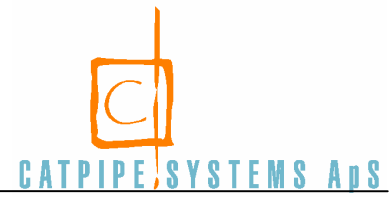
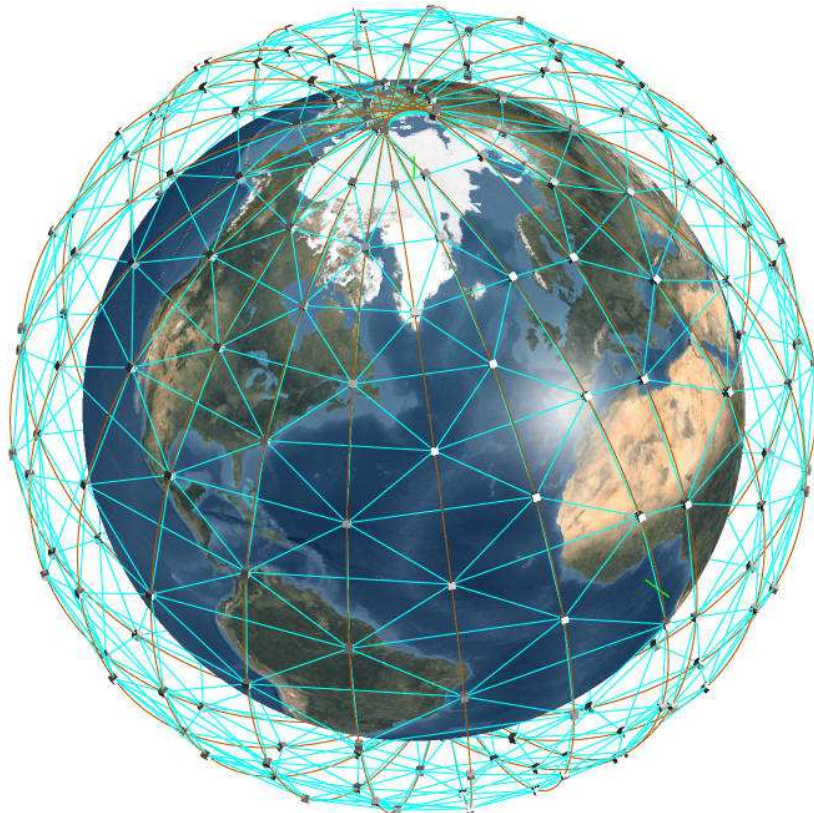


Layer 2 topology discovery



Midterm project – COM•DTU
Technical University of Denmark



January 20th, 2006

Supervisor: Anders Fosgerau

Simon Thoustrup – s021739

January 20th, 2006

Written by

Simon Thoustrup
s021739

*Front page picture:
Overview of SC Modeler2.7
Copyright © 2000-2006 AVM Dynamics Inc.
<http://www.avmdynamics.com/software1.htm>*

Resumé

Metoder til bestemmelse af netværkstopologi på lag 2 i Ethernet netværk er blevet udviklet ud fra krav som generalisering og skalering i form af hastighed i afvikling.

Første metode baserer sig på en analyse af forwarding databaser fra bridge, switche og routere. Herunder redefineres det dobbelt-gældende komplekskrav til et enkeltvejs komplekskrav samt et minimumskrav.

Anden metode baserer sig på en analyse af Spanning Tree Protokollens informationer, der er gemt i noder, der deltager i protokollen.

Tredje metode baserer sig på en analyse af Cisco Discovery Protocol-informationer, der er gemt i noder, der deltager i et CDP netværk.

Det er blevet vist, at det ikke er muligt for en enkelt metode at påvise alle lag 2 forbindelser i et givent netværk, men at de tre metoder i kombination kan give et velkvalificeret gæt på topologien.

Projektet blev udført som et Polyteknisk Midtvejs Projekt hos virksomheden catpipe Systems ApS i perioden 1. september 2005 – 20. januar 2006. Fra catpipe Systems deltog Nicolai Petri samt Philippe Regnauld. Fra catpipe Systems, senere NNIT, deltog Benny Kjærgaard i projektets udformning.

Den udviklede kildekode er ikke medtaget i rapporten, da koden efter aftale med catpipe Systems ikke er tilgængelig for offentligheden.

January 20th, 2006

THIS PAGE WAS INTENTIONALLY LEFT BLANK

Abstract

Methods for discovery of layer 2 network topology for an Ethernet network have been developed based on requirements such as generalness and scalability represented as speed in execution.

The first method is based on an analysis of forwarding databases from bridges, switches and routers. The method redefines the existing double completeness requirement into a single completeness requirement and a minimum requirement.

The second method is based on an analysis of the Spanning Tree Protocol information stored in each node participating in the protocol.

The third method is based on an analysis of the Cisco Discovery Protocol information stored in each node participating in a CDP network.

It has been shown that it is not possible for a single method to reliably return all layer 2 links in any given network but that the three methods in combination can return an educated guess of the topology.

The project was carried out as a Polytechnic Midterm Project at catpipe Systems ApS starting the 1st of September 2005 and ending at the 20th of January 2006. From catpipe Systems Nicolai Petri and Philippe Regnaud were involved in the project. Benny Kjærgaard from catpipe Systems, later NNIT, was involved in the start-up of the project.

The source code for the developed scripts and database structure is in agreement with catpipe Systems not available to the general public and is therefore not included in this report.

January 20th, 2006

THIS PAGE WAS INTENTIONALLY LEFT BLANK

Table of Content

1	INTRODUCTION	1
2	PROBLEM DESCRIPTION	3
2.1	ANALYSIS	3
3	THEORY	5
3.1	MATCH IN-OUT TRAFFIC COUNTERS	5
3.2	CONFIGURATION PARSING.....	5
3.3	EXPLORATION OF KNOWN PROCESSING DELAYS.....	5
3.4	SPANNING TREE PROTOCOL.....	5
3.5	FORWARDING DATABASES.....	6
3.6	DISCOVERY WITHOUT NETWORK ASSISTANCE	7
3.7	CISCO DISCOVERY PROTOCOL	8
3.8	SNMP.....	8
3.8.1	<i>Physical Topology MIB</i>	9
3.9	802.1AB	9
3.10	POSTGRESQL.....	10
3.10.1	<i>Views</i>	10
3.10.2	<i>Rules</i>	11
3.10.3	<i>Functions</i>	11
4	DESIGN.....	12
4.1	THE TECHNOLOGIES CHOSEN	12
4.2	FORWARDING DATABASE.....	12
4.3	SPANNING TREE.....	17
4.4	CISCO DISCOVERY PROTOCOL	18
5	IMPLEMENTATION	19
5.1	DATABASE LAYOUT	19
5.2	SCRIPTS.....	21
5.2.1	<i>s-*</i>	21
5.2.1.1	<i>s-iphavest.pl</i>	21
5.2.1.2	<i>s-stpinfo.pl & s-fdbinfo.pl</i>	21
5.2.2	<i>parse-ipharvest.pl</i>	22
5.2.3	<i>u-*</i>	22
5.2.3.1	<i>u-fdb-portwise.pl</i>	22
5.2.3.2	<i>u-fdb-single.pl</i>	23
5.2.4	<i>u-cdp.pl</i>	24
5.2.5	<i>graph.pl</i>	24
6	TEST.....	25
7	RESULTS & DISCUSSION	27
7.1	REAL-LIFE NETWORKS	27
7.2	FURTHER DEVELOPMENT	28
8	CONCLUDING REMARKS	29
9	BIBLIOGRAPHY.....	31

January 20th, 2006

List of acronyms and abbreviations

s

1 Introduction

For a network operator or the person in charge of the network in the company it is of great importance to know how the network is laid out – to know the network topology. Today this knowledge is typically stored in the memory of the people responsible for the network which makes it difficult for other systems, people and processes to obtain the information.

For the business manager it is important to know which entities are present in the network, how they perform and when they eventually should be replaced. It is important to have knowledge of which entities are crucial to the operation of the network in order to lay out budgets for purchasing replacements and/or upgrades.

For the software developer of distributed systems, it is important to know which entities are in use in the path used by the application in development. The software developer is interested in latency, reachability and throughput – information all dependent of the topology. By having an (not necessarily full-detailed) idea of the topology may guide the developer in designing backup-procedures in case of a link fail etc.

The network engineer or the network manager is usually the person who has the greatest value of and need for knowing the topology.

With respect to monitoring service availability, it is of crucial importance to the perception of alarms that the network topology is known. A manual analysis of alarms is however not always the most feasible approach depending on the number of alarms.

If a large number of monitored services is dependent of the availability of a third service, the alarm for that third service may drown in the large number of total alarms.

If the monitoring system is aware of the topology and dependencies, it is possible for the system to build a string of dependencies and mark the alarm which should be dealt with.

In networks comprising of many thousand nodes¹ it can be a difficult task to define these dependencies manually. A manual definition usually means a manual update of the definition which potentially could be an even more difficult task in large-scale dynamic networks. Neglecting the update of the definition can lead to the monitoring system giving false advise on which alarms to deal with and lead to the system being of no value at all.

*"We're engineers. To us,
data is protocol overhead."
- Unknown.*

For these reasons an automated topology discovery would be feasible.

Existing methods to discover the topology is usually based on the topology seen from layer 3.

For cases with physical errors in equipment, a topology definition based on layer 2 is interesting.

¹ For the rest of this report, the term "node" is used for switches, bridges and routers.

January 20th, 2006

This report focuses on the development of a method to perform automated topology discovery for layers down to layer 2. The method is if not fully implemented then prepared for implementation as a module to catpipe Systems' monitoring product "d-mon".

2 Problem description

A method for automatic network topology discovery in real-life Ethernet networks should be developed. The method should be able to map topologies at layer 2.

The method should be prepared to deal with VLANs as well as virtual layer 2 interfaces (tunnels etc.).

An option to map the layer 2 topology to layer 3 topology as well as being able to work with multiple layer 3 subnets in the same network segment would be feasible.

Furthermore, the following is required:

- The developed algorithms should be as general as possible.
- The developed code should be able to work in multi-vendor and multi-model networks.
- The developed method should be able to scale to networks comprised of many thousand nodes.
- The results generated by the implementation should be presented to the user in some way or the other.

It is considered given that:

- The nodes in the network supports SNMP and SNMP is activated and configured.
- The SNMP management interface for the nodes is accessible either via a dedicated network or in-band.
- The nodes in the network and the network itself is correctly configured (loops is non-existing or blocked via STP).

If time permits, the developed method should be implemented as a module for the surveillance system “d-mon”.

2.1 Analysis

Any given network is comprised of nodes (switches, routers, bridges), devices² (PC, printer) and edges (links e.g. connections) between the nodes and the devices.

The main focus is to be able to map the interconnections of nodes.

In order to be able to design a method to clarify how these nodes are interconnected one must have an overview of which properties are available from the different entities. In the following some of these properties are listed, and each will be dealt with more carefully in the next section.

- Nodes as well as devices usually implement the Internet Control Message Protocol (ICMP). This protocol enables the measurement of round-trip times from one node/device to another – the longer the distance, the larger the round-trip time.

² For the rest of this report, the term “device” is used for end user equipment (PCs, printers etc.)

- The round-trip time increases with distance due to physics, but is also subject to the processing time taken up by the nodes the packet passes.
- Nodes such as bridges, switches (multiport bridges) and routers implement a forwarding database (FDB). This FDB holds information of which layer 2 nodes and devices are available via which interfaces. An FDB entry is subject to ageing, and is at some point aged sufficiently to be purged from the database. This occurs when the address has not been in use (i.e. a packet with the entry as source address has not been seen) for the defined age-period.
- Similarly such nodes often hold a Spanning Tree Protocol (STP) database in case of the node participating in the STP. The STP ensures that no loops are present in the network, as well as the links chosen by the network administrator are used. In case of a link failure, the STP re-calculates the path to be used, once again taking the priorities made by the administrator into account. The STP database holds information of which participants in the spanning tree is available on which interfaces, and whether these interfaces are currently the ones used (in forwarding state) or not used (in blocking state).
- Nodes and devices keep track of how many packets and bytes are sent and received on the individual interfaces.
- The configuration of nodes is usually done by technicians that possess knowledge of the immediate neighbours to which the particular node is connected. In order to make configuration (and especially adjusting the configuration later on) more intuitive, technicians often assign names to interfaces that reveal what the particular interface is connected to.

3 Theory

The properties listed in the previous section opens for a set of different methods. They are each being dealt with more carefully in the following.

3.1 Match in-out traffic counters



By comparing the in-counter on one port with the out-counter on other ports it may be possible to pin-point the port a given device is placed on. This does require control of many of the devices and nodes in the network, since one must make sure only the device(s) used in the check is transmitting.

Furthermore a problem may arise regarding broadcasting and multicasting traffic and the effect they have on the counters.

Figure 1 – Counters [16]

3.2 Configuration parsing

Parsing the configurations from the nodes might be able to give enough information to build the topology of the network.

This approach does however rely much on the employees in respect of giving the right information in the configurations.

Furthermore the format of configuration files has been seen to change from firmware version to firmware version which imposes yet another possible point of error in the method.

Finally the programming made to retrieve and parse information from nodes of one vendor will probably not work for nodes of another vendor.

3.3 Exploration of known processing delays

If the processing delay for the nodes in the network is known, the number of nodes a given packet travels through may be calculated by analysing the round-trip times.

This approach requires extremely well-known devices in the network since some of the processing delay will be contributed by the devices. In case of measuring the round-trip time to/from a node, the time measured may be heavily dependent on the load of the node. Implementations of the ICMP in nodes are usually made so that functions for the operation of the node receive CPU time prior to the ICMP. The result is that the processing time before the reply to an ICMP packet is sent can be anything between a few microseconds and up to a few seconds.

The method may be interesting in the case of a hidden switch in the network – i.e. non-manageable switches or HUBs.

3.4 Spanning Tree Protocol

An Ethernet network only works for as long as there are no existing loops in the network. The IEEE has specified an algorithm to ensure this – the IEEE 802.1D Spanning Tree Protocol. If a network is represented with a graph, the spanning tree maintains connectivity by including every node in the network as a node in the graph but removes any links causing loops.

The STP requires all bridges in the network to have a unique bridge id, each interface

within each bridge must have a unique port id and each interface within each bridge has to be assigned a cost.

When the algorithm is enabled, all bridges elect a root bridge which is the bridge with the lowest bridge id.

Each bridge then determines the root port which is the port with the least-cost path to the root bridge – the cost path is the sum of costs for the interfaces in the path. In case of ties the root port is elected by the lowest port id.

Finally each bridge determines the designated bridge, which is the bridge that offers the least-cost path to the root bridge, i.e. the first (STP-enabled) bridge reached via the root port. Once again in case of ties, the bridge with the lowest bridge id is elected as the designated bridge.

The information required by the algorithm is exchanged between bridges by using Bridge Protocol Data Units (BPDUs).

A BPDU holds among other information the root bridge id, the root path cost, the sender's bridge id and the sender's port id.

By exploring the STP database in nodes it is possible to obtain information regarding the links between nodes taking part in the spanning tree.

Since this information has to be present at all times, and the STP ensures the information is up-to-date, this can be used to build the topology of the spanning tree. If all nodes in the network participate in the spanning tree and runs the STP, all nodes will be accounted for. Usually some nodes do not participate in the spanning tree, and neither do devices. To find and map these nodes and devices in the topology some other technique must be put to use.

3.5 Forwarding Databases

Every node has a forwarding database (FDB). This database contains an entry for each source MAC address seen on the interfaces of the node.

By extracting, ordering and combining this information a topology description could be built. The requirement is for these FDBs to hold information of all addresses in the broadcast domain i.e. they have to fulfil the “completeness requirement”.

This technique does have some limitations however. First of all entries in a FDB are aged, and when they reach a certain age they are purged from the database. This is due to the fact that the nodes in the network need to be aware of where a device is located. If a device is moved from one part of the network to the other, the nodes in the network need to update their FDB entry.

Another reason for purging the FDB is performance. The smaller the FDB, the lesser the CPU power to process the information.

In real-life networks the FDBs in the nodes seldom hold information of all devices in the broadcast domain. If some devices have been idle for a period, the FDB entry is most probably removed. When the device is active (i.e. transmitting) again a new entry is placed in the FDB.

This poses a problem on the method of extracting all FDBs from the network since these will most probably never be complete.

Different approaches to solving this problem have been suggested [1, 2].

One method is based on generating traffic to ensure that the FDB entries don't age [2]. By generating ICMP ping packets with source and destination pairs widely across the network, the FDB entries are kept in the database. One problem may be how to generate these packets. If they are to be generated from the management station running the topology discovery application, this station needs to be in-band with the user-traffic. Generation of the traffic on the management network will most probably not populate the FDB database sufficiently. Some network operators may have disabled the ICMP or parts of it in nodes and/or devices. This will result in missing information in the FDBs. Finally, to create a ping-storm on a network with 1500+ nodes and even more devices may not be a good idea. To generate all these ICMP ping packets and to receive all FDBs within the default ageing time of 300 seconds may prove difficult for large scale networks.

Another method is to accumulate the FDBs during a fixed period of time before making any analysis [2]. This violates the ageing principle, but in corporate networks devices do not usually move around that much. A problem may arise with the increasing number of WLAN devices but if one is able to separate information regarding these devices from the rest of the FDB the method may prove valid in many network environments.

A third method is based on an opposite constraint [1]. The method defines the "Simple Connection" as opposed to the "Direct Connection". A Simple Connection is defined as a connection between two bridges, where there may be nodes in between. In other words, any thought connection which doesn't place nodes or devices in two different places in the network *can* be a valid Simple Connection. Whether it is a valid connection is determined via the opposite constraint. The constraint given is that if just a single entry of the FDBs mismatches the connection is not valid.

This approach relaxes the completeness requirement by requiring only enough information to determine that all but one possible connection is invalid.

3.6 Discovery without Network Assistance

It has been suggested to perform a topology discovery without any assistance from the network [4].

The method relies on having all devices in the network conform to a special network discovery protocol. The technique is based on two properties of a switched Ethernet: 1) Devices on the same switch segment can see all traffic on that segment while in promiscuous mode. 2) A packet entering a switch with a specific source address prevents the switch from sending out packets with the same address as destination to any other port. Furthermore this approach requires one terminal "the mapper", which centrally controls the algorithm. This "mapper" needs to have access to the daemon running on all (or most) devices in the network in order to control the discovery protocol session running on them.

Microsoft seem to be willing to proceed with this approach and are working on bringing the method into their operating systems which will (provided Microsoft operating systems are used heavily among the devices) make the approach a bit more appealing: "*we are working with product teams to get this functionality into a future release of Windows*"[4].

3.7 Cisco Discovery Protocol

Cisco Systems Inc. have created their proprietary Cisco Discovery Protocol (CDP). It operates at layer 2 which enables it to support different layer 3 protocols while maintaining connectivity with other CDP-enabled nodes.

It is primarily used for layer 3 protocol address-exchange as well as to obtain certain interface parameters of neighbouring nodes. The exchange of information is taking place by means of special Cisco BPDUs sent to the Cisco multicast address of 01:00:0C:CC:CC:CC every 60 seconds by default [5]. The BPDUs contain information such as device id, layer 3 address, port id, capabilities and platform [6]. This information enables each CDP-enabled node to know exactly which CDP neighbours it has.

In case of a HUB or a CDP-unaware bridge being immediate “neighbour”, CDP will report either the sender of the last received BPDU or all senders of all BPDUs received in the interface as the neighbour(s) depending on the implementation [7, 8]. There is no way to tell whether this is the case or not, which is why CDP might give direct connections and it might give simple connections.

In case of a CDP-aware bridge being the immediate “neighbour”, the BPDUs are discarded by this node, and the CDP information never reaches any CDP-enabled nodes. Thus, the information stored in the CDP-enabled nodes is more likely to give the direct connections but the topology generated may consist of groups of nodes not connected to one another.

Even though CDP is developed proprietarily by Cisco other vendors have implemented the protocol in their equipment (HP as an example). Yet other vendors have made their equipment CDP-aware but not fully implemented the protocol, probably due to political or legal reasons.

Apart from the technologies dealt with above, a few additional protocols and technologies are interesting when designing a layer 2 topology discovery application.

3.8 SNMP

The Simple Network Management Protocol (SNMP) provides the means of automated management of network devices. Defined in 1988 as RFC 1067 the SNMP allows for network administrators to receive information from and send configuration to any network node provided it implements the SNMP.

All information available to the SNMP agent is stored in the node in MIBs – Management Information Bases. A MIB database holds a set of objects described by a unique identifier which among other things include the type of the object (string, integer etc.), access level (read, write read/write) and size restrictions if any.

The SNMP protocol has been improved and enhanced several times since the original RFC. Today version 2c (RFC 1901+) or version 3 (RFC 2271) are the versions commonly used. The ‘c’ in version 2c is pointing to the fact that it is a version using communities as access protection. One needs to have the correct community (string) in order for the SNMP node to provide or respond to any information.

The information available via SNMP is usually available via an interface (serial, telnet or ssh). Scripting the way through the various existing (vendor specific) interfaces will likely pose the same problems as doing configuration parsing.

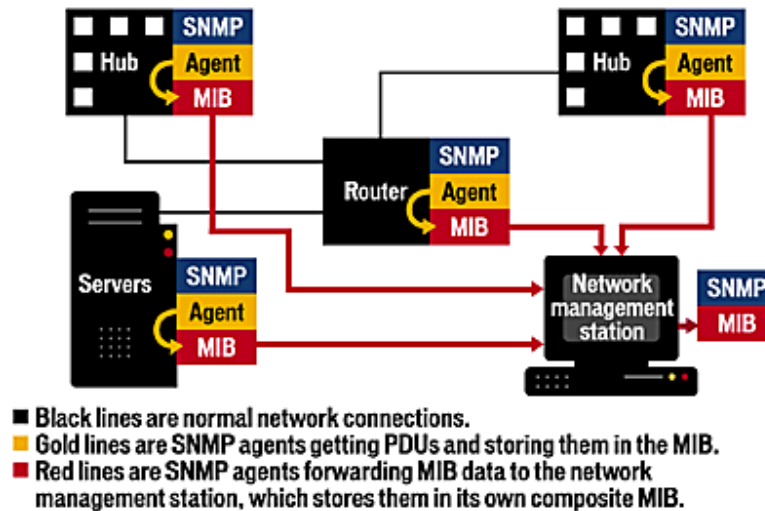


Figure 2 - SNMP Agents and MIBS [9]

3.8.1 Physical Topology MIB

One MIB is particularly interesting for topology discovery. The Physical Topology MIB (RFC 2922) or PTOPO-MIB was defined by representatives of Cisco Systems Inc. and Nortel Networks in September 2000. The RFC defines how physical network information is kept within this MIB and how this information relates to the data kept in other MIBs. The PTOPO-MIB is however just a MIB. The RFC does not describe a protocol to allow the information to be exchanged within groups of devices. Thus, it has not succeeded fully in becoming the de-facto standard MIB since it makes little sense to provide data which cannot be distributed to other nodes.

3.9 802.1ab

The recently approved IEEE standard 802.1ab³ defines the Link Layer Discovery Protocol (LLDP) which is a neighbour discovery protocol. It will enable nodes in the network to advertise information about themselves to other nodes in the network and store the information they receive.

The information they advertise must include the fields “device chassis ID” and “port ID” which enables other nodes to know by which interface on the remote node they are connected.

The information is transmitted periodically – the IEEE recommends a transmission rate of 30 seconds, but the value can be adjusted.

The information received by the nodes is stored in SNMP MIBs especially designed for LLDP.

³ Approved March 20th 2005 by IEEE

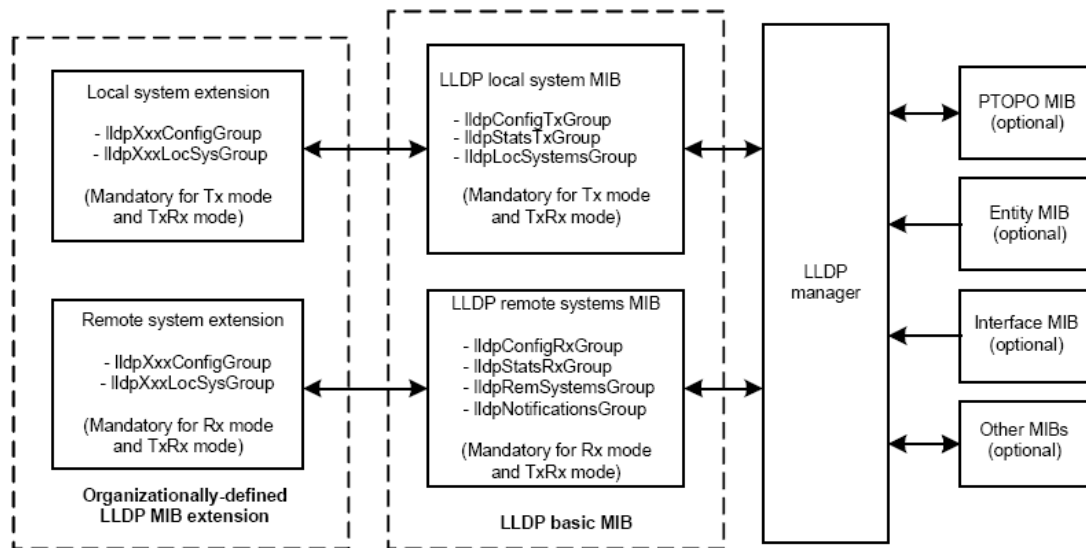


Figure 3 - LLDP MIB Block Diagram [3]

The information will be aged and removed when no update to the information has been received for a period of time. This is to ensure that only valid information is present in the LLDP MIBs.

Since the information is based on periodically advertised information about the nodes in the network, the 802.1ab has solved some of the problems existing in the exploration of FDBs.

Once this protocol is present in all nodes in the network, a layer 2 topology can be created by retrieving and ordering the information given in the LLDP MIBs.

3.10 PostgreSQL

When dealing with datasets of a reasonable size it will often be an advantage to place the data in a database. PostgreSQL is one example of a database using the Structured Query Language (SQL). PostgreSQL (originally Postgres) started as a research project at the University of California, Berkeley and has evolved to being the most advanced open source database as of today [10].

A few interesting capabilities of PostgreSQL will be briefly described.

3.10.1 Views

Views are pseudo-tables. That is, they are not real tables, but conform to a SELECT statement in the ordinary way. A view can represent a subsection of a real table, a full real table (should anybody need that) or joined tables.

Since a view is not a real table, UPDATE, DELETE and INSERT statements have no effect on a view. This behaviour can be changed by a concept called “rules”.

Views are available in the MySQL database as well from version 5.0 (production release 19th of October 2005).

3.10.2 Rules

Rules allows for predefined actions to take place when a table or view is accessed. A most useful property of rules is that they can manipulate the original SQL query to the table and execute the manipulated query.

This enables views to accept UPDATE, DELETE and INSERT statements, and real table to perform an UPDATE instead of an INSERT in case of the entry already existing.

Rules are not yet available in the MySQL database.

3.10.3 Functions

Functions (or stored procedures) are pieces of code to be executed server side when used in a query. A function may be given one or more arguments and return a result which is then placed in the query instead of the function.

Functions are available in the MySQL database from version 5.0.

4 Design

4.1 The technologies chosen

Relying on traffic counters, parsing of configuration, and processing delays seems to be a fairly complicated, unstable and unpredictable way to implement anything. All of the above are heavily dependent on the environment in which they are sought brought to use and are therefore deemed as the wrong path to take.

The 802.1ab is absolutely the way to go in layer 2 network topology discovery, but it will probably be a while before the majority of nodes (and devices) are shipped with an implementation of this protocol. This is probably why to the knowledge of the author any applications utilizing this protocol have not been built. The protocol will for the same reason not be sought implemented in this project.

With respect to Microsoft's "Discovery without Network Assistance", since this technique requires all devices to participate in the discovery by implementing the suggested algorithms, it has been judged as unsuitable for this project.

STP, FDB and CDP information will definitely give some useful information, but it might not give a complete 100% accurate layer 2 topology for the network. It is, however, the best approach available at the moment and is therefore the technologies this project will focus on.

SNMP is the way to retrieve information from nodes and devices in the network, and will be the basis of this project.

For database, PostgreSQL is chosen in favour of MySQL since PostgreSQL supports rules and PostgreSQL is the database used by the monitoring system "d-mon".

4.2 Forwarding database

In order for a switch to do any switching, it must be aware of how other nodes and devices in the network are reached.

The switch gathers this information by analysing the packets it receives. When a packet arrives at the switch, the FDB is updated with an entry mapping the source address of the packet to the interface on which the packet arrived. If the destination of the packet is found in the FDB the switch forwards the packet to that interface. If the destination is unknown i.e. not found in the FDB, the switch broadcasts the packet to all ports. Once the destination device sends a response, the switch will know which interface the device is located at and insert an entry in the FDB.

When an entry in the FDB has not been in use for a defined amount of time (typical default of 300 seconds), the entry is purged from the FDB as mentioned in section 3.5.

It is important to keep in mind that only nodes and devices on the same broadcast domain communicate directly by layer 2 i.e. MAC addresses. When a cross-broadcast domain communication occurs the third layer is responsible for having the communication pass through a router. This router or gateway must have a layer 2 address on the same

broadcast domain as the sender, and it is thus this gateway that is the destination for the packets on the broadcast domain. The same applies to the receivers end.

Some nodes are routing-switches. These are switches that have the capabilities to route packets based on the layer 3 address but still act as a switch. Nodes connected to a routing-switch where the routing engine is active for the broadcast domain of which the connecting interface is taking part, will have the MAC address of the interface of the routing-switch in the nodes FDB.

In an Ethernet network, MAC addresses worldwide must be unique. This enables vendors of switches and routing-switches to keep only one FDB even though the node is part of many different broadcast domains. A MAC address should never appear on two different broadcast domains and thus, a single FDB can be used. Vendors may include other information in the FDB than just a MAC address and an interface. For instance the VLAN on which the interface belongs, or the MAC was seen may be kept as well enabling operators to have different ageing times depending on VLAN.

A few exceptions to this rule of uniqueness do exist: Cisco's Hot Standby Routing Protocol (HSRP) utilises what is called "Virtual Layer 2 Addresses". These are defined in RFC 2281 (Cisco HSRP) as 00:00:0C:07:AC:XX where XX represents the HSRP group number, in case of the layer 2 network being anything else than Token Ring. Similar for the Virtual Router Redundancy Protocol (VRRP) defined in RFC 3768, the address group of 00:00:5E:00:01:XX where XX represents the VRRP Virtual Router Identifier, is reserved for VRRP operation and can as such exist on any given broadcast domain at the same time. Such exceptions must be known in order to filter these addresses from the dataset. The first three octets in the VRRP addresses are derived from the block of addresses assigned to IANA [14]. IANA also owns the block of 01:00:5E of which the first half is used for multicast addresses. Since IANA is not producing any NICs, these blocks may just as well be filtered from the list.

Karl Auerbach who has been a long-time member of the Internet Engineering Task Force⁴ has compiled a list [15] of these and other MAC addresses which cannot be expected to be unique.

In order to utilise the forwarding databases to find layer 2 links with 100% certainty between nodes, one would need the FDBs to be complete at the time of investigation.

One could argue that in order to do a valid match, tables do not necessarily need to be populated with all possible information from the network.

Observing a network with two neighbouring nodes, it is assumed that Node 1 holds complete information of Node 2 i.e. the FDB of Node 1 is complete with respect to the FDB on Node 2.

This alone narrows the possible Node 2 to nodes having at least the FDB entries that is present at the interface in question on Node 1. In a tree-like network this will usually be

⁴ Mr. Auerbach has also been the North American representative on the ICANN Board of Directors and a former senior researcher in the Advanced Internet Architecture group in the Office of the Chief Strategy Officer at Cisco Systems.

nodes further down in the tree. That may still be a lot of nodes and definitely a lot of different interfaces on these nodes. A further requirement is needed:

- An interface on Node 2 needs to hold the entries matching at least one of the entries on other interfaces than the one in question on Node 1.

The completeness requirement is approached but not to extent. The requirement only states that Node 2 holds a limited amount of information of Node 1. This ensures that neither of the interfaces in question points to each other for the same address, and thereby excludes a lot of interfaces and nodes.

A few additional requirements must be adhered to however. In order for Node 2 to avoid being linked to a node “higher up the tree” than Node 1 (should such a node exist and have no information besides what is reachable via Node 2), Node 2 needs to hold one of two types of entries:

- Node 2 must have at least one entry for an interface on Node 1 which is not an aggregated (uplink) interface (i.e. a MAC directly accessible from Node 1) or
- Node 2 must have at least two entries which are on two different interfaces on Node 1.

For both requirements it is required that the interface in question on Node 1 has to be another interface than the one connected to Node 2.

The completeness requirement has by now been relaxed into a completeness requirement for one direction and a minimum requirement for the other direction.

This method does have some limitations however. The method has been sought developed so that it would produce as few false positives as possible. Even if that means fewer correct links as well. Unfortunately a few scenarios will still make the method produce false positives.

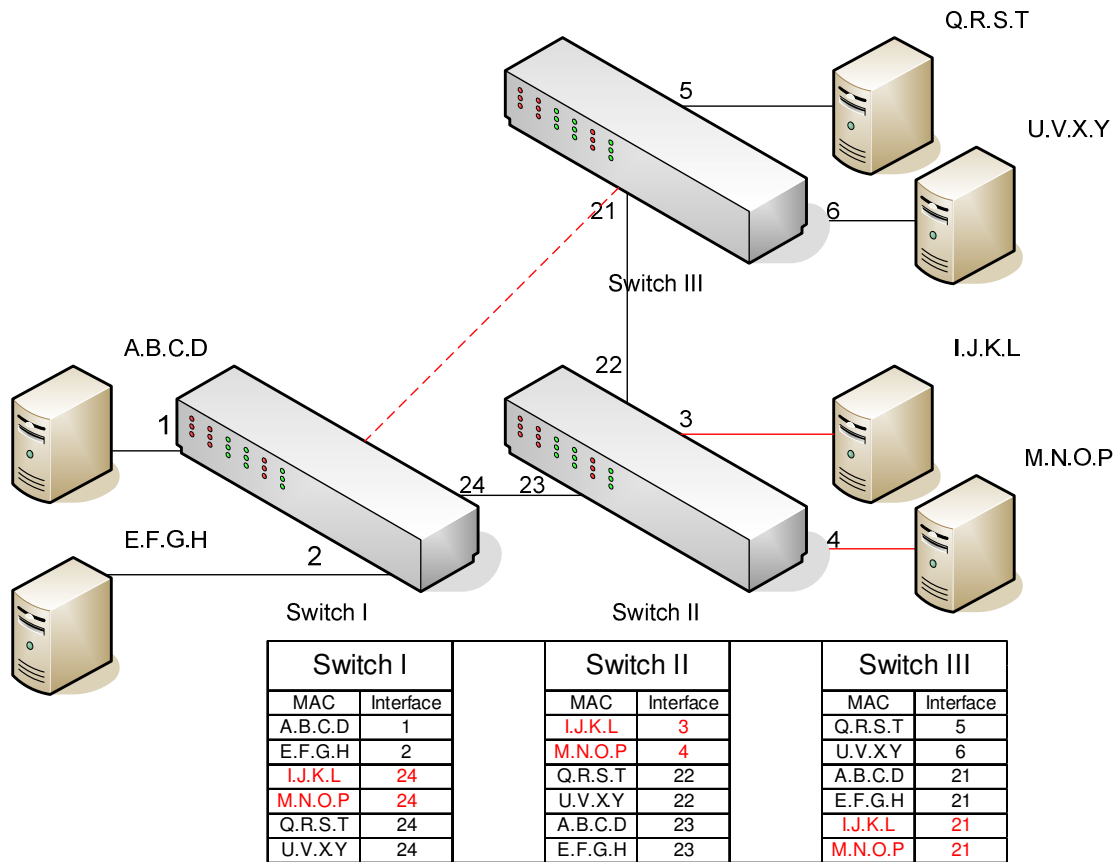


Figure 4 - Problem scenario 1

Observing the network in Figure 4 it is seen that three switches are connected to each other as well as six devices. The layer 2 addresses of the devices are written next to them. The devices having the addresses I.J.K.L and M.N.O.P are either turned off or have not been transmitting in the network long enough for the entries to be removed from all three switches.

In this scenario the two original links between the switches will be found but a third false link between Switch I and Switch III will be found as well.

The scenario in Figure 5 displays the same situation in a less extreme way. Here it is only Switch I that is unaware of the devices connected to Switch II but this still results in the false link between Switch I and Switch III.

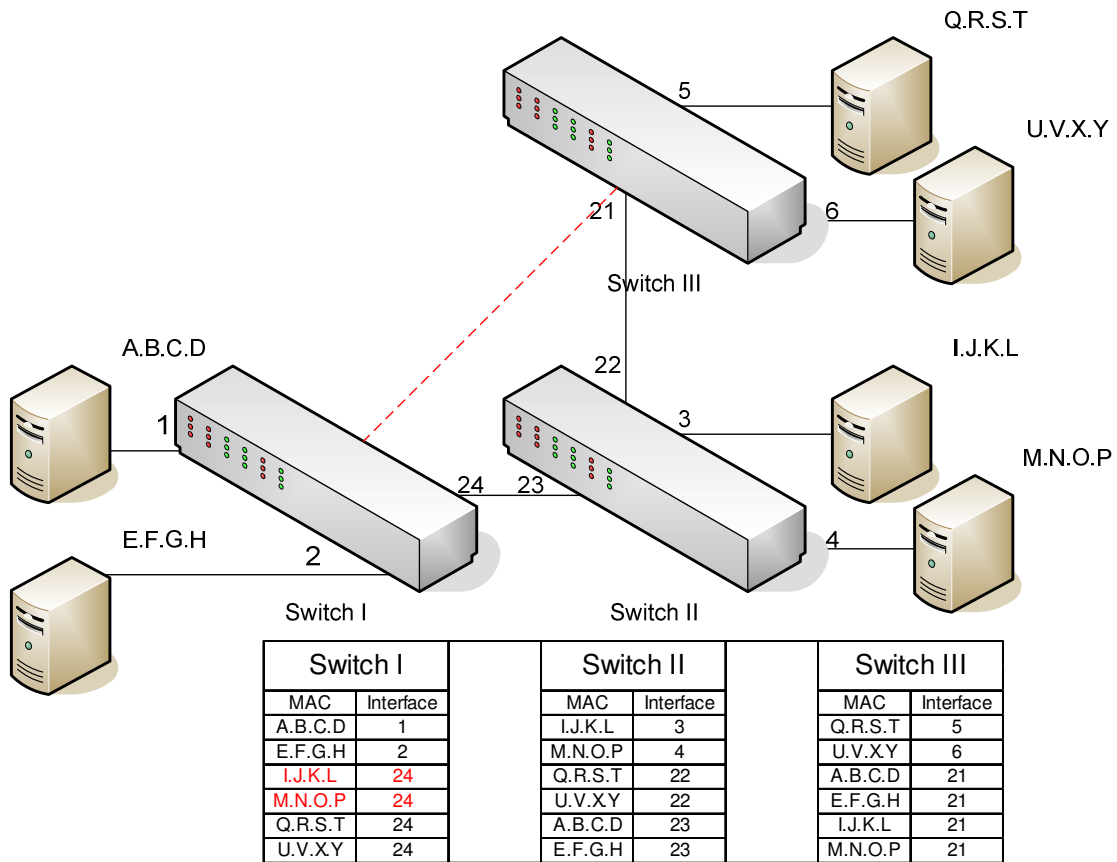


Figure 5 - Problem scenario 2

In the scenario in Figure 6, Switch II has no knowledge of the devices connected to Switch I. It cannot take the role of Node 1 since it does not have full information of Switch 1, and does not conform to any of the two last listed requirements either, which results in the link between Switch I and Switch II not being found.

In case of all nodes having complete information a link will be found from node X to node Y when examining node X, and from node Y to node X when examining node Y. Due to the requirements set up above it is expected that in real-life networks only one link will be found. If both links are found one could just as well apply the completeness requirement since this removes the risk of false positives.

Even though the method can produce false positives it is considered “good enough”^(TM). The method fails and produces a false link in the problem in Figure 5, but if the FDB of Switch I is incomplete, chances are so will the FDBs of Switch II and/or Switch III be. If Switch III has no knowledge of the devices connected to Switch I, the false link will not be found but the two right links will be.

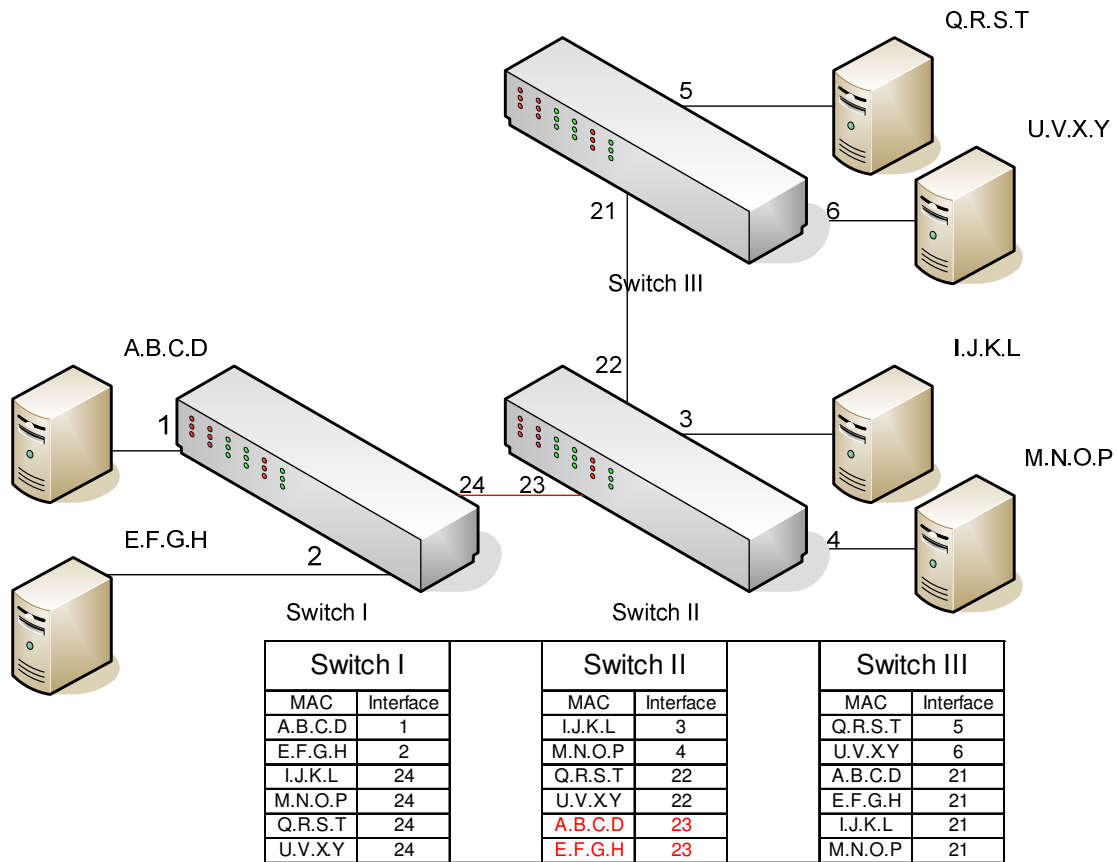


Figure 6 - Problem scenario 3

The presence routing-switches pose yet a challenge. The challenge is the same as for nodes offering a layer 2 tunnel to another node for traffic to pass through. If such nodes exist in the network, interface MAC addresses may be present in other nodes and devices. An interface MAC may literally be the MAC address of the physical interface on the node a MAC address of the tunnelling interface, or a MAC address of the VLAN interface in case of a inter-VLAN routing-switch.

There is no guaranteed way of telling whether the network contains routing-switches (nodes may be queried by the *ipr_info()* object of *SNMP::Info*, but this gives no guarantee). If such nodes do exist the interface MACs should be taken into account when doing the analysis – if such nodes do not exist it cannot be required for any node to have complete information of another node including interface MACs since these will not be in active use and thus not be present in the tables.

Layer 2 tunnels that work by providing bridging between the tunnel ends will not bring any new problems to the method since they will be treated like any other bridge interface on the node.

4.3 Spanning Tree

Utilising the information from the STP is fairly straight forward. The nodes hold the following information which is to be retrieved and stored:

- The layer 2 address of the node
- The bridge identifier which can be converted to an interface identifier by performing a lookup in the associated table.
- The layer 2 address of the designated bridge (node)
- The designated interface i.e. the bridge identifier of the interface on the designated bridge to which the interface on this node is connected

Once the information is retrieved from the devices and stored in the database there is little left to do in terms of data analysis in order to be provided with a set of nodes and links.

4.4 Cisco Discovery Protocol

The data available from the CDP MIB is very similar to the STP data. Because of this the same approach can be used for the CDP data as well. The following information is to be retrieved and stored:

- The layer 2 address of the node
- The interfaces associated with and the layer 3 addresses of each neighbouring node

Once this information is stored the layer 2 addresses of the neighbouring nodes can be found by a lookup in the database. The interface of the neighbouring node can be found as well when assuming that each neighbouring pair is connected with one connection only. That is, interfaces on the same node can not have the same neighbouring node.

5 Implementation

5.1 Database layout

public.stp_data	
guid	serial
systemobj	bool
created	timestampz
modified	timestampz
deleted	timestampz
nodeid	int4
b_port	text (-1)
port	text (-1)
desig_bridge	text (-1)
desig_port	text (-1)
root_bridge	text (-1)

public.graph_links	
guid	serial
systemobj	bool
created	timestampz
modified	timestampz
deleted	timestampz
node_a	int4
port_a	text (-1)
node_b	int4
port_b	text (-1)

public.nodes_imac	
guid	serial
systemobj	bool
created	timestampz
modified	timestampz
deleted	timestampz
nodeid	int4
imac	text (-1)
port	text (-1)

public.nodes	
guid	serial
systemobj	bool
created	timestampz
modified	timestampz
deleted	timestampz
mac	text (-1)
ip	text (-1)
community	text (-1)

public.cdp_data	
guid	serial
systemobj	bool
created	timestampz
modified	timestampz
deleted	timestampz
node_a_ip	text (-1)
port_a	text (-1)
node_b_ip	text (-1)

public.fdb_data	
guid	serial
systemobj	bool
created	timestampz
modified	timestampz
deleted	timestampz
port	text (-1)
fdb_mac	text (-1)
nodeid	int4

public.nodes_alias	
guid	serial
systemobj	bool
created	timestampz
modified	timestampz
deleted	timestampz
mac	text (-1)
ip	text (-1)

public.object	
guid	serial
systemobj	bool
created	timestampz
modified	timestampz
deleted	timestampz

Figure 7 - Database layout, tables

The database structure is built upon the tables listed in Figure 7. The *public.object* table is inherited in all other tables giving each table the parameters listed in this table.

When nodes are scanned, they are stored in *public.nodes*. In case of the scanned node being an alias of a previously scanned node, this alias information is stored in *public.nodes_alias*. This enables the topology map to give information about all layer 3 addresses associated with any given node.

The layer 2 addresses of the interfaces in the node scanned are stored in *public.nodes_imac* to be able to perform a matching using this information.

The *public.fdb_data*, *public.stp_data* and *public.cdp_data* tables hold the information gathered from the FDB, the STP database and the CDP database of the scanned node.

The table *public.graph_links* is the table where the links calculated via the FDB method are stored.

public.all_graph_links	
node_a	int4
ip_a	text (-1)
mac_a	text (-1)
port_a	text (-1)
node_b	int4
ip_b	text (-1)
mac_b	text (-1)
port_b	text (-1)
type	text (-1)

public.cdp_graph_links	
node_a	int4
ip_a	text (-1)
mac_a	text (-1)
port_a	text (-1)
node_b	int4
ip_b	text (-1)
mac_b	text (-1)
port_b	text (-1)

public.fdb_graph_links	
node_a	int4
ip_a	text (-1)
mac_a	text (-1)
port_a	text (-1)
node_b	int4
ip_b	text (-1)
mac_b	text (-1)
port_b	text (-1)

public.stp_graph_links	
node_a	int4
ip_a	text (-1)
mac_a	text (-1)
port_a	text (-1)
node_b	int4
ip_b	text (-1)
mac_b	text (-1)
port_b	text (-1)

public.fdb_nodes	
nodeid	int4
port	text (-1)
count	int8

public.fdb_list_ports_count_2	
port	text (-1)
nodeid	int4

Figure 8 - Database layout, views

The *public.fdb_graph_links*, *public.stp_graph_links* and *public.cdp_graph_links* are views listing the links found with each method. The *public.all_graph_links* view unions the three views into one view for graphing with *graph.pl*.

The view *public.fdb_nodes* lists the nodes and interfaces with only one entry in the FDB for that interface. This view is used in the *u-fdb-single.pl* script to find router-to-router “point-to-point” links.

The view *public.fdb_list_ports_count_2* is used to isolate the nodes with FDB entries on a defined minimum of interfaces. It is used in the *u-fdb-portwise.pl* where a link can only be found if FDB data for more than one interface exist.

A few functions have been created as well:

- *public.mac_to_guid*: This function resolves a MAC address into a nodeid (*guid* in *public.nodes*).

- *public.mac_to_ip*: This function resolves a MAC address into an IP address
- *public.b_port_to_port*: This function resolves a bridge port as seen in the STP information into a true interface identifier. It is used in the *public.stp_graph_links* view in order to give the correct interface identifier from the interface table and not the identifier from the bridge table.

Finally a rule has been defined in each table (except *public.object*) to ensure that an INSERT statement with values already existing in the table is modified to an UPDATE statement updating the *modified* column with a new timestamp.

This allows for the data to be aggregated over time by limiting the SELECT statements to select only rows modified after a certain point in time.

5.2 Scripts

Several scripts have been created in order to do a proof-of-concept implementation of the design.

5.2.1 s-*

The scripts starting with *s-* are scripts doing a scan of nodes. The scan is preformed by retrieving information from the nodes via SNMP. A perl module `SNMP::Info` [11] is used to provide an object oriented interface to the SNMP. This perl module is responsible for handling all vendor specific specialities thereby presenting a common interface to the application in which it is used, no matter the equipment connected to.

5.2.1.1 s-iphavest.pl

The *s-iphavest.pl* script is utilising the CDP-information stored in devices to find IP addresses of nodes. One node and a (set of possible) community(ies) is needed as a seed, and the script will connect to that IP address and retrieve the IP addresses of the CDP neighbours. This set of addresses is then connected to and the IP addresses of the CDP neighbours of all of these nodes are retrieved. This continues and eventually no new addresses are discovered.

The addresses found for each device is dumped in an output file (using the `-o` option) for parsing by either *parse-iphavest.pl* or *u-cdp.pl*.

This script is a modified version of an original script made for the “*d-mon*” product.

5.2.1.2 s-stpinfo.pl & s-fdbinfo.pl

The STP information is retrieved from each node passed to the *s-stpinfo.pl* script and stored in the *public.stp_data* table. The FDB is retrieved from each node passed to the *s-fdbinfo.pl* script and stored in the *public.fdb_data* table.

Both scripts store information of each node in the *public.nodes* table. In case of the node checked being an alias for a previously checked node, the node information is stored in the *public.nodes_alias* table. Finally, all the interface layer 2 addresses of the node are stored in the *public.nodes_imac* table.

The scripts read the nodes to scan and the community to use for each node from an input file which can be generated from *s-ipharvest.pl* and *parse-ipharvest.pl* or created manually.

5.2.2 parse-ipharvest.pl

The output file generated by “*s-ipharvest.pl*” is parsed by this script in order to provide a list if the format <IP>,<community> which is used in *s-stpinfo.pl* and *s-fdbinfo.pl*.

5.2.3 u-*

The *u-* scripts provides the analysis needed in order to bring conclusions from the data gathered by the *s-* scripts as to where links exist. They either fetch the required data from the database or from files. They analyse the data and store the resulting links in the database.

Notice that there is no *u-stp*-file. Such a script is not needed for STP-data since all analysis can be done by the database on the fly in the view *public.stp_graph_links*.

5.2.3.1 u-fdb-portwise.pl

The method of finding links by analysing incomplete FDB data is implemented in *u-fdb-portwise.pl*. The reason for this naming of the script is that the method was first tried implemented by means of a MAC-wise approach meaning that the implementation tried to find links by analysing each MAC address. The idea was to find all possible links with offset in any given MAC address and evaluate on the sum of possible links in the end. This implementation strategy proved to be hard to get right and a new implementation relying on an interface-wise approach was commenced.

This proved to be a much easier and faster way of implementing the method.

The basic flow of the script is shown in Figure 9.

The reason for returning to Step III from Step V is found in the recognition of the weakness in the method that false links may occur. If a return to Step I was made after storing the link-information for the link just found in the database (Step V) there would be no chance of other false interfaces being found as well, removing the last resort of informing the user that a problem with the validity of this link exists.

With respect to interface addresses, and whether to include them or not, an option can be given to the script and it will either include or exclude them based on the value of the option.

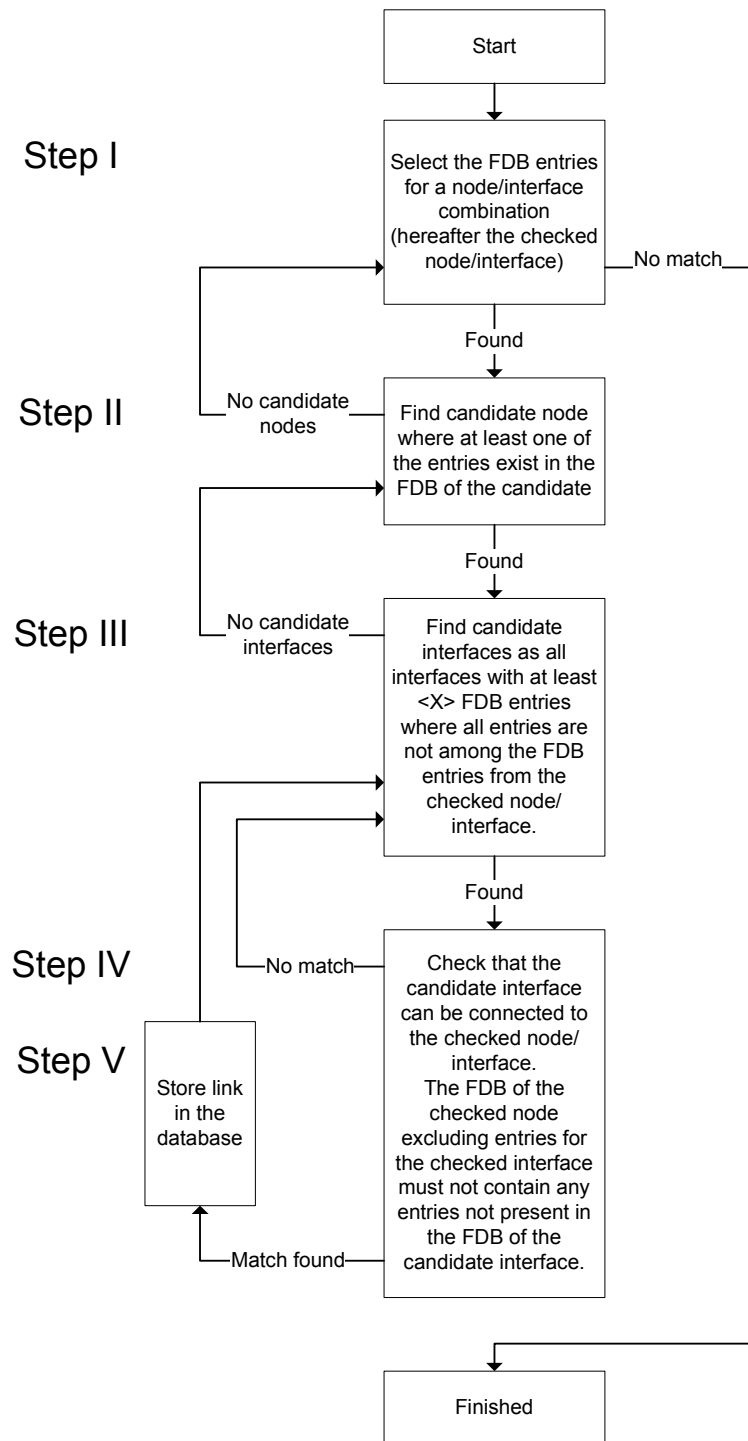


Figure 9 - Flow diagram of u-fdb-portwise.pl

5.2.3.2 u-fdb-single.pl

The purpose of this script is to find point-to-point links between routers in the network. Such links will be characterized by FDBs like the ones in Figure 10. The script will find nodes/interface combinations with only one FDB entry. This entry is then sought in the

public.nodes_imac table. The matching node/interface is then checked in order to establish that it as well only has one FDB entry, and that the entry matches the address of the first interface.

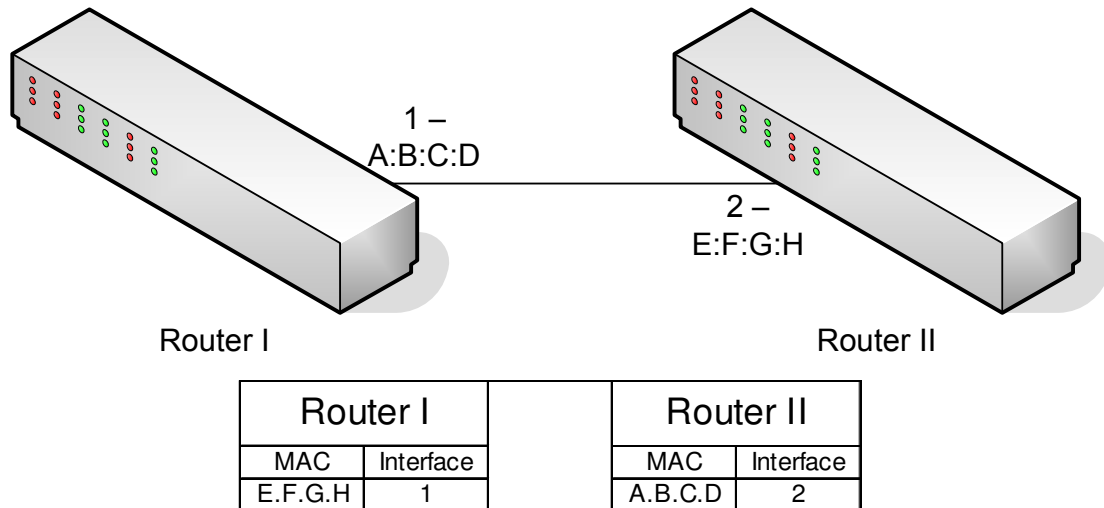


Figure 10 - Point-to-point link between routers

5.2.4 u-cdp.pl

The output file generated by “*s-ipharvest.pl*” is parsed by this script in order to find links by analysing the information retrieved from the CDP MIB. This prototype-script does not store node information in the database, and is thus required to be run after *s-fdbinfo.pl* or *s-stpinfo.pl*. The deeper analysis for the generation of links is done by the database in the appropriate view *public.cdp_graph_links*.

5.2.5 graph.pl

The topology map is graphed by the perl module [12] of Graphviz [13].

Since all data describing the links is stored in the database, *graph.pl* is only providing the means to represent this data in an organized behaviour.

The script will graph the topology using the information found via the three different methods or using only information from one of the three. For topologies made from all three methods, links will be coloured according to the method with which they were found. In case of more than one method giving any particular link, only one link will be drawn but the colour of the link will be changed to black.

In case of a possible conflict i.e. two links connected to the same interface on the same node both links will be marked by drawing the links bold.

In case of a method being unable to determine the precise interface, which can be a problem in some hardware versions from some vendors, the link will be drawn with a dashed line instead of a solid line. Furthermore the link will not be aggregated into a common link even if other “good” links between the same nodes exist.

6 Test

The interesting part to test is the *u-fdb-portwise.pl* script and the method to find links by analysing FDB data.

The network in Figure 4 (p. 15) was manually stored in the database with all FDBs being complete, and the *u-fdb-portwise.pl* script was executed. With the completeness requirement fulfilled, all links were returned twice – one in each direction – as can be seen in Figure 11. This was as expected.

	node_a	ip_a	mac_a	port_a	node_b	ip_b	mac_b	port_b
1	1	SWITCH I	00:0e:7f:8e:f3:40	24	140	SWITCH II	00:14:c2:2c:d6:40	23
2	140	SWITCH II	00:14:c2:2c:d6:40	22	270	SWITCH III	00:0e:7f:5a:c2:40	21
3	140	SWITCH II	00:14:c2:2c:d6:40	23	1	SWITCH I	00:0e:7f:8e:f3:40	24
4	270	SWITCH III	00:0e:7f:5a:c2:40	21	140	SWITCH II	00:14:c2:2c:d6:40	22

Figure 11 - Test result on complete FDBs

The FDBs were then changed into the FDBs from Figure 5 (p. 16) by deleting the two red entries from Switch I. The *public.graph_links* table was truncated and *u-fdb-portwise.pl* executed once again.

	node_a	ip_a	mac_a	port_a	node_b	ip_b	mac_b	port_b
1	1	SWITCH I	00:0e:7f:8e:f3:40	24	140	SWITCH II	00:14:c2:2c:d6:40	23
2	1	SWITCH I	00:0e:7f:8e:f3:40	24	270	SWITCH III	00:0e:7f:5a:c2:40	21
3	140	SWITCH II	00:14:c2:2c:d6:40	22	270	SWITCH III	00:0e:7f:5a:c2:40	21
4	270	SWITCH III	00:0e:7f:5a:c2:40	21	1	SWITCH I	00:0e:7f:8e:f3:40	24
5	270	SWITCH III	00:0e:7f:5a:c2:40	21	140	SWITCH II	00:14:c2:2c:d6:40	22

Figure 12 - Test result on FDBs not conforming to the requirements

Since the FDBs do not conform to the two requirements set up, false positives are returned in the link-base. The red dashed line in Figure 5, the false non-existing link, is found in the result as row 2 and row 4.

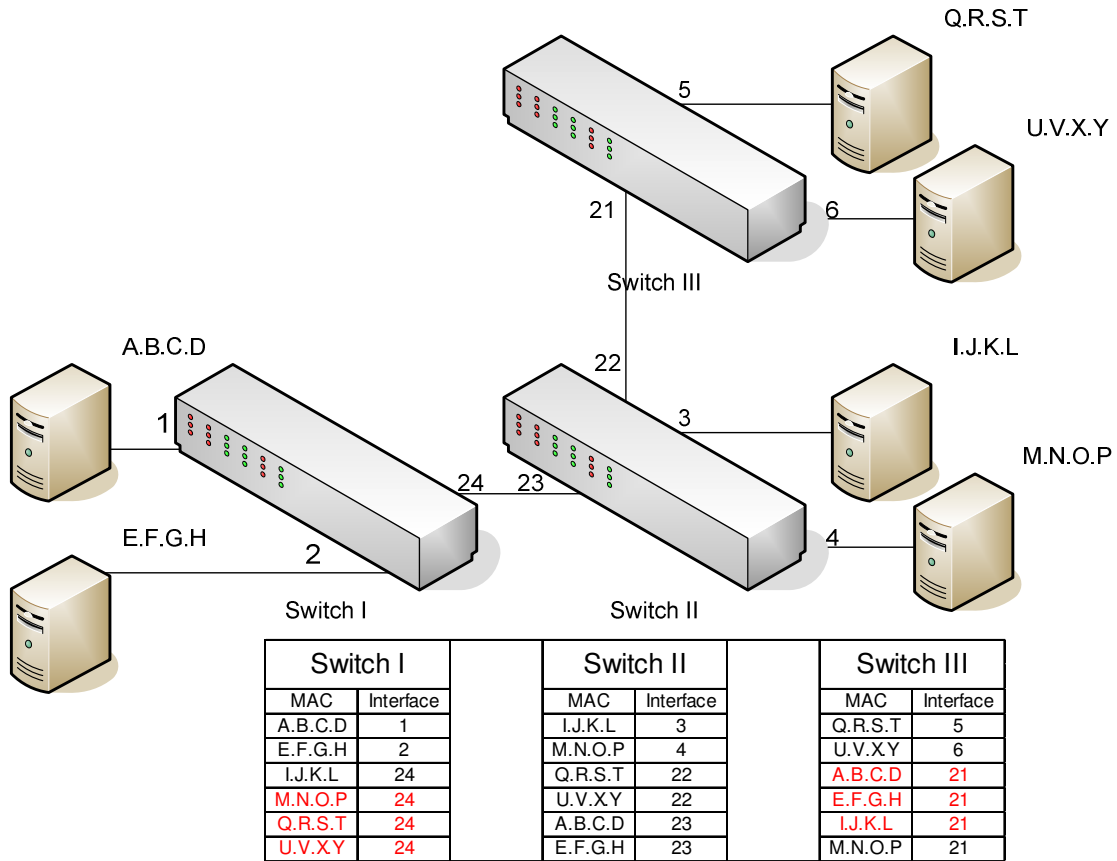


Figure 13 - Test with lots of missing information

The FDBs in Figure 13 are certainly not conforming to the completeness requirement but they do conform to the requirements set up in section 4.2 (p. 12).

	node_a	ip_a	mac_a	port_a	node_b	ip_b	mac_b	port_b
1	1	SWITCH I	00:0e:7f:8e:f3:40	24	140	SWITCH II	00:14:c2:2c:d6:40	23
2	270	SWITCH III	00:0e:7f:5a:c2:40	21	140	SWITCH II	00:14:c2:2c:d6:40	22

Figure 14 - Result of test with missing information

The result is that as long as the requirements set up are adhered to the method will return the links as shown in Figure 14.

7 Results & Discussion

The scripts developed are able to collect and analyze data from nodes using SNMP and PostgreSQL.

VLANs are handled in the way that their presence is mostly ignored since no VLAN information is given by the FDB in the nodes available for test. Since MAC addresses are unique, this is not a problem as the FDB method holds the minimum requirement which must be met. Only interfaces in the proper VLAN will be able to meet that requirement. The interface MAC of all interfaces is stored in the database. This includes the interface MACs for VLAN interfaces in case of routing-switches, which can be either included or excluded in the analysis as mentioned in section 4.2 (p. 12).

7.1 Real-life networks

The links generated are stored in the database and a script is available for graphing the topology. This script takes into account by which method the individual links were found, and passes this information on to the user enabling the user to judge whether to trust that a given link exist.

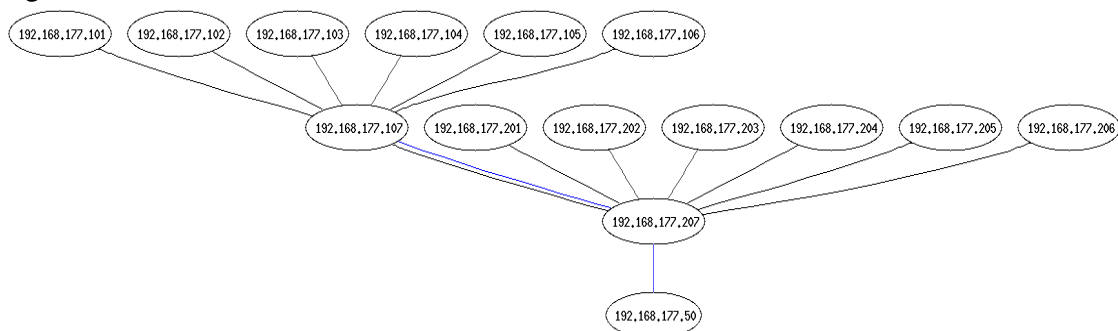


Figure 15 - All links returned for test network

Figure 15 is the result of the scripts being run on the network of P.O. Pedersen Kollegiet in Kgs. Lyngby. The network is built as two wiring closets each with 7 nodes. Between the two wiring closets two links exist for redundancy. From one wiring closet there is link to the server room where an additional node is placed.

As can be seen from the graph all links are found. The black links are found by more than one method and since only the FDB and STP method was used, all links are found by analysing the STP data (blue links) and nearly all links were found by analysing the FDB data (yellow links). As mentioned, once a link is found by more than one method the colour is changed to black.

It is interesting to see that the STP find the redundant link not currently in use between the wiring closets whereas the FDB only finds the link currently in use.

One potential user of the topology discovery method is the NNIT company. The network they manage at Novo Nordisk is comprised of about 1.600 nodes. Running the scripts on the management network at Novo Nordisk does give some links, but not nearly as many

as expected. The STP and CDP links found are valid, but very much grouped together in unconnected groups. Less than 30 links is found using the FDB method, but almost all of them are validated by a matching STP or CDP link.

When analysing the raw data it is noticeable that 1.600 nodes only return about 10.400 FDB entries whereas the 15 nodes at P.O. Pedersen Kollegiet return just over 1.100 FDB entries.

The nodes at P.O. Pedersen Kollegiet is HP 2524 switches whereas the nodes at Novo Nordisk is a lot of different primarily Cisco equipment. It has been observed that nodes return 0 FDB entries which points to a problem reading the FDB of the nodes.

It is the author's belief that the reason of the more incomplete result for the Novo Nordisk network is due to the data not being sufficient.

Regarding speed, the 15 links at P.O. Pedersen were analysed in a matter of seconds. For the links at Novo Nordisk, the `u-fdb-portwise.pl` completes it's analysis in about 3-10 minutes depending on processor speed, memory configuration etc.

7.2 Further development

It would be feasible to be able to select a subset of the nodes to be graphed based on layer 3 address. For networks of more than a handful of nodes, it is seldom interesting to receive a map of all nodes and layer 2 links.

Likewise, the data returned by `graph.pl` could include a HTML map enabling the user to click a node and receive information of layer 3 alias addresses, layer 2 addresses etc.

Clicking on a link could then bring up information about the interfaces and nodes that particular link is connected to, and information about by which method(s) the link was found.

All the information required to do so is available in the database.

The scripts could be adjusted in order to enter the d-mon monitoring system. The database structure is founded in the one used by d-mon in order to make this task as easy as possible.

Nearly all scripts could be rewritten in order to execute in a parallel structure instead of a sequential structure. This will limit the execution time to a minimum and enable the implementation to scale to very large networks.

To be on top of things and in order to keep parallelisation problems from interfering with the methods in development this has not been a priority.

Based on the topology and the information gathered from the FDBs, a device location application could be designed. The FDBs provide information of where any given MAC address is seen. When excluding the interfaces found as connecting to other nodes, it will be possible to pinpoint locate the layer 2 address of a device to a given node and interface.

8 Concluding remarks

This project has aimed at finding and implementing methods for doing topology discovery at layer 2 in Ethernet networks.

- The methods developed are very generic and can be used on any given network comprised of nodes from different vendors.
- The resulting topology can be drawn with details on how each link was found.
- The current implementation of the method does have it's limitations with respect to scalability due to the sequential execution.

Based on the above I find the project to have been a success with room for improvement. catpipe Systems are satisfied with the results achieved so far and I will continue to work with them at improving the methods in order for the methods to be implemented in the monitoring system "d-mon".

January 20th, 2006

THIS PAGE WAS INTENTIONALLY LEFT BLANK

9 Bibliography

- [1] "Topology Discovery for Large Ethernet Networks". Bruce Lowekamp et al., College of William and Mary Williamsburg, VA, USA.
- [2] "Topology Discovery in Heterogeneous IP Networks". Yuri Breibart et al., Bell Labs, Lucent Technologies, NJ, USA.
- [3] "802.1AB-2005. IEEE Standard for Local and metropolitan area networks. Station and Media Access Control Connectivity Discovery". IEEE.
- [4] "Ethernet Topology Discovery without Network Assistance". Richard Black et al., Microsoft Research, UK.
- [5] "Troubleshooting ... IOS". Cisco Systems Inc.
<http://www.cisco.com/warp/public/473/162.html>
- [6] "Frame Formats". Cisco Systems Inc.
<http://www.cisco.com/univercd/cc/td/doc/product/lan/trsrb/frames.htm>
- [7] "Release Notes for HP Procurve Switches". Hewlett-Packard Company.
ftp://ftp.hp.com/pub/networking/software/59692375_e1.pdf
- [8] "Analysis, Implementation and Enhancement of Vendor dependent and independent Layer-2 Network Topology Discovery". Alexander Barthel. Chemnitz University of Technology.
- [9] "SNMP Overview & SNMP Security". SolarWinds.net.
<http://www.solarwinds.net/Tools/SNMP.htm>
- [10] "PostgreSQL - Introduction and Concepts". Bruce Mornjian. ISBN 0-201-70331-9.
- [11] SNMP::Info. Max Baker & Eric Miller.
<http://snmp-info.sourceforge.net/>
- [12] GraphViz Perl Module. Leon Brocard.
<http://search.cpan.org/dist/GraphViz/>
- [13] Graphviz - Graph Visualization Software.
<http://www.graphviz.org>
- [14] "Ether Types". IANA.
<http://www.iana.org/assignments/ethernet-numbers>
- [15] "Multicast (including Broadcast) Addresses". Karl Auerbach.
<http://www.cavebear.com/CaveBear/Ethernet/multicast.html>
- [16] Image Copyright, Lån & Spar Bank.
<http://www.lsb.dk>