. Continue until all nodes are in the same connected component.*

5.2 Line graph

We have analysed the performance of Strategy I on a line graph. In a line graph on n vertices, there are two vertices which have degree 1 and all other vertices have degree 2. An example is given in Figure 5.1.

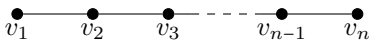


Figure 5.1: A line graph on n vertices.

For $n \in \{3, 4, 5\}$ we have explicitly calculated the expected value of the number of probes needed to map a line graph using Strategy I and Strategy II. The expected values are 5/3, 13/6 and 77/30 respectively and they are equal for both Strategies. Since $n = 5$ is already an extensive case and both strategies lead to the same expected value for all three values of n, we suspect that for a line graph the strategies are equivalent in performance. However, we found no proof of this yet. It does motivate why we have only analysed Strategy I for a line graph.

According to Strategy I a pair of vertices is selected at random among the currently unobserved vertices. So in the first step any pair of vertices is just as likely. There is a possibility that we are very lucky in choosing the first pair as two end nodes of the line. There are $\binom{n}{2} = n(n-1)/2$ possible first pairs and therefore with probability $1/\binom{n}{2}$ we need only one probe.

Define Λ_n as the number of probes we need to fully identify a line on n vertices when following Strategy I. Then what can we say about Λ_n? We have already seen that $1 \leq \Lambda_n \leq n-1$ and as a next step we will look at the expected value of Λ_n. First recall that the definition of the expected value applied to Λ_n gives us

$$\mathbb{E}[\Lambda_n] = \sum_{y=1}^{n-1} y\mathbb{P}[\Lambda_n = y]. \tag{5.1}$$

We will exploit the conditional expectation in which we condition on the number of edges discovered at the first probe. So, let X_n be the number of edges discovered in the first probe of a line on n vertices. Then $\mathbb{E}[\Lambda_n] = \mathbb{E}[\mathbb{E}[\Lambda_n \mid X_n]]$. We will first show that $\mathbb{E}[\Lambda_n \mid X_n = x] = 1 + \mathbb{E}[\Lambda_{n-x}]$ in Lemma 1.

Lemma 1. *The expected number of probes needed to fully identify a line on n vertices satisfies*

$$\mathbb{E}[\Lambda_n] = 1 + \sum_{x=1}^{n-1} \frac{n-x}{\binom{n}{2}} \mathbb{E}[\Lambda_{n-x}].$$

Proof. Observe that for $1 \leq x \leq n-1$

$$\begin{aligned}\mathbb{E}[\Lambda_n \mid X_n = x] &= \sum_{y=1}^{n-1} y\mathbb{P}[\Lambda_n = y \mid X_n = x] \\ &= \sum_{y=1}^{n-1} y \frac{\mathbb{P}[\Lambda_n = y, X_n = x]}{\mathbb{P}[X_n = x]}.\end{aligned}$$

Recall that for n points we need at most $n-1$ probes. At the first probe we have identified x edges, leaving us with an unknown line on $n-x$ vertices. Thus for $y > n-x$ we obtain $\mathbb{P}[\Lambda_n = y \mid X_n = x] = 0$. In addition, the event $(\Lambda_n = y) \cap (X_n = x)$ only happens when $X_n = x$ and $\Lambda_{n-x} = y-1$. For sure we need to discover x edges at the first probe, which happens with probability $(n-x)/\binom{n}{2}$. Secondly, if in the first step we discovered x edges, we have $n-1-x$ edges left between $n-x$ vertices and we must spend $y-1$ probes on these $n-x$ vertices. This gives us:

$$\begin{aligned}\mathbb{E}[\Lambda_n \mid X_n = x] &= \sum_{y=1}^{n-1} y \frac{\mathbb{P}[\Lambda_n = y, X_n = x]}{\mathbb{P}[X_n = x]} \\ &= \sum_{y=1}^{n-x} y \frac{\mathbb{P}[\Lambda_n = y, X_n = x]}{\mathbb{P}[X_n = x]} \\ &= \sum_{y=1}^{n-x} y \frac{\mathbb{P}[\Lambda_{n-x} = y-1]\,\mathbb{P}[X_n = x]}{\mathbb{P}[X_n = x]} \\ &= \sum_{y=1}^{n-x} y\mathbb{P}[\Lambda_{n-x} = y-1] \\ &= \sum_{y=1}^{n-x} (y-1+1)\mathbb{P}[\Lambda_{n-x} = y-1] \\ &= 1 + \sum_{y=0}^{n-x-1} y\mathbb{P}[\Lambda_{n-x} = y].\end{aligned}$$

Since in the last expression we have $0 \cdot \mathbb{P}[\Lambda_{n-x} = 0] = 0$ we obtain

$$1 + \sum_{y=1}^{n-x-1} y\mathbb{P}[\Lambda_{n-x} = y] = 1 + \mathbb{E}[\Lambda_{n-x}]$$

and therefore we conclude

$$\mathbb{E}[\Lambda_n \mid X_n = x] = 1 + \mathbb{E}[\Lambda_{n-x}]. \tag{5.2}$$

From 5.2 it follows that

$$\begin{aligned}\mathbb{E}[\Lambda_n] &= \mathbb{E}[\mathbb{E}[\Lambda_n \mid X_n]] \\ &= \sum_{x=1}^{n-1} \mathbb{P}[X_n = x](1 + \mathbb{E}[\Lambda_{n-x}]) \\ &= 1 + \sum_{x=1}^{n-1} \frac{n-x}{\binom{n}{2}} \mathbb{E}[\Lambda_{n-x}].\end{aligned}$$

The equality $\mathbb{P}[X_n = x] = (n-x)/\binom{n}{2}$ is shown in Lemma 2. □

The recursive expression for $\mathbb{E}[\Lambda_n]$ can be used to calculate all expected values, but it does not directly give insight in how many probes we need proportional to the number of vertices. We would expect that in each step the same proportion of the current unknown edges is revealed and that therefore the expected number of probes needed to identify a line on n vertices grows logarithmically with n. Theorem 1 shows that $\mathbb{E}[\Lambda_n]$ is upper bounded by a logarithmic function of n.

Theorem 1. *Let Λ_n be the number of probes needed to recover a line on n vertices, then*

$$\mathbb{E}[\Lambda_n] \leq \log_{\frac{2}{3}}(2) + \log_{\frac{3}{2}}(n+1),$$

for $n \geq 3$.

We will first show how to get to the logarithmic expression and then present the proof of the theorem. Consider the first probe on a line on n vertices and $m = n-1$ edges. The number of edges we expect to discover in the first probe is $(m+2)/3$ as shown in the proof of Lemma 2.

Lemma 2. *Given a line graph G on n vertices and $m = n-1$ edges and two random vertices u and v $(u \neq v)$ in this line graph, the expected number of edges revealed by $\rho(u,v)$ is $(m+2)/3$.*

Proof. The operation $\rho(u,v)$ reveals edges on the shortest path between u and v. In a tree graph, and thus in a line graph, there is only one path from u to v which is automatically also the shortest path. Suppose $\rho(u,v)$ reveals $1 \leq k \leq n-1$ edges. Using Figure 5.1 we see that there are $n-k$ such possible pairs: (v_1, v_k), (v_2, v_{k+1}), …, (v_{n-k}, v_n). Thus the probability that $\rho(u,v)$ reveals k edges is

$$\mathbb{P}[\rho(u,v) \text{ reveals } k \text{ edges}] = \frac{n-k}{\binom{n}{2}}.$$

A quick sanity check tells us that if we sum over all $1 \leq k \leq n-1$ we get

$$\begin{aligned}
\sum_{k=1}^{n-1} \frac{n-k}{\binom{n}{2}} &= \frac{1}{\binom{n}{2}} \left(n \left(\sum_{k=1}^{n-1} 1 \right) - \left(\sum_{k=1}^{n-1} k \right) \right) \\
&= \frac{1}{\binom{n}{2}} \left(n(n-1) - \frac{n(n-1)}{2} \right) \\
&= \frac{\binom{n}{2}}{\binom{n}{2}} = 1.
\end{aligned}$$

This means that we have valid probabilities. It follows that the expected number of edges revealed by $\rho(u,v)$ is:

$$\begin{aligned}
\mathbb{E}[\text{Number of edges revealed by } \rho(u,v)] &= \sum_{k=1}^{n-1} \frac{n-k}{\binom{n}{2}} k \\
&= \frac{1}{\binom{n}{2}} \sum_{k=1}^{n-1} nk - k^2 \\
&= \frac{2}{n(n-1)} \left(n \left(\sum_{k=1}^{n-1} k \right) - \left(\sum_{k=1}^{n-1} k^2 \right) \right) \\
&= \frac{2}{n(n-1)} \left(\frac{n^2(n-1)}{2} - \frac{n(n-1)(2(n-1)+1)}{6} \right) \\
&= n - \frac{2n-1}{3} = \frac{n+1}{3} = \frac{m+2}{3}.
\end{aligned}$$

□

Consequently, the expected number of unobserved edges after a first probe will be

$$m - \frac{m+2}{3} = \frac{2(m-1)}{3}. \tag{5.3}$$

Continuing to apply Strategy I we shrink $\rho(u,v)$ into a single vertex and we obtain a new line. This new line has, on expectation $m - (m+2)/3) = 2(m-1)/3$ undiscovered edges. We pretend that we are again in a situation in which we can issue a "first" probe but this time on a line on $2(m-1)/3$ edges. The

expected number of edges we discover if we probe a line on $2(m-1)/3$ edges will be $(2(m-1)/3+2)/3$ and thus the number of unobserved edges will be

$$\frac{2(m-1)}{3} - \frac{\frac{2(m-1)}{3}+2}{3} = \left(\frac{2}{3}\right)^2 (m-1) - \frac{2}{3}.$$

If we define N_i to be the number of unobserved edges at the i^{th}step of this process, then we claim that

$$N_i = \left(\frac{2}{3}\right)^i (m+2) - 2. \tag{5.4}$$

This can be shown by induction. For $i = 0$, the situation in which we have not done any probing, we obtain $N_0 = m$. Assuming the statement is true for N_i, it follows that

$$\begin{aligned} N_{i+1} &= \frac{2}{3}(N_i - 1) \\ &= \frac{2}{3}\left(\left(\frac{2}{3}\right)^i (m+2) - 2 - 1\right) \\ &= \left(\frac{2}{3}\right)^{i+1} (m+2) - \frac{2}{3}(3) \\ &= \left(\frac{2}{3}\right)^{i+1} (m+2) - 2. \end{aligned}$$

Thus indeed, for this process expression 5.4 is correct. We are interested in the value of i for which we have 0 unobserved edges left. In other words; the value of i for which N_i is 0. We solve equation $N_i = 0$ for i and obtain

$$i = \log_{\frac{2}{3}}(2) + \log_{\frac{3}{2}}(m+2) = \log_{\frac{2}{3}}(2) + \log_{\frac{3}{2}}(n+1).$$

Theorem 1 states that this expression is an upper bound.

Proof of Theorem 1. We will prove this by induction. For the base case take $n = 3$. We can easily calculate $\mathbb{E}[\Lambda_n]$ by using that $\mathbb{P}[\Lambda_n = 1] = 1/3$ and $\mathbb{P}[\Lambda_n = 2] = 2/3$ and we get $\mathbb{E}[\Lambda_n] = 5/3$. On the other hand we have $\log_{\frac{2}{3}}(2) + \log_{\frac{3}{2}}(3+1) \approx 1.7095 \geq 1.7 \geq 5/3$. We see that for $n = 3$ the bound holds.

Next we assume that the bound holds for Λ_n and we consider Λ_{n+1}. Using Lemma 1 we write

$$\mathbb{E}[\Lambda_{n+1}] = 1 + \sum_{x=1}^{n} \frac{n+1-x}{\binom{n+1}{2}} \mathbb{E}[\Lambda_{n+1-x}]. \tag{5.5}$$

Observe that $1 \leq x \leq n$ which means that $n+1-x \leq n$ and thus by our induction hypothesis we can use

$$\mathbb{E}[\Lambda_{n+1-x}] \leq \log_{\frac{2}{3}}(2) + \log_{\frac{3}{2}}((n+1-x)+1).$$

If we substitute this in Equation 5.5 we obtain

$$\mathbb{E}[\Lambda_{n+1}] \leq 1 + \sum_{x=1}^{n} \frac{n+1-x}{\binom{n+1}{2}} \left(\log_{\frac{2}{3}}(2) + \log_{\frac{3}{2}}(n+2-x)\right) \tag{5.6}$$

$$= 1 + \log_{\frac{2}{3}}(2) + \sum_{x=1}^{n} \frac{n+1-x}{\binom{n+1}{2}} \log_{\frac{3}{2}}(n+2) + \log_{\frac{3}{2}}\left(1 - \frac{x}{n+2}\right) \tag{5.7}$$

$$= 1 + \log_{\frac{2}{3}}(2) + \log_{\frac{3}{2}}(n+2) + \sum_{x=1}^{n} \frac{n+1-x}{\binom{n+1}{2}} \log_{\frac{3}{2}}\left(1 - \frac{x}{n+2}\right). \tag{5.8}$$

For the upper bound to hold we need $\sum_{x=1}^{n} \frac{n+1-x}{\binom{n+1}{2}} \log_{\frac{3}{2}}\left(1 - \frac{x}{n+2}\right) \leq -1$. The proof of this inequality is in Appendix C. Observe that this is an almost tight bound:

$$\lim_{n\to\infty} \sum_{x=1}^{n} \frac{n+1-x}{\binom{n+1}{2}} \log_{\frac{3}{2}}\left(1 - \frac{x}{n+2}\right) = -\frac{1}{\log(9/4)} \approx -1.233.$$

This completes the proof of the theorem. □

In short we now know that using Strategy I, the number of probes needed to discover a line graph is lower bounded by 1 (constant), upper bounded by $n-1$ (linear) and on expectation at most logarithmic in n.

5.3 Star graph

Define a star graph on n vertices as follows. There are $n-1$ vertices, $\{v_1, \ldots, v_{n-1}\}$, with degree 1, and there is one vertex v_n with degree $n-1$. An example for $n = 6$ is given in Figure 5.2. Let Λ_n^I be

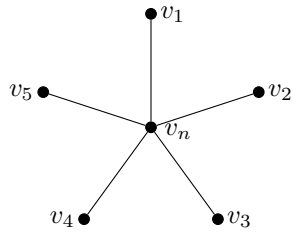


Figure 5.2: A star graph on $n = 6$ vertices.

the number of probes needed to identify a star graph on n vertices using Strategy I and let Λ_n^{II} be the number of probes needed to identify a star graph on n vertices using Strategy II. Note that this time we need at least $(n-1)/2$ probes rather than only 1 because we can discover at most 2 edges in each probe. That means we have

$$\frac{n-1}{2} \leq \Lambda_n^I \leq n-1.$$

Moreover, because Strategy II will not choose vertex v_n again (it is always discovered in the first probe), it will need at most $\lceil (n-2)/2 \rceil$ probes. That means we have

$$\frac{n-1}{2} \leq \Lambda_n^{II} \leq 1 + \left\lceil \frac{n-2}{2} \right\rceil.$$

The upper and lower bound for Λ_n^I and Λ_n^{II} are both linear. Let us look at the expected values $\mathbb{E}[\Lambda_n^{II}]$ and $\mathbb{E}[\Lambda_n^I]$.

First we consider $\mathbb{E}[\Lambda_n^{II}]$. Note that whatever the first pair (u, v) is, vertex v_n will always be discovered at the first probe. Therefore we condition on whether or not v_n is used as probing point. Let X_n be the number of edges discovered in the first probe on a star graph on n vertices. Note that $X_n \in \{1, 2\}$ is independent of the used strategy and

$$\mathbb{P}[X_n = 1] = \frac{n-1}{\binom{n}{2}} = \frac{2}{n}$$

and

$$\mathbb{P}[X_n = 2] = \frac{\binom{n}{2} - (n-1)}{\binom{n}{2}} = \frac{n-2}{n}.$$

When $X_n = 1$ we have used 2 nodes and we have not discovered a new node. That means we have $n-2$ unobserved nodes left. In the case that n is even we can form $(n-2)/2$ pairs and thus we need $(n-2)/2$ extra probes (after the first one). In the case that n is odd we have that $n-2$ is also odd and we need $(n-3)/2 + 1$ extra probes. A similar analysis for $X_n = 2$ shows that if n is even we need $(n-4)/3 + 1$ extra probes and if n is odd we need $(n-3)/2$ extra probes. Conditioning on X_n in the case n is even results in:

$$\begin{aligned}
\mathbb{E}[\Lambda_n^{II}] &= \mathbb{E}[\mathbb{E}[\Lambda_n^{II} \mid X_n]] \\
&= \mathbb{P}[X_n = 1] \cdot \left(1 + \frac{n-2}{2}\right) + \mathbb{P}[X_n = 2] \cdot \left(1 + \frac{n-4}{2} + 1\right) \\
&= \frac{2}{n} \cdot \left(1 + \frac{n-2}{2}\right) + \frac{n-2}{n} \cdot \left(1 + \frac{n-4}{2} + 1\right) \\
&= 1 + \frac{n-2}{n} + \frac{(n-2)^2}{2n} = 1 + \frac{n-2}{2}.
\end{aligned}$$

In the case that n is odd we have

$$\begin{aligned}\mathbb{E}[\Lambda_n^{II}] &= \mathbb{E}[\mathbb{E}[\Lambda_n^{II} \mid X_n]] \\ &= \mathbb{P}[X_n = 1] \cdot \left(1 + \frac{n-3}{2} + 1\right) + \mathbb{P}[X_n = 2] \cdot \left(1 + \frac{n-3}{2}\right) \\ &= \frac{2}{n} \cdot \left(1 + \frac{n-3}{2} + 1\right) + \frac{n-2}{n} \cdot \left(1 + \frac{n-3}{2}\right) \\ &= 1 + \frac{2}{n} + \frac{n-3}{2}.\end{aligned}$$

We see that we have an exact expression for $\mathbb{E}[\Lambda_n^{II}]$. It is not so easy to find an exact expression for $\mathbb{E}[\Lambda_n^{I}]$ but similar to the case of the line graph (Section 5.2) we can see that

$$\mathbb{E}\left[\Lambda_n^I \mid X_n = x\right] = 1 + \mathbb{E}\left[\Lambda_{n-x}^I\right]. \tag{5.9}$$

Using Equation 5.9 we can then see that

$$\mathbb{E}\left[\Lambda_n^I\right] = \mathbb{E}[\mathbb{E}[\Lambda_n^I \mid X_n]] \tag{5.10}$$

$$= 1 + \frac{2}{n}\mathbb{E}[\Lambda_{n-1}^I] + \frac{n-2}{n}\mathbb{E}[\Lambda_{n-2}^I]. \tag{5.11}$$

In the setting of a star graph we would expect that Strategy II performs better than Strategy I because there is a chance that Strategy I chooses the centre node v_n multiple times. Indeed Theorem 2 shows that on expectation Strategy I performs worse than Strategy II.

Theorem 2. *Let Λ_n^I be the number of probes needed to identify a star graph on n vertices using Strategy I and similarly define Λ_n^{II}. Then*

$$\mathbb{E}[\Lambda_n^I] \geq \mathbb{E}[\Lambda_n^{II}]. \tag{5.12}$$

Proof. This proof works by induction. As base case take $n = 3$ and observe that

$$\mathbb{E}[\Lambda_3^I] = 1 \cdot \frac{1}{3} + 2 \cdot \frac{2}{3} = \frac{5}{3} = \mathbb{E}[\Lambda_3^{II}].$$

For the induction hypothesis, assume that $\mathbb{E}[\Lambda_n^I] \geq \mathbb{E}[\Lambda_n^{II}]$ for all integers up to and including n. In the induction step we distinguish two cases: $n+1$ is even and $n+1$ is odd.

- $n+1$ **is even:** When $n+1$ is even we have n is odd and $n-1$ is again even. We take (5.11) and use the induction hypothesis to write

$$\begin{aligned}\mathbb{E}[\Lambda_{n+1}^I] &= 1 + \frac{2}{n+1}\mathbb{E}[\Lambda_n^I] + \frac{n-1}{n+1}\mathbb{E}[\Lambda_{n-1}^I] \\ &\geq 1 + \frac{2}{n+1}\left(1 + \frac{2}{n} + \frac{n-3}{2}\right) + \frac{n-1}{n+1}\left(1 + \frac{n-3}{2}\right) \\ &= 1 + \frac{n^3 - n + 8}{2n(n+1)} \\ &= 1 + \frac{(n+1)(n-1)}{2(n+1)} + \frac{8}{2n(n+1)} \\ &\geq 1 + \frac{n-1}{2} = \mathbb{E}[\Lambda_{n+1}^{II}].\end{aligned}$$

- $n+1$ **is odd:** When $n+1$ is odd we have n is even and $n-1$ is odd again. We take (5.11) and use

the induction hypothesis to get

$$\begin{aligned}
\mathbb{E}[\Lambda^I_{n+1}] &= 1 + \frac{2}{n+1}\mathbb{E}[\Lambda^I_n] + \frac{n-1}{n+1}\mathbb{E}[\Lambda^I_{n-1}] \\
&\geq 1 + \frac{2}{n+1}\left(1 + \frac{n-2}{2}\right) + \frac{n-1}{n+1}\left(1 + \frac{2}{n-1} + \frac{n-4}{2}\right) \\
&= 1 + 2\frac{2}{n+1} + \frac{2(n-2)}{2(n+1)} + \frac{2(n-1)}{2(n+1)} + \frac{(n-1)(n-4)}{2(n+1)} \\
&= 1 + 2\frac{2}{n+1} + \frac{n-2}{2} \\
&\geq 1 + \frac{2}{n+1} + \frac{n-2}{2} = \mathbb{E}[\Lambda^{II}_{n+1}].
\end{aligned}$$

This concludes the proof that $\mathbb{E}[\Lambda^I_n] \geq \mathbb{E}[\Lambda^{II}_n]$. □

Theorem 2 implies that for a star graph Strategy II better on average. Also, observe that $\mathbb{E}[\Lambda^{II}_n]$ is very close to $(n-1)/2$, meaning that Strategy II performs on average very close to the minimum number of probes needed.

5.4 General (tree) graphs

When we know nothing more than that the structure of the graph is a tree, we know at least when we can stop: as soon as we have discovered $n-1$ edges or, equivalently, as soon as we have connected all vertices. In Section 5.1 we have seen that the upper bound for the number of probes needed to identify a tree graph on n nodes is given by $n-1$. Concerning the lower bound; in Section 5.2 we have seen an example of a tree graph for which the minimum number of probes needed to identify the graph is 1.

As a final remark, let us divert from the assumption that the graph is a tree and suppose that we are dealing with a general (undirected) graph. When dealing with a general graph the only way to know that we have discovered the complete topology is when we have used all possible vertex pairs as probing pairs. Strictly speaking we do not need to have used all pairs because some probes may give information on vertex pairs other than the source and target vertices. For example, suppose that in some graph we get $\rho(u,v) = (u, v_1, v_2, v_3, v)$. Then we automatically know that $\rho(v_1, v_3) = (v_1, v_2, v_3)$ because ρ returns the shortest (u,v)-path. Unfortunately not all graphs will give that much information. An example of such a graph is K_n with one edge removed. The number of edges of that graph is $\binom{n}{2} - 1 = n(n-1)/2 - 1 = \mathcal{O}(n^2)$. Almost every pair of vertices will reveal only one edge and only one pair will reveal two edges. Therefore we need as many probes as the number of edges plus one[1]. Even when we know that the graph we try to map is K_n with one edge missing, we still need many probes. The chance that we pick the pair of vertices between which no edge exists is very small ($1/\binom{n-i}{2}$ in for the i^{th}probe) and thus it may take many probes before we figure out which edge is missing.

5.5 Summary

In this chapter we have focused on a more abstract form of topology discovery, applicable to any type (data, electricity, etc.) of network - represented as a graph - for which there exists a probing operation ρ that reveals the shortest path between two graph vertices. We showed that for any tree graph ρ can be used to recover the graph topology. Moreover, we presented two probing strategies and analysed their performance on two types of tree graphs: a line graph and a star graph. Strategy I uses at most a logarithmic number of probes on average to identify a line graph. In the case of a star graph, the average is linear in the number of vertices for both strategies although Strategy II performs better on average than Strategy I. The average performance of the two strategies on a general tree graph remains open.

[1] We also need to use the pair of vertices between which no edge exists to discover that one edge is missing.

Chapter 6

Disjoint Path Computation

In this chapter we reverse the problem of testing if two existing connections are disjoint. In an earlier stage of the network configuration, when the network is there but the connections have not yet been set up, it is possible to choose the connections such that they are disjoint in the first place. To do that we need the topology of the network. Coincidently we have seen two chapters dealing with recovering the topology of a network, so we assume that at this point that the network topology is known. In addition, long and extensive monitoring of a network can provide information on the reliability of the links in the network. As a result, we can assign values to the edges which represent the reliability of that edge in terms of failure probabilities. In this chapter we will present different measures for the robustness of a pair of disjoint connections (or paths). In addition, we will discuss the computational difficulty of finding the best pair of disjoint paths regarding these measures. We will also show a new relation between the optimal solutions of two different measures.

6.1 Network model

Consider a graph $G = (V, E)$ with $|V| = n$, $|E| = m$, source node $s \in V$, target node $t \in V$ and for each edge $e \in E$ a probability $p_e \in [0, 1)$ that edge e fails (these probabilities are mutually independent). Note that if $p_e = 1$ then edge e would always fail and we can just exclude it from the set of edges E. Therefore we assume $p_e < 1$. We are interested in finding two node disjoint (s, t)-paths A and B such that the resulting pair A and B is as "robust" as possible. In terms of notation, paths A and B may be represented by both an ordered list of vertices and an unordered list of edges. We propose some different measures for robustness, because depending on the interest of the network operator robustness may be defined in different ways.

When presented with a general graph, one can easily check if G contains a pair of disjoint (s, t)-paths by applying a maximum flow algorithm with source s, sink t and edge and node capacities 1. The resulting maximum flow (max-flow) then represents the amount of vertex disjoint (s, t)-paths. So, if this max-flow value is ≤ 1, then a pair of disjoint (s, t)-paths does not exist. For the remainder of this chapter we assume that the graph G contains at least one pair of disjoint (s, t)-paths.

The setting described in the Motivation implies that we are interested in finding vertex-disjoint paths. The problem of finding vertex disjoint paths is equivalent to finding edge-disjoint paths by means of splitting vertices. Starting with an undirected graph G, replace all undirected edges $\{u, v\}$ by two directed edges (u, v) and (v, u) resulting in a directed graph. Then, split every vertex v in two vertices v_1 and v_2. Replace every directed (incoming) edge (u, v) with (u, v_1) and replace every (outgoing) edge (v, u) with (v_2, u). Finally add the directed edge (v_1, v_2) and call the resulting directed graph G'. Now observe that every edge-disjoint path in G' is equivalent to a vertex-disjoint path in G and vice versa. In the remainder of this chapter we will therefore mainly focus on edge-disjoint paths.

6.1.1 Measures for robustness

We propose four measures for the robustness of the two paths A and B. These measures can be seen as optimisation problems. For the optimisation problems, denote

$$D(s, t)_G := \{\{A, B\} \mid A \text{ and } B \text{ are two vertex disjoint } (s, t)\text{-paths in } G\}.$$

Also, call a path A operational if none of its edges is failing. A path is not operational or failing if at least one of its edges is failing. In three of the four following proposed measures/optimisation problems

it is possible to translate the problem into a problem which minimizes path lengths via a logarithmic transformation.

6.1.1.1 Probability that both paths are operational

An obvious measure is the probability that both connections are up: $\mathbb{P}[A \text{ and } B \text{ operational}]$. A higher value of this probability coincides with a better choice of A and B. Therefore, the optimisation problem that belongs to this measure is: given a graph G as described above, find

$$\max_{\{A,B\} \in D(s,t)_G} \{\mathbb{P}[A \text{ and } B \text{ operational}]\}.$$

This optimisation problem can be formulated as an optimisation problem which can be solved in polynomial time. First of all, because the link failures are independent and A and B are disjoint it follows that

$$\begin{aligned}\mathbb{P}[A \text{ and } B \text{ operational}] &= \mathbb{P}[A \text{ is operational}] \cdot \mathbb{P}[B \text{ is operational}] \\ &= \prod_{e \in A}(1-p_e) \prod_{e \in B}(1-p_e).\end{aligned}$$

That means that

$$\max_{\{A,B\} \in D(s,t)_G} \{\mathbb{P}[A \text{ and } B \text{ operational}]\} = \max_{\{A,B\} \in D(s,t)_G} \left\{ \prod_{e \in A}(1-p_e) \prod_{e \in B}(1-p_e) \right\}.$$

Next we use the logarithm to formulate an equivalent optimisation problem. Maximizing over a product is equivalent to maximizing over the sum of the logarithms. Observe that $p_e \in [0,1)$, and thus $1-p_e \in (0,1]$. From this it follows that $\log(1-p_e) < 0$ and thus $-\log(1-p_e) > 0$. This results in:

$$\begin{aligned}\arg\max_{\{A,B\} \in D(s,t)_G} \left\{ \prod_{e \in A}(1-p_e) \prod_{e \in B}(1-p_e) \right\} &= \arg\max_{\{A,B\} \in D(s,t)_G} \left\{ \sum_{e \in A} \log(1-p_e) + \sum_{e \in B} \log(1-p_e) \right\} \\ &= \arg\min_{\{A,B\} \in D(s,t)_G} \left\{ \sum_{e \in A} -\log(1-p_e) + \sum_{e \in B} -\log(1-p_e) \right\}.\end{aligned}$$

The latter is equivalent to finding two vertex disjoint paths that minimize their total cost, if set the costs to be $c(e) = -\log(1-p_e)$. Suurballe and Tarjan [ST84] showed that this can be solved in polynomial time $\mathcal{O}(m \log_{(1+m/n)} n)$.

This measure is interesting for companies that use both connections simultaneously, particularly when both connections have a high load. In such case it can happen that when one connection fails the other connection does not have enough bandwidth left to redirect the traffic of the failed connection over the connection which is still operational. This would result in a loss of connection or in a congestion on the operational connection.

Another purpose for using two connections at the same time is for security reasons. Splitting and directing a data flow along two paths ensures that no one who is listening in on one connection can view the complete data flow. However, if one of the connections is down and all data traffic is routed along the other single connection there is suddenly a security threat.

6.1.1.2 Strength of the weakest path

The next measure is the "strength" of the weakest path. In this sense, a pair is weaker when it has a higher probability that one of its paths fails. In mathematical syntax the measure is described by: $\max\{\mathbb{P}[A \text{ fails}], \mathbb{P}[B \text{ fails}]\}$. Intuitively one would want the weakest path to be as strong as possible and thus the maximum to be as low as possible. That results in the optimisation problem:

$$\min_{\{A,B\} \in D(s,t)_G} \{\max\{\mathbb{P}[A \text{ not operational}], \mathbb{P}[B \text{ not operational}]\}\}.$$

Similar to the derivation we did in Section 6.1.1.1, using the logarithm we can see

$$\begin{aligned}
&\arg\min_{\{A,B\}\in D(s,t)_G} \{\max\{\mathbb{P}[A \text{ fails}], \mathbb{P}[B \text{ fails}]\}\} \\
&\qquad = \arg\min_{\{A,B\}\in D(s,t)_G} \left\{\max\left\{1-\prod_{e\in A}(1-p_e), 1-\prod_{e\in B}(1-p_e)\right\}\right\} \\
&\qquad = \arg\max_{\{A,B\}\in D(s,t)_G} \left\{\min\left\{\prod_{e\in A}(1-p_e), \prod_{e\in B}(1-p_e)\right\}\right\} \\
&\qquad = \arg\min_{\{A,B\}\in D(s,t)_G} \left\{\max\left\{\sum_{e\in A}-\log(1-p_e), \sum_{e\in B}-\log(1-p_e)\right\}\right\}.
\end{aligned}$$

This problem is then equivalent to finding $\{A, B\} \in D(s,t)_G$ such that the cost of the most expensive path is minimized, when taking the cost equal to $c(e) = -\log(1-p_e)$. In Section 6.2 we will see that Li et al [LMSL90] have shown that this problem is NP-complete. In the same paper they also propose a heuristic which is simply Suurballe and Tarjans algorithm and show that this has a worst-case ratio equal to 2, i.e. in the worst case the heuristic will output a solution which is twice as bad as an optimal solution.

This measure is very interesting in terms of redundancy. Once the primary connection fails it is important to have a reliable backup connection. A natural thing to do is to select the primary connection as the connection with the lowest probability of failing. A secondary connection would be almost useless if the probability that it fails is very high: in the case that you are already unlucky and your primary connection goes down you would not want to have a high chance of getting in a worse situation. Intuitively this measure makes sure that the two failure probabilities do not differ too much.

6.1.1.3 Probability of at least one operational path

The third proposed measure comes down to the probability that at least one path is operational ($\mathbb{P}[A \text{ or } B \text{ operational}]$) or, equivalently, the probability that not both paths fail simultaneously: $1 - \mathbb{P}[A \text{ and } B \text{ fail}]$. Intuitively, the probability that both paths fail should be as small as possible, which gives rise to the following optimisation problem:

$$\max_{\{A,B\}\in D(s,t)_G} \{\mathbb{P}[A \text{ or } B \text{ operational}]\}.$$

In trying to find an equivalent form; we derived:

$$\begin{aligned}
\arg\max_{\{A,B\}\in D(s,t)_G} \{\mathbb{P}[A \text{ or } B \text{ operational}]\} &= \arg\max_{\{A,B\}\in D(s,t)} \{1-\mathbb{P}[A \text{ and } B \text{ fail}]\} \\
&= \arg\min_{\{A,B\}\in D(s,t)_G} \{\mathbb{P}[A \text{ and } B \text{ fail}]\} \\
&= \arg\min_{\{A,B\}\in D(s,t)_G} \{\mathbb{P}[A \text{ fails}]\cdot\mathbb{P}[B \text{ fails}]\} \\
&= \arg\min_{\{A,B\}\in D(s,t)_G} \left\{\left(1-\prod_{e\in A}(1-p_e)\right)\left(1-\prod_{e\in B}(1-p_e)\right)\right\}.
\end{aligned}$$

When we expand the last line we obtain

$$\arg\min_{\{A,B\}\in D(s,t)_G} \left\{\left(\prod_{e\in A}(1-p_e)\right)\left(\prod_{e\in B}(1-p_e)\right) - \left(\prod_{e\in A}(1-p_e)+\prod_{e\in B}(1-p_e)\right)\right\}.$$

This measure is very interesting in terms of connectivity connectivity. It maximizes the probability that at least one of the connections works although it does not give any information on what the situation is when a link breaks down. This measure might very well choose a very robust connection and a very fragile connection. That would mean that if the primary connection (chosen as the most robust one) breaks down then almost surely there is no connection at all. That would of course not do well in terms of redundancy.

6.1.1.4 Maximum operational probability

This measure is the maximum of the probabilities: path A being operational, path B being operational. In mathematical syntax:

$$\max_{\{A,B\}\epsilon D(s,t)_G} \{\max\{\mathbb{P}[A \text{ operational}], \mathbb{P}[B \text{ operational}]\}\}.$$

Again by using the logarithm we can see that this problem is equivalent to

$$\min_{\{A,B\}\epsilon D(s,t)_G} \left\{\min\left\{\sum_{e\epsilon A} -\log(1-p_e), \sum_{e\epsilon B} -\log(1-p_e)\right\}\right\}.$$

This measure comes down to optimizing the primary connection such that it is as reliable as possible. It is particularly interesting when a party heavily depends on the primary path and is reluctant to switch to the secondary path, e.g. when the secondary path has very low bandwidth or high operation costs.

6.2 Complexity of disjoint path problems

We chose to further investigate the problem of finding two connections such that the weakest is as strong as possible. Call this objective the Min-Max objective. This seems the most interesting problem in terms of redundancy. In this section we cover the computational difficulty of finding such a pair of connections. For the remainder of this chapter we will talk about paths rather than connections since we will also switch to the notion of graphs.

Li et al. [LMSL90] have shown that the problem of finding two disjoint paths such that the length of the longer path is minimized is strongly NP-complete. The proof is based on a reduction from the Maximum 2-Satisfiability Problem [GJ79]. An instance of the Maximum 2-Satisfiability Problem consists of variables $z_1, \ldots, z_r$, giving rise to literals z_i and $\bar{z}_i$, and clauses $C_1, \ldots, C_q$: each a subset of 2 literals. We say that C_j is satisfied under a truth assignment $\tau : z_i \to \{\text{true}, \text{false}\}$ if C_j contains a literal z_i with $\tau(z_i) = \text{true}$ or $\tau(\bar{z}_i) = \text{true}$. The problem is; given a bound $K \leq q$, is there a truth assignment τ which satisfies $\geq K$ clauses?

The reduction presented in [LMSL90] starts with an instance of the Maximum 2-Satisfiability Problem. Then, based on the literals z_i and clauses C_j from the instance, Li et al construct a graph G with specific edges and edge weights. In particular G contains also two special vertices s and t. They show that a pair of disjoint (s,t)-paths in G with a Min-Max value of $\tilde{K}$ coincides with a truth assignment τ which satisfies at least K clauses. The value of $\tilde{K}$ is based on the structure of the graph. Since the construction of the graph can be done in polynomial time there is a polynomial time reduction from Maximum 2-Satisfiability to finding a pair of disjoint paths with a Min-Max objective function.

Since the problem of finding two disjoint paths with a Min-Max objective function is NP-complete one would typically fall back to using heuristics. Li et al proposed to use the Min-Sum solution as an approximation of the Min-Max problem and show that this has a worst-case bound equal to 2. Again, let P_1^* and P_2^* represent an optimal solution to the Min-Sum problem and let Q_1^* and Q_2^* represent an optimal solution to the Min-Max problem. Then

$$c(P_1^*) + c(P_2^*) \leq c(Q_1^*) + c(Q_2^*).$$

From this it follows that

$$\max_{i=1,2}\{c(P_i^*)\} \leq c(Q_1^*) + c(Q_2^*) \leq 2\max_{i=1,2}\{c(Q_i^*)\}$$

and thus

$$\frac{\max_{i=1,2}\{c(P_i^*)\}}{\max_{i=1,2}\{c(Q_i^*)\}} \leq 2.$$

Proving tightness is done using Figure 6.1. An optimal solution to the Min-Max problem on the graph of Figure 6.1 has $\max_{i=1,2}\{c(Q_i^*)\} = a_1 + a_2$. The pair of paths $P_1^* = (s,1,t)$ and $P_2^* = (s,2,1,3,t)$ is an optimal solution for the Min-Sum problem with $c(P_1^*) = 0$ and $c(P_2) = 2(a_1 + a_2)$. Hence $\max_{i=1,2}\{c(P_i^*)\}/\max_{i=1,2}\{c(Q_i^*)\} = 2$.

Moreover, Li et al proved that, for any directed graph, there exists no algorithm which has a better worst-case behaviour unless $P = NP$:

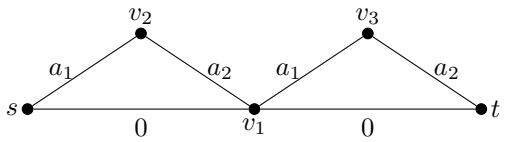


Figure 6.1: A graph which attains the worst-case bound of 2 for the Min-Max problem using the Min-Sum solution.

Theorem 3. *([LMSL90, Theorem 2]) If $P \neq NP$, then any psuedo-polynomial-time heuristic for finding two Min-Max disjoint $s \to t$ paths in a directed network has a worst-case bound of at least 2.*

The proof of this theorem relies on a result of Fortune, Hopcroft and Wyllie [FHW80]. Given any directed graph $G = (V, E)$ and four vertices $s_1, s_2, t_1, t_2 \in V$, the problem of finding two vertex-disjoint (or arc-disjoint) paths, one from s_1 to t_1 and another from s_2 to t_2 is NP-complete. We will refer to this problem as the 2DDP (2 Directed Disjoint Paths) problem.

Proof of Theorem 3. (Li et al, [LMSL90]) Suppose we have an instance of 2DDP with $G = (V, E)$ and the four vertices $s_1, s_2, t_1, t_2 \in V$. Create two new vertices s and t and define a new directed graph $\hat{G} = (\hat{V}, \hat{E})$ by:

$$\hat{V} = V \cup \{s, t\},$$
$$\hat{E} = E \cup \{(s \to s_1), (s \to s_2), (t_1 \to t), (t_2 \to t)\}.$$

The arcs $(s \to s_1)$ and $(t_2 \to t)$ get cost 1 and all other arcs get cost 0. Then the most expensive path among any pair of disjoint (s, t)-paths in $\hat{G}$ has cost 1 or 2. Furthermore, if there exists a pair of disjoint (s, t)-paths in $\hat{G}$, the most expensive path has cost 1 if and only if there exists two disjoint paths in G connecting s_1 with t_1 and s_2 with t_2.

In other words, if there is a pseudo-polynomial-time heuristic for finding two Min-Max disjoint (s, t)-paths with a worst-case bound less than 2, then this heuristic would solve the 2DDP problem in G. This would contradict the assumption that $P \neq NP$. □

6.3 Different solutions with different measures of robustness

In the previous section we have seen a few different measures. An interesting question is in what type of graphs, networks and situations do these measures result in the same solution? To investigate this question, let us consider the first two measures: "the probability that both paths are operational" and "the strength of the weakest path". We have seen that using the logarithm we can translate these two measures to:

- $L_\Sigma(A, B)$: the total costs of the two paths A and B,
- $L_{\max}(A, B)$: the cost of the most expensive path, i.e. $\max\{A, B\}$.

Suppose we are given a network $G = (V, E)$ with a source node s and a target node t and for every edge $e \in E$ a cost $c(e) \geq 0$. Also, if A is a path, let $c(A) = \sum_{e \in A} c(e)$ denote the cost of a path A. Let the two paths P_1^* and P_2^* be paths which minimize L_Σ, and the two paths Q_1^* and Q_2^* be the paths which minimize $L_{\max}$. In other words:

$$\{P_1^*, P_2^*\} = \operatorname*{arg\,min}_{\{A,B\} \in D(s,t)_G} \{L_\Sigma(A, B)\} = \operatorname*{arg\,min}_{\{A,B\} \in D(s,t)_G} \{c(A) + c(B)\} \tag{6.1}$$

$$\{Q_1^*, Q_2^*\} = \operatorname*{arg\,min}_{\{A,B\} \in D(s,t)_G} \{L_{\max}(A, B)\} = \operatorname*{arg\,min}_{\{A,B\} \in D(s,t)_G} \{\max\{c(A), c(B)\}\} \tag{6.2}$$

Without loss of generality we may assume $c(P_1^*) \leq c(P_2^*)$ and $c(Q_1^*) \leq c(Q_2^*)$. Using this we can prove the following claim.

Claim 1. *$c(P_1^*) \leq c(Q_1^*)$ and $c(Q_2^*) \leq c(P_2^*)$.*

Proof. First we show $c(Q_2^*) \leq c(P_2^*)$ by contradiction. Suppose $c(Q_2^*) > c(P_2^*)$. Then

$$\max\{c(Q_1^*), c(Q_2^*)\} = c(Q_2^*) > c(P_2^*) = \max\{c(P_1^*), c(P_2^*)\}.$$

That means that choosing $\{P_1^*, P_2^*\}$ leads to a better (lower) value for L_2 which contradicts the minimality of $\{Q_1^*, Q_2^*\}$. Note that we can do this because P_1^* and P_2^* are also disjoint paths. Therefore $c(Q_2^*) \leq c(P_2^*)$.

Secondly we show $c(P_1^*) \leq c(Q_1^*)$, also by contradiction. Suppose $c(Q_1^*) < c(P_1^*)$. Using $c(Q_2^*) \leq c(P_2^*)$ we can then derive $c(Q_1^*) + c(Q_2^*) \leq c(Q_1^*) + c(P_2^*) < c(P_1^*) + c(P_2^*)$. That contradicts $\{P_1^*, P_2^*\}$ being an optimal solution to (6.1) and thus we can conclude $c(P_1^*) \leq c(Q_1^*)$. □

From Claim 1 and the assumptions $c(P_1^*) \leq c(P_2^*)$ and $c(Q_1^*) \leq c(Q_2^*)$ it follows trivially that

$$c(P_1^*) \leq c(Q_1^*) \leq c(Q_2^*) \leq c(P_2^*).$$

Observe that if $c(Q_2^*) = c(P_2^*)$ then $\max\{c(Q_1^*), c(Q_2^*)\} = c(Q_2^*) = c(P_2^*) = \max\{c(P_1^*), c(P_2^*)\}$ and then $\{P_1^*, P_2^*\}$ is also an optimal solution for (6.2).

Claim 2. *Every (s,t)-path R which has $c(R) < c(P_2^*)$ also satisfies $R \cap P_1^* \neq \emptyset$, i.e. R intersects P_1^*.*

Proof. This proof is by contradiction. For that purpose, suppose $c(R) < c(P_2^*)$ and that R and P_1^* are disjoint. Then, the pair $\{R, P_2^*\}$ is a pair of disjoint (s,t)-paths and $c(P_1^*) + c(R) < c(P_1^*) + l(P_2^*)$ which contradicts the optimality of $\{P_1^*, P_2^*\}$ under L_Σ. Therefore path R can not be disjoint from path P_1^*. □

6.3.1 A special type of graph

Due to its significance in Telecom operator networks we will consider the following type of graph, a so called dual-homed fully-meshed network. This type of network consists of $K_{n\geq 4}$, the complete graph on n vertices with $n \geq 4$ and two more vertices, s and t. Moreover, s is connected to only two nodes of K_n: s_1 and s_2 such that $s_1 \neq s_2$. Also, t is connected to only two nodes of K_n, t_1 and t_2 with $t_1 \neq t_2$. Lastly, all four nodes that connect to s and t are different, thus for $i, j \in \{1, 2\}$ we have $s_i \neq t_j$ (which also motivates why n should be bigger than or equal to 4). Denote such a graph with $FM(n)$, $n \geq 4$. Figure 6.2 shows $FM(4)$.

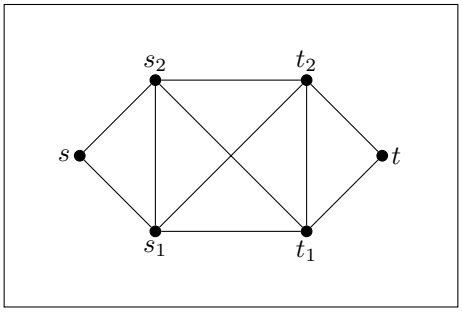


Figure 6.2: Example of dual-homed fully-meshed network with $n = 4$ ($FM(4)$).

We are interested in this graph because typically a Telecom providers core network is a fully meshed network. Then, when a party wishes to connect two sites by a pair of disjoint connections; these two sites together with the core network form a dual-homed fully-meshed network.

First consider a setting in which all the weights (representing the failure probabilities) of the edges are equal; for example 1 (after scaling them). Then it is obvious that the pair of paths

$$((s, s_1, t_1, t), (s, s_2, t_2, t))$$

is an optimal solution for both the Min-Sum and the Min-Max problem.

Next, consider a setting in which the weights satisfy for every three nodes $a, b, c \in V(FM(n))$: $c(a,b) + c(b,c) \geq c(a,c)$. In other words, choosing paths with less number of links may only improve the Min-Sum and Min-Max. That means that the only two pairs of paths that we need to consider are the pairs

$$\{(s, s_1, t_1, t), (s, s_2, t_2, t)\} \text{ and } \{(s, s_1, t_2, t), (s, s_2, t_1, t)\},$$

since they will be at least as good as any other pair. Checking these two pairs for optimality for Min-Sum and Min-Max is then fairly easy.

The real challenge is when the weights do not have the structure as in the previous paragraph. When we assume the weights to be random then there is no guaranteed structure to exploit and we are back to square one. If we want to find the optimal solution for the Min-Max problem then the best thing to do is to try all possible pairs of disjoint (s,t)-paths. The number of pairs of vertex-disjoint (s,t)-paths in $FM(n)$ quickly grows out of hands as we can see in Lemma 3.

Lemma 3. *The number of different pairs (A,B) of disjoint (s,t)-paths in $FM(n)$ is*

$$\sum_{i=0}^{n-4} \sum_{j=0}^{n-4-i} 2 \frac{(n-4)!}{(n-4-i-j)!} = \sum_{i=0}^{n-4} \sum_{j=0}^{n-4-i} 2 \prod_{k=n-4-i-j+1}^{n-4} k.$$

Proof. We will count the pairs as follows: first we will count pairs of paths with a specific number of edges and then we will sum over the possible number of edges. Let

$$D(s,t)_{(G,i,j)} = \{(A,B) \in D(s,t)_G \mid l(A) - 3 = i, l(B) - 3 = j\},\ i,j \geq 0,\ i + j \leq n - 4,$$

with $l(A)$ the number of edges of path A. Intuitively $D(s,t)_{(G,i,j)}$ consists of all pairs (A,B) of vertex-disjoint (s,t)-paths where A consists of $i+3$ links and B consists of $j+3$ links. The terms $i+3$ and $j+3$ follow from the fact than any (s,t)-path in $FM(n)$ consists of at least 3 edges. In other words, for A we have chosen i extra vertices, and for B we have chosen j extra vertices (from $V(FM(n))\backslash\{s, s_1, s_2, t, t_1, t_2\}$). Recall that $FM(n)$ has $n+2$ vertices, of which s, t, s_1, s_2, t_1 and t_2 are always used. That leaves $n + 2 - 6 = n - 4$ free vertices which can also be used, which explains why $i + j \leq n - 4$.

As an example: consider paths with 3 edges. There are 4 such paths:

1. (s, s_1, t_1, t),
2. (s, s_1, t_2, t),
3. (s, s_2, t_1, t), and
4. (s, s_2, t_2, t).

These four paths can form two pairs of vertex-disjoint paths:

1. $\{(s, s_1, t_1, t), (s, s_2, t_2, t)\}$, and
2. $\{(s, s_2, t_1, t), (s, s_1, t_2, t)\}$.

Choosing a path A with $l(A) = 3$ means we can choose none of the remaining free $n-4$ vertices. That means $|D(s,t)_{(FM(n),0,0)}| = 2$ as we have just seen.

We will determine the value of $|D(s,t)_{(FM(n),i,j)}|$. A (s,t)-path in $FM(n)$ either passes through s_1 or through s_2. It also passes through either t_1 or t_2. Choosing through which vertices (out of s_1, s_2, t_1, t_2) A passes determines through which vertices B will pass. That means that for counting the pairs we only need to consider $\{(s_1 \to t_1), (s_2 \to t_2)\}$ and $\{(s_1 \to t_2), (s_2 \to t_1)\}$. In other words, we may assume A passes through s_1 and B passes through s_2 (if A passes through s_2 we rename the paths).

Definition 1. *For the sake of writing, define $\mathcal{P}_{a_1,a_2,\ldots,a_k}(i)$ to be the number of (a_1,a_k)-paths via at least $a_2,\ldots,a_{k-1}$ (not necessarily in this order) in $FM(n)$ consisting of $i \geq k-1$ edges. Formally, for $a_1,a_2,\ldots,a_k \in V(FM(n))$, $i \in \{0,\ldots,n-4\}$:*

$$\mathcal{P}_{a_1,a_2,\ldots,a_k}(i) = |\{(a_1,a_k)\text{-paths } P \text{ in } FM(n) \mid P \text{ goes through all of } \{a_2,\ldots,a_{k-1}\}, l(P) = i\}|$$

$\mathcal{P}_{a_1,a_2,\ldots,a_k}(i)$ counts paths which do not go via s or t. The special vertices s and t can only appear as end point in these paths. In addition, $\mathcal{P}_{a_1,a_2,\ldots,a_k}(i)$ only counts paths which do not go through both s_1 and s_2 (so it does count paths which go through only one of s_1 or s_2). The same holds for t_1 and t_2.

The number of paths from s to t via s_1 is equal to the number of paths from s_1 to t by construction of $FM(n)$. Equivalently, the number of paths from s to t via t_1 is equal to the number of paths from s to t_1. From this it follows that the number of (s,t)-paths via s_1 and t_1 having $i+3$ edges is equal to the number of (s_1,t_1)-paths containing $i+1$ edges. Thus $\mathcal{P}_{s,s_1,t_1,t}(i+3) = \mathcal{P}_{s_1,t_1}(i+1)$. Then the number of (s_1,t_1)-paths A having $i+3$ edges equals $\mathcal{P}_{s_1,t_1}(i+1)+\mathcal{P}_{s_1,t_2}(i+1)$. Note that $\mathcal{P}_{s_1,t_1}(i+1) = \mathcal{P}_{s_1,t_2}(i+1)$ by construction of $FM(n)$. That means that the number of paths A consisting of $i+3$ edges is $2\mathcal{P}_{s_1,t_1}(i+1)$.

Fortunately we can compute $\mathcal{P}_{s_1,t_1}(i+1)$. For a path from s_1 to t_1 to have $i+1$ edges we must choose i extra vertices out of $n-4$ available vertices. This can be done in $\binom{n-4}{i}$ ways. Also, these vertices can be ordered in $i!$ different ways. So, $\mathcal{P}_{s_1,t_1}(i+1) = \binom{n-4}{i}i!$. Given a path A, how many choices for path B have we left? Note that for path A we used i vertices, so for path B we can only choose from $n-4-i$ vertices. Thus, given A, we have $\binom{n-4-i}{j}j!$ paths left for B. This shows that

$$D(s,t)_{(FM(n),i,j)} = 2\binom{n-4}{i}i!\binom{n-4-i}{j}j! = 2\frac{(n-4)!}{(n-4-i-j)!}.$$

Here, the factor 2 comes from the fact that the paths A and B either connect s_1 with t_1 or s_1 with t_2. Summing over all possible combinations of i and j such that $i+j \leq n-4$ and $i,j \geq 0$ gives us the desired result:

$$\sum_{i=0}^{n-4}\sum_{j=0}^{n-4-i} D(s,t)_{i,j} = \sum_{i=0}^{n-4}\sum_{j=0}^{n-4-i} 2\frac{(n-4)!}{(n-4-i-j)!} = \sum_{i=0}^{n-4}\sum_{j=0}^{n-4-i} 2\prod_{k=n-4-i-j+1}^{n-4} k.$$

□

Lemma 3 shows that the number of pairs of vertex-disjoint (s,t)-paths grows rapidly with n. The values of the number of pairs is shown in Table 6.1 for some values of n. Suppose we have a computer

n	# pairs
4	2
5	6
6	22
10	$\approx 23 \cdot 10^3$
15	$\approx 23 \cdot 10^8$
20	$\approx 18 \cdot 10^{14}$
25	$\approx 58 \cdot 10^{20}$
50	$\approx 13 \cdot 10^{59}$
60	$\approx 21 \cdot 10^{77}$
100	$\approx 51 \cdot 10^{151}$

Table 6.1: The number of pairs of vertex-disjoint (s,t)-paths in $FM(n)$ for some values of n.

which can evaluate a billion (10^9) pairs of paths per second. For $n = 50$, for example, it would take $13 * 10^{59}/10^9/60/60/24/365 \approx 3 * 10^{42}$ years to evaluate all pairs. As a reference, $3 * 10^{42}$ is approximately $7 * 10^{32}$ times the age of the earth (estimated at $4.54 * 10^9$ years). Since the Min-Max problem is NP-complete we know that in order to find a guaranteed optimal solution we should evaluate all pairs (assuming $P \neq NP$). The above calculations show that this is impossible within a lifetime for $FM(n \geq 50)$. Therefore we wish to apply the approximation algorithm (choosing (P_1^*, P_2^*)) and do a post-process algorithm which is described in Section 6.4.

6.4 Running the post-process algorithm

From Claim 2 it follows that when $\{P_1^*, P_2^*\}$ is not an optimal solution for (6.2) and thus $c(Q_1^*) \leq c(Q_2^*) < c(P_2^*)$, then both Q_1^* and Q_2^* have a non-empty intersection with P_1^*. This motivates the following idea for a new algorithm:

1. Select the first edge e_1 on the path P_1^* and remove it from G, obtaining the graph denoted by $G - e_1$.
2. In $G - e_1$, find the shortest (s,t)-path R.
3. If $c(P_1^*) < c(R) < c(P_2^*)$, take G again and remove the path R from G (denoted $G - R$). If $c(R) \geq c(P_2^*)$ or $c(R) \leq c(P_1^*)$, move to step 5.
4. In $G - R$, find a shortest (s,t)-path R' (if it exists, otherwise move to step 5). If $c(P_1^*) < c(R') < c(P_2^*)$, then $\{R, R'\}$ is a better option for (6.2) than $\{P_1^*, P_2^*\}$ and the algorithm terminates with outputting $\{R, R'\}$. Otherwise, move to step 5.
5. Suppose we write the vertex representation of P_1^* as $P_1^* = (s, v_1, \ldots, v_k, t)$. Then in the previous step we deleted edge $e_{i-1} = \{v_{i-1}, v_i\}$ (where $v_0 = s$ and $v_{k+1} = t$). Now we wish to look for a path which starts at s, then goes to v_i via P_1^* and then leaves the path P_1^*. Thus remove $e_i = \{v_i, v_{i+1}\} \in E$ as well and look for a shortest (v_i, t)-path R_i. Then the new path becomes $R = P_1^*(s, v_i) \cup R_i$ and repeat steps 3 and 4 and see if we obtained a better solution.
6. If in the end no better solution than $\{P_1^*, P_2^*\}$ has been found the algorithm outputs $\{P_1^*, P_2^*\}$ as solution.

This algorithm tries to use the results of Claim 2, which imply that if $\{P_1^*, P_2^*\}$ is not an optimal solution for (6.2) then Q_1^* intersects P_1^*. Basically, after determining $\{P_1^*, P_2^*\}$ (can be done in polynomial time, see [ST84]) it runs at most $2k$ times the Dijkstra's shortest path algorithm, where k is the number of nodes on path P_1^* excluding s and t (thus $k \leq n$). These steps run in polynomial time and therefore this algorithm runs in polynomial time. So unless $P = NP$, this algorithm cannot always find an optimal solution to (6.2). For example, Figure 6.3 is an instance of a graph for which the Post-Process algorithm does not find an improvement. The optimal value for the Min-Sum is 12: $c(P_1^*) + c(P_2^*) = 2 + 10 = 12$.

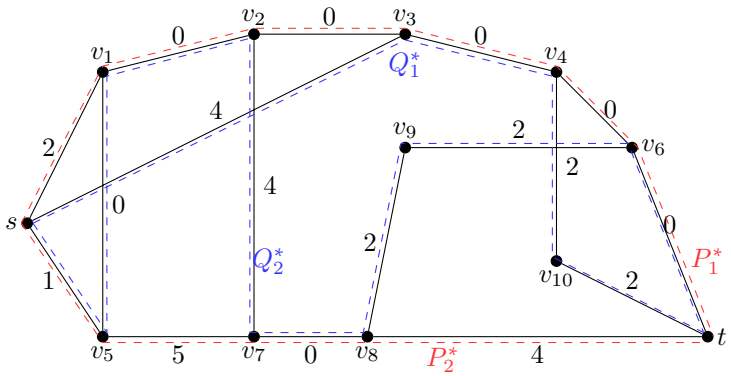


Figure 6.3: Instance for which the Post-Process algorithm does not find an improvement.

The optimal value of the Min-Max is 9: $\max\{c(Q_1^*), c(Q_2^*)\} = \max\{8, 9\} = 9$. The paths are:

$$P_1^* = (s, v_1, v_2, v_3, v_4, v_6, t),$$
$$P_2^* = (s, v_5, v_7, v_8, t),$$
$$Q_1^* = (s, v_3, v_4, v_{10}, t),$$
$$Q_2^* = (s, v_5, v_1, v_2, v_7, v_8, v_9, v_6, t).$$

The algorithm was implemented in Sage [Sag14] as shown in Appendix D. The algorithm was then tested on graphs of type $FM(n)$. The failing probabilities of the edges (s, s_1), (s, s_2), (t_1, t) and (t_2, t) are all taken to be equal to 0.0001 resulting in a weigth of $-\log(0.0001) \approx 9.2$. The failing probabilities of the other edges are taken uniformly at random from $[0, 0.4]$ since failure probabilities of real network links are typically small. However, it should not really matter since it is a matter of scaling. The algorithm was repeated many times for different n. The amount of times it is repeated is given by the number of simulations and the number of improvements is shown in Table 6.2.

n	time in seconds	# improvements
1	11.44	102
2	14.11	112
5	27.94	100
10	72.04	120
15	158.23	128
20	288.81	114
25	476.44	127
50	3615.37	148

(a) 1.000 simulations

n	time in seconds	# improvements
1	104.14	906
2	136.78	1016
5	302.84	1078
10	747.52	1254
15	1564.86	1264
20	2923.75	1346
25	4838.56	1413
50	88765.59	1548

(b) 10.000 simulations

Table 6.2: Results of running the post-process algorithms on $FM(n)$ with random edge weights.

The results in Table 6.2 suggest that in around 10-12% the post-process algorithm finds an improvement in $FM(n)$. In particular, the Post-Process Algorithm's chance of finding an improvement seems to increase as n grows. This can be explained by the fact that the chance that there exist pairs (A, B) which have a better value for the Min-Max than (P_1^*, P_2^*) also increases when n grows. Consequently the algorithm also has a higher chance of finding a better solution.

6.5 Summary

Based on a network in which each edge has a failure probability, we proposed four different measures for the optimality of a pair of disjoint paths. Two of these measures are particularly interesting in relation to the robustness of a backup connection: the *probability that both paths are operational* and the *strength of the weakest path*. Using the logarithmic function we transformed these measures into respectively a Min-Sum and a Min-Max measure. Finding a pair of disjoint paths with a Min-Sum objective can be done in polynomial time using Suurballe and Tarjans algorithm [ST84]. Finding a pair of disjoint paths with a Min-Max objective is an NP-complete problem and Li et al. [LMSL90] proved that using a Min-Sum solution is a 2-approximation to finding a Min-Max solution. We presented a new relation between the two solutions. Using this relation; we made a new algorithm which runs in polynomial time and performs a search for a better pair of disjoint paths than the Min-Sum solution. For a specific type of graph, interesting for ISPs, we have shown that our algorithm finds improvements in approximately 10-12% of the cases.

Chapter 7

Conclusions

Considering the first problem described in the problem description, we can conclude that it is theoretically possible to recover the network topology using CFM. Depending on the presence of non-responsive devices the resulting topology may be incomplete. Such incompleteness can result in a false sense of robustness when two connections run through the same non-responsive switch. Also, in a real ISP network there are some limitations that prevent the use of the algorithms proposed in Section 4.3. To overcome these limitations and ensure that all switches respond to the Linktrace protocol we propose to offer a measurement service along the same path as the connections we wish to test on disjointness. Such a service is technically possible according to experts from an ISP and an external party can issue its own measurements. The measurements can then be used to check whether two connections run through the same devices.

Even though we have descended from the IP layer to the Data Link layer and concluded that we can (presently only in theory) recover the network topology, we can not ensure complete disjointness. The Data Link layer is on top of the Physical layer and the connections that we wish to test for disjointness can still share Physical layer devices while being disjoint on the Data Link layer. Even then we have not yet considered geographical disjointness. Two fibre cables laying in the same trench but routed through different devices still poses a threat when both cables are cut simultaneously. To gain a truthful and trustworthy answer on the question if two connections are disjoint, more research is needed to find and evaluate topology discovery methods on the Physical layer.

Concerning the second problem that we considered: we concluded that in a tree graph it is possible to recover the graph structure using the probing operation ρ with at most $n-1$ probes. We have more thoroughly analysed the performance of two probing strategies on a line graph and a star graph. We discovered that on expectation Strategy I (Shrinking Random Pairs) needs at most a logarithmic number of probes. This is a good improvement compared to the linear upper bound. For a star graph we have computed the exact expected number of probes needed when using Strategy II (Unused Nodes First) and showed that for this graph type it performs better than Strategy I. The expected number of required probes for discovering the topology of general tree graphs and also for general graphs remains unknown, yet.

Finally we showed that constructing a best pair of disjoint connections with failure probabilities is equivalent to well studied problems such as the 2 Disjoint Path Problem with a Min-Sum or Min-Max objective. The problem of finding a pair of disjoint paths with a Min-Sum objective can be solved in polynomial time using Suurballe and Tarjans algorithm [ST84]. The problem of finding a pair of disjoint paths with a Min-Max objective is NP-complete. We compared the optimal solutions to these two problems and have shown a property which links the two optimal solutions. Using this property we suggested an algorithm, running in polynomial time, that uses the solution of the Min-Sum (polynomial time solvable) to do a smart search for a better approximation to the Min-Max problem. This algorithm still has the same worst-case ratio as when using the Min-Sum solution directly, although simulations on dual-homed fully-meshed network show that our new algorithm finds improvements in approximately 10-12% of the cases. It remains to show how it performs on general networks and how it performs on expectation.

Appendices

Appendix A

Connectivity Fault Management information requests

A.1 Remote MEP Parameters

Table A.1 explains the parameters of the remote MEP overview showed in Section 3.1, Table 3.1. The Remote MEP overview was obtained from switch S_1 in the Alfa set-up.

Parameter	Description	Values
Mepid	The MEP ID of the remote MEPs.	Integer between 1-8191.
Mac Address	The MAC address of the remote MEP.	A MAC address.
State	Indicates whether the MEP is enabled.	Enabled (en) or disabled (dis).
Total Rx CCM	Specifies the total number of CCM messages received.	Integer
Seq Error	Specifies the number of CCM sequence errors.	Integer
Last Seq Num	Provides the last sequence number.	Integer
Fault	Remote MEP fault indicators.	All defects currently in effect for a Remote MEP are designated with an "X".

Table A.1: Description of Remote MEP parameters.

A.2 Information of a Single Remote MEP

Table A.2 shows the output of remote MEP information of a single remote MEP. In this case, the remote MEP was the MEP wit MEP ID 2 in the Alfa set-up. The information was obtained from switch S_1.

CFM REMOTE MEP INFO	
Parameter	Value
Service	alfa
MEPID	2
Mac Address	AA:BB:CC:DD:EE:03
Service Network	1
MD Level	2
Admin State	enabled
Operational State	enabled
Hold State	enable
Time since last state change (ms)	156490482
CCM Sequence Errors	0
Total CCM Lost	0
Total Rx CCM	1006
CCM Failure Defect	No
Port Status Defect	Up
Interface Status Defect	Up
RDI Error Defect	No
DMM Tx Count	0
DMR Rx Count	0
DMM Min Delay (us)	0
DMM Ave Delay (us)	0
DMM Max Delay (us)	0
DMM Ave Jitter (us)	0
LMM Tx Count	0
LMR Rx Count	0
LMM Frame Loss Near	0
LMM Frame Loss Far	0
LMR Bad Sequence Number	0
CCM Loss Statistics	Disabled
CCM Loss Statistics Wrapped	No
CCM Current Loss Statistics Index	0
CCM Loss Bucket Size (minutes)	15

Table A.2: Output of remote MEP information request for a given, single, MEP ID.

A.3 Linktrace Reply Parameters

Parameter	Description	Values
Ttl	Time To Live	Integer.
Ttl Idx	Time To Live Index.	Integer.
Relay Action	Indicates how the LTM was handled by the MEP or MIP.	Not relevant.
Flags	Reports two flags in the Linktrace responses.	Not relevant.
Ingress Port	Name of the port on which the LTM was received.	Port number or aggregation name.
Ingress Action	Reports how a data frame to the LTM target MAC would be handled at the ingress port.	Not relevant.
Egress Port	Name of the port that is the target of the LTM, or the port that a data frame to the LTM target MAC would be transmitted out when forwarded.	Port number or aggregation name.
Egress Action	Reports how a frame to the LTM target MAC would be handled as it passes through the egress port.	Not relevant.

Table A.3: Description of Linktrace Reply Parameters.

Appendix B

Topology discovery algorithm using LLDP

As described in Section 4.1, there is a straightforward method to reveal the topology of the network using the Link Layer Discovery Protocol: crawl through the network, node by node, and at every node ask for all its neighbours. Pseudo code for this algorithm is shown in Algorithm 3. Because it is only possible to ask for neighbours of switches that the algorithm has access to, it is assumed that if the algorithm has no access to a switch S then $Neighbours(S) = \varnothing$.

Note that the algorithm does not check if a switch S is already in the node set V but just sets $V \leftarrow V \cup \{S\}$. This is a valid and proper operation since in set theory for an element S and a set $\mathcal{S}$ of switches:

$$S \in \mathcal{S} \iff \mathcal{S} = \mathcal{S} \cup \{S\},$$

or in other words; adding an element to set while that element is already in that set does nothing.

```
input  : Starting point/switch S_0
output : A graph representing the network
initialize: Set S = {S_0}, X = ∅, V = {S_0}, E = ∅, G = (V, E).
while S ≠ ∅ do
    Pick S ∈ S;
    if S ∉ X then
        foreach v ∈ Neighbours(S) do
            V ← V ∪ {S};
            E ← E ∪ {(S, v)};
            if v ∉ X then
                S ← S ∪ {v};
            end
        end
        S ← S \ {S};
        X ← X ∪ {S};
    else
        S ← S \ {S};
    end
end
return G = (V, E);
```

Algorithm 3: Topology discovery with LLDP

Also, for this algorithm the following assumptions are needed:

1. Can connect to all switches and query it for the LLDP neighbours,
2. LLDP must be configured and enabled on all switches (IEEE 802.1AB-2009 standard).

Appendix C

Proof of inequality

Theorem 4. *Let $n \geq 3$. Then*

$$\sum_{x=1}^{n} \frac{n+1-x}{\binom{n+1}{2}} \log_{\frac{3}{2}} \left(1 - \frac{x}{n+2}\right) \leq -1. \tag{C.1}$$

Proof. We introduce a new variable

$$y = n + 1 - x.$$

Using y we can write

$$\sum_{x=1}^{n} \frac{n+1-x}{\binom{n+1}{2}} \log_{\frac{3}{2}} \left(1 - \frac{x}{n+2}\right) = \frac{1}{\binom{n+1}{2}} \sum_{y=1}^{n} y \log_{\frac{3}{2}} \left(\frac{y+1}{n+2}\right) \tag{C.2}$$

$$= \frac{1}{\binom{n+1}{2}} \sum_{y=1}^{n} y \left(\log_{\frac{3}{2}}(y+1) - \log_{\frac{3}{2}}(n+2)\right) \tag{C.3}$$

$$= -\log_{\frac{3}{2}}(n+2) + \frac{1}{\binom{n+1}{2}} \sum_{y=1}^{n} y \log_{\frac{3}{2}}(y+1) \tag{C.4}$$

$$\tag{C.5}$$

Notice that $\sum_{y=1}^{n} y \log_{\frac{3}{2}}(y+1)$ is a right Riemann sum and the function $y \log_{3/2}(y+1)$ is increasing for $y \geq 1$. Therefore the right Riemann sum can be (upper) bounded:

$$\begin{aligned}
\sum_{y=1}^{n} y \log_{\frac{3}{2}}(y+1) &\leq \int_{1}^{n+1} y \log_{\frac{3}{2}}(y+1)\, dy \\
&= \left[\frac{\left(2y^2 \log(y+1) - y^2 + 2y - 2\log(y+1)\right)}{4 \log\left(\frac{3}{2}\right)} \right]_{1}^{n+1} \\
&= \frac{2(n+1) - (n+1)^2 + 2(n+1)^2 \log(n+2) - 2\log(n+2) - 1}{4 \log\left(\frac{3}{2}\right)}.
\end{aligned}$$

We plug this into C.4 and (rigorously) simplify to obtain

$$\begin{aligned}
-\log_{\frac{3}{2}}(n+2) + \frac{1}{\binom{n+1}{2}} \sum_{y=1}^{n} y \log_{\frac{3}{2}}(y+1) &\leq \log_{\frac{3}{2}}(n+2) \\
&\quad + \frac{1}{\binom{n+1}{2}} \left(\frac{2(n+1) - (n+1)^2 + 2((n+1)^2 - 1)\log(n+2) - 1}{4 \log\left(\frac{3}{2}\right)} \right) \\
&= \frac{2\log(n+2) - n}{(n+1)\log\left(\frac{9}{4}\right)}.
\end{aligned}$$

That means that we have bounded our original sum by

$$\sum_{x=1}^{n} \frac{n+1-x}{\binom{n+1}{2}} \log_{\frac{3}{2}} \left(1 - \frac{x}{n+2}\right) \leq \frac{2\log(n+2) - n}{(n+1)\log\left(\frac{9}{4}\right)} = \mathcal{B}(n). \tag{C.6}$$

Let $\mathcal{B}(n)$ be as in Equation C.6. Then we notice that for small n $\mathcal{B}(n) \geq -1$ which works against us. However, when we take $n = 50$ then

$$\mathcal{B}(50) = -1.01789\ldots \leq -1.$$

Moreover,

$$\frac{d}{dn}\mathcal{B}(n) = \frac{n - 2(n+2)\log(n+2)}{(n+1)^2(n+2)\log(9/4)} \leq 0,$$

for $n \geq 3$. That means that $\mathcal{B}(n)$ is descending for $n \geq 3$ and thus $\mathcal{B}(n) \leq -1$ for $n \geq 50$. This implies that for $n \geq 50$:

$$\sum_{x=1}^{n} \frac{n+1-x}{\binom{n+1}{2}} \log_{\frac{3}{2}}\left(1 - \frac{x}{n+2}\right) \leq \mathcal{B}(n) \leq -1.$$

All that is left to do is to check the values $n \in \{3, \ldots, 49\}$ which can be easily done using a computer. The values for $n \in \{3, \ldots, 49\}$ are in Table C.1 This concludes the proof that for $n \geq 3$:

$$\sum_{x=1}^{n} \frac{n+1-x}{\binom{n+1}{2}} \log_{\frac{3}{2}}\left(1 - \frac{x}{n+2}\right) \leq -1.$$

□

n	Value of the sum
3	-1.07176
4	-1.09272
5	-1.10866
6	-1.12123
7	-1.13141
8	-1.13984
9	-1.14693
10	-1.153
11	-1.15824
12	-1.16282
13	-1.16686
14	-1.17045
15	-1.17366
16	-1.17656
17	-1.17917
18	-1.18156
19	-1.18373
20	-1.18573
21	-1.18757
22	-1.18926
23	-1.19084
24	-1.1923
25	-1.19366
26	-1.19494
27	-1.19613
28	-1.19725
29	-1.1983
30	-1.1993
31	-1.20023
32	-1.20112
33	-1.20196
34	-1.20275
35	-1.20351
36	-1.20422
37	-1.20491
38	-1.20556
39	-1.20618
40	-1.20677
41	-1.20734
42	-1.20788
43	-1.20841
44	-1.20891
45	-1.20939
46	-1.20985
47	-1.21029
48	-1.21072
49	-1.21113

Table C.1: The values of $\sum_{x=1}^{n} \frac{n+1-x}{\binom{n+1}{2}} \log_{\frac{3}{2}} \left(1 - \frac{x}{n+2}\right)$ for $n \in \{3, \ldots, 49\}$.

Appendix D

Implementation of Post-Process algorithm in Sage

```
import time;
def STWeight():
    return -log(0.0001);

def RandomEdgeWeight():
    return -log(RR.random_element(0,0.4));

def FM(n):
    G = DiGraph(n);
    G.add_vertices(['s', 's1', 's2', 't', 't1', 't2']);
    G.add_edges([('s','s1', STWeight()), ('s', 's2', STWeight()), ('t1', 't', STWeight()
        ), ('t2', 't', STWeight()), ('s1', 't1', RandomEdgeWeight()), ('s1', 't2',
        RandomEdgeWeight()), ('s2', 't1', RandomEdgeWeight()), ('s2', 't2',
        RandomEdgeWeight())]);
    for x in range(n):
        G.add_edges([('s1', x, RandomEdgeWeight()), ('s2',x, RandomEdgeWeight()), (x, '
            t1', RandomEdgeWeight()), (x, 't2', RandomEdgeWeight())]);
        for y in [x+1..n-1]:
            G.add_edges([(x,y, RandomEdgeWeight()), (y,x, RandomEdgeWeight())]);
    b = {'s':2, 't':-2, 's1':0, 's2':0, 't1':0, 't2':0}
    for x in range(n):
        b[x] = 0;
    return {'G':G, 'b':b};

def RunAlg(n, displayTime):
    time1 = time.time();
    G = DiGraph(n);
    G.add_vertices(['s', 's1', 's2', 't', 't1', 't2']);
    G.add_edges([('s','s1', STWeight()), ('s', 's2', STWeight()), ('t1', 't', STWeight()
        ), ('t2', 't', STWeight()), ('s1', 't1', RandomEdgeWeight()), ('s1', 't2',
        RandomEdgeWeight()), ('s2', 't1', RandomEdgeWeight()), ('s2', 't2',
        RandomEdgeWeight())]);
    for x in range(n):
        G.add_edges([('s1', x, RandomEdgeWeight()), ('s2',x, RandomEdgeWeight()), (x, '
            t1', RandomEdgeWeight()), (x, 't2', RandomEdgeWeight())]);
        for y in [x+1..n-1]:
            G.add_edges([(x,y, RandomEdgeWeight()), (y,x, RandomEdgeWeight())]);
    b = {'s':2, 't':-2, 's1':0, 's2':0, 't1':0, 't2':0}
    for x in range(n):
        b[x] = 0;
    time2 = time.time();
    if displayTime:
        print 'Creating G took %f seconds.' % (time2-time1);

    time1 = time.time();
    E = G.edges();
    M = MixedIntegerLinearProgram(maximization=False, solver = "GLPK");
    f = M.new_variable(nonnegative=True, integer=True);
    M.set_objective(sum(f[x]*x[2] for x in E));
    for x in E:
        M.add_constraint(f[x] <= 1);
    Verz = G.vertices();
    Verz.remove('s');
    Verz.remove('t');
    for v in Verz:
```

```
        IncomingEdges = G.incoming_edges(v);
        M.add_constraint(sum(f[e] for e in IncomingEdges) <= 1);
    V = G.vertices();
    for v in V:
        M.add_constraint(sum(f[e] for e in G.outgoing_edges(v)) - sum(f[e] for e in G.
            incoming_edges(v)) == b[v]);
    M.set_binary(f);
    time2 = time.time();
    if displayTime:
        print 'Initializing MILP took %f seconds.' % (time2-time1);

    time1 = time.time();
    M.solve(objective_only=False)
    time2 = time.time();
    if displayTime:
        print 'MILP took %f seconds.' % (time2-time1);
    flow = M.get_values(f);


    time1 = time.time();
    H = DiGraph();
    G.add_vertices(G.vertices());
    for e in flow.keys():
        if flow[e] >0:
            H.add_edge(e);
    #H.show()
    p1 = H.shortest_path('s', 't', by_weight=True);
    p1length = H.shortest_path_length('s', 't', by_weight=True);
    H.delete_vertex(p1[1]);
    p2 = H.shortest_path('s', 't', by_weight=True);
    p2length = H.shortest_path_length('s', 't', by_weight=True);
    time2 = time.time();
    if displayTime:
        print 'Creating H took %f seconds.' % (time2-time1);

    q1 = [];
    q2 = [];
    q1length = p2length;
    q2length = p2length;
    time1 = time.time();
    for i in [1..len(p1)-3]:
        Gcopy = G.copy();
        Gcopy.delete_edge((p1[i], p1[i+1]));
        q1lastpart = Gcopy.shortest_path(p1[i], 't', by_weight = True);
        q1new = [];
        for j in [0..i]:
            q1new.append(p1[j]);
        for j in [1..len(q1lastpart)-1]:
            q1new.append(q1lastpart[j]);
        q1newlength = Gcopy.shortest_path_length(p1[i], 't', by_weight = True) + sum(G.
            distance(p1[j], p1[j+1], by_weight=True) for j in [0..i-1]);
        if q1newlength < p2length:
            for j in [1..len(q1new)-2]:
                if Gcopy.has_vertex(q1new[j]):
                    Gcopy.delete_vertex(q1new[j]);
            q2newlength = Gcopy.shortest_path_length('s', 't', by_weight = True);
            if q2newlength < p2length:
                q2new = Gcopy.shortest_path('s', 't', by_weight = True);
                if max(q1newlength, q2newlength) < max(q1length, q2length):
                    q1 = q1new;
                    q2 = q2new;
                    q1length = q1newlength;
                    q2length = q2newlength;
    time2 = time.time();
    if displayTime:
        print 'Iterating to find Q1 and Q2 took %f seconds.' % (time2-time1);
    return {'P1Length':p1length, 'P2Length':p2length, 'Q1':q1, 'Q2':q2, 'Q1Length':
        q1length, 'Q2Length':q2length}
```

```
def Ratio(P1, P2, Q1, Q2):
    return 1 - (max(Q1, Q2) - (P1+P2)/2)/(P2-(P1+P2)/2);

# k is number of simulations
# n is number of vertices other than s, s1, s2, t, t1 and t2.
def CountAlgIsBetter(n, simulations, displayTime, boxes):
    better = 0;
    minimprove = 0;
    maximprove = 0;
    averageimprove = 0;
    boxcount = boxes*[0];
    output = {};
    for k in [1..simulations]:
        output = RunAlg(n, displayTime);
        if output['Q1'] != []:
            better += 1;
            newratio = Ratio(output['P1Length'], output['P2Length'], output['Q1Length'],
                output['Q2Length']);
            newratioindex = floor(boxes*newratio);
            boxcount[newratioindex] += 1;
            if minimprove == 0:
                minimprove = newratio;
                maximprove = newratio;
            else:
                if newratio < minimprove:
                    minimprove = newratio;
                if newratio > maximprove:
                    maximprove = newratio;
            averageimprove += newratio;
    averageimprove = averageimprove/better;
    return {'Better':better, 'Min':minimprove, 'Max':maximprove, 'Average':
        averageimprove, 'Boxcount':boxcount};


boxintervals = 10;
time1 = time.time();
Improve = CountAlgIsBetter(5, 10000, False, boxintervals);
time2 = time.time();
print 'Algorithm took %f seconds' % (time2-time1)
print 'Number of improvements: %i' % Improve['Better'];
print 'Minimal improvement: %f' % Improve['Min'];
print 'Maximal improvement: %f' % Improve['Max'];
print 'Average improvement: %f' % Improve['Average'];
for i in range(boxintervals):
    print 'In box %i : %i' % (i, Improve['Boxcount'][i]);
```

Programming Code D.1: Implementation of Post-Process Algorithm in Sage

List of Symbols

A	A path in a graph, represented by its set of edges.
B	A path in a graph, represented by its set of edges.
$c(e)$	The cost of an edge e.
$c(P)$	The total cost of a path P: the sum of the weights of the edges of P.
$D(s,t)_G$	The set of pairs of vertex-disjoint (s,t)-paths in a graph G.
e	An edge of a graph.
$E(G)$ **or** E_G	The edge set of a graph G.
F	A forest graph; a graph consisting of trees.
G	A Graph with vertex set $V(G)$ and edge set $E(G)$.
$G = (V, E)$	A Graph with vertex set V and edge set E.
i, j	Integers acting as iterators.
k	Natural number.
λ	Number of Linktraces.
Λ_n	The number of probes needed to recover the network topology on n nodes.
$l(P)$	The length (amount of edges) of path P.
m	Number of edges.
M	A set of MEPs.
▶	A MEP; Maintenance domain End Point.
●	A MIP; Maintenance domain Intermediate Point.
n	Number of vertices.
$\mathcal{O}$	Big O notation.
P	A path in a graph, represented by its set of edges.
p_e	The failure probability of an edge e.
$\rho(u, v)$	A probing operation which returns the shortest path between vertices u and v.
s	A special vertex of a graph that acts as a source.
S **or** S_i	A switch, possibly indexed by a number i.
t	A special vertex of a graph that acts as a target.
T	A tree graph.
$V(G)$ **or** V_G	The vertex set of a graph G.

List of Abbreviations

CCM	Continuity Check Message
DDoS	Distributed Denial of Service
DNS	Domain Name System
E-LMI	Ethernet Local Management Interface
EVC	Ethernet Virtual Connection
IEEE	Institute of Electrical and Electronics Engineers
IP	Internet Protocol
ITU	International Telecommunications Union
LAN	Local Area Network
LBM	Loopback Message
LBR	Loopback Reply
LTM	Linktrace Message
LTR	Linktrace Reply
MA	Maintenance Association
MAC	Media Access Control
MAID	Maintenance Association Identifier
MD	Maintenance Domain
MEF	Metro Ethernet Forum
MEG	Maintenance Entity Group
MEP	Maintenance domain End Point
MIP	Maintenance domain Intermediate Point
OAM	Operations, Administration and Maintenance
OSI	Open System Interconnectino
PBB-TE	Provider Backbone Bridge - Traffic Engineering
STP	Spanning Tree Protocol
VLAN	Virtual Local Area Network
WAN	Wide Area Network

Bibliography

[Cas+90] J Case et al. *A Simple Network Management Protocol*. Tech. rep. Cambridge MA: Internet Engineering Task Force (IETF), 1990.

[Cis] *Manual for Cisco Catalyst 4500 series switch*. Release 12.2(54)SG. Chapter 55 - Configuring Ethernet OAM and CFM. Cisco Systems Inc.

[Das09] Santanu Dasgupta. *Understanding the Benefits of Ethernet OAM*. PDF slides. Cisco Systems InC., 2009.

[FHW80] S Fortune, J Hopcroft and J Wyllie. 'The directed subgraph homeomorphism problem'. In: *Theoret. Comput. Sci.* 10 (1980), pp. 111–121.

[Fin04] Norman Finn. *Connectivity Fault Management, IEEE 802.1 tutorial*. PDF slides. Cisco Systems Inc., 2004.

[GJ79] M. R. Garey and D. S. Johnson. *Computers and Intractability: A Guide to the Theory of NP-Completeness*. W. H. Freeman and Company, 1979.

[IEE04] IEEE. *IEEE std. 802.1D-2004, Media Access Control (MAC) Bridges*. Tech. rep. New York: Institute of Electrical and Electronics Engineers Inc., 2004.

[IEE09] IEEE. *IEEE standard 802.1AB-2009. IEEE Standard for Local and metropolitan area networks - Station and Media Access Control Connectivity Discovery*. Tech. rep. New York: Institute of Electrical and Electronics Engineers Inc., 2009.

[IEE11] IEEE. *IEEE standard 802.1Q-2011. IEEE Standard for Local and metropolitan area networks - Media Access Control (MAC) Bridges and Virtual Bridge Local Area Networks*. Tech. rep. New York: Institute of Electrical and Electronics Engineers Inc., 2011.

[IEE12] IEEE. *IEEE std. 802.3-2012. IEEE Standard for Ethernet*. Tech. rep. New York: Institute of Electrical and Electronics Engineers Inc., 2012.

[ITU11] Telecommunication Standardization Sector of ITU. *Recommendation ITU-T G.8013/Y.1731: OAM functions and mechanisms for Ethernet based networks*. Tech. rep. International Telecommunications Union, 2011.

[JN10] Inc. Juniper Networks. *Ethernet OAM, A Technical Overview*. Tech. rep. Juniper Networks Inc., 2010.

[JN14] Inc. Juniper Networks. *Configuring a Maintenance Endpoint*. 2014. URL: http://www.juniper.net/techpubs/en_US/junos13.3/topics/usage-guidelines/interfaces-creating-a-maintenance-end-point.html (visited on 14/01/2014).

[LMSL90] Chung-Lun Li, S. T. McCormick and D Simchi-Levi. 'The complexity of finding two disjoint paths with min-max objective function'. In: *Discrete Applied Mathematics* 26 (1990), pp. 105–115.

[MEF06] MEF. *Technical Specification MEF 16, Ethernet Local Management Interface (E-LMI)*. Tech. rep. Metro Ethernet Forum, 2006.

[MEF12] MEF. *MEF on the GO, MEF-CECP Study Guide*. Metro Ethernet Forum, 2012.

[PM11] M Prins and R Malhotra. *Ethernet Operation Administration and Maintenance, Opportunities for the NREN community*. Tech. rep. TNO, SURFnet, 2011.

[Sag14] SageMath. *Sage - Open-Source Mathematical Software System*. 2014. URL: http://www.sagemath.org (visited on 29/09/2014).

[Spr04] N. Spring. 'Efficient discovery of network topology and routing policy in the Internet. A dissertation submitted in partial fulfillment of the requirements for the degree of Doctor of Philosophy'. PhD thesis. University of Washington, 2004.

[ST84] J. W. Suurballe and R. E. Tarjan. 'A Quick Method for Finding Shortest Pairs of Disjoint Paths'. In: *Networks* 14 (1984), pp. 325–336.

[Wik14] Wikipedia. *Overlay Network*. 2014. URL: http://en.wikipedia.org/wiki/Overlay_network (visited on 02/10/2014).