EBU – Tech 3346

End-to-End IP Network Measurement for Broadcast Applications

EisStream Software package description

Geneva
May 2011

Conformance Notation

This document contains both **normative** text and **informative** text.

All text is normative except for that in the Introduction, any section explicitly labelled as ‘Informative’ or individual paragraphs that start with ‘Note:’.

Normative text describes indispensable or mandatory elements. It contains the conformance keywords ‘shall’, ‘should’ or ‘may’, defined as follows:

‘Shall’ and ‘shall not’:
(mandatory)

Indicate requirements to be followed strictly and from which no deviation is permitted in order to conform to the document.

‘Should’ and ‘should not’:
(recommended)

Indicate that, among several possibilities, one is recommended as particularly suitable, without mentioning or excluding others.

OR indicate that a certain course of action is preferred but not necessarily required.

OR indicate that (in the negative form) a certain possibility or course of action is deprecated but not prohibited.

‘May’ and ‘need not’:
(optional)

Indicate a course of action permissible within the limits of the document.

Default identifies mandatory (in phrases containing “shall”) or recommended (in phrases containing “should”) presets that can, optionally, be overwritten by user action or supplemented with other options in advanced applications. Mandatory defaults must be supported. The support of recommended defaults is preferred, but not necessarily required.

Informative text is potentially helpful to the user, but it is not indispensable and it can be removed, changed or added editorially without affecting the normative text. Informative text does not contain any conformance keywords.

A conformant implementation is one that includes all mandatory provisions (‘shall’) and, if implemented, all recommended provisions (‘should’) as described. A conformant implementation need not implement optional provisions (‘may’) and need not implement them as described.

Contents

Page is deliberately blank

End-to-End IP Network Measurement for Broadcast Applications

EisStream software package description

EBU Committee	*First Issued*	*Revised*	*Re-issued*
EC-N	2011		

Keywords: Internet Protocol, Network Management, Broadcasting, MIB, EisStream

1. Introduction

In recent years, EBU Members have been increasingly adopting IP networks for the contribution of audio and video in real-time. It is well known that although IP networks are of lower cost and provide more flexibility compared with circuit switched networks, they suffer from longer delays and have much larger jitter, while broadcasters' tolerance to these variables is much less than that of normal business IT traffic.

To respond to Members' use of IP the EBU set up two groups, ECN-ACIP (Audio contribution over IP) and ECN-VCIP (Video contribution over IP) with the tasks of drawing up recommended codes of practice[1].

It was also recognised that there would be a strong demand for tools that would enable broadcasters to measure and manage their IP networks properly to suit the many time-critical broadcast applications they would be subjected to. To this end, the ECN-IPM (IP measurement) group was set up. The relationships between these three groups are shown in the following figure.

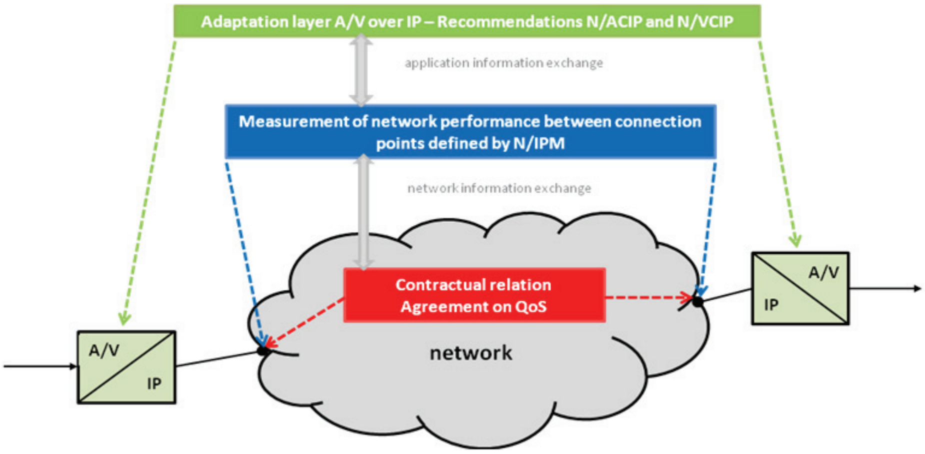

Adaptation layer A/V over IP – Recommendations N/ACIP and N/VCIP
application information exchange
Measurement of network performance between connection points defined by N/IPM
network information exchange
Contractual relation Agreement on QoS
network
A/V
IP
A/V
IP

Figure 1: Relationships between ECN groups ACIP, VCIP and IPM

The goals of the ECN-IPM group were:

[1] ECN-ACIP and ECN-VCIP were formerly known as N/ACIP and N/VCIP respectively.

- To define a Quality of Service classification to achieve the requested A/V transmission quality for broadcast applications.
- To standardise network information exchange between EBU Members and Network providers.
- To propose a method of collecting end-to-end performance information for management purposes.

In achieving these goals the ECN-IPM Group has specified a set of parameters that are important for broadcasters when using IP networks for audio and video transmission and has developed a software mechanism to probe a network for device and topology discovery, physical path tracing for both end-to-end communication and multicast streams, with the potential for multilayer monitoring for streams on a multi-vendor network with fully media-specific parameters.

The specified parameters cover both the network layer and application layer (for video and audio). SNMP is employed to collect information on the status of networked devices, such as the transmission rate, error rate, the codec used and multicast streams status.

To ensure that all the parameters can be recovered from a variety of different manufacturers' IP equipment, the group has designed a MIB (Management Information Base). Although many MIB files have been published over the years, especially on the network side, very little standardisation work has been done on A/V codec MIB files. The EBU ECN-IPM group has therefore proposed a new standard, based upon IEC 62379 (Common Control Interface for Networked Audio and Video Systems) to address this issue.

Two EBU technical publications have been produced by the ECN-IPM group:

EBU Tech 3345 defines the parameters and the new MIB Information.

This document, **EBU Tech 3346**, is a description of the software mechanism, **EisStream**[2]. The software package is written in Java and it provides physical path tracing for IP traffic using SNMP.

2. Features & Functionalities

The new monitoring standard defined in EBU Tech 3345 addresses the unique challenge faced by broadcasters in monitoring end-to-end media traffic, especially on multi-vendor networks. It also enables broadcasters to deploy unified third-party monitoring tools across their infrastructure by allowing a larger set of parameters to be measured with a common method.

In order to incentives the manufacturers to adopt the new standard, it is necessary to demonstrate the potential advantage their products will have by supporting the new standard. Hence the EBU-IPM group needs to provide the broadcasters a software tool that can monitor the media traffic using the new standard.

This software needs to be independent of any operating system, equipment manufacturer or any proprietary technology. Otherwise it would risk the new standard being perceived as being associated with a particular supplier.

The lack of any existing suitable licence-free product led the EBU ECN-IPM group to commission the development work on its own software, the EBU IPM SNMP Stream Monitor (EisStream). The objective was to develop a fully automated network monitoring tool that is open-source, standard-based and platform-independent.

[2] EBU Integrated Monitoring Solution for Media Streams on IP Networks, http://eisstream.sourceforge.net/

As shown in Figure 2, EisStream is designed to monitor data traffic on multiple layers in the OSI model. The software performs all its discovery, analysis and monitoring functions through processing a small set of common MIB data extracted from different network devices via SNMP (Simple Network Management Protocol).

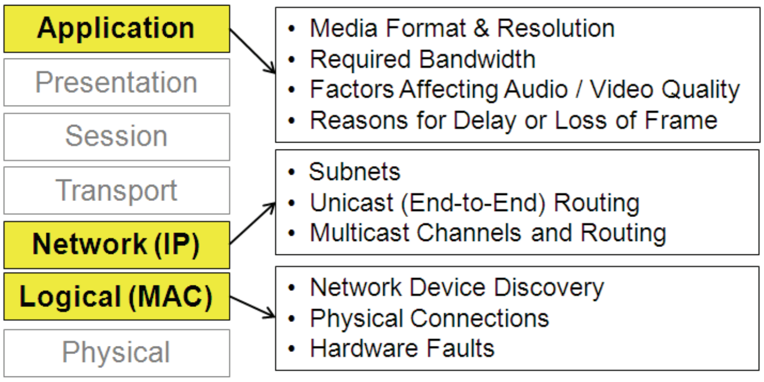

Application
Presentation
Session
Transport
Network (IP)
Logical (MAC)
Physical
• Media Format & Resolution
• Required Bandwidth
• Factors Affecting Audio / Video Quality
• Reasons for Delay or Loss of Frame
• Subnets
• Unicast (End-to-End) Routing
• Multicast Channels and Routing
• Network Device Discovery
• Physical Connections
• Hardware Faults

Figure 2: Monitoring Level of EisStream

Due to the fact that the development work on EisStream took place in parallel with the drafting of EBU Tech 3345, the first part of the software release only consists of network discovery and analysis functions in Layers 2 and 3 using the existing standard MIB objects that have been widely adopted by all major manufacturers. Table 1 shows a list of these existing MIB objects used by the EisStream software.

Table 1: MIB support required by EisStream

MIB	Table	Object	Required Support
MIB-II	(Leaf Objects)	sysDescr	All Devices
		sysServices	
		ipForwarding	
	ifTable	ifIndex	
		ifDescr	
		ifPhysAddress	
	ipAddrTable	ipAdEntAddr	
		ipAdEntIfIndex	
		ipAdEntNetMask	
	ipRouteTable	ipRouteDest	
		ipRouteIfIndex	
		ipRouteNextHop	
		ipRouteType	
	ipNetToMediaTable	ipNetToMediaIfIndex	
		ipNetToMediaPhysAddress	
		ipNetToMediaNetAddress	
IP-FORWARD-MIB	ipCidrRouteTable	ipCidrRouteDest	Routers Only
		ipCidrRouteMask	
		ipCidrRouteNextHop	
		ipCidrRouteIfIndex	
		ipCidrRouteType	
BRIDGE-MIB	dot1dTpFdbTable	dot1dTpFdbAddress	Multilayer Switches & Bridges
		dot1dTpFdbPort	
		dot1dTpFdbStatus	
IPMROUTE-STD-MIB	ipMRouteTable	ipMRouteSource	Multicast Routers
		ipMRouteSourceMask	
		ipMRouteUpstreamNeighbor	
		ipMRouteInIfIndex	
		ipMRouteAddress	
		ipMRouteRtMask	
	ipMRouteNextHopTable	ipMRouteNextHopIfIndex	
		ipMRouteNextHopState	
	ipMRouteScopeNameTable	ipMRouteScopeNameString	

EisStream was designed to provide a purely SNMP-based, universal and open source platform for broadcasters and manufacturers to implement the new monitoring MIB objects proposed in EBU

Tech 3345. Therefore once EBU Tech 3345 is fully adopted, the software can be easily updated to provide network monitoring information on the Application Layer, thus providing a single solution for monitoring a multi-vender network.

The source codes of the **EisStream** software have been released on SourceForge: (http://eisstream.sourceforge.net/).

The current version of EisStream provides five main features:

- Device Discovery – discovering all devices on an unknown network
- Physical Topology – identifying all physical connections on this network
- IP Layer Information – gathering subnet and Layer 3 routing information
- End-to-End Physical Path – tracing all the nodes involved in delivering a IP packet
- Multicast Maps – discovering and mapping the routing structures for all channels

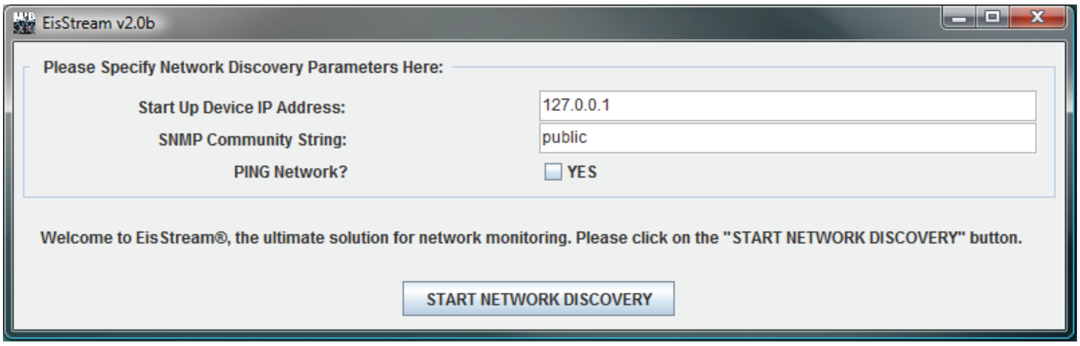
2.1 Device Discovery

EisStream is able to start the network discovery from any SNMP-compliant host and discover all the devices in an unknown network, providing the entire network infrastructure supports SNMP and their SNMP community strings are known.

This feature not only enables a fully automated analysis of a completely unknown network, but it is also extremely useful for managing networks whose topologies are already known, as it is increasingly difficult to manually keep information up-to-date over time for large-scale networks.

Since an SNMP message is only transmitted as a UDP packet in the IP Layer , it can only be sent to a known IP address. Therefore the software will start the discovery process by sending its first SNMP messages to either the local address, 127.0.0.1 or to a user-specified IP address. After obtaining information on the first device, EisStream will then explore the IP addresses of its logical neighbours. The software repeats this process until it has discovered all the devices in the network.

As shown in Figure 3, unless the user needs to change the start-up IP address or a non-default SNMP password is being used, EisStream will require no initial input in order to discover the unknown network.



Please Specify Network Discovery Parameters Here:

Start Up Device IP Address: 127.0.0.1

SNMP Community String: public

PING Network? □ YES

Welcome to EisStream®, the ultimate solution for network monitoring. Please click on the "START NETWORK DISCOVERY" button.

Figure 3: Start-up screen of EisStream

The discovered devices, including the start-up device, are categorised into three main groups.

- Routers: devices that forward IP packets. Routers can be identified by reading the value of "ipForwarding" object in MIB-II. Therefore Gateway PCs are also considered as routers. Routers with bridge ports (non-routed ports) are known as Multilayer Switches.
- Bridges: often referred to as Layer 2 Switches, bridges are devices that do not forward IP packets and do not have any data entry in the "ipRouteTable".

- Hosts: network-attached audiovisual equipment and devices including PCs and workstations, that do not forward IP packets but have data entries in the “ipRouteTable”.

The detailed process for device discovery can be found in § 4, Algorithms.

2.2 Physical Topology

EisStream is capable of automatically generating a physical network topology by analysing the information collected through the network discovery stage. For the same reason as for Device Inventory, this feature will save network administrators significant amounts of time and effort either in familiarising themselves with a new network or in the day-to-day management of existing networks. More importantly this function also allows any change in the physical connections to be discovered more easily, enabling effective and accurate root-cause analysis to be carried out for network failures.

In general every host on the network is connected to a router or a bridge, which is further connected by other routers or bridges. Therefore the hosts are considered as “leaves” in a physical topology, whereas routers and bridges are considered as “branches”.

EisStream analyses the physical network structure by determining connections between the branches first. Once the topology of the branches is established, the leaves can be simply attached to the branches.

EisStream identifies the type of connection between any two physical neighbours as one of the following:

- “L2-L2”, between two bridge ports on bridges and multilayer switches.
- “L2-L3”, between a bridge port and a routed port.
- “L3-L3”, between two routed ports.

The connection type of any given port is identified by examining the number of Address Resolution Protocol (ARP) records of this port and its associated entries on the Forwarding Information Base (FIB) of the device. The connection type of a port must be reciprocated by the other port connected to it.

The details of the methods for analysing connection types, determining physical neighbourhood and establishing the network topology are described in Section 4 Algorithms.

2.3 IP Layer Information

EisStream is also capable of providing accurate information on IP subnets and Layer 3 routing. Currently it fully supports all IPv4 specifications, including classless addressing and subnet zero. These functionalities can also be expanded to support IPv6, although no tests have been carried out to verify this at the time of writing (April 2011).

Having a set of comprehensive Layer 3 information is essential for monitoring IP networks. Network administrators rely on this information to find the individual devices and perform higher level tasks, such as monitoring and configuring.

As mentioned above, as well as discovering the devices, EisStream also collects the entire set of network data associated with the devices. The Layer 3 information of every network port on a device can be found in this data. IP subnets are identified by calculating the network addresses from the IP addresses and the associated subnet mask values. The Layer 3 routing information for routers can be found in the data for their virtual interfaces.

This is a relatively well-developed functionality in network monitoring tools; hence no further detail is required to describe the methods used in gathering the IP Layer information by EisStream.

2.4 End-to-End Physical Path

One of the most important features of EisStream is its ability to trace the end-to-end physical path between any two hosts in a network and identify every individual port number of the various devices involved in the connection chain.

Monitoring a media stream between two hosts is currently limited to the application layer only. All physical nodes, such as bridges and routers that are involved in establishing the connection between the hosts, are transparent to most software monitoring tools as a result of the generic characteristics of the layered approach to telecommunication. Many specialist hardware tools or integrated software tools provided by network equipment manufacturers have the capability of monitoring across the layers but they are either proprietary (and therefore are very expensive) or are only applicable to devices made by the same manufacturer.

As shown in Figure 4, traditional monitoring software can only provide information about the data stream in the Application Layer, represented by the blue line in the figure. With the new EBU standard MIB and EisStream, it is now possible to do hop-by-hop tracing in the IP and MAC Layers as shown by the green line in the figure, which represents the physical path of a data packet. This function enables EisStream to identify and monitor all the nodes on the path.

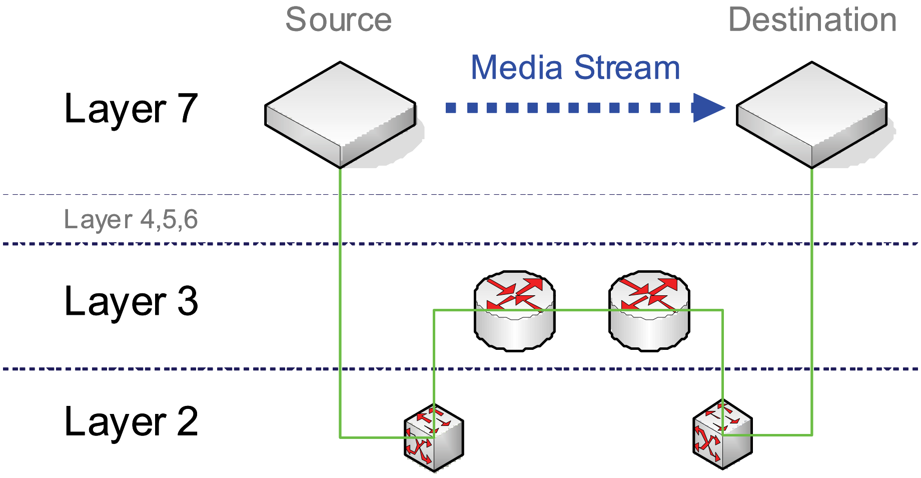


Figure 4: Physical Path of a Media Stream

Using the IP layer information collected during the discovery process, EisStream first determines the logical routes between two hosts. If they are found on different subnets, the EisStream would then identify all the routers that established the Layer 3 connection between the subnets. Finally the physical path between the boundary routers and the hosts are mapped out from the physical topology.

The detailed elaboration of the end-to-end physical path tracing function of EisStream is explained in § 4, Algorithms.

2.5 Multicast Maps

Another unique feature of EisStream is the capability of discovering multicast streams and their routing structures. For every multicast channel/stream it discovers, EisStream also maps all the routers and their place in the forwarding structure of this particular channel/stream. Using this function together with the end-to-end physical path-tracing function, EisStream is able to monitor the traffic and physical path between any multicast source and client.

Multicast technology is used extensively for automated routing and network management protocols, such as Routing Information Protocol (RIP), Open Shortest Path First (OSPF) and Network Time Protocol. It is also used widely by broadcasters for audio and video delivery over IP, both for contribution and distribution chains.

EisStream gathers all multicast information from the multicast routers through the network discovery stage. It first identifies all the multicast routers, then from the information contained on these routers, it determines multicast channels and their associated multicast addresses. Each of these channels contains a number of hosts as source and clients, and the routers that form the branches of a particular multicast tree.

It is possible for a router in a multicast channel to enter an idle state where it does not forward any multicast traffic because no client is subscribing to the channel through it, as shown in Figure 5. EisStream will still include this router in the multicast tree as a branch but with no further branches or leaves, whereas other network monitoring tools based on traffic-sniffing would still exclude such routers from the channel.

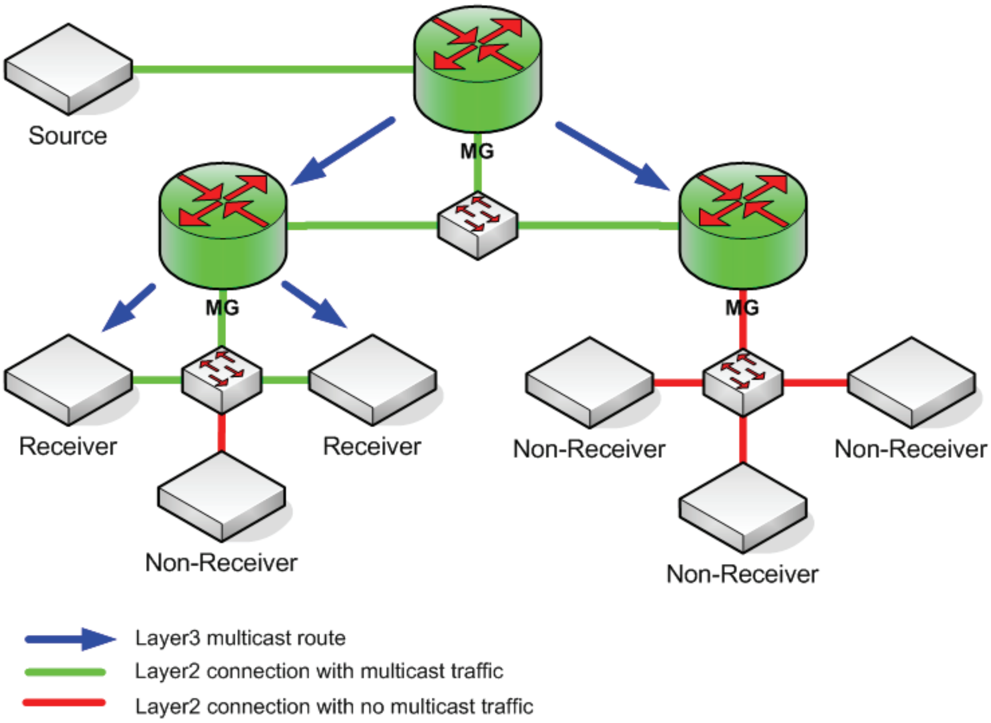


Figure 5: Idle Router in a Multicast Tree

EisStream is able to discover idle routers in a multicast channel because it only gathers information independent of any manual configurations, such as PIM mode, IGMP version, bootstrap router priority and Rendezvous-Point settings. In other words, the software will only learn the true routing behaviour of the routers. Hence as long as a router has the forwarding information of a multicast channel, it is considered as part of the routing structure, regardless of whether it is actively involved or not.

Further details of how EisStream discovers and analyses the multicast channels can be found in § 4, Algorithms.

3. Software Architecture

The EisStream software was developed in Java for platform independence and potential implementation as web service. This section is to give Java developers a general introduction to the software structure, including the classes of the generic packages and the key methods of each class in the source code of EisStream.

In order to be released open-source, the software has included very limited number of licence-free external packages as listed below:

- SNMP4J: used for sending and receiving SNMP messages.
- JGraphT: used for generating graphical interactive network diagrams.
- JGraph: required by JGraphT.

Apart from the packages listed above, EisStream has three internal packages that were developed completely in-house by the BBC.

- **NMS**: (Network Monitoring Station) manages communications with devices on the network.
- **NETWORK**: responsible for all network discovery and analysis functions.
- **GUI**: generating and driving the user interface functions.

The core functions of the EisStream software are built entirely within the NETWORK package, with NMS being the interfaces to the network and users respectively. The external packages are simply employed for handling basic graphical display and network messaging. Figure 6 shows the relationship between all the EisStream packages.

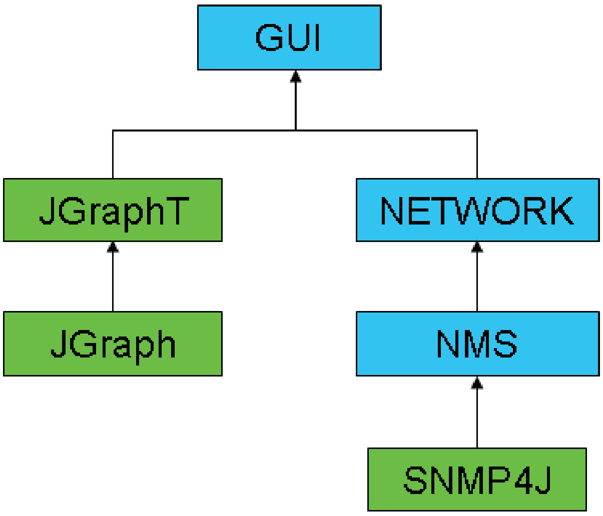


Figure 6: EisStream Package Dependency Structure

3.1 The NMS Package

In a typical SNMP monitoring scenario the Network Monitoring Station (NMS) communicates with devices on the monitored network through SNMP messages. Normally the NMS is a physical server that sends and receives UDP packets containing SNMP message strings in the form of dotted decimals. In addition to this, the NMS package in EisStream is also responsible for scrambling the SNMP message string from a MIB file, validating and extracting the data from the replied strings, and sending other types of messages, such as ping.

There are 5 objects in the EisStream NMS package.

- NMS.java: public class for scrambling SNMP queries and interpreting SNMP replies.
- MibFileAccessor.java: private class for finding the SNMP message string from a standard MIB file.
- SnmpMessage.java: private class for containing a single SNMP message object to be used by SNMP4J.
- SnmpTransponder.java: private class for driving the SNMP4J to send and receive SNMP messages.
- IcmpSender.java: public class for sending a single Ping packet to a given IP address.

Figure 7 shows the relationships between the objects in the NMS package. The NMS object makes exclusive use of all private classes in the package for exchanging SNMP messages with network devices. This is the class that is used by other packages to fetch MIB data from the network devices. The use of the IcmpSender object is explained in § 4, Algorithms.

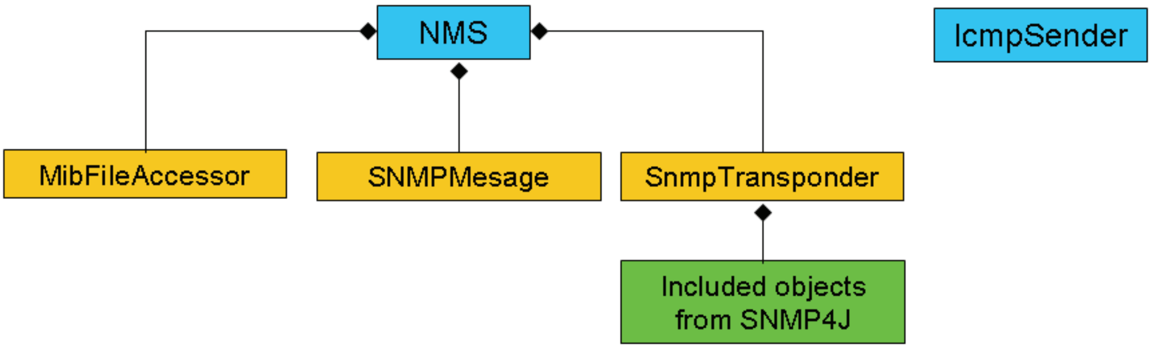


Figure 7: Simplified UML object structure of the NMS package

Once commanded to send an SNMP message to an IP address, the NMS object first calls the “MibFileAccessor.getSnmpMessageByName(String)” to obtain SNMP message content from the local MIB file library and stores it as a SnmpMessage object.

The NMS object then passes the message object to the SnmpTransponder object and calls the “SnmpTransponder.SendSnmpMessage(SnmpMessage)” to send and receive the SNMP message from the IP address. Upon receiving of the replied message, the NMS object validates the data and uses the “validSnmpReply()” method to indicate the data acceptance.

3.2 The NETWORK Package

The NETWORK package is the computation and data centre of the EisStream software, responsible for all its functionalities. It was designed to have no direct interface with the users or networks, therefore making it easy to implement as a web service in the future.

There are 10 objects in the EisStream NETWORK package. All of them are public classes because they all need to interact fully with other parts of the software.

- Interface.java: represents a physical port of a device;
- Device.java: represents a generic object that can either be a host, router or bridge;
- Switch.java: inheriting Device.java, represents a generic object that can either be a bridge or multilayer switch, both are the most common types of modern routers;
- Router.java: inheriting Switch.java, represents a router, which could also be a multilayer switch;

- Discoverer.java: performs network device discovery function;
- Layer2Map.java: performs physical topology analysis function;
- Layer3Map.java: performs Layer 3 information gathering function;
- Path.java: performs end-to-end physical path tracing function;
- MulticastGroup.java: represents a multicast forwarding structure consisting of multicast routers, sources and the physical ports on those devices;
- MulticastMap.java: performs multicast channel mapping function;

Figure 8 shows the relationships between the objects in the NETWORK package. The Interface and Device classes are the two fundamental building blocks of all the functionalities, as naturally all networks are made of devices connected together via ports. The three basic types of devices are each represented by the Device, Switch and Router objects. Every basic function of EisStream is also defined as an individual object in the NETWORK package. MulticastGroup is a data object required to represent the special structural relationships of a group of devices.

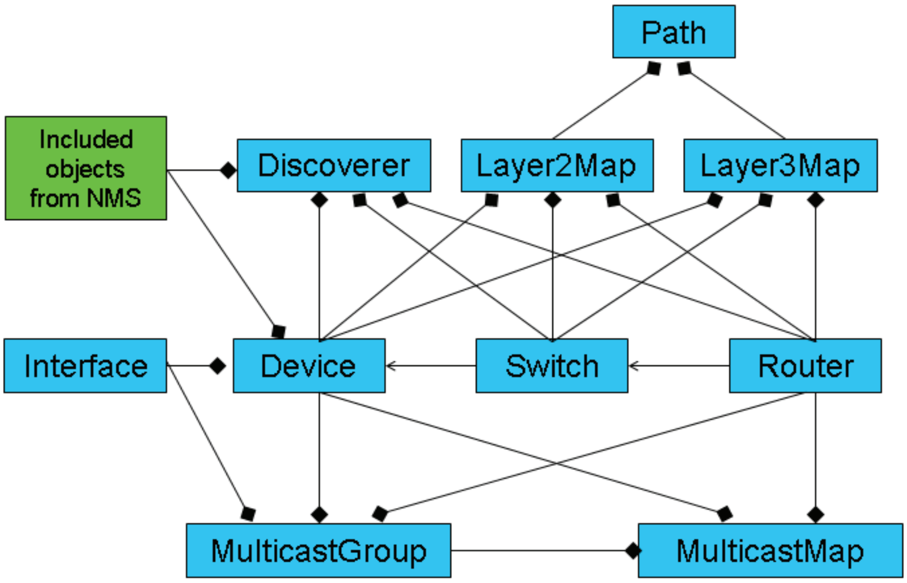


Figure 8: Simplified UML object structure of the NETWORK package

The method for commanding the NMS package to send and receive SNMP messages to devices resides within the Device object, thus also inherited by Switch and Router objects. This allows the sending of SNMP messages from certain MIBs to be associated with the appropriate device type, for example only the Switch object can send SNMP messages from the BRIDGE-MIB. Hence in future when the BRIDGE-MIB is changed, only the Switch object needs to be updated.

The Discoverer object launches a single thread process through the “DiscoverAll(boolean)” method. It creates a new Device object each time a new IP address is learned through the recursive “nextHopDiscovery(Device)” or “arpCacheDiscovery(Device)” methods. The new Device object then populates itself using the “Device.populate()” method. An Interface object is created and populated within the Device object for every physical port (it discovered??). Then the Discoverer will use the “isLayer2Switch(Device)” and “isLayer3Router(Device)” methods to identify the Device as Host, Switch or Router, and cast it into appropriate class if necessary. Both Switch and Router objects have further data fields to populate, which are specific to their routing and bridging functionalities. The cyclic discovery algorithm continues until no new IP address can be found. The software then exists from the discovery process.

Once the “update(ArrayList<Router>, ArrayList<Switch>, ArrayList<Device>)” method is called, the Layer2Map object takes over the newly discovered set of Routers, Switches and Hosts, and finds the core of the network using the "findCore(ArrayList<Router>)" method. It then runs the various algorithms for physical connection analysis through the "buildMap()" method, which calls the recursive "expand(Switch)" and "findBranchConnections(Interface)" methods to establish all the branch connections among the Routers and Switches. Then the leaf connections for the Hosts are established using the "findLeafConnection(Interface)" method. Once the analysis is finished, the Layer2Map object validates the results by calling the "validateConnections()" method. There is also a "getPath(Interface, Device, Interface, ArrayList<Interface>)" method, which is used for finding the physical path between two devices.

Similarly the Layer3Map also has a method called "update(ArrayList<Router>, ArrayList<Device>)", which receives data from the Discoverer object for processing Layer 3 information. Apart from the "buildMap()" method, which produces a HashMap of subnets and their members, the Layer3Map also provides a "routeSubnets(String, String)" method for finding the IP route among different subnets, which is useful for the end-to-end physical path tracing. The Layer3Map object also provides a set of static functions for IP information processing, such as "findIPMask(String)", "findNetworkNumber(String, String)" and "sameSubnet(Interface, Interface)".

The Path object makes use of the Layer2Map and Layer3Maps objects and finds the physical path between two IP addresses by calling the "getPath(String, String)" method. It first makes use of the "Layer3Map.routeSubnets(String, String)" to locate the Layer 3 nodes, then uses the "Layer2Map.getPath(Interface, Device, Interface, ArrayList<Interface>)", the Path object finds all physical path between every pair of Layer 3 nodes. Joining these paths together according to the order of the Layer 3 nodes, the Path object then returns the full physical path as an indexed HashMap of Interfaces and Devices.

In the same way as the Layer2Map and Layer3Map objects, the MulticastMap object uses "update(ArrayList<Router>, ArrayList<Device>)" and "buildMap()" to receive and processes data to discover the multicast groups and their structures. It stores all the information of a multicast group in a multicastGroup object, and keeps a list of these objects for the entire network.

3.2 The GUI Package

The GUI package has been developed to handle the graphical representation of all the network information in the various layers. The main user-interface has a window containing three tabs for displaying Physical Topology, subnets and Multicast. There is also a pop-up window for displaying detailed information and user messages.

There are 6 objects in the GUI package.

- Main.java: starts and stops the EisStream software.
- GUIWindow.java: creates the main user window object.
- L2MapTab.java: creates the tab for the interactive physical topology map
- L3MapTab.java: creates the tab for the interactive IP Subnet information table
- MulticastMapTab.java: creates the tab for the maps of the multicast channels.
- InfoWindow.java: creates the pop-up window object.

Figure 9 shows the relationships between the objects in the GUI package.

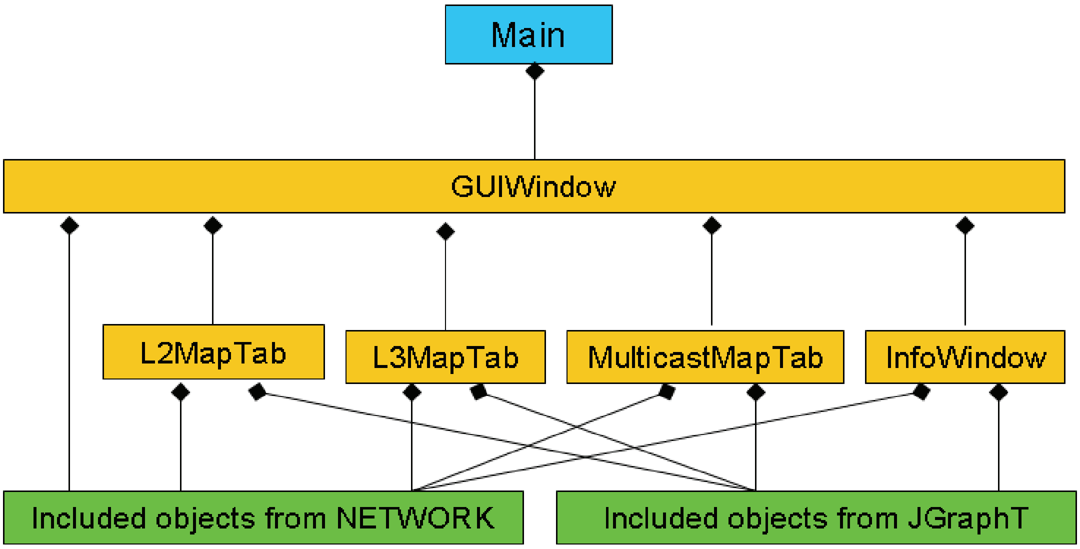


Figure 9: Simplified UML object structure of the GUI package

The Main class is principally responsible for launching the GUIWindow, which has to be started from another class. As soon as the window is started, the GUIWindow object assumes the role of the driver of the EisStream software. Before live monitoring functionalities are implemented, the tabs and pop-up window are only there to present information to the user. The discovery and analysis functions of the software are solely driven by the GUIWindow object.

On start up the Main object creates a GUIWindow object within itself. The GUIWindow object then takes a few parameter values from the user, who then commences the discovery and analysis process. Once finished, the GUIWindow object will create the L2MapTab, L3MapTab and MulticastMapTab objects in the user window. The tabs use colour-coded boxes in the JGraphT to represent the different types of devices. Once a device box is clicked, an InfoWindow object is created by the tab object calling the "displayInfo()" method.

4. Algorithms

Due to the requirement that EisStream must be open-source and that no proprietary library could be used and because of the lack of suitable licence-free packages, all of the core functionalities of EisStream were developed from scratch. Many original, innovative methods and algorithms were developed as a result.

Apart from the function for Layer 3 information, all other features of EisStream are delivered using these original methods and algorithms.

4.1 Device Discovery

To start discovering the network, EisStream must be given the IP address of any device that supports SNMP. If this address is not specified, the software can also start from the loopback address of the local machine where the software is running, provided that the local machine supports SNMP.

Figure 10 shows the entire discovery process.

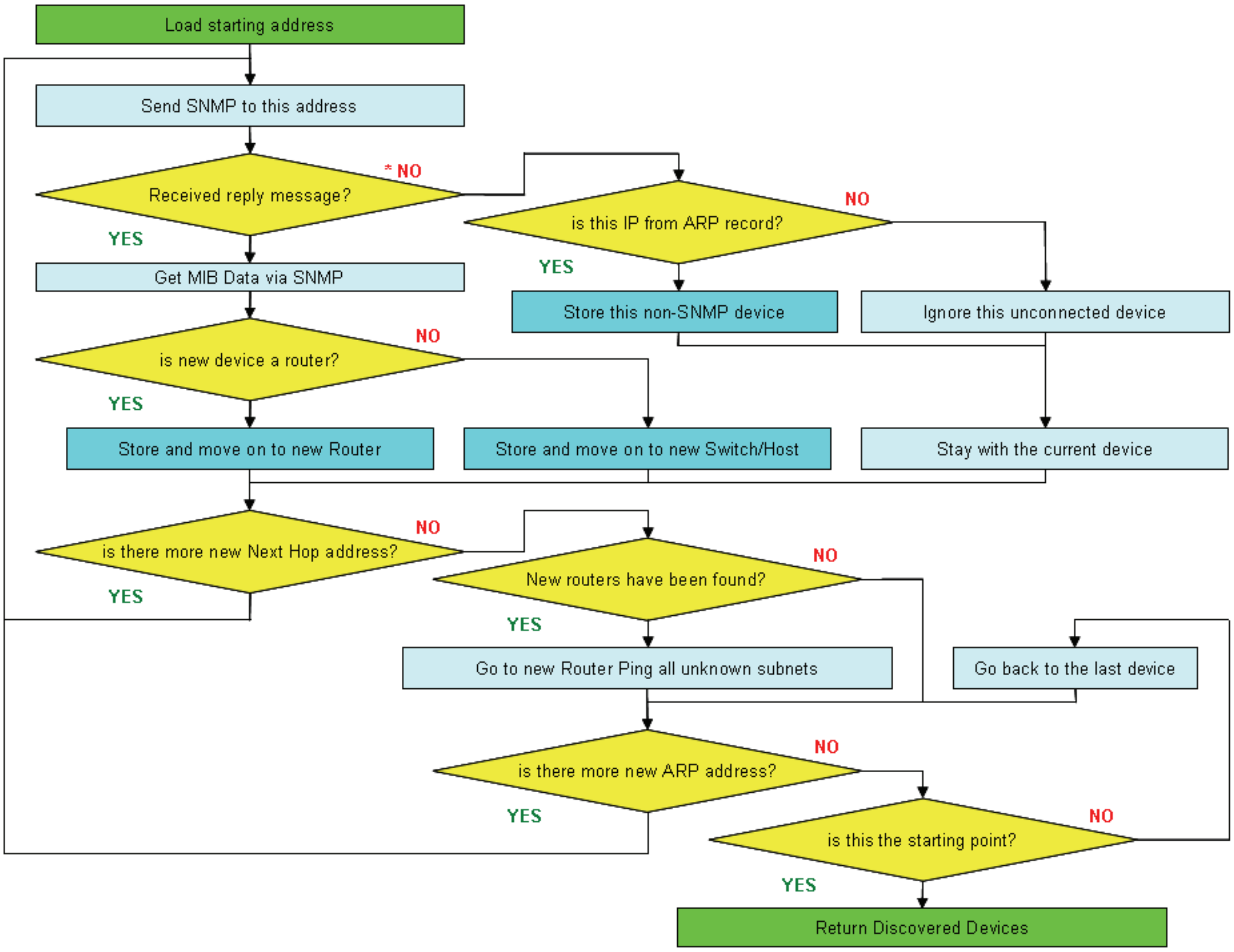

Load starting address
Send SNMP to this address
Received reply message?
* NO
YES
is this IP from ARP record?
NO
YES
Get MIB Data via SNMP
Store this non-SNMP device
Ignore this unconnected device
is new device a router?
NO
YES
Store and move on to new Router
Store and move on to new Switch/Host
Stay with the current device
is there more new Next Hop address?
NO
YES
New routers have been found?
NO
YES
Go to new Router Ping all unknown subnets
Go back to the last device
is there more new ARP address?
NO
YES
is this the starting point?
NO
YES
Return Discovered Devices

Figure 10: Discovery Algorithm

Two groups of MIB objects are used for discovering new devices: the “ipRouteTable” and “ipNetToMediaTable”. The “ipRouteTable” contains Layer 3 routing information, also known as Next Hop, which is used for discovering the IP addresses of the gateway router of a device. Once a new router is found, the Next Hop of this router leads to the discovery of another router. The “ipNetToMediaTable” contains ARP records, which are used for discovering IP addresses of other logical neighbours of the device. Using the ARP record of all devices the recursive algorithm could potentially span over the entire network.

Discovery through Next Hop is preferred over ARP, as it leads to the discovery of new routers and therefore has higher chance of discovering more subnets, which leads to the discovery of more devices.

Whenever a new device is found, either through Next Hop or ARP from the previous device, the first data to look at is always the Next Hop of the current device. The discovery through ARP only starts when no new router has been found through Next Hop.

When the discovered IP address does not respond to SNMP, it is possible that either the device does not support SNMP, or it has been disconnected. If this IP address is a Next Hop, indicating the device is a router, it is more likely that the router is disconnected because it is highly uncommon for a router not to support SNMP. For the same reason given in the next chapter if the device is discovered through ARP, it is more likely to be a connected device without SNMP support.

The EisStream has a certain degree of tolerance for devices that do not support SNMP. Instead of returning an error, it creates a “dummy” device, with a “dummy” interface for the non-SNMP device. This device is taken as a Host as it is impossible to identify the correct type without getting any device information through SNMP.

In a typical broadcaster's network, devices are most likely to be communicating frequently with one another with up-to-date information on its logical neighbours. This scenario is perfect for using EisStream, which depends solely on collecting network information from individual devices via SNMP messages.

In certain cases where a network device remains idle for a long period of time, the information about its logical neighbours is often deleted as a result of the house-keeping feature of the device. This presents a challenge for complete network discovery by the EisStream software. Unless the software is able to artificially generate some traffic to or from the IP addresses of these devices, these devices will never be discovered by the software.

This problem also has a direct impact on discoverability of the bridges, which under normal operations are transparent to the IP layer. If the IP address of a transparent bridge is not shown on any device's logical neighbour list, the software would not be able to discover them neither.

To solve this issue a single ping packet (Internet Control Message Protocol, ICMP) is sent sequentially to every unknown addresses on a known subnet. This will refresh the ARP record for a brief period of time without straining the network.

Given the fact that most broadcasters' networks have separate management subnets with proper separation from the production subnets, it is recommended to run EisStream in "Ping-Enabled" mode on a management subnet. It is possible to limit the ICMP destinations to the management network only. This gives absolute guarantee that the production network is not affected by any extra traffic.

4.2 Physical Topology

As mentioned previously, all hosts are considered as leaves and all Bridges and Routers as branches on any IP network. Therefore each Bridge or Router is considered as a "Branch Point".

To establish the full topology of a network, the connections between all the branch points must be determined first.

On an overall scope, an algorithm is needed to span from one single branch point to another recursively until the entire network is covered. On a microscopic scale, several other methods are needed to determine the type of the physical connection for a port and find where the other port it is connected to.

4.2.1 Overall Algorithm for Branch Topology

The process for finding branch topology starts by determining a hypothetical "core" of the network, which is the router that has been the most popular HextNext Hop destination of all other routers. The more a device is routed to by other routers, the more likely this device is located at the centre of the network.

Figure 11 shows the algorithm written in Java language for finding the network core. This may not be the designed core router of the network but serves as a very good starting point.

```
int max_count = 0;
router core = new router ();

for (every CURRENT_Router) {
      int current_count = 0;
      for (every OTHER_Router) {
         for (every “ipRouteNextHop” entry) {
             if (“ipRouteType” = indirect) {
                 if (CURRENT_Router’s IP address list.contains(“ifRouteNextHop”)) {
                     ++current_count;
                 }
             }
         }
      }
      if (current_count > max_count) {
         core = CURRENT_Router;
         max_count = current_count;
      }
}
return core;
```

Figure 11: Algorithm for finding hypothetical network core in Java syntax.

With the core chosen, a neighbourhood structure around it can be built up by finding directly connected branch points. For each of its neighbours, the same process continues until all branch points have been placed onto this core neighbourhood. There is also a “catch-all” function for branch points that cannot be reached from the core. Each of these branch points is expanded to reach other branch points that are already found on the core neighbourhood as shown by Figure 12.

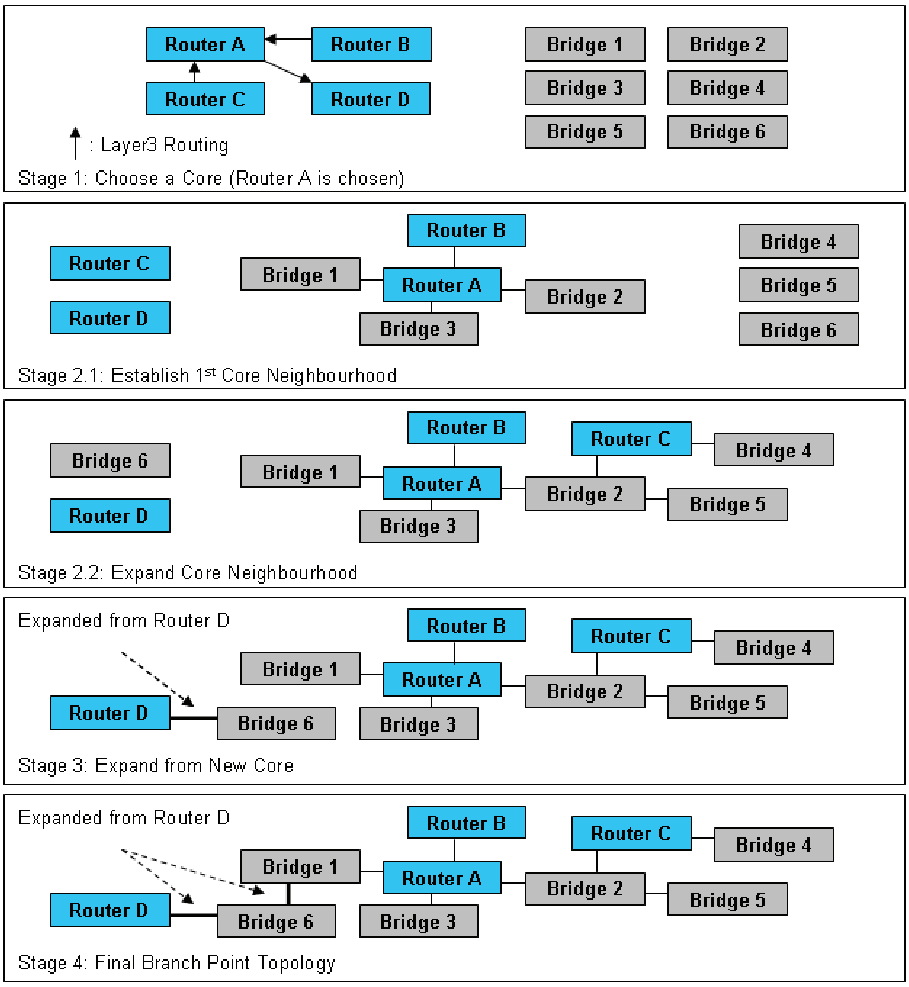

Router A
Router B
Router C
Router D
Bridge 1
Bridge 2
Bridge 3
Bridge 4
Bridge 5
Bridge 6
: Layer3 Routing
Stage 1: Choose a Core (Router A is chosen)
Stage 2.1: Establish 1st Core Neighbourhood
Stage 2.2: Expand Core Neighbourhood
Expanded from Router D
Stage 3: Expand from New Core
Expanded from Router D
Stage 4: Final Branch Point Topology

Figure 12: Example of finding the Branch Point Topology

4.2.2 Specific Methods for Branch Point Connectivity

The backbone of a network consists of interconnected branch points. As mentioned earlier, there are three types of physical connections; therefore any two physically connected ports must have their corresponding connection types.

To find a connected port for any given port, all other device ports with matching connection types are selected. Then the connectivity analysis suitable for the type of connection can be carried out among the candidates to find the actual connected port.

4.2.2.1 Determining Connection Type

The algorithm for examining physical connectivity starts by analysing the type of connection a port is likely to have. Based on the connection type, an appropriate algorithm is used for finding the physical neighbour of this port.

Using the logic shown in Table 2, the connection type of a port on a router or a bridge can be accurately decided.

Table 2: Decision Table for Determining Connection Type

Connection Type		Number Of Entries In FIB		
		0	1	n
Number of ARP Entries	0	n/a	L2-L2	L2-L2
	1	L3-L3	L2-L3	L2-L2
	n	L3-L2	L2-L2	L2-L2

For example if a port does not appear on the MAC address Forwarding Information Base (FIB), it is likely to be a routed port. Then looking at the ARP record of the port, if it has multiple data entries, the port is likely to be connected to a switch port of another branch point. Therefore the connection to this port is likely to be a L3-L2 connection.

4.2.2.2 L2-L2 connectivity Analysis

As shown in Figure 13, the FIB of a bridge or multilayer switch specifies the port to which a Layer 2 packet with an external destination MAC address should be forwarded. The FIB data is contained in “dot1dTpFdbTable” in BRIDGE-MIB, with the external destination MAC address of the Layer 2 packet stored in a “dot1dTpFdbAddress” object and the index of the forwarded port in “dot1dTpFdbPort”.

The set of all “dot1dTpFdbAddress” entries on a bridge with the same “dot1dTpFdbPort” is a collection of all the destination MAC addresses of all Layer 2 packets sent by this port.

On the other hand, all Layer 2 packets arriving at a port on a bridge will be forwarded onto the other ports. Therefore the set of all “dot1dTpFdbAddress” entries less those forwarded to a particular port, plus the MAC address of that port, is a collection of all possible destinations of the Layer 2 packet received by this port.

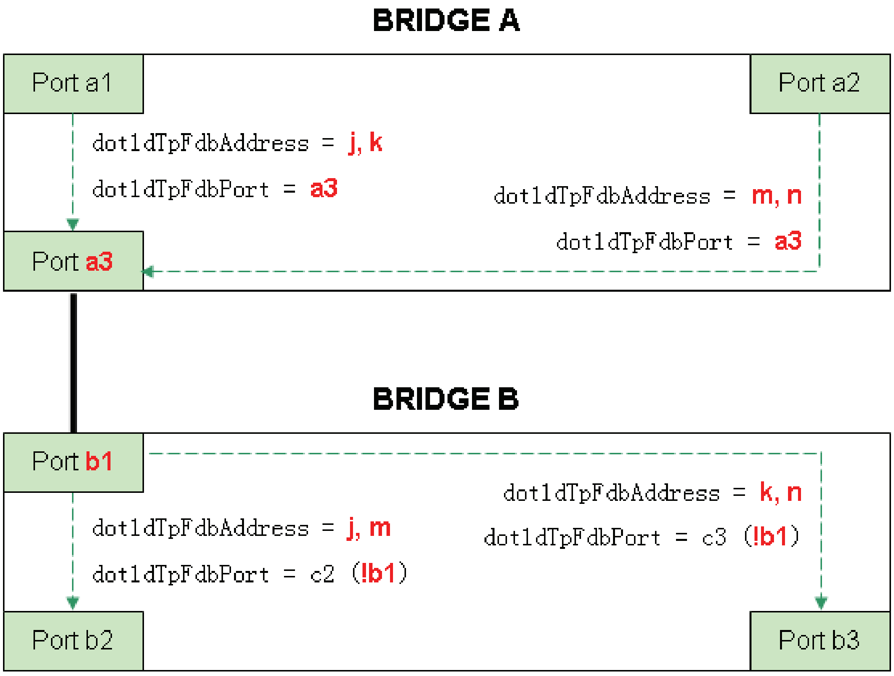


Figure 13: Layer2 Connectivity Theorem

In a physical port-to-port connection, all packets sent by a port must be received by the connected port. Hence by comparing the Layer 2 packets sent and received by two ports, the L2-L2 connectivity between two ports can be successfully established.

LAYER 2 CONNECTIVITY THEOREM (proposed by EisStream developer):

“If port x on bridge A is connected to port y on bridge B, then the set of all the "dot1dTpFdbAddress" entries of A with their "dot1dTpFdbPort" value equal to x, less any entry equal to the ‘ifPhysAddress’ of B, should be equal to the set of all "dot1dTpFdbAddress" with their “dot1dTpFdbPort” values unequal to y.”

As shown in Figure 14, F is the set of all destination MAC addresses of the packets leaving x, B is the set of all "ifPhysAddress" on all ports of B, and F´ is the set of all "dot1dTpFdbAddress" with their values unequal to y:

$$F - B = F'$$

This theorem applies to all L2-L2 connections including those between trunk ports or bridges with multiple redundancies enabled with Spanning Tree.

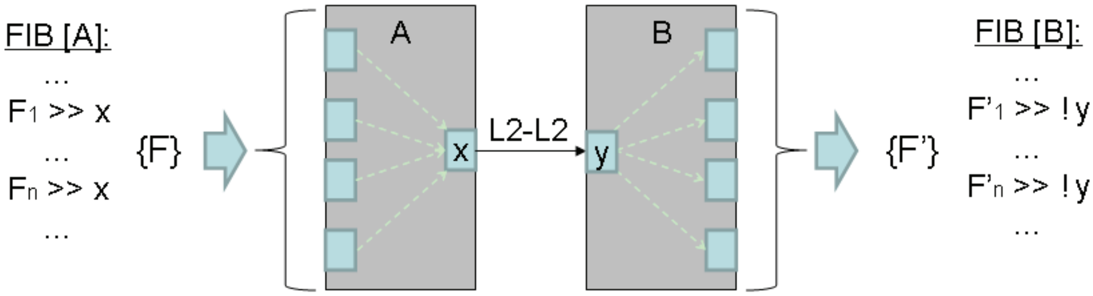


Figure 14: Bridges connected in a daisy chain

Due to the complex nature of the Layer 2 connections, it is unrealistic to expect that applying a universal algorithm will result in a guaranteed accuracy and success rate. In order to be robust in handling practical issues such as the presence of a hub, incomplete MIB data or incorrect manual configurations, an element of fuzzy logic is introduced to the algorithm.

Instead of looking for an exact match, the algorithm will first select a group of possible candidates against a certain threshold of criteria. It then applies filters to the selected candidates to eventually find the most likely match. Instead of looking for the perfect match between both sides of the equation, therefore, a threshold can be specified and expressed as:

$$(F - B) \cap F' = (F - B) \text{ or } F'$$

Hence the theorem could also be written as:

"If port x on bridge A is connected to port y on bridge B, then the set of all the "dot1dTpFdbAddress" entries of A with their "dot1dTpFdbPort" value equal to x, less any entry equal to the 'ifPhysAddress' of B, should be A SUBSET OF all "dot1dTpFdbAddress" with their "dot1dTpFdbPort" values unequal to y or vice versa."

However the fuzzy logic causes problems when determining connections involving 3 or more bridges in daisy-chain or ring structures. In the example shown in Figure 15, a3 can be determined by the fuzzy algorithm as being connected to either port b1 or c1. This is because the sum of all possible destinations of the packets leaving a3 is a subset of all "dot1dTpFdbAddress" on port b1 as well as being a subset of those on c1.

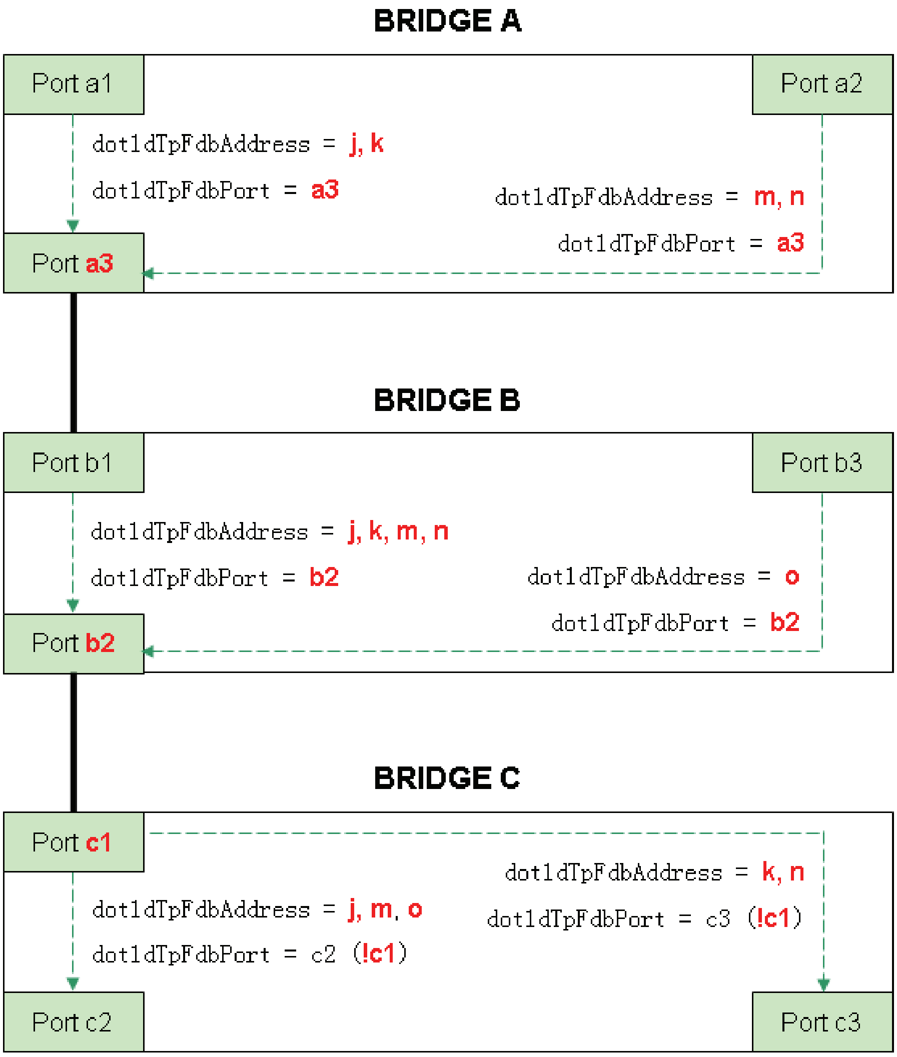


Figure 15: Bridges connected in a daisy chain

This problem is solved by further investigating the "dot1dTpFdbAddress" entries on the ports, as c1 is also forwarding packets to MAC address o, which can only be sent from b2.

This fuzzy algorithm also accommodates the presence of hubs. When a hub is present, one port on a branch point will seem to be connected to multiple hosts, as determined by the algorithm.

4.2.2.3 L2-L3 connectivity Analysis

A L2-L3 connection only exists between a switch port and a routed port, where the MAC address of the routed port is the only “dot1dTpFdbAddress” entry in the bridge’s FIB with the “dot1dTpFdbPort” value pointing to the switch port.

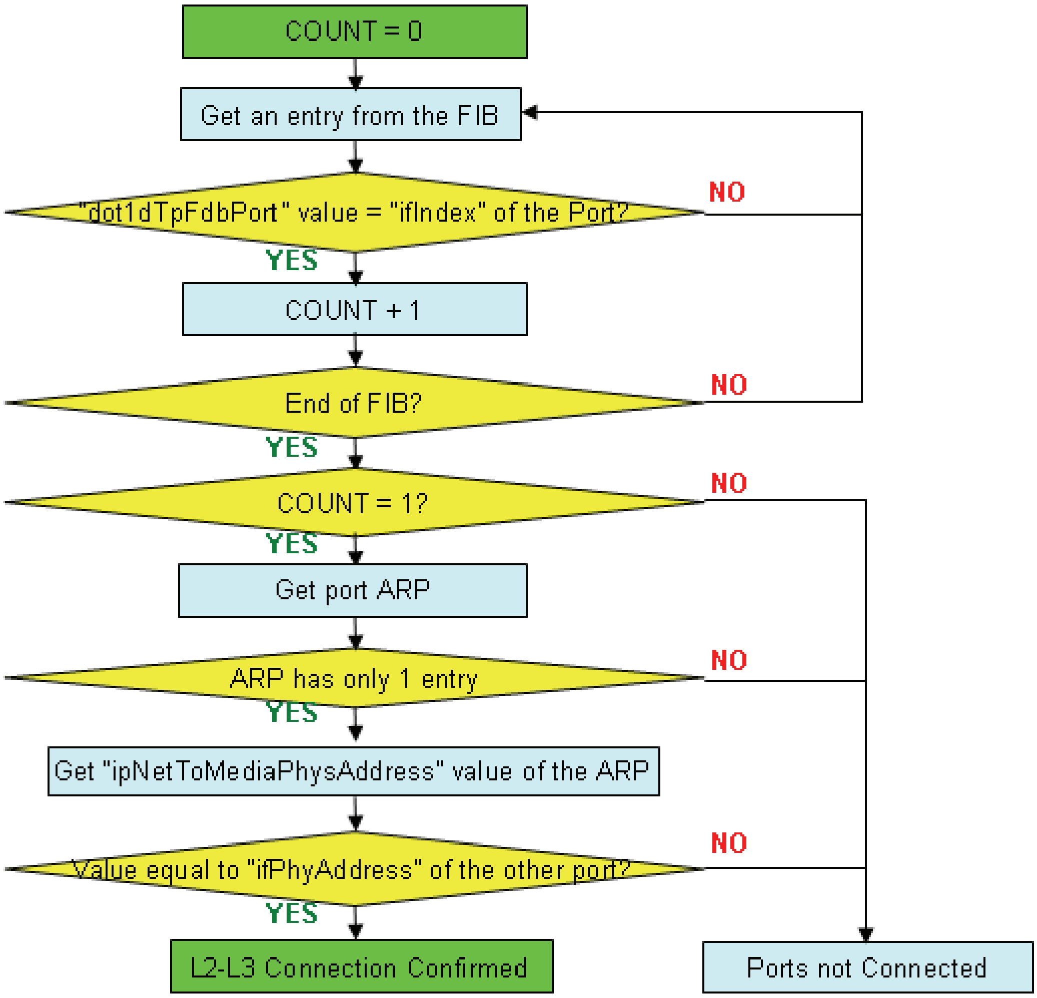

COUNT = 0
Get an entry from the FIB
"dot1dTpFdbPort" value = "ifIndex" of the Port?
NO
YES
COUNT + 1
End of FIB?
NO
YES
COUNT = 1?
NO
YES
Get port ARP
ARP has only 1 entry
NO
YES
Get "ipNetToMediaPhysAddress" value of the ARP
Value equal to "ifPhyAddress" of the other port?
NO
YES
L2-L3 Connection Confirmed
Ports not Connected

Figure 16: L2-L3 Connectivity Discovery Algorithm

As shown in Figure 16, L2-L3 connectivity is discovered through the following steps, which are also used for discovering the L3-L2 connectivity.

1) Find the number of entries in the bridge’s FIB with “dot1dTpFdbPort” equal to the “ifIndex” of the port;
2) If only one such entry is found, check the number of entries in the ARP table of the port;
3) If only one ARP entry is found, check if the “ipNetToMediaPhysAddress” matches the “ifPhysAddress” of the other port;
4) If matches, the L2-L3 connection is determined.

4.3 End-to-End Physical Path

To trace the end-to-end physical path between two hosts in a network is the ultimate goal of building a complete network diagram. Nevertheless without correctly utilizing the Layer 3 routing information on all the routers, it is still impossible to accurately identify the path an IP packet must travel from source to destination.

Figure 17 is an example of a typical modern network of 3 subnets.

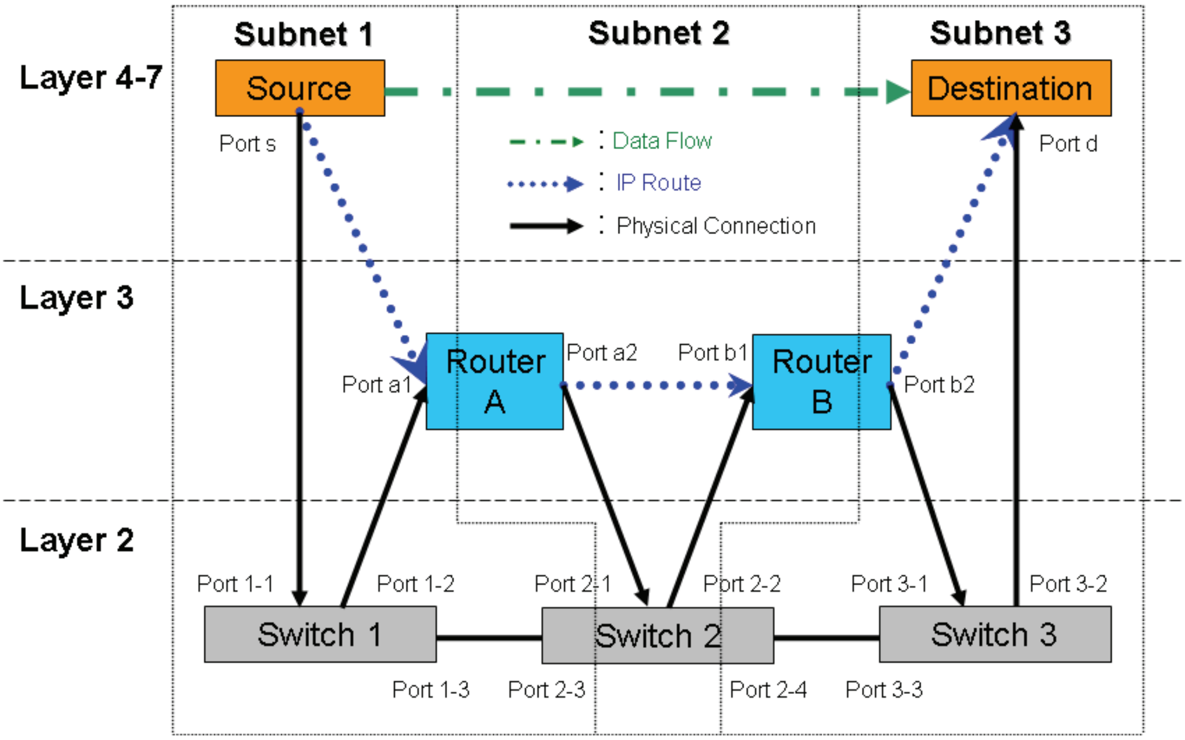


Figure 17: Example of a typical modern network of 3 subnets

Subnet 1 and 2 are routed via Router A and Subnet 2 and 3 via Router B.

Bridge 2 is a backbone switch with ports configured for all subnets, where port 2-3 is connected to port 1-3 on Bridge 1 in subnet 1 and port 2-4 is connected to port 3-3 on Bridge 3 in subnet 3.

In terms of physical ports, any IP packet from the source in subnet 1 to the destination in subnet 3 would follow this path:

s >>1-1>1-2 >>a1>a2 >>2-1>2-2 >>b1>b2 >>3-1>3-2 >>d

Where ">>" indicates a physical cable and ">" indicates the internal switching or routing in a device from one port to another.

However on a simple physical topology diagram without Layer 3 information the path could easily be misinterpreted as:

s >> 1-1>1-3 >> 2-3>2-4 >> 3-3>3-2 >> d

To correctly identify the physical path for any end-to-end connection, it is important to identify the Layer 3 routes between the source and destination.

4.3.1 Determining Layer 3 Routes

By investigating the source's Default Gateway setting in the "ipRouteNextHop" object, the first router on the Layer 3 route can be identified easily. For hosts without this setting, their network numbers can be deduced from the IP address in the "ipAdEntAddr" object and the subnet mask in "ipAdEntNetMask". The router can be found by comparing the host's network number with every router's locally routed subnets, stored in "ipRouteDest" entries with "ipRouteType" equal to 3.

Once the router on the host network is found, by looking for the destination network number or the default route in its routing table, "ipRouteTable" or "ipCidrRouteTable", the next router can be found. Repeat this step until the router in the destination network is found.

It is important that information of the downstream interface to the next router is recorded for every router. For multilayer switches the downstream interface could refer to a subnet with multiple switch ports. In this case, information about every individual switch port is to be collected and kept with the router.

4.3.2 Determining Layer 2 Paths

Layer 2 paths are to be determined individually between every two adjacent routers, as well as between routers and hosts at each end of the route. It is relatively easy to spot the path on a diagram between two points. Nevertheless an algorithm similar to Spanning Tree is required to find the path programmatically.

4.3.2.1 Layer 2 Path between Routers

Within the same subnet, using the physical topology information, the Layer 2 connection from the all the downstream ports of the first router is traced to the immediate neighbours, then to the neighbours' neighbours, and so on until the next router on the path is reached.

4.3.2.2 Layer 2 Path between Router and Host

The same algorithm applies for finding the physical path between a router and a host, except that the starting device is always the router. This is to accommodate the fact that a host can be connected to a router via a hub and some of the hosts may not support SNMP.

4.4 Multicast

4.4.1 Multicast Groups

By extracting the "ipMRouteScopeNameString" data from all routers on the network, a full list of available multicast groups can be made. A multicast address is the textual name that uniquely identifies a multicast group.

Every multicast group contains a source and a number of receivers. It is possible to have multicast groups with no receiver. The IP address and subnet mask of the sources are stored in the "ipMRouteSource" and "ipMRouteSourceMask" objects respectively.

The router IP address and subnet mask for the Rendezvous Point (RP) of a multicast group is found in the "ipMRouteRtAddress" and "ipMRouteRtMask" objects respectively. IP address of a router, from which a particular multicast stream is received, is found in the "ipMRouteUpstreamNeighbor" object.

4.4.2 Multicast Channels

For a particular multicast group, a complete channel structure can be established from the sources down to all the routers involved in forwarding the traffic of this multicast group.

By reading values of the "ipMRouteInIfIndex" and "ipMRouteNextHopIfIndex", the physical port where a multicast stream is received from and the interface where the stream is forwarded to, can be obtained respectively. When a router is enabled with IGMP, the forwarding interface is a physical port; when not, the forwarding interface is a subnet. In the case of forwarding to a subnet, the multicast stream is broadcasted on all switch ports in that subnet.

EisStream constructs the multicast maps solely based on the information gathered from the multicast routers. This is because the multicast forwarding structure is decided entirely by the multicast routers, as they negotiate among themselves to decide the final path of the multicast traffic. Bridges that support IGMP, which serve the only function of stopping the Layer 2 multicast packets being broadcast to all the ports, deliver the packets passively from the multicast routers to the receiving hosts. Their behaviour is irrelevant to how the multicast route is constructed. Hence bridges are excluded from the multicast map by EisStream. Their existence in a multicast traffic path can be worked out simply by finding the end-to-end physical path between the last multicast router and the receiving host.

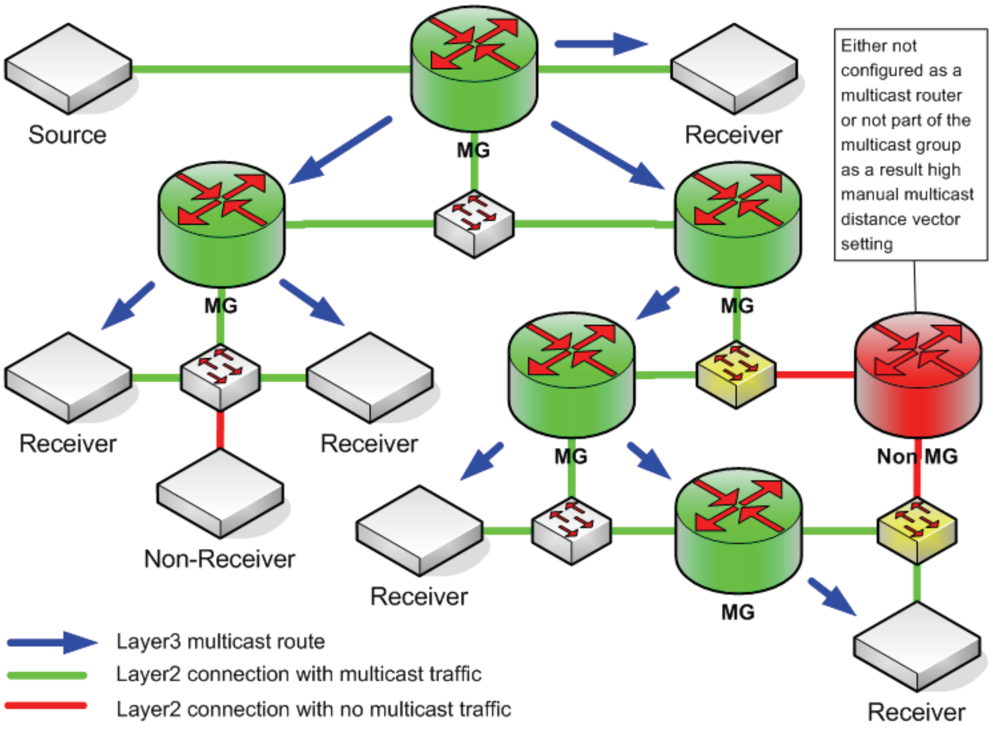

Source
MG
Receiver
Either not configured as a multicast router or not part of the multicast group as a result high manual multicast distance vector setting
MG
MG
Receiver
Receiver
Non-Receiver
MG
Receiver
Non MG
MG
Receiver
Layer3 multicast route
Layer2 connection with multicast traffic
Layer2 connection with no multicast traffic

Figure 18: Route Map of a Multicast Channel

For example in Figure 18, even though the yellow bridges know the shorter path through the red router, because the red router is not part of the multicast structure, they must obey the Layer 3 route and forward the multicast packets to the green routers only. For this reason, bridges are excluded by EisStream from the multicast map. Their existence in the path of multicast traffic could nonetheless be worked out simply by finding the end-to-end physical path between the last multicast router and the receiving host.

4.4.3 Physical Path for Multicast Traffic

Using the mechanism for end-to-end physical path discovery and the information on the channel structure of all multicast groups, every node involved in the delivery of a particular multicast stream can be identified.

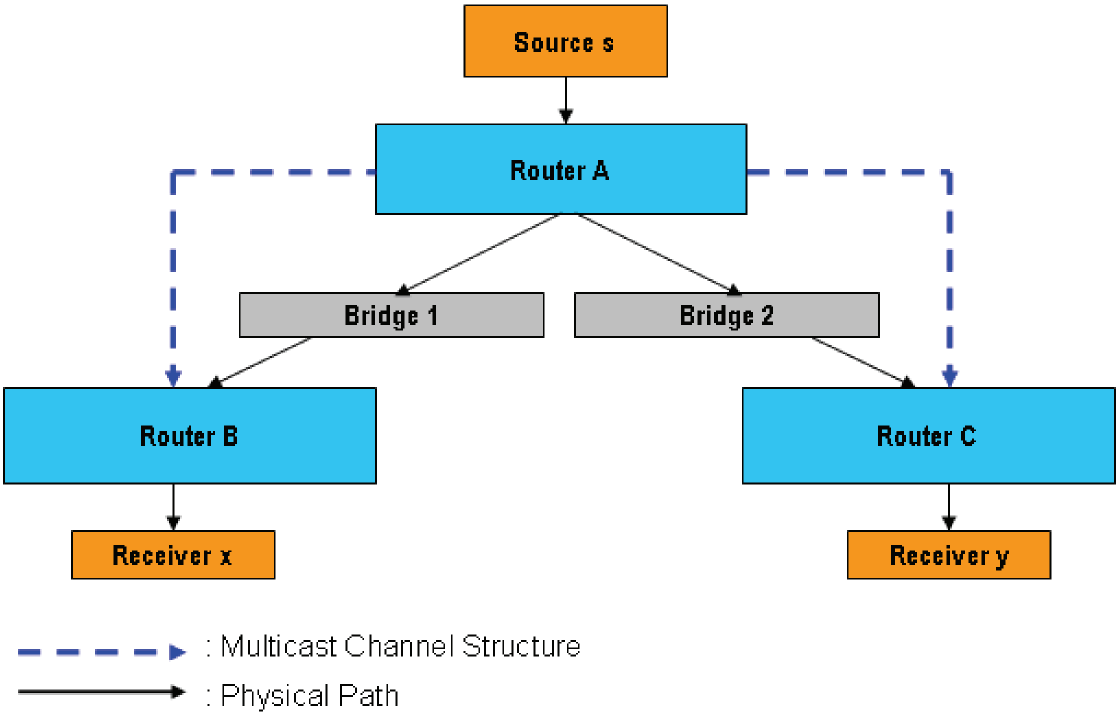


Figure 19: Physical Path for Multicast Traffic

As shown in Figure 19, the physical path for connections between routers, connections between a source and its RP, and connections between receivers and multicast routers can be determined in exactly the same manner as for an end-to-end connection.

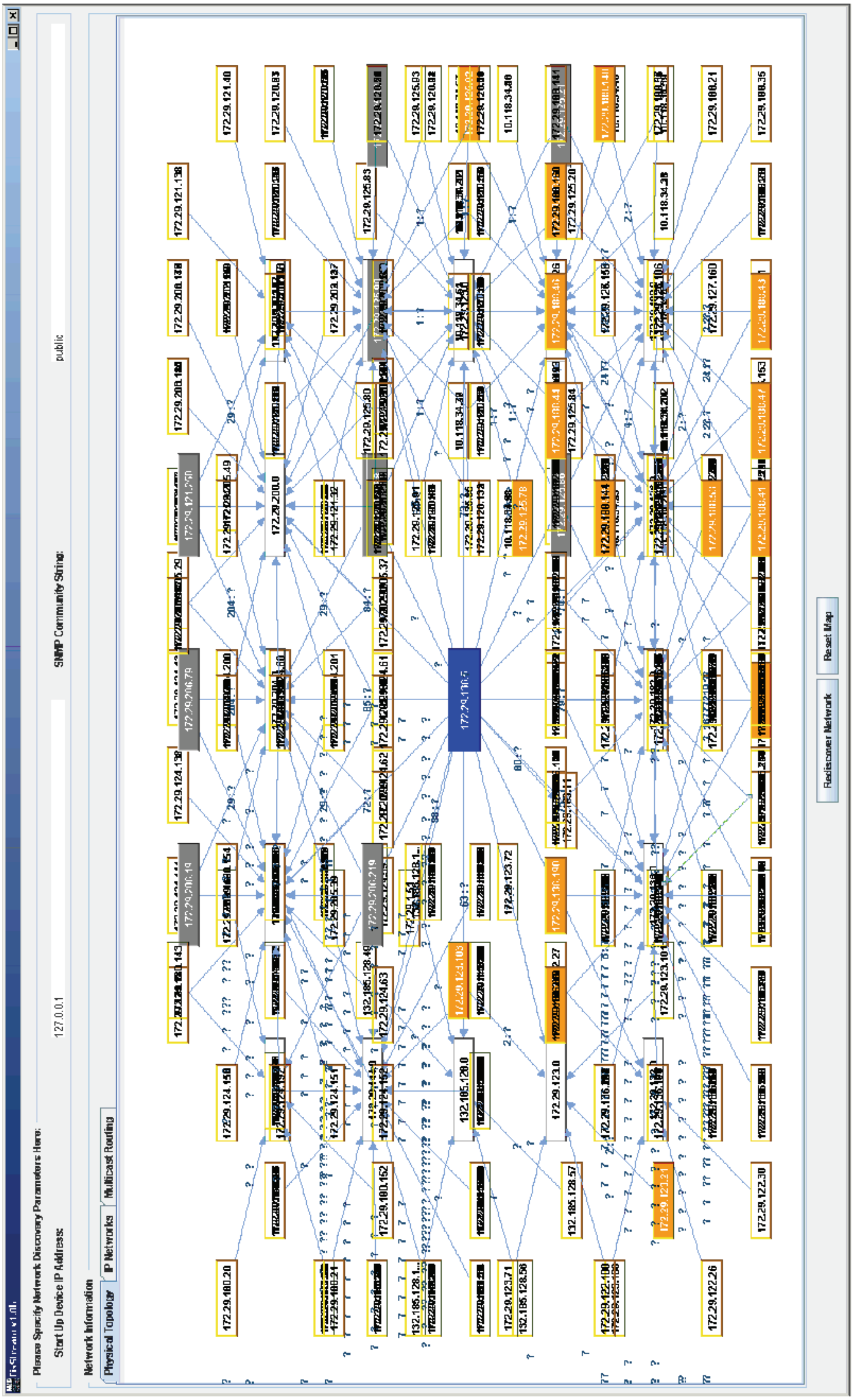
5. Test Results

The EisStream software was developed by BBC R&D using the Agile Software Development method and it was rigorously tested on the BBC R&D network at every iteration cycle.

Figure 20 shows that the BBC R&D network consists of a single router and many bridges and hosts; over 250 in total.

Each transparent box in the figure indicates a device that does not support SNMP (EisStream times out when attempting to contact such devices).

The software was also tested live on VRT's distribution network in Belgium. The output of the topology analysis is shown in Figure 21. The software was able to discover all the routers and bridges in the network. There are also a large number of hosts that do not support SNMP in this network, which resulted in the software running in excess of 20 minutes to finish its analysis.



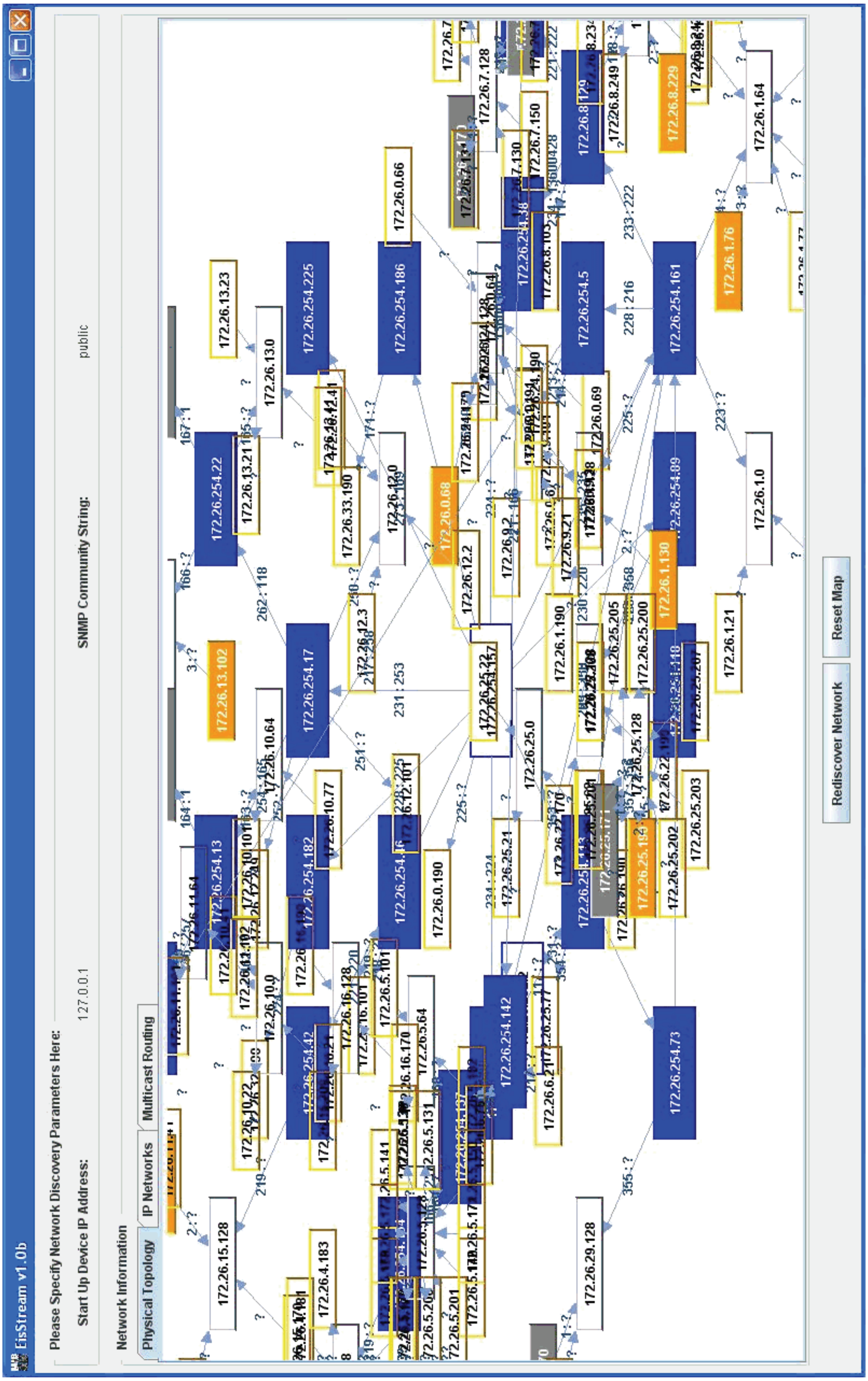
Please Specify Network Discovery Parameters Here:

Start Up Device IP Address: 127.0.0.1

SNMP Community String: public

Network Information

Figure 20: EisStream analysis of the BBC R&D Office Network



EisStream v1.0b

Please Specify Network Discovery Parameters Here:

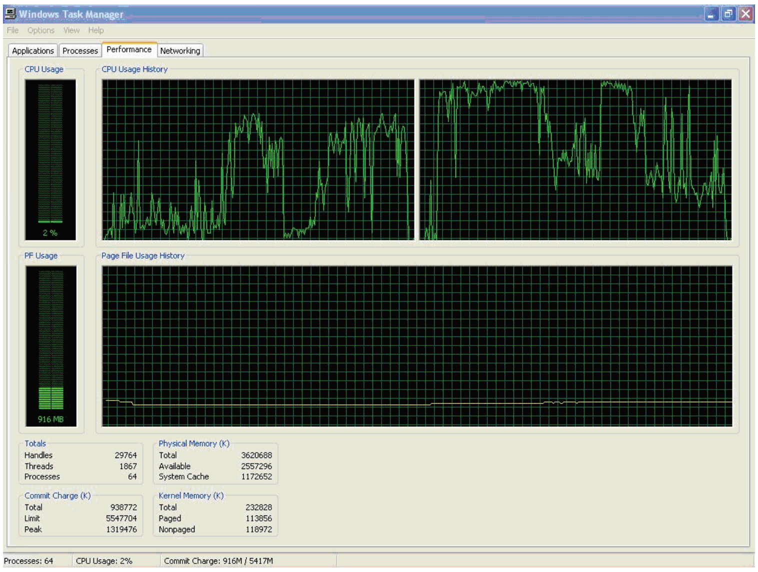
Start Up Device IP Address: 127.0.0.1

SNMP Community String: public

Network Information

Figure 21: EisStream analysis of VRT's Distribution Network

Throughout the entire discovery process, the overall network traffic did not much exceed 500 kbit/s but the CPU utilisation saturated at 50%, as shown in Figures 22 & 23. It is recommended that the software be run on a CPU of at least 1.3 GHz clock frequency.



Totals	
Handles	29764
Threads	1867
Processes	64

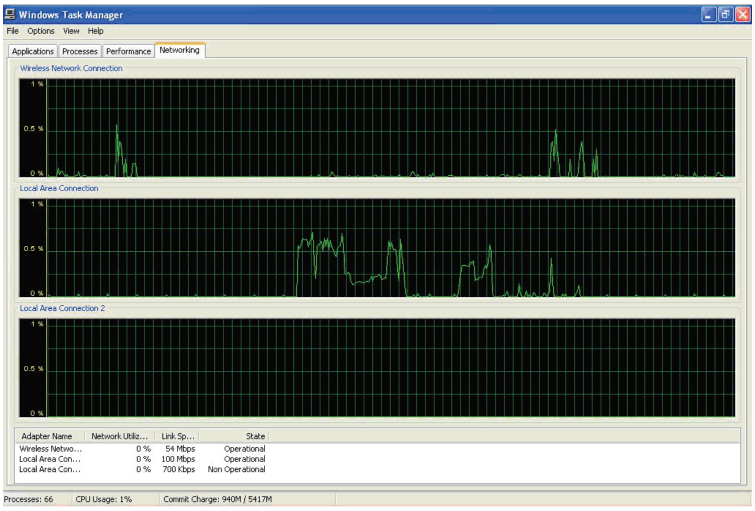
Physical Memory (K)	
Total	3620688
Available	2557296
System Cache	1172652

Commit Charge (K)	
Total	938772
Limit	5547704
Peak	1319476

Kernel Memory (K)	
Total	232828
Paged	113856
Nonpaged	118972

Processes: 64 CPU Usage: 2% Commit Charge: 916M / 5417M

Figure 22:
CPU load of PC running EisStream



Adapter Name	Network Utiliz…	Link Sp…	State
Wireless Netwo…	0 %	54 Mbps	Operational
Local Area Con…	0 %	100 Mbps	Operational
Local Area Con…	0 %	700 Kbps	Non Operational

Processes: 66 CPU Usage: 1% Commit Charge: 940M / 5417M

Figure 23:
Network use during EisStream analysis

6. Next Steps & Further Developments

With EBU Tech 3345 and Tech 3346 (this document), describing respectively a new network measurement standard tailored for media requirements and a universal software platform capable of monitoring any device port for a media stream, an integrated solution for fully audiovisual-oriented network monitoring is now in place.

The next step is to promote the new standard and tool with a view to an industry-wide adoption by major manufacturers. Once the new MIB standard achieves widespread adoption the EisStream software will be able to achieve the ultimate goal of multilayer monitoring for streams on a multi-vendor network with fully media-specific parameters.

The functions of the EisStream can potentially be developed into web services and deployed in a remote monitoring scenario as shown in Figure 24. APIs for distributed or embedded deployment can also be developed from the existing software packages.

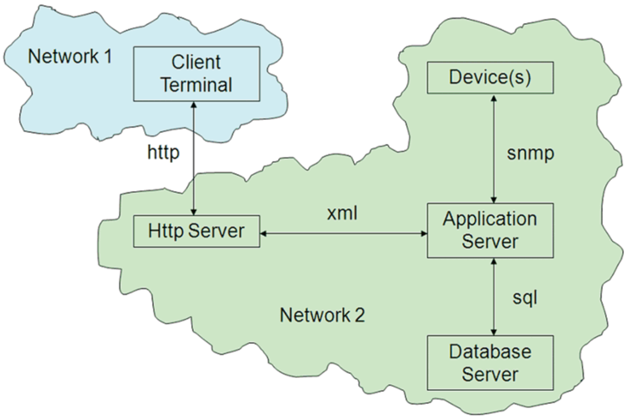


Figure 24: EisStream as a Web Service