PCT WORLD INTELLECTUAL PROPERTY ORGANIZATION
International Bureau

INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(51) International Patent Classification : Not classified	A2	**(11) International Publication Number:** **WO 98/18306** **(43) International Publication Date:** 7 May 1998 (07.05.98)

(21) International Application Number: PCT/US97/19485

(22) International Filing Date: 27 October 1997 (27.10.97)

(30) Priority Data:
08/742,566 28 October 1996 (28.10.96) US

(71) Applicant: SWITCHSOFT SYSTEMS, INC. [US/US]; 1010 North State, Orem, UT 84057 (US).

(72) Inventors: EKSTROM, Joseph, A.; 131 East 300 South, Lindon, UT 84042 (US). GILLE, J., Bernard; 307 East 60 North, Lindon, UT 94042 (US). MC NEILL, Thomas, G.; 1072 North 1000 East, Orem, UT 94097 (US). YANG, Hui; 835 East 1st. Street #B4, Salt Lake City, UT 94103 (US).

(74) Agent: HICKMAN, Paul, L.; Hickman Beyer & Weaver, P.O. Box 61059, Palo Alto, CA 94306 (US).

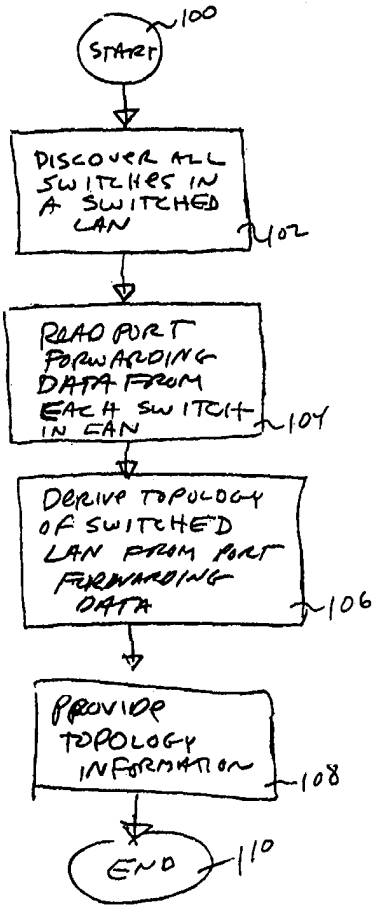
(81) Designated States: JP, European patent (AT, BE, CH, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE).

Published
Without international search report and to be republished upon receipt of that report.

(54) Title: METHOD AND APPARATUS FOR GENERATING A NETWORK TOPOLOGY

(57) Abstract

A method for determining the topology of a computer network includes reading port forwarding data from each device in at least a subset of a computer network. This port forwarding data is then processed and analyzed to derive the topology of at least the subset of the computer network. The reading of the port forwarding data is preferably proceeded by discovering all the switches in the at least subset of the computer network to provide the maximum available port forwarding data for subsequent analysis. The topological analysis uses a recursive function to derive the topology from the derived port forwarding data by placing the discovered switches in a tree structure that permits, the physical interconnections of the switches to be efficiently displayed.



FOR THE PURPOSES OF INFORMATION ONLY

Codes used to identify States party to the PCT on the front pages of pamphlets publishing international applications under the PCT.

AL	Albania
AM	Armenia
AT	Austria
AU	Australia
AZ	Azerbaijan
BA	Bosnia and Herzegovina
BB	Barbados
BE	Belgium
BF	Burkina Faso
BG	Bulgaria
BJ	Benin
BR	Brazil
BY	Belarus
CA	Canada
CF	Central African Republic
CG	Congo
CH	Switzerland
CI	Côte d'Ivoire
CM	Cameroon
CN	China
CU	Cuba
CZ	Czech Republic
DE	Germany
DK	Denmark
EE	Estonia
ES	Spain
FI	Finland
FR	France
GA	Gabon
GB	United Kingdom
GE	Georgia
GH	Ghana
GN	Guinea
GR	Greece
HU	Hungary
IE	Ireland
IL	Israel
IS	Iceland
IT	Italy
JP	Japan
KE	Kenya
KG	Kyrgyzstan
KP	Democratic People's Republic of Korea
KR	Republic of Korea
KZ	Kazakstan
LC	Saint Lucia
LI	Liechtenstein
LK	Sri Lanka
LR	Liberia
LS	Lesotho
LT	Lithuania
LU	Luxembourg
LV	Latvia
MC	Monaco
MD	Republic of Moldova
MG	Madagascar
MK	The former Yugoslav Republic of Macedonia
ML	Mali
MN	Mongolia
MR	Mauritania
MW	Malawi
MX	Mexico
NE	Niger
NL	Netherlands
NO	Norway
NZ	New Zealand
PL	Poland
PT	Portugal
RO	Romania
RU	Russian Federation
SD	Sudan
SE	Sweden
SG	Singapore
SI	Slovenia
SK	Slovakia
SN	Senegal
SZ	Swaziland
TD	Chad
TG	Togo
TJ	Tajikistan
TM	Turkmenistan
TR	Turkey
TT	Trinidad and Tobago
UA	Ukraine
UG	Uganda
US	United States of America
UZ	Uzbekistan
VN	Viet Nam
YU	Yugoslavia
ZW	Zimbabwe

METHOD AND APPARATUS FOR GENERATING A NETWORK TOPOLOGY

Technical Field

This invention relates generally to computer network systems, and more particularly to methods and apparatus for determining and displaying the interconnections of devices within a computer network system.

Background Art

In Fig. 1, an example of a prior art computer network system 10 includes a first local area network ("LAN") 12, a second LAN 14, and a wide area network ("WAN") 16. While the distinction between a LAN (such as LANs 12 and 14) and a WAN (such as WAN 16) is not always clear, generally a LAN is used within a relatively small geographical area, such as within a building or within a small group of buildings, and a WAN is used over a large geographical area, such as within an entire county, within a state, within a country, or even worldwide. While both LANs and WANs perform somewhat similar functions in interconnecting digital devices of the network for data communication, the devices used and the architecture of these two types of networks tend to be somewhat different, as will be explained subsequently.

Taking, as an example, the LAN 12 of Fig. 1, a LAN typically includes a number of "switches" 18, each having a number of "ports" 20. Some or all of the ports are connected to other devices, such as other switches 18 or to computers 22. Switches 18 (also referred to sometimes as "bridges") serve the function of dividing the local area network 12 into distinct regions or subsets of the local area network 12 to reduce data collisions on the interconnecting media 24 of the computer system. These media 24 (which typically comprise twisted pairs of wires, coaxial cables, or optical fiber) can only carry data for one network device at a time and, in consequence, create a communications bottleneck if there are a several devices attempting to use the media at a given point in time. The switches 18 reduce this problem by isolating individual media 24 from other individual media 24 of the local area network 12 so that data communication on one medium does not necessarily affect data communication on another medium.

Switches 18 operate in a theoretically rather simple fashion. When a switch 18 senses a data communication at one of its ports 20, it determines to which port, if any, it should forward the data. If, for example, the data that it sees at a particular port pertains to a device that is already coupled to that port (*i.e.* the sending device and the receiving device are on the same medium 24) that data will not be forwarded to the other ports of the switch, thereby eliminating unnecessary

traffic on other media of the network. If, for another example, a switch receives data at a first port which is destined for a device coupled to a second port, the switch will forward ("switch") the data to that second port. In consequence, only the devices connected to the first port media and to the second port media will be affected by this data transfer. If a switch 18 receives data at a first port, but does not know which device to send the data to, it will typically send the data out all of its ports and will wait for a response from the recipient device to determine which port that device is on. The device port number is then stored in a "table" so that, in the future, data for that device will be sent directly to the correct port.

Large LANs can assume a "topology" of mind-numbing complexity. As used herein, "topology" means the relative physical interconnections of the various devices within a computer network system 10, within a LAN, or within a subset of a LAN. While the appropriate "domain" for the topology is typically referred to as a LAN (or a subset of a LAN), the present invention is capable of determining the topology of any set of switches that define a broadcast domain. As is well known to those skilled in the art, a "broadcast domain" is a set of devices that receive data packets sent to the broadcast address of that domain. By "device" it is meant any digital electronic apparatus connected to the network that is provided with a unique network address within at least that portion of the network. Therefore, the devices illustrated in the LAN 12, for example, include the switches 18 and the computers 22. Of course, other devices can also be used in computer network system.

Typically, one or more computers 22 are coupled to the media 24 which, in turn, are coupled to one or more ports 20 of the switches 18. When multiple computers are coupled to a single medium 24, the medium is referred to as a "shared medium." In some instances, a "backbone" connection (*i.e. medium)* is made between switches 18, such as the backbone 26 shown in Fig. 1. The purpose of the backbone 26 is to provide very high speed communication between various switches of the local area network 12 for the rapid transfer of large quantities of data. Such backbone 26 connections are useful since switches 18, while very fast, do produce a delay or "latency" in the transmission of data.

A WAN 16 is typically used to coupled multiple LAN networks together for data communication. One of the best known WANs is the so-called Internet® which is used to couple many thousands of computers and LANs together for data communication. WANs typically use another type of network device known as a router 28 to direct data from one LAN to another. Most of the routers for the Internet are provide by Cisco Systems, Inc. of San Jose, California. However, it is becoming more prevalent to use switches (similar to switches 18) within WANs to perform this data directing function because switches are inherently faster than routers.

Most LANS and WANS transmit data in discrete chunks called "packets." A router 28 of a WAN will read the destination of the data packet, and then directs the data packet either directly or indirectly to a destination LAN or other computer device.

As noted previously, the Internet is the largest, best known, and most frequently used WAN in the world. The Internet utilizes a protocol known as TCP/IP for transmitting data through its routers. Therefore, a LAN 12 or a LAN 14 wishing to communicate over an Internet WAN 16 needs to send and receive data over a WAN connection 30 or 32 using the appropriate TCP/IP protocols. By providing a universal protocol for the transmission of data over the Internet, LANs and other devices of many different types can effectively communicate with each other over a worldwide WAN.

Fig. 1a is a block diagram of a typical prior art switch 18. A switch 18 includes logic 34, a receive buffer 36, a forwarding table 38, and a spanning tree table 40. The logic 34 communicates with the receive buffer 36, the forwarding table 38, the spanning tree table 40 via busses labeled 42, 44, and 46, respectively. Of course, these buses 42-46 illustrate logical buses, and may physically share some or all of the same conductive lines. A receive buffer 36 has a number of ports "n" which are typically numbered from 0 to n-1. Some or all of these ports may be connected to the media 24 of the network.

When data is received at one of the ports {0::n-1} of the receive buffer 36, the logic 34 reads the forwarding table 38 (stored in memory) to determine to which port(s) the data should forwarded. For example, data received at port 1 may be determined by the forwarding table 38 to be destined for a device on port n-2. The logic 34 will then instruct the receive buffer 36 to forward the data from port 1 to port n-2.

Sometimes data at a particular port is destined for multiple other ports. The logic 34 can also determine this from the forwarding table 38. If the destination for particular a data packet is not in the forwarding table 38, the logic 34 will instruct the receiver buffer to send the data out of all of the ports, and then will wait for a response from the destination device. When the response is received on a port of the switch, the forwarding table is updated so that, in the future, the receiving port of the switch will be associated with that device.

As noted previously, computer networks communicate using a number of protocols. While the biggest WAN in the world, the Internet, communicates with the TCP/IP protocol, many LANs communicate with other protocols such as the Novell IPX network protocol, the AppleTalk protocol, etc. However, regardless of the protocol, data tends to be sent in discrete chunks or "packets," such as the packet 48 illustrated in Fig. 1a. A packet 48 includes a destination D address, a source S address, and the data DATA being sent. Data is typically sent on networks in "packets" to promote better sharing of the media and other resources of the network.

The spanning tree table 40 (also stored in memory) ensures that the topology of the network assumes a "tree" structure. As it is well-known to those skilled in the art, a tree structure is one that does not include loop that can create uncontrolled packet duplication on the network. Therefore, networks having switches 18 with spanning tree tables can be conclusively presumed to form orderly tree structures. The standard for a spanning tree is defined by the IEEE 802.1 committee in standard 802.1D, incorporated herein by reference, which is available from the Institute for Electrical and Electronic Engineers (IEEE) New York, New York.

As will be apparent from this foregoing discussion, network topology, can rapidly become very complex. A switch, can have anywhere from a few to many hundreds of ports. When switches have only a few ports, they are often referred to as "bridges", while if they have many ports, they are usually referred to as "switches" or "switching hubs." In a LAN with, for example, several hundreds computer systems and perhaps a dozen switches, the number of possible interconnections number is in the many thousands.

To properly manage and/or diagnose a LAN, it is necessary to know the actual physical interconnections of the various devices of the LAN. While the media could be physically traced, this can be a huge job for a LAN of any size. As an alternative, U.S. Patent No. 5,226,120 of Brown et al. teaches a method for automatically generating the topology of a LAN. The solution suggested by Brown et al. was to provide specialized software at each of the "hubs" of their network to provide topology information that can be displayed at a control console.

The method disclosed by Brown et al., however, has several drawbacks. First, the hubs need to be provided with specialized software to provide the topology information to the control console. This, increases the complexity of the hubs and reduces their performance, since the hubs must provide computational tasks in addition to performing their core switching function. In addition, the solution of Brown et al. requires that each hub subscribe to the topology collecting protocols, meaning that network devices from only a single manufacturer can be used. This is a disadvantage in heterogeneous local area network configurations where devices (*e.g.* switches, hubs, bridges, routers) from various manufacturers may be used in a single LAN.

Disclosure of the Invention

The present invention provides a method and apparatus for efficiently determining the topology of a network of heterogeneous or homogeneous devices without resorting to wire-tracing, and without the requirement of specialized topology software running on the individual devices. In consequence, a LAN manager or LAN consultant can quickly determine the topology of a local area network for diagnostic and network optimization purposes.

In a broad sense, a method for determining the topology of a computer network includes reading port forwarding data from each device in at least a subset of a computer network. This port forwarding data is then processed and analyzed to derive the topology of at least the subset of the computer network. Preferably, the reading of the port forwarding data is preceded by discovering all the switches in the at least subset of the computer network to provide the maximum available port forwarding data for subsequent analysis.

More particularly, all of the devices are discovered by querying (*i.e.* "pinging") device addresses within a range of addresses that correspond to the at least subset of a computer network and monitoring for responses. Each responding device is categorizes as one of a configured device type (such as a switch type or a host type) and an unknown device type. Information about the device is store in a subnet device list, and if the device was a switch, information is also stored in a subnet switch list. As well known to those skilled in the art, in the context of the Internet Protocol ("IP") the "subnet" is a group of addresses within a broadcast domain. In the context of the Novell IPX protocol, the same construct (*i.e.* that which is referred to herein as a "subnet") is referred to as a "network."

For each device in the subnet device list, a port object is created for each port of the device, along with addresses and masks associated with that port. If the device is a switch type, the address is associated with that switch type. More particularly, if the network operates with a layer 3 protocol, the address is associated with the port includes Media Access Control ("MAC") addresses and protocol addresses, and the mask comprises a subnet mask specific to the IP protocol.

The preferred method of the present invention uses a recursive function to derive the topology from the resultant port forwarding data. More particularly, the recursive function efficiently places the devices in a tree structure that permits the physical interconnections of the switches to be efficiently displayed.

The present invention further includes a computer readable medium including program instructions for a computer system to perform the above-described method. In addition, an apparatus for determining the topology of a computer network in accordance with the present invention, utilizes a computer system to implement the above-described method.

As noted previously, the present invention is advantageous in that it operates over networks having devices of any type, and does not require the use of devices running specialized topology software for the development of the topology of the network. For example, a network consultant can use a portable computer implementing the method of the present invention to connect to a local area network (either directly or indirectly through a WAN) having devices of any type and efficiently develop the topology of the network for diagnostic and/or optimization purposes.

These and other advantages of the present invention will become apparent upon reading the following detailed descriptions and studying the various figures of the drawings.

Brief Description of the Drawings

Fig. 1 is an example of a prior art computer network system including several local area networks (LANs) and a wide area network (WAN);

Fig. 1a is an illustration of a prior art switch used in local area network;

Fig. 2 is an illustration of an apparatus of the present invention connect to a computer network for determining the topology of a LAN of the computer network;

Fig. 2a is a block-diagram of the apparatus of Fig. 2;

Fig. 3 is a flow-diagram of a process in accordance with the present invention for determining the topology of a computer network;

Fig. 4 is a flow-diagram illustrating the "Discover All Switches In A Switched LAN" of Fig. 3;

Fig. 5 illustrates the device pointer list and switch pointer list developed by the process of the present invention;

Figs. 5a, 5b, and 5c illustrate the device object for a generic host object, a specific switch module object, and a generic switch module object, respectively, of the present invention;

Fig. 6 is a flow-diagram illustrating the "Read Port Forwarding Data From Each Switch In LAN" step of Fig. 3;

Fig. 6a is an illustration used to describe the operation of the process of Fig. 6;

Fig. 7 is a flow-diagram of the "Derive Topology Of Switched LAN From Port Forwarding Data" step of Fig. 3;

Fig. 8 is a flow-diagram of the recursive function "PlaceInSubtree" of Fig. 7;

Fig. 9 is a flow-diagram of the method "isBelow" of Fig. 8;

Fig. 10 is an example of a network topology that is to be discovered by the method of the present invention;

Figs. 11a and 11b illustrate the device list and the scan list for the topology of Fig. 10;

Figs. 12a-12d illustrate the sequential development of the topology of the switched LAN using the process of Fig. 7;

Fig. 13 illustrate the "Provide Topology Information" step of Fig. 3;

Fig. 14 illustrates the "CreateMedia" function of Fig. 13; and

Fig. 15 illustrates a display of the topology information derived by the process of the present invention.

Best Modes for Carrying out the Invention

Figs. 1 and 1a were discussed previously in terms of the prior art. In Fig. 2, an apparatus 50 of the present invention is used to determine the topology of at least the portion or subset of a broadcast domain or LAN 12. The apparatus 50 can be directly coupled to the LAN 12 as indicated at 52 to develop the topology of LAN 12, or it can be indirectly coupled to LAN 12 through LAN 14 as indicated at 52a or through WAN 16 as indicated at 52b to also develop the topology of the LAN 12.

A block diagram of the apparatus 50 of the present invention is illustrated in Fig. 2a. It should be noted that the block diagram of Fig. 2a is illustrative of a typical personal computer (PC) which, under the control of programming instructions implementing the method of the present invention, perform the desired functionality of the present invention. Other computer architectures capable of implementing the method present invention can also be used.

In Fig. 2a, the apparatus 50 of the present invention includes a microprocessor 54 which is coupled to a high speed memory bus 56 by a bus 58 and to an input/output (I/O) bus 60 by a bus 62. The microprocessor 54 can be any one of a number of commonly used microprocessor, such as an Intel "x-86" compatible microprocessor, a Motorola power PC microprocessor, a SPARC processor, etc. The microprocessor 54 is coupled to the memory bus 56 and the I/O bus 60 by specialized chips (not known) known as "chipsets", which control the flow of data and which performs other functions within the computer apparatus 50.

Random access memory (RAM) 64 and read only memory (ROM) 66 are typically coupled to the memory bus 56. As noted, RAM 64 is coupled to bus 56 by a bus 68, and ROM 66 is coupled to memory bus 56 by a bus 70. The RAM 64 is typically volatile or "scratch-pad" memory, while ROM is non-volatile memory which often includes the start-up instructions for "booting" the apparatus 50.

A number of devices can be coupled to the I/O bus 60. These devices are typically referred to as "peripherals." Examples of such peripherals include a floppy drive 72, a CD ROM drive 74, a hard disk drive 76, a network card 78, and other I/O interfaces generically shown at 80. Each is coupled by its own bus to the I/O bus 60, *i.e.*, peripherals 72, 74, 76, 78, and 80 are coupled to the I/O bus 60 by buses 82, 84, 86, 88, and 90, respectively.

Some peripherals, such as floppy drive 72 and CD-ROM drive 74, are provided with removable computer readable medium. More particularly, floppy drive 72 is provided with a computer readable medium in the form of a floppy disk 92 and CD ROM drive 74 is provided with a removable computer readable medium in the form of a CD ROM disk 94. Hard drive 76 typically does not have a removable medium, *i.e.* its computer readable medium is fixed within the hard drive unit. Network card 78 is used to couple the apparatus 50 to a computer network, *e.g.*

the network card 78 can be coupled to a medium 24 of the local area network 12. The generic I/O interface 80 can, for example, be an Ethernet card, an RS-232 port, etc., and is coupled to external devices by a bus 96.

While conventional memory such as RAM 64 and ROM 66 are shown coupled to the high speed memory bus, and while the "peripherals" are shown to be coupled to the I/O bus, it should be noted that in many modern personal computer architectures, other busses are supported. In addition, certain high speed peripherals, such as peripherals dealing with real time video data, may be coupled to the memory bus 56, while some expansion memory may be coupled to the I/O bus 60. Therefore, the architecture illustrated in Fig. 2a is provided as an illustration of suitable hardware for the apparatus 50 of the present invention, and not by way of limitation.

It should be noted that virtually every device in the apparatus 50 of the present invention can provide some form of memory function. While the RAM 64, ROM 66, floppy drive 72, CD ROM 74, and hard drive 76 are more conventional "memory devices," other peripherals such as the network card 70 and I/O interface 80 also typically include buffers and other memories which can be accessed directly or indirectly by the microprocessor 50. In addition, memory can be accessed over the network via the network card 78, and from other devices over the I/O interface 80. Therefore, by "memory," it is meant herein any form of digital storage that can be accessed directly or indirectly by the microprocessor 54. Likewise, the term "computer readable medium" will be used herein to refer any device or media that supports memory. Traditional computer readable media include the floppy disk 92, the CD ROM 94, the contents of the hard drive 76, RAM 64, and ROM 66. However, the computer readable media can also be accessible over a network by the network card 78 or from other devices over the I/O interface 80.

In Fig. 3, a process 98 in accordance with the present invention begins at 100 and, in a step 102, discovers all of the switches in at least a subset of a switched LAN. Next, in a step 104, the port forwarding data is read from each switch in the LAN. In a step 106, the topology of the switched LAN is derived from a port forwarding data, and in a step 108, the topology information is provided. The process is then completed at 110.

Therefore, in a broad sense, a method for determining the topology of a computer network in accordance with the present invention includes reading port forwarding data from each device in at least a subset of a computer network. This port forwarding data is then processed and analyzed to derive the topology of at least the subset of the computer network. Preferably, the reading of the port forwarding data is proceeded by discovering all the switches in the at least subset of the computer network to provide the maximum available port forwarding data for subsequent analysis. The topology can then be processed and displayed in any convenient form.

In Fig. 4, the process 102 of Fig. 3 is illustrated in greater detail. More particularly, process 102 begins at 112 and an iterative loop is begun at 114. In the iterative loop, a counter i is

initialized to 0 and is compared to the number of addresses NUM_ADDRESSES within the LAN (or subset of the LAN) of interest. These addresses are typically sequentially assigned, and start at an offset address OFFSET. If the counter i is less than total number of addresses of interest, a step 116 "pings" the device at ADDRESS(OFFSET+i). The term "ping" is well-known to those skilled in the art as meaning sending a message to the designated address (*i.e.* "querying" a device address) and waiting for a reply. If a reply or response is not detected by step 118 within a "timeout" period, process control is returned to the iterative loop 114 to iterate the counter i by 1.

If a response to the ping of step 116 is detected by step 118, it is then determined in a step 120 if this is an MIB-II device. MIB-II devices are defined in the industry standard "Simple Network Management Protocol" or "SNMP" specifications well-known to those skilled in the art, and incorporated herein by reference. If step 120 determines that it is not an MIB-II device, a step 122 creates an unknown device object with one port, and saves the IP address. If step 120 determines that it is an MIB-II device, a step 124 determines whether the MIB-II device is a recognized switch. This is determined by making an SNMP "get request" for a MIB-II object. If the request is successful, then the device supports MIB-II. If step 124 does recognize the switch, a specific switch module object is created in step 126. If step 124 does not recognize the switch, a step 128 determines whether the device supports "bridge MIB group." Again, the bridge MIB group is defined by the Internet Engineering Task Force ("IETF") Requests for Comments (RFCs). Also defined by the IETF RFCs are IP and subnet masks. If step 128 determines that the device supports the bridge MIB group, a generic switch module object is created in step 132. If step 128 determines that it does not support bridge MIB group, then a generic host object is created in step 130.

It should be noted that while the present invention utilizes the SNMP protocol and MIB-II objects, that other management protocols can be used. For example, CMIP, the OSI management protocol can also be used for the same purpose. The OSI model and its associated protocols are described, for example, in The Open Book, by Marshall Rose, which is incorporated herein by reference.

After the completion of step 122, 126, 130, or 132, the device object is put in a subnet device list in a step 134. Step 136 determines whether the device is a switch and, if so, the switch is put in the subnet switch list. If step 136 determines that the device is not a switch, or after the performance of step 138, process control is returned to the iterative loop step 114 to iterate the counter i. After the counter has stepped through the total number of addresses NUM_ADDRESSES, the process is completed as indicated at 140.

Fig. 5 illustrates a data structure for the device pointer list and switch pointer list of the present invention. More particularly, the device pointer list 142 includes a number of pointers 144 which point to the "n" devices discovered by the step 102. Similarly, the switch pointer list 146

includes a number of pointers 148 which point to the switches discovered by step 102. It therefore follows that device pointer list 142 includes end pointers to point to devices ranging from device 0 to device n-1, *i.e.* from {0::n-1}. The switch pointer list 146 will include fewer pointers than the device pointer list unless all of the devices are switches. In this example, device 0 is on the device pointer list, but is not on the switch pointer list because it is not a switch.

Figs. 5a, 5b, and 5c, device objects 150, 152, and 154 that are produced by the method 102 of the present invention are respectively illustrated. In Fig. 5a, the device object 150 is a generic host object and includes the IP address, the MAC address, the type of device, the name of device, and the number of ports of the device represented by the generic host object 150. In Fig. 5b, the device object 152 is a specific switch module object which includes an IP address, a MAC address, the type of device, the name of device, the number of ports, and specific methods. In Fig. 5c, device object 154 is a generic switch module object and includes the IP address, the MAC address, the type of device, the name of device, the number of ports, and generic methods. The terms "IP address" is defined in the aforementioned IETF RFCs, and "MAC address" is defined in the aforementioned IEEE 802.1D standard.

As well-known to those skilled in the art, a software "object" is a construct including data and methods (i.e., specialized software procedures) which have certain attributes and which can perform designated functions in response to external calls and parameters. Object-oriented programming is used in the present invention in order to provide a flexible, scaleable, and powerful implementation of the processes of the present invention.

The process or method 102 is alternately described herein as "Phase I" of the process of the present invention. The basic purpose of Phase I is to construct the list of device types as illustrated in Fig. 5. In the following Table I, a pseudo-code listing is used to illustrate the process of Fig. 4. As it will be appreciated by those skilled in the art, the pseudo-code is readily transferable into a high-level object-oriented language such as C++.

TABLE 1 - PHASE 1 PSEUDO-CODE
(Construct Lists Of Device Types)

```
For each IP address that responds to a ping:
        Attempt MIB-II Get
        If not MIB-II device
                Create an unknown device with one port. Save IP address with
                port
        else it is a MIB-II device
                Read MIB-II systems group
                if recognized switch type
                        create specific switch module object
                else
                        if supports bridge MIB group
                                create generic switch object
                        else
                                create generic host object
```

put device in subnet's device list.
If it's a switch, also put it in the subnet's switch list.

It is therefore clear that at the end of Phase I, a list of devices for the subnet as well as a list of switches of the subnet is provided.

In Fig. 6, the process 104 of Fig. 3 is illustrated in greater detail. The process 104 begins at 156 and, in an iterative loop step 158, a counter i is initialized to 0. Next, in a step 160, the number of ports NUMPORTS(i) for DEVICE(i) is obtained. Next, in an iterative loop step 162, a counter j is initialized to 0 and, in a step 164, a port PORTOBJECT(j) is created. Next, in a step 166, the MAC address, the IP address, and the subnet mask are obtained and put into the OBJECT(j). A step 168 then determines if DEVICE(i) is a switch, and if not, process control is returned to iterative step 168 to iterate the counter j.

If the DEVICE(i) is a switch, a step 170 obtains a set of visible MAC addresses. By "visible", it is meant that the forwarding table includes the MAC address and the port that it was last seen on. If the MAC address is not in the forwarding table for this port, it is not "visible" to this port within this definition. Next, a step 172 associates the IP address to the MAC address if possible. In any subnet, an IP address is "associated with" or "bound to" only one MAC address. At any point in time, the system may "know" an IP address without knowing the associated MAC address, and vice versa. If both are known, the "associates" or "associating" step 172 is a straightforward task, as will be appreciated by those skilled in the art. For example, many devices include an ARP cache that can be queried for this association information. Process control is returned to iterative step 162 until the counter j is less than NUMPORT(i), at which time process control is returned to iterative loop step 168 as indicated by A. When the counter i is equal to NUM_DEV, the iterative loop 158 is exited and the process 104 is completed as indicated at 174.

In Fig. 6A, data structures created by the process 104 are illustrated. The device object 176 created by the process 104 includes the number of ports and the port list pointer for the device. The port list pointer points to the point list 178 which, in turn, points to the port objects 180. There will be one port object 180 created for each of the ports, i.e., if there are n ports for the device, there will be created port objects in the range {0::n-1}. In turn, each of the port objects will point to a "Port Child List" 182 and to a "MAC Address Seen" list 184. The Port Child List is filled-in by process 106 of Fig. 3. The MAC Address Seen list 184 includes the MAC number and the associated IP address for that MAC number. In the example of Fig. 6a, MAC number 0 has no associated IP address, MAC address 1 has the associated address 128.203.124.001 and MAC address 2 has no associated IP address.

As will be appreciated by those skilled in the art, for each device in the subnet device list, a port object is created for each port of the device, along with addresses and masks associated with that port. If the device is a switch type, the address is associated with that switch type. More

particularly, if the network operates with a layer 3 protocol, the address is associated with the port includes Media Access Control ("MAC") addresses and protocol addresses, and the mask comprises a subnet mask for the IP. Layer 3 protocols (including protocol addresses) are defined by various organizations including the IETF, the ISO, Novell, Inc., and Apple Computer, Inc. Examples include the IETF IP, the Novell IPX, and the Apple AppleTalk layer 3 protocols.

The process 104 of the present invention will also be referred herein as "Phase II" and, essentially, collects detailed information about the devices in the switched LAN. In Table II, below, the process 104 is further illustrated with pseudo-code. Again, it will apparent to those skilled in the art that this pseudo-code can be implemented in an appropriate object-oriented programming language such as C++.

TABLE 2 - PHASE 2 PSEUDO-CODE
(Collect Detailed Information About The Devices)

```
For each device in the subnet
        Get number of ports
        Create port object for each port
        for each port
                get MAC address
                get IP address
                get Subnet mask
                if it's a switch
                        Get set of MAC addresses visible
                        Associate IP address to MAC if possible
```

At the end of Phase II there is a collection of detailed information about all of the devices of the switched LAN. This data preferably completed in a separate pass from Phase I because it is possible to derive additional information in Phase II given the complete device list generated by Phase I.

In Fig. 7, the process 106 of Fig. 3 begins at 186 and, in a step 188, an arbitrary root r of the connectivity tree is chosen. For example, the first device on the list can be chosen as the root r, or any other arbitrary device on the list can be chosen as the root r. Next, in an iterative loop step 190, the counter os is initialized to 0. Then, a step 192 determines whether os is equal to the root r of the connectivity tree. If it is, process control is returned to iterative loop 190 to increment the counter os. If os is not equal to r, a recursive function PlaceInSubtree(os) for the switch os is called in step 194. The recursive function 194 will properly place the switch into the connectivity tree, as will be discussed in greater detail subsequently. After the completion of the recursive function 194, process control is returned to the iterative step 190 to iterate the counter os. When os is equal to the number of switches NUMSWITCH the process is completed as indicated in 196.

In Fig. 8, the cursive function PlaceInSubtree is entered at 198 and, in a step 200, the variables pr and ps are initialized. The variable pr is the port upon which the current switch sees s. The variable ps is the port upon which s sees this switch. Next, a function 202 isBelow is called to determine whether s is below pc, where pc is one of the children of pr. The function isBelow will be described in greater detail with reference to Fig. 9. Briefly, the function isBelow 202 returns a "TRUE" if s is below pc, (which is one of the children of pr) and returns a "FALSE" if s is not below pc. If the function 202 returns a TRUE, a recursive call is made to PlaceInSubtree with the parameter s which on the switch that owns pc. The process is then completed as indicated at 206.

If the function isBelow returns a FALSE, a step 208 gets the port pcr, which is the child of pr, where s is visible on pcr. A step 210 then determines if all ports are processed. If not, a step 212 removes pcr as the child of pr and a recursive call is made to function PlaceInSubtree passing the parameter sw on switch s. The parameter "sw" is the switch to which pcr is attached. After the return of the function call 214, process control is returned to step 208. When step 210 determines that all ports are processed, process 194 is completed as indicated at 206.

The function isBelow 202 is illustrated in greater detail in Fig. 9. The function isBelow begins at 216 and, in a step 218, variables sp and st are initialized. The variable sp is the switch to which p is attached. The variable st is the switch to which the receiver port is attached. Next, in a step 220, it is determined whether st sees p on the receiver port. If it does, a FALSE is returned by the function in a step 222 and the process is completed at 224.

If st does not see p on the receiver port, a step 226 gets the port pc of st which is not the receiver port. Next, a step 228 determines whether all the ports are processed and, if so, a FALSE is returned in step 222 and the process is completed at 224. If all the ports have not been processed, a step 230 determines whether sp is visible on pc. "Visible" is being used as previously described. If not, process control is returned to step 226. If, however, sp is visible on pc as determined by step 230, a TRUE is returned by step 232 and the process is completed at 224. As a result, the function 222 will return a TRUE if s is below pc, and will return a FALSE if s above or on the same level as pc.

Fig. 10 is a simple example of a network topology. There are seven switches in this example of a network, namely switches R, A, B, C, D, E, and F. The ports of each of the switches have been numbered. For example, the switch R has three ports numbered 1, 2, and 3. The switches A and B have ports numbered 1 and 2, and the switches C, D, E, and F have single ports numbered 1.

Figs. 11a and 11b illustrate the device list and the scan list developed by Phases I and II of the present invention. The device list of Fig. 11a includes the devices R, F, C, D, B, and E. The scan list of Fig. 11b indicates what switches are seen on the various ports of the variable switches.

In this list, a two-digit designation indicates the switch and port. For example, port 1 of switch R is "R1." Therefore, in this example, R1 sees switches A and D, R2 sees switches B, E, and F, R3 sees C, A1 sees R, A2 sees D, B1 sees R, B2 sees E and F, C1 sees R, F1 sees R, B, and E, and D1 sees A and R. As will be explained subsequently, these lists are used by the process 106 of Fig. 7 to properly generate the topology of the network as illustrated in Fig. 10.

As illustrated in Fig. 12a, the devices are conveniently read from the device list of Fig. 11a in sequential order. The switch R is first placed as the root of the tree. The device F is then selected and placed in the tree by the method of step 106 such that it is below port 2 of switch R. Device C is then selected from the device list 11a and placed by the process 106 at R3. The device D is then selected from the list and placed at R1.

As is evident from the foregoing example, the scan list is used to determine which port of the root the subsequent switches are "seen" on, and places those subsequent switches accordingly. Continuing with this example, in Fig. 12b the device D is selected from the device list of Fig. 11a, and the process 106 determines that D is below switch A and places switch D accordingly. Next, as seen in Fig. 12c, the switch B is chosen from the device list of Fig. 11a and the process 106 determines that the switch F is below switch B and places it accordingly. Finally, as seen in Fig. 12d, the device E is selected from the device list 11a and the process 106 determines that E is below B, but is not below F. Therefore, the devices E and F are placed at the same level below switch B.

The step 106 as described above will be also referred to herein as "Phase III." In the following Table 3, pseudo-code illustrating Phase III process is shown. Essentially, this phase III process is the derivation of the topology of the switched LAN. In the following Table 4, the port method isBelow is also illustrated in pseudo-code. It will be appreciated by those skilled in the art that the pseudo-code of Tables 3 and 4 can be easily translated into an object oriented programming language such as C++.

Since the spanning tree algorithm of the switches of the LAN prevent loops, we know that the actual topology of the network is a tree structure. The objective of the method of Phase III is to derive a connectivity tree. Since there are no cycles any switch can serve as a root for this connectivity tree. The branches of the tree are defined by port connectivity. Each switch has an associated set of ports called its children. A child port is bound to a set of other ports which, in turn, are bound to switches. The switches can therefore be thought of as a parent/child hierarchy with the ports in between.

TABLE 3 - PHASE 3 PSEUDO-CODE
(Topology derivation)

Pick a switch **r** to be the root of the connectivity tree.

For each other switch os call **PlaceInSubtree** on the root passing **os** as a parameter.

Let **PlaceInSubtree** be a method on the class *switch* which places switch **s** into the subtree for which the receiver is the root.

Let **pr** be the port upon which **this** switch sees **s**.
Let **ps** be the port upon which **s** sees **this** switch
If **s** is below **pc,** one of the children of **pr**,
then call **PlaceInSubtree** on the switch that owns **pc**, passing **s** as a parameter
else
for each port **pcr**, child of **pr**, such that **s** is visible on **pcr**
remove **pcr** as child of **pr**
call **PlaceInSubtree** on switch **s** passing switch **sw**, the
switch to which **pcr** is attached, as a parameter.

TABLE 4 - PORT METHOD PSEUDO-CODE

Let **isBelow** be a method on the class *port* which accepts a port **p** as a parameter. The method returns true if the switch that **p** is attached to is in the set of switches below the receiver *port*.

Let **sp** be the switch that **p** is attached to.
Let **st** be the switch that the receiver port is attached to.
If **st** sees **p** on the receiver port, then return false.
Otherwise, for each port **pc** of the **st** which is not the receiver port,
If **sp** is visible on **pc**, then return true
Else return false.

In Fig. 13, the process 108 of Fig. 3 is illustrated in greater detail. Process 108 begins at 234 and, in a step 236, a function call is made to CreateMedia on the root with the uplink equals null. This function call 236 will be discussed in greater detail with reference to Fig. 14. Next, a step 238 updates and displays the topology of the network. A step 240 senses user input on the display (i.e., the display is interactive) and will update and display in step 238 in response to the user input. The user input can be made, for example, with a mouse or other pointing device, or by a keyboard command.

In Fig. 14, the process 236 of Fig. 13 begins at 242, with the receipt of the parameter "u" in a step 244. An iterative loop step 246 initializes a counter i to 0, and in a step 248, a medium object M is created. If p=u process control is returned to the iterative loop step 246 to iterate the counter i. A step 250 determines that p is not equal to u, p is added to the object M in a step 252 and a step 254 gets a child pc of p. If step 256 determines that all the children have been processed, process control is returned to iterative loop step 246 to iterate the counter i. If not all the children have been processed, as determined by step 256, a step 258 adds pc to the object M. Process control is then returned to step 254. After the iterative loop step 246 determines that i is equal to NUMPORT(s), the process is then completed as indicated at step 260.

Fig. 15 illustrates one possible display that can be produced by the step 138 of Fig. 13. Of course, there are many other ways well known to those skilled in the art for displaying network

topologies. In this display, there are a number of media represented at 262 and a number of ports as indicated 264. The display illustrated in Fig. 15 is a interactive interface preferably used in a mouse of other pointing device type system. In this example, the media ethernet 0 has been selected and the ports P1 of two devices are indicated to be coupled to ethernet0, both graphically with the two dots 266 which connect P1 ports to the ethernet0 medium.

While this invention has been described in terms of several preferred embodiments, there are alterations, permutations, and equivalents which fall within the scope of this invention. It should also be noted that there are may alternative ways of implementing both the process and apparatus of the present invention. It is therefore intended that the following appended claims be interpreted as including all such alterations, permutations, and equivalents as fall within the true spirit and scope of the present invention.

Claims

1. A method for determining the topology of a computer network comprising:

reading port forwarding data from each switch in at least a subset of a computer network; and

deriving a topology of said at least a subset of a computer network from said port forwarding data.

2. A method for determining the topology of a computer network as recited in claim 1 further comprising:

discovering all switches in said at least subset of a computer network prior to reading port forwarding data from each switch in said at least subset of a computer network.

3. A method for determining the topology of a computer network as recited in claim 2 wherein discovering all switches comprises:

querying device addresses within a range of addresses that corresponds to said at least subset of a computer network and monitoring for responses; and

for each device address for which there was a response, categorizing the device at that address as one of a switch type, a host type, and an unknown device type.

4. A method for determining the topology of a computer network as recited in claim 3 wherein said switch type is represented by one of a switch module object and a generic switch module object, and said host type is represented by a generic host object.

5. A method for determining the topology of a computer network as recited in claim 4 where, for each device address for which there was a response from a device, storing information concerning said device in a subnet device list and, if said device is a switch type, storing information concerning said switch type in a subnet switch list.

6. A method for determining the topology of a computer network as recited in claim 5 wherein reading port data comprises:

for each device in said subnet device list, creating a port object for each port in said device and, for each port, obtaining addresses and mask associated with said port and, if said device is a switch type, associating said addresses.

7. A method for determining the topology of a computer network as recited in claim 6 wherein said network operates with a layer 3 protocol, and wherein said addresses associated with each port comprises MAC addresses and protocol addresses, and wherein said mask comprises a subnet mask.

8. A method for determining the topology of a computer network as recited in claim 7 wherein said layer 3 protocol is a TCP/IP protocol, and wherein said protocol addresses are IP addresses.

9. A method for determining the topology of a computer network as recited in claim 2 wherein associating said addresses includes obtaining a set of MAC addresses visible at said port, and associating the IP address of said port to said MAC addresses visible, if possible.

10. A method for determining the topology of a computer network as recited in claim 1 wherein deriving a topology of at least one subnet includes a recursive function.

11. A method for determining the topology of a computer network as recited in claim 10 wherein said recursive function includes placing a switch type device in a tree structure.

12. A computer readable medium including program instructions for a computer system coupled to a computer network to determine the topology of said computer network comprising:

program instructions for a computer system coupled to a computer network to read port forwarding data from each switch in at least a subset of said computer network; and

program instructions for said computer system to derive a topology of said at least a subset of a computer network from said port forwarding data.

13. A computer readable medium including program instructions as recited in claim 12 further comprising program instructions for:

discovering all switches in said at least subset of a computer network prior to reading port forwarding data from each switch in said at least subset of a computer network.

14. A computer readable medium including program instructions as recited in claim 13 wherein said program instructions for discovering all switches comprise:

querying device addresses within a range of addresses that corresponds to said at least subset of a computer network and monitoring for responses; and

for each device address for which there was a response, categorizing the device at that address as one of a switch type, a host type, and an unknown device type.

15. A computer readable medium including program instructions as recited in claim 14 wherein said switch type includes a switch module object and a generic switch object, and said host type includes a generic host object.

16. A computer readable medium including program instructions as recited in claim 15 where, for each device address for which there was a response from a device, there are program instructions for storing information concerning said device in a subnet device list and, if said device is a switch type, storing information concerning said switch type in a subnet switch list.

17. A computer readable medium including program instructions as recited in claim 16 wherein said program instructions for reading port data comprises:

for each device in said subnet device list, creating a port object for each port in said device and, for each port, obtaining addresses and mask associated with said port and, if said device is a switch type, associating said addresses.

18. A computer readable medium including program instructions as recited in claim 17 wherein said network operates with a TCP/IP protocol, and wherein said addresses associated with each port comprises MAC addresses and IP addresses, and wherein said mask comprises a subnet mask.

19. A computer readable medium including program instructions as recited in claim 13 wherein said program instructions for associating said addresses includes obtaining a set of MAC addresses visible at said port, and for associating the IP address of said port to said MAC addresses visible, if possible.

20. A computer readable medium including program instructions as recited in claim 12 wherein said program instructions for deriving a topology of at least one subset includes a recursive function.

21. A computer readable medium including program instructions as recited in claim 20 wherein said program instructions for a recursive function includes placing a switch type device in a tree structure.

22. A method for determining the topology of a computer network comprising:

step for reading port forwarding data from each switch in at least a subset of a computer network; and

step for deriving a topology of said at least a subset of a computer network from said port forwarding data.

23. A method for determining the topology of a computer network as recited in claim 22 further comprising:

step for discovering all switches in said at least subset of a computer network prior to reading port forwarding data from each switch in said at least subset of a computer network.

24. A method for determining the topology of a computer network as recited in claim 23 wherein said step for discovering all switches comprises:

step for querying device addresses within a range of addresses that corresponds to said at least subset of a computer network and monitoring for responses; and

for each device address for which there was a response, step for categorizing the device at that address as one of a switch type, a host type, and an unknown device type.

25. A method for determining the topology of a computer network as recited in claim 24 wherein said switch type includes a switch module object and a generic switch object, and said host type includes a generic host object.

26. A method for determining the topology of a computer network as recited in claim 25 where, for each device address for which there was a response from a device, there is a step for storing information concerning said device in a subnet device list and, if said device is a switch type, there is step for storing information concerning said switch type in a subnet switch list.

27. A method for determining the topology of a computer network as recited in claim 26 wherein reading port data comprises:

for each device in said subnet device list, step for creating a port object for each port in said device and, for each port, step for obtaining addresses and mask associated with said port and, if said device is a switch type, step for associating said addresses.

28. A method for determining the topology of a computer network as recited in claim 27 wherein said network operates with a TCP/IP protocol, and wherein said addresses associated with each port comprises MAC addresses and IP addresses, and wherein said mask comprises a subnet mask.

29. A method for determining the topology of a computer network as recited in claim 23 wherein said step for associating said addresses includes step for obtaining a set of MAC addresses visible at said port, and step for associating the IP address of said port to said MAC addresses visible, if possible.

30. A method for determining the topology of a computer network as recited in claim 22 wherein said step for deriving a topology of at least one subset includes step for performing a recursive function.

31. A method for determining the topology of a computer network as recited in claim 30 wherein said step for performing a recursive function includes step for placing a switch type device in a tree structure.

32. An apparatus for determining the topology of a computer network comprising:

means for reading port forwarding data from each switch in at least a subset of a computer network; and

means for deriving a topology of said at least a subset of a computer network from said port forwarding data.

33. An apparatus for determining the topology of a computer network as recited in claim 32 further comprising:

means for discovering all switches in said at least subset of a computer network prior to reading port forwarding data from each switch in said at least subset of a computer network.

34. An apparatus for determining the topology of a computer network as recited in claim 33 wherein said means for discovering all switches comprises:

means for querying device addresses within a range of addresses that corresponds to said at least subset of a computer network and monitoring for responses; and

for each device address for which there was a response, means for categorizing the device at that address as one of a switch type, a host type, and an unknown device type.

35. An apparatus for determining the topology of a computer network as recited in claim 34 wherein said switch type includes a switch module object and a generic switch object, and said host type includes a generic host object.

36. An apparatus for determining the topology of a computer network as recited in claim 35 where, for each device address for which there was a response from a device, there is a means for storing information concerning said device in a subnet device list and, if said device is a switch type, there is means for storing information concerning said switch type in a subnet switch list.

37. A method for determining the topology of a computer network as recited in claim 36 wherein reading port data comprises:

for each device in said subnet device list, means for creating a port object for each port in said device and, for each port, means for obtaining addresses and mask associated with said port and, if said device is a switch type, means for associating said addresses.

38. An apparatus for determining the topology of a computer network as recited in claim 37 wherein said network operates with a TCP/IP protocol, and wherein said addresses associated with each port comprises MAC addresses and IP addresses, and wherein said mask comprises a subnet mask.

39. An apparatus for determining the topology of a computer network as recited in claim 33 wherein said means for associating said addresses includes means for obtaining a set of MAC addresses visible at said port, and means for associating the IP address of said port to said MAC addresses visible, if possible.

40. An apparatus for determining the topology of a computer network as recited in claim 32 wherein said means for deriving a topology of at least one subset includes means for performing a recursive function.

41. An apparatus for determining the topology of a computer network as recited in claim 40 wherein said means for performing a recursive function includes means for placing a switch type device in a tree structure.

42. An apparatus for determining the topology of a computer network comprising:

a central processing unit;

program instruction memory coupled to said central processing unit;

port forwarding data memory coupled to said central processing unit;

an I/O port coupled to said central processing unit;

a switch port reader comprising said central processing unit and switch port reader program instructions stored in said program instruction memory, said switch port reader being coupled to said computer network through said I/O port to read port forwarding data from switches in at least a subset of a computer network and to store said port forwarding data in port forwarding data memory; and

a port forwarding data analyzer comprising said central processing unit and port forwarding data analyzer program instructions stored in said program instruction memory, said port forwarding data analyzer deriving the topology of said at least subset of a computer network using said port forwarding data stored in said port forwarding data memory.

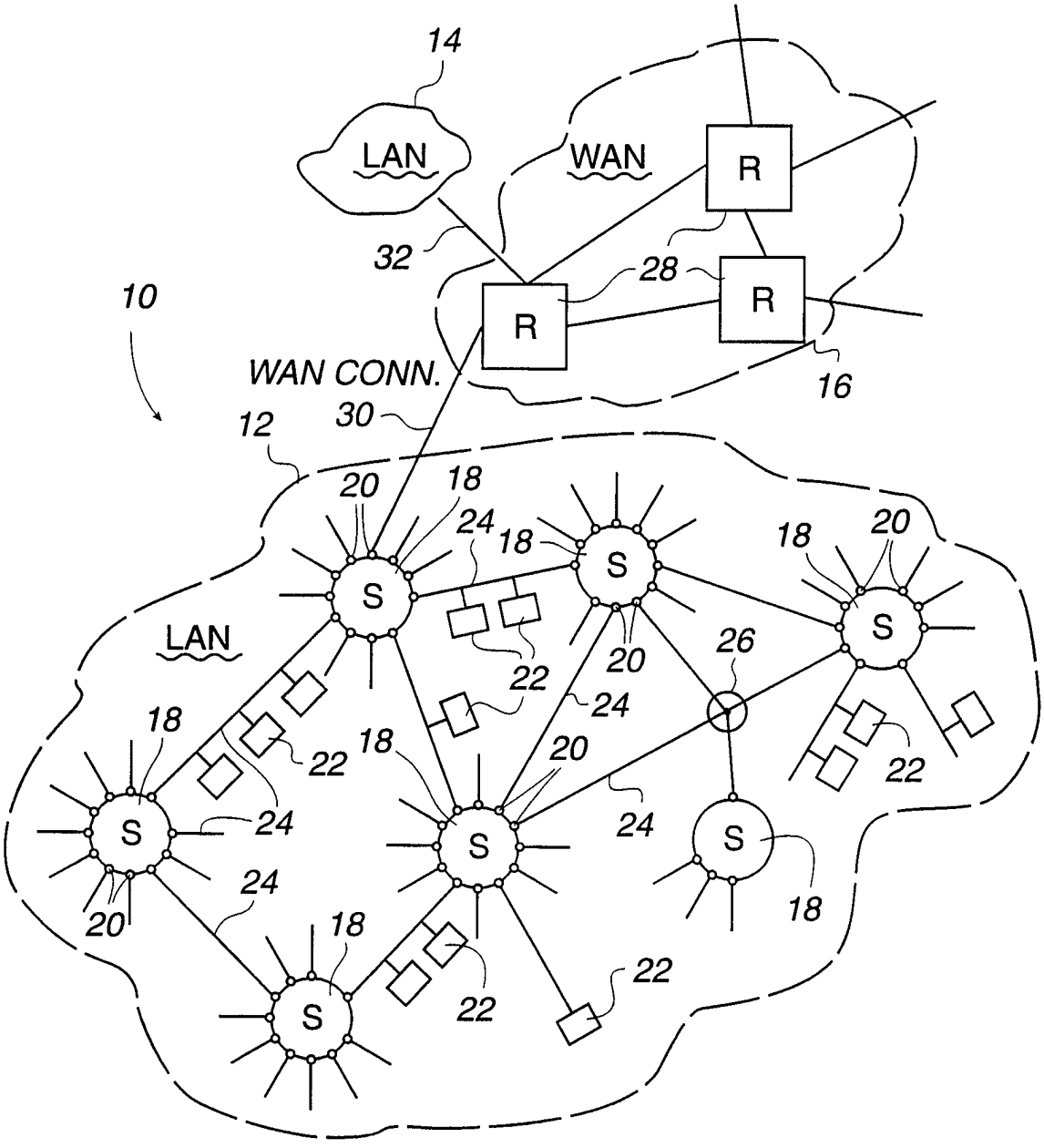


Fig. 1
Prior Art

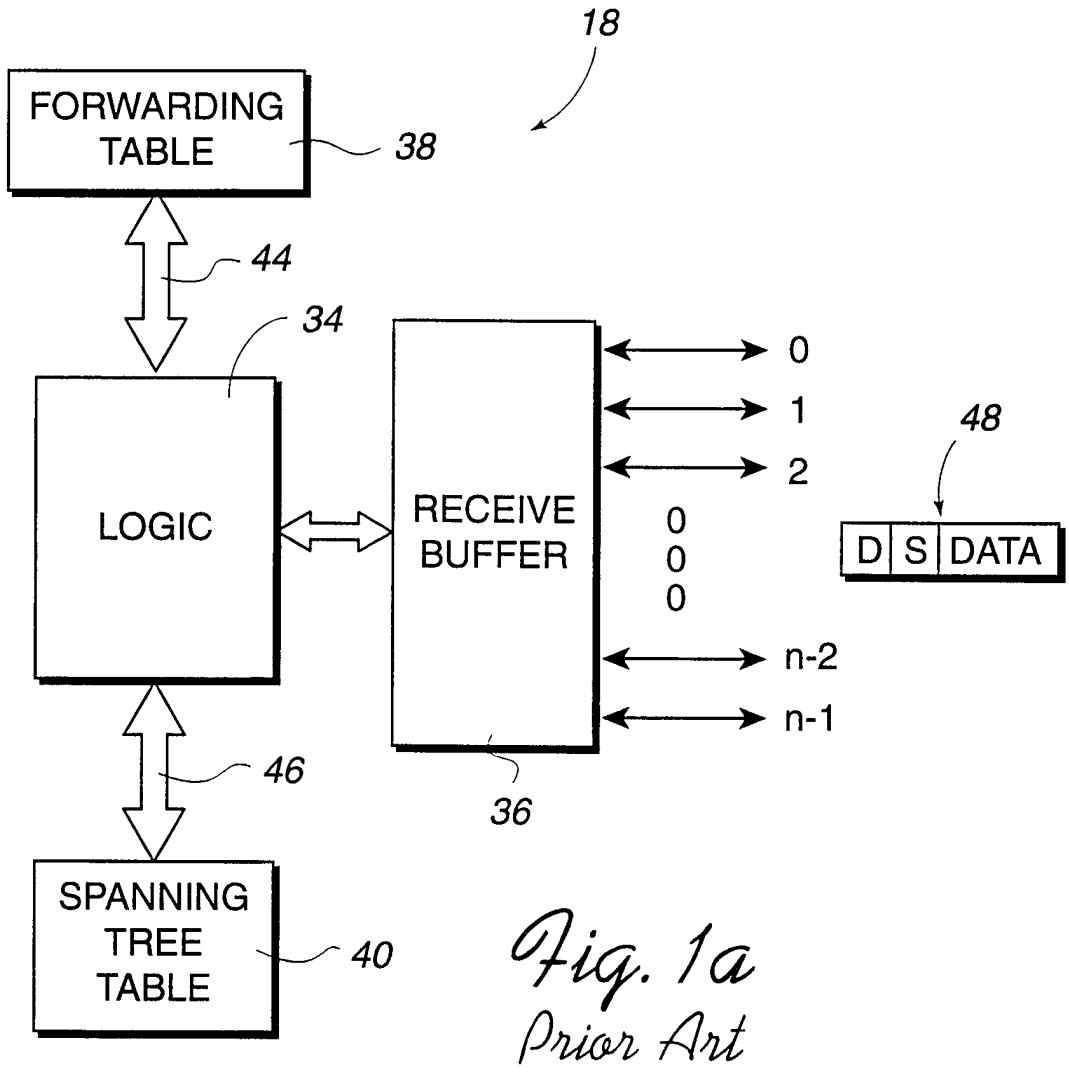


Fig. 1a
Prior Art

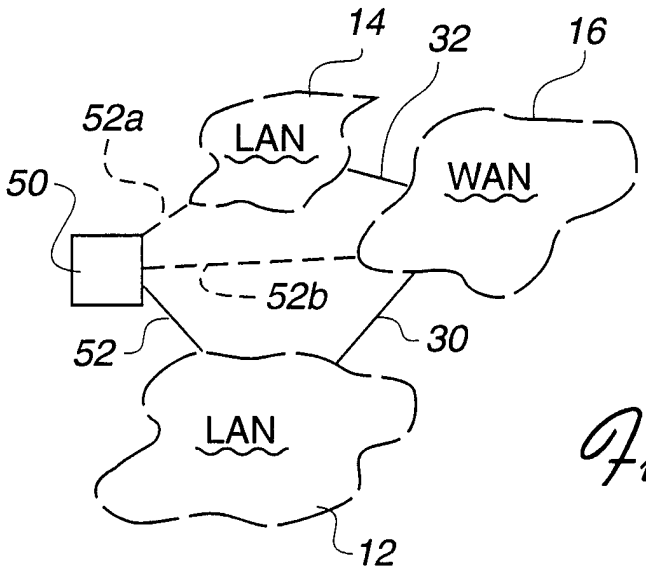

14
32
16
52a
LAN
WAN
50
52b
52
30
LAN
12

Fig. 2

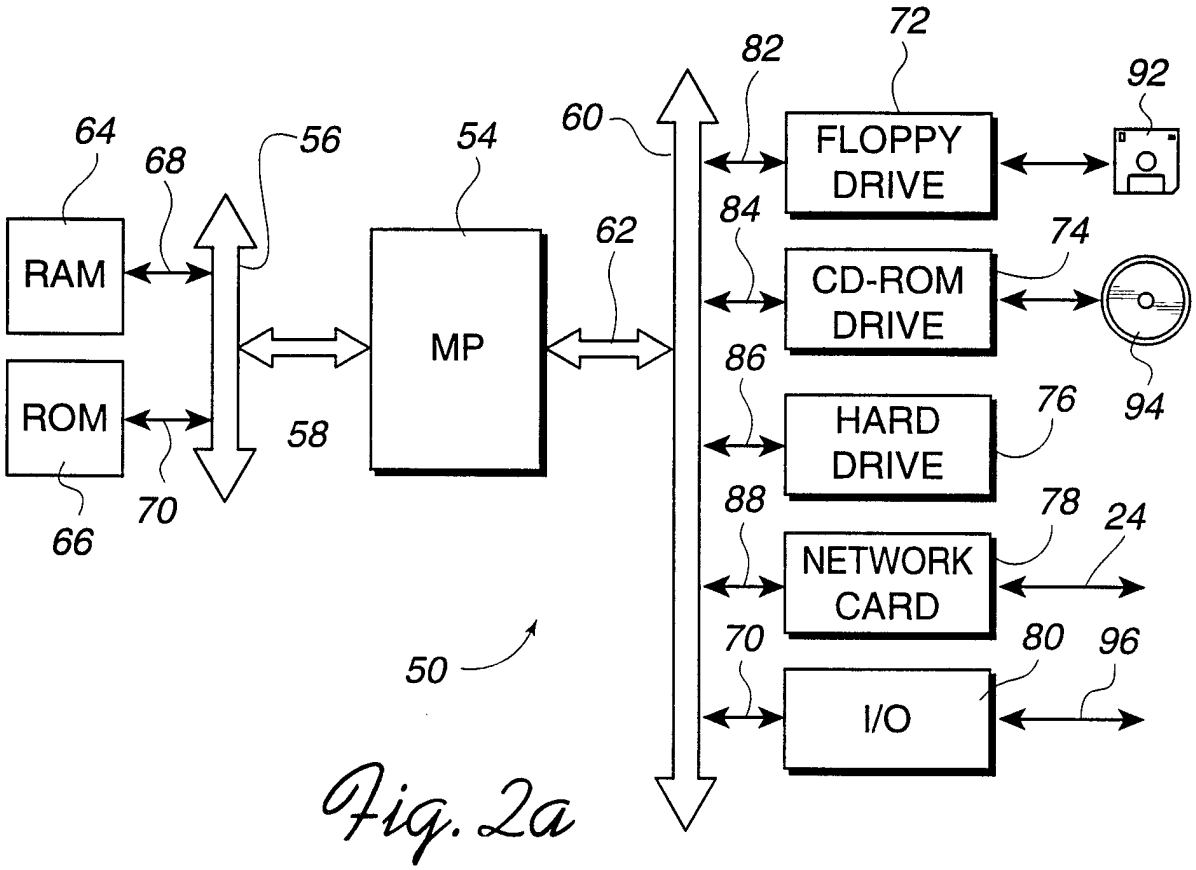

72
82
92
60
64
68
56
54
FLOPPY DRIVE
62
84
74
RAM
MP
CD-ROM DRIVE
86
ROM
58
HARD DRIVE
76
94
88
78
24
70
NETWORK CARD
66
70
50
80
96
I/O

Fig. 2a

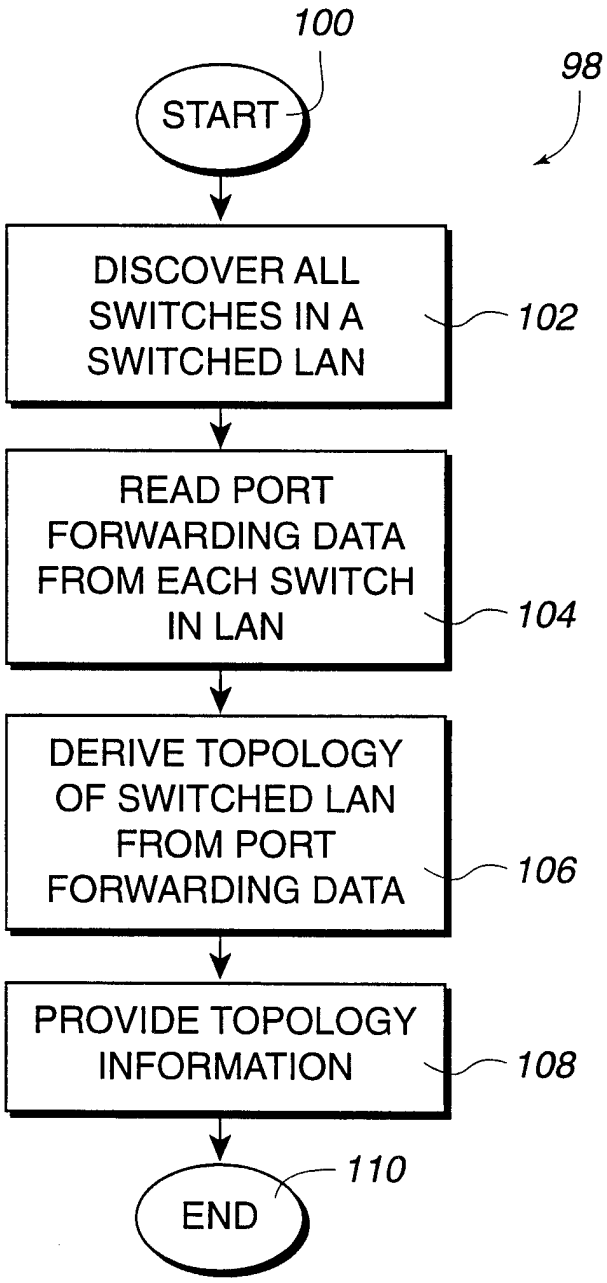

100
98
START
DISCOVER ALL SWITCHES IN A SWITCHED LAN
102
READ PORT FORWARDING DATA FROM EACH SWITCH IN LAN
104
DERIVE TOPOLOGY OF SWITCHED LAN FROM PORT FORWARDING DATA
106
PROVIDE TOPOLOGY INFORMATION
108
110
END

Fig. 3

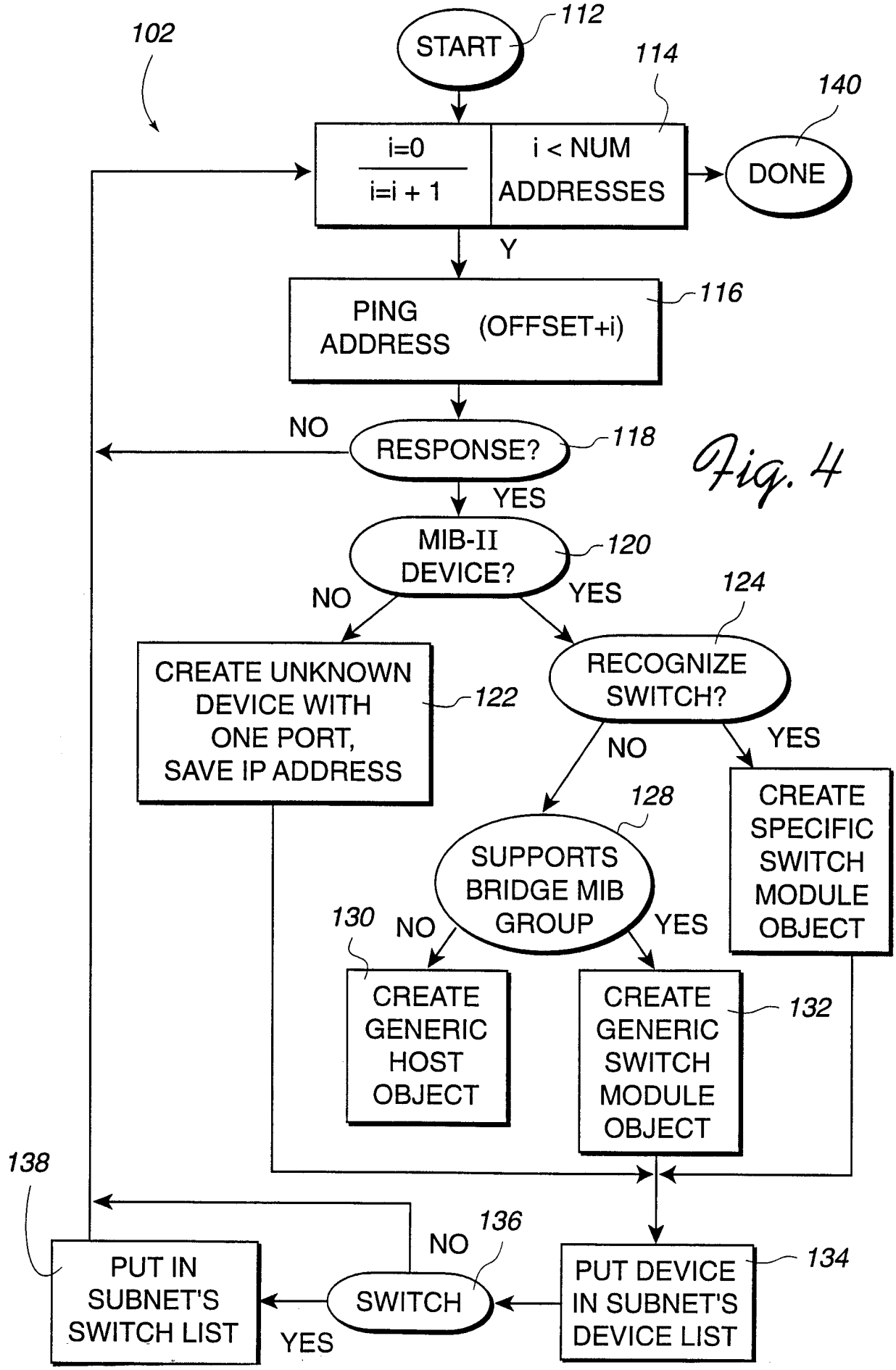

102
START
112
114
140
i=0
i=i + 1
i < NUM ADDRESSES
DONE
Y
PING ADDRESS (OFFSET+i)
116
NO
RESPONSE?
118
YES
MIB-II DEVICE?
120
NO
YES
124
CREATE UNKNOWN DEVICE WITH ONE PORT, SAVE IP ADDRESS
122
RECOGNIZE SWITCH?
NO
YES
CREATE SPECIFIC SWITCH MODULE OBJECT
128
SUPPORTS BRIDGE MIB GROUP
130
NO
YES
CREATE GENERIC HOST OBJECT
CREATE GENERIC SWITCH MODULE OBJECT
132
138
NO
136
PUT IN SUBNET'S SWITCH LIST
SWITCH
YES
PUT DEVICE IN SUBNET'S DEVICE LIST
134

Fig. 4

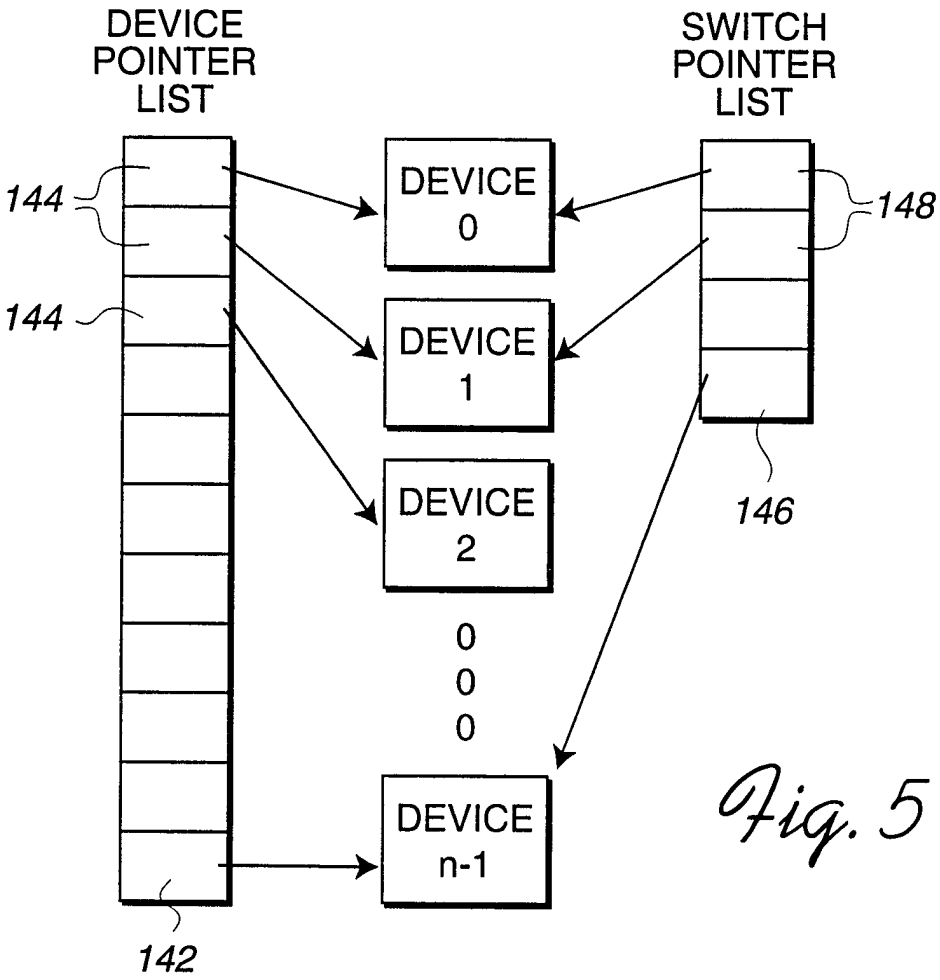

DEVICE POINTER LIST
SWITCH POINTER LIST
144
144
148
146
142
DEVICE 0
DEVICE 1
DEVICE 2
0
0
0
DEVICE n-1

Fig. 5

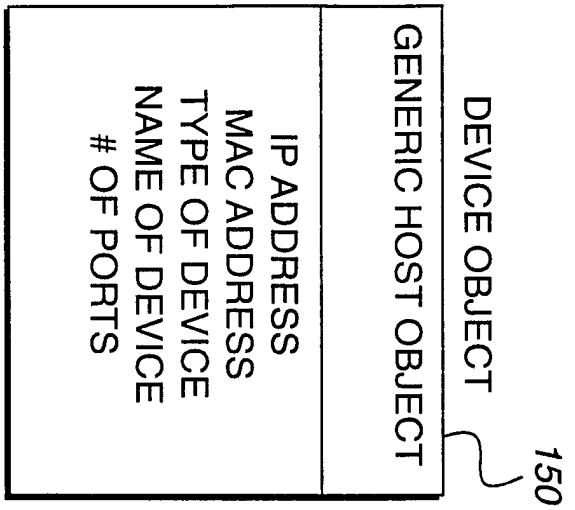


Fig. 5a

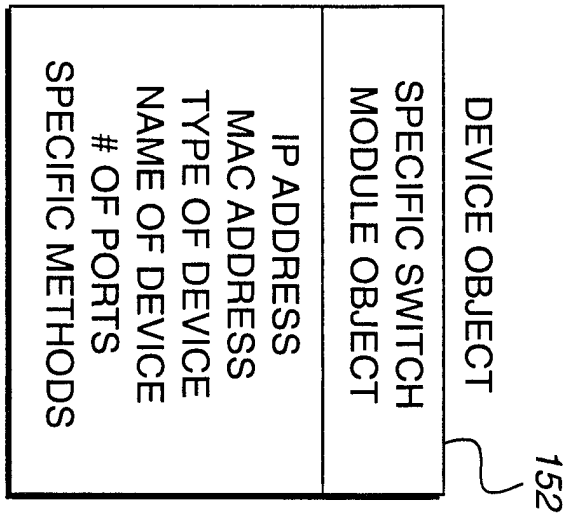


Fig. 5b

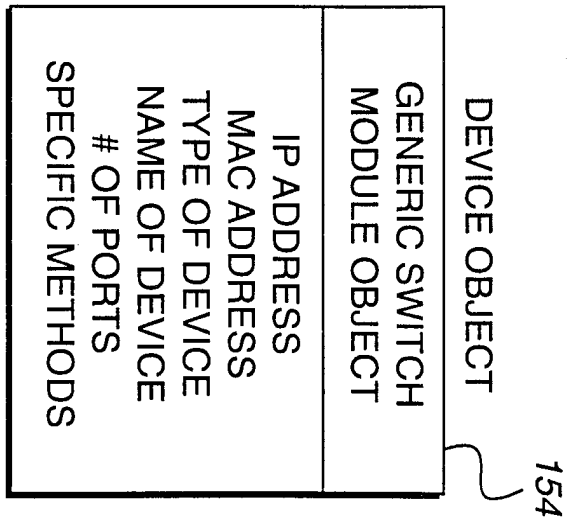


Fig. 5c

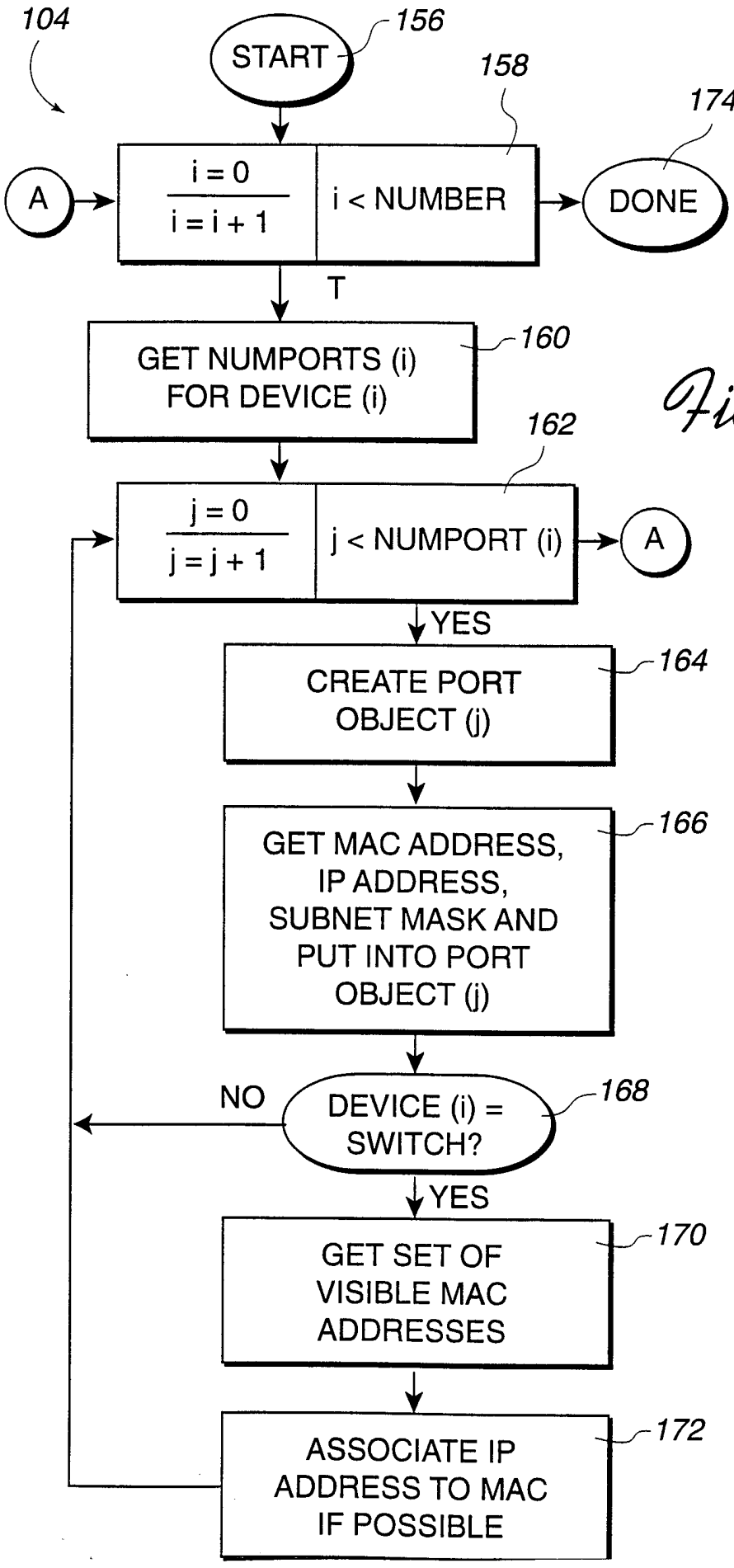

104
START
156
158
174
A
i = 0
i = i + 1
i < NUMBER
DONE
T
GET NUMPORTS (i)
FOR DEVICE (i)
160
162
j = 0
j = j + 1
j < NUMPORT (i)
A
YES
CREATE PORT
OBJECT (j)
164
GET MAC ADDRESS,
IP ADDRESS,
SUBNET MASK AND
PUT INTO PORT
OBJECT (j)
166
NO
DEVICE (i) =
SWITCH?
168
YES
GET SET OF
VISIBLE MAC
ADDRESSES
170
ASSOCIATE IP
ADDRESS TO MAC
IF POSSIBLE
172

Fig. 6

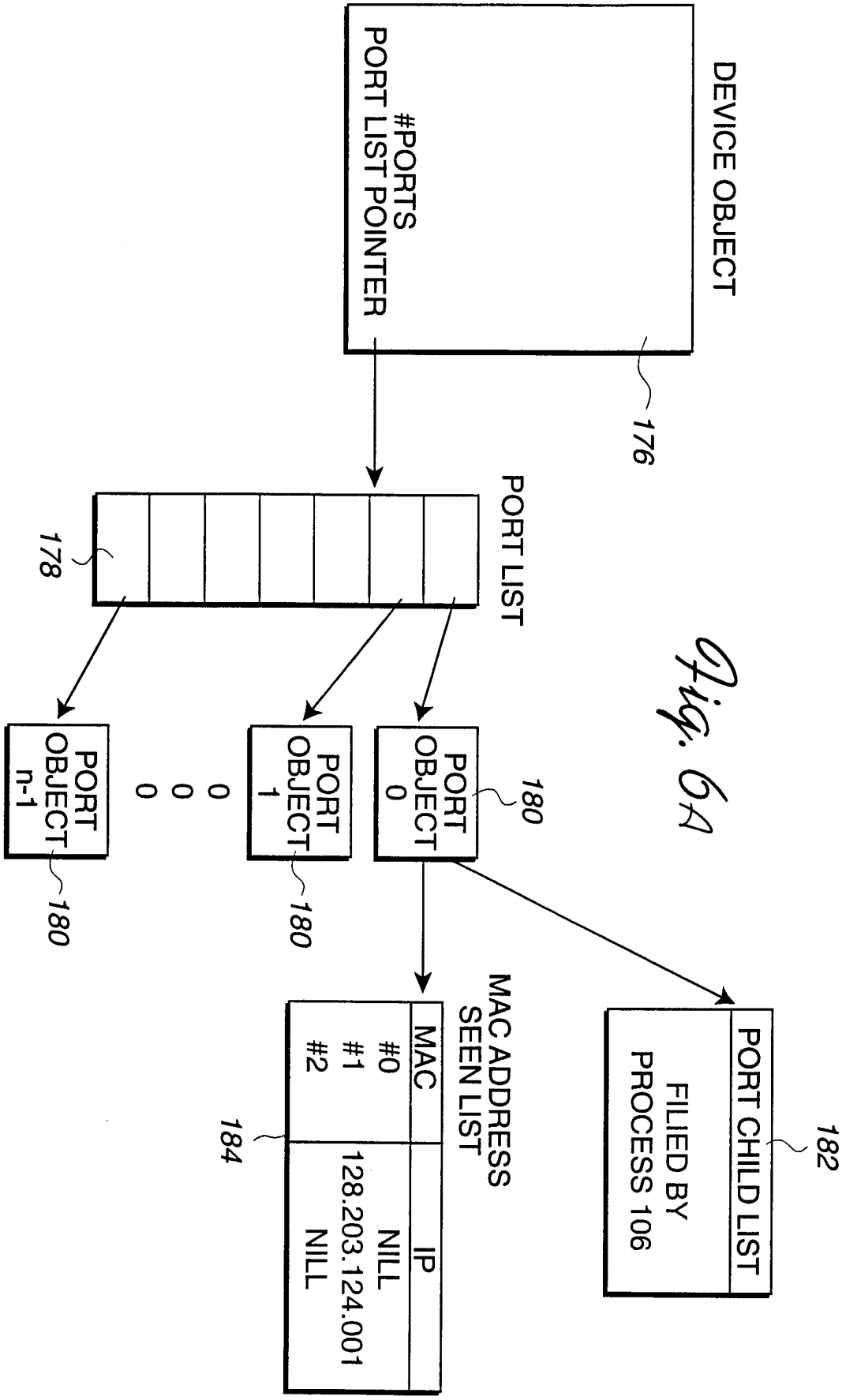

DEVICE OBJECT
#PORTS
PORT LIST POINTER
176
PORT LIST
178
PORT OBJECT 0
PORT OBJECT 1
PORT OBJECT n-1
180
0
0
0
0
180
180
PORT CHILD LIST
FILIED BY PROCESS 106
182
MAC ADDRESS SEEN LIST
MAC
IP
#0
#1
#2
NILL
128.203.124.001
NILL
184

Fig. 6A

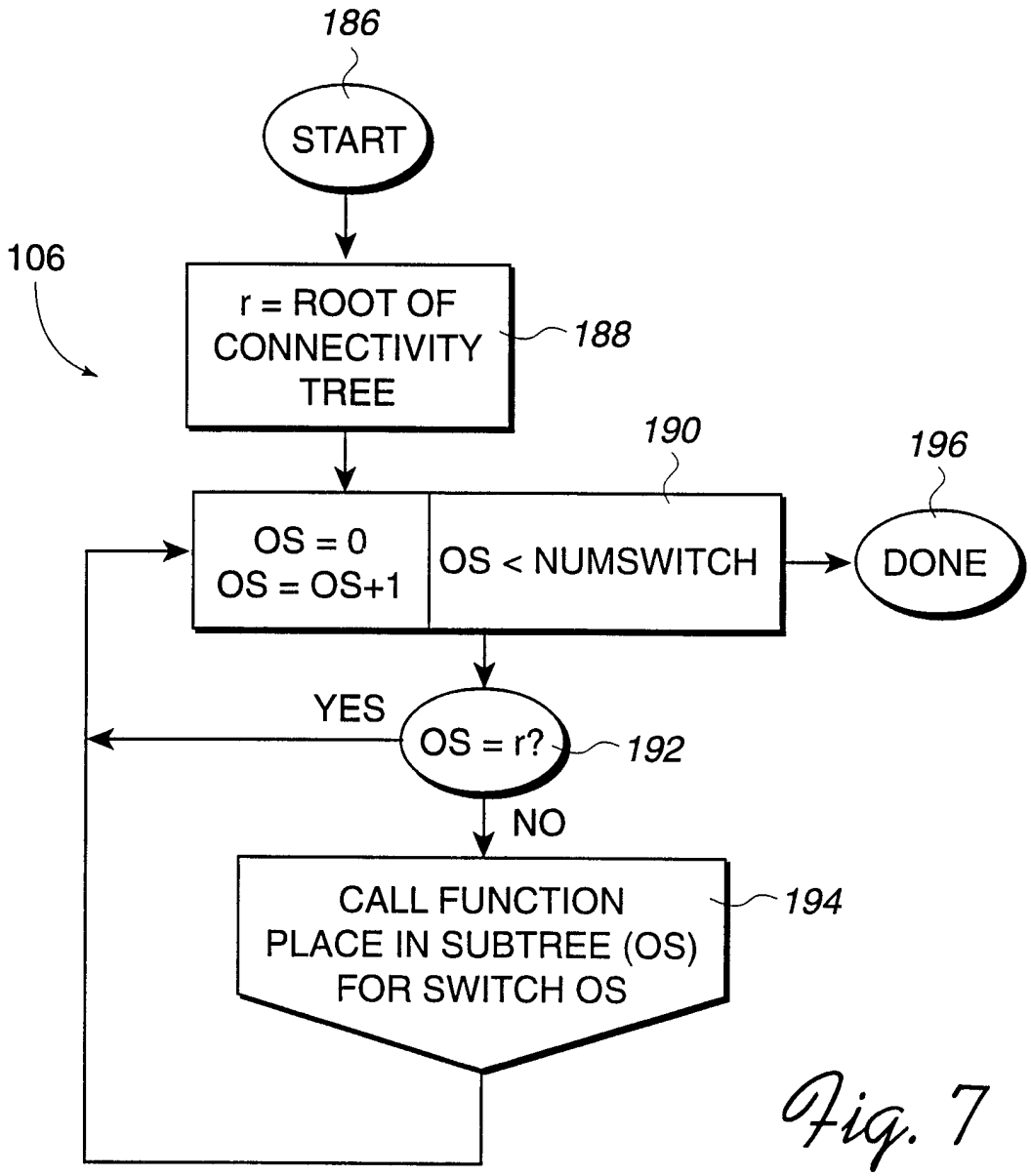

186
START
106
r = ROOT OF CONNECTIVITY TREE
188
190
196
OS = 0
OS = OS+1
OS < NUMSWITCH
DONE
YES
OS = r?
192
NO
CALL FUNCTION PLACE IN SUBTREE (OS) FOR SWITCH OS
194

Fig. 7

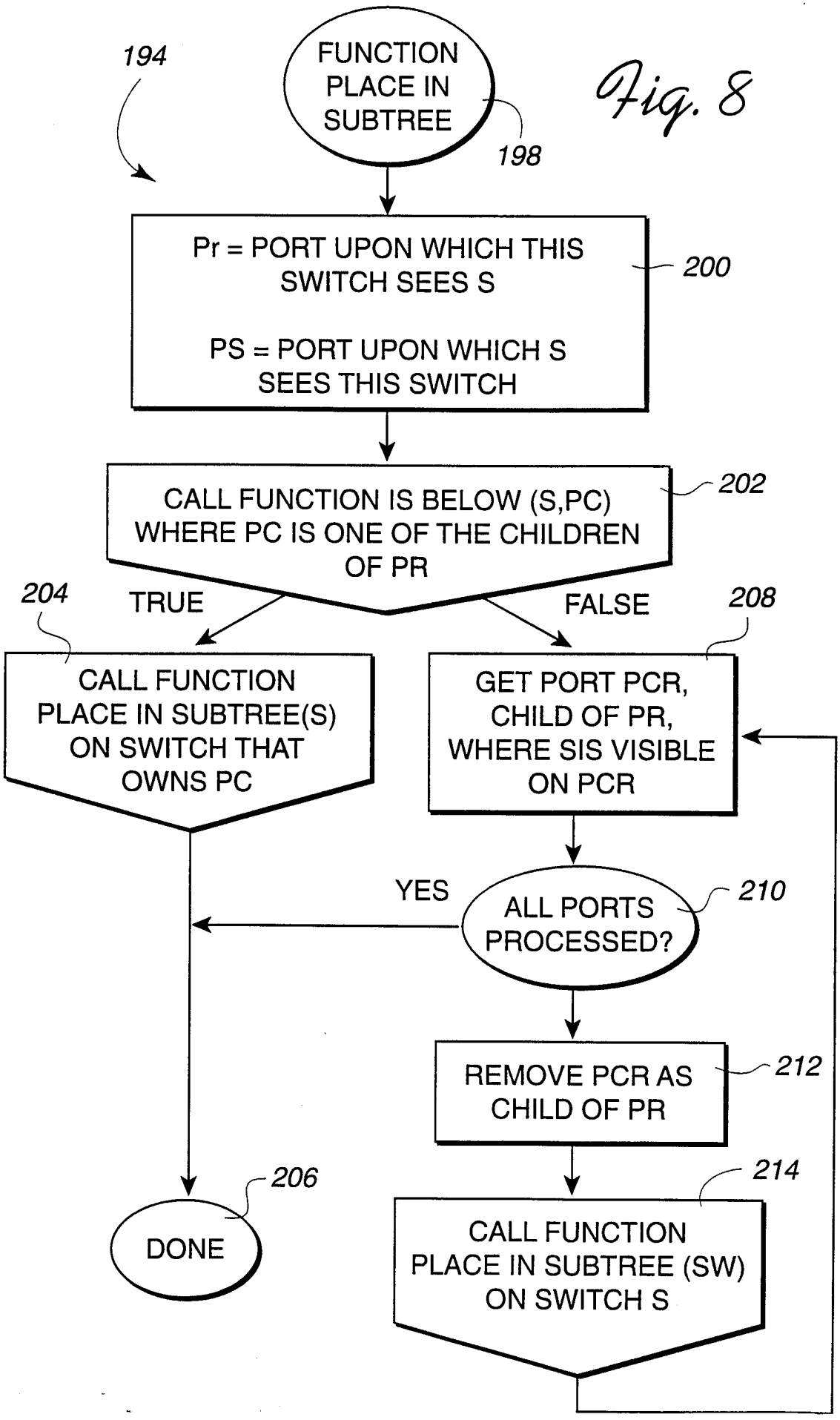

194
FUNCTION PLACE IN SUBTREE
198
Pr = PORT UPON WHICH THIS SWITCH SEES S
PS = PORT UPON WHICH S SEES THIS SWITCH
200
CALL FUNCTION IS BELOW (S,PC) WHERE PC IS ONE OF THE CHILDREN OF PR
202
204
TRUE
FALSE
208
CALL FUNCTION PLACE IN SUBTREE(S) ON SWITCH THAT OWNS PC
GET PORT PCR, CHILD OF PR, WHERE SIS VISIBLE ON PCR
YES
ALL PORTS PROCESSED?
210
REMOVE PCR AS CHILD OF PR
212
206
DONE
214
CALL FUNCTION PLACE IN SUBTREE (SW) ON SWITCH S

Fig. 8

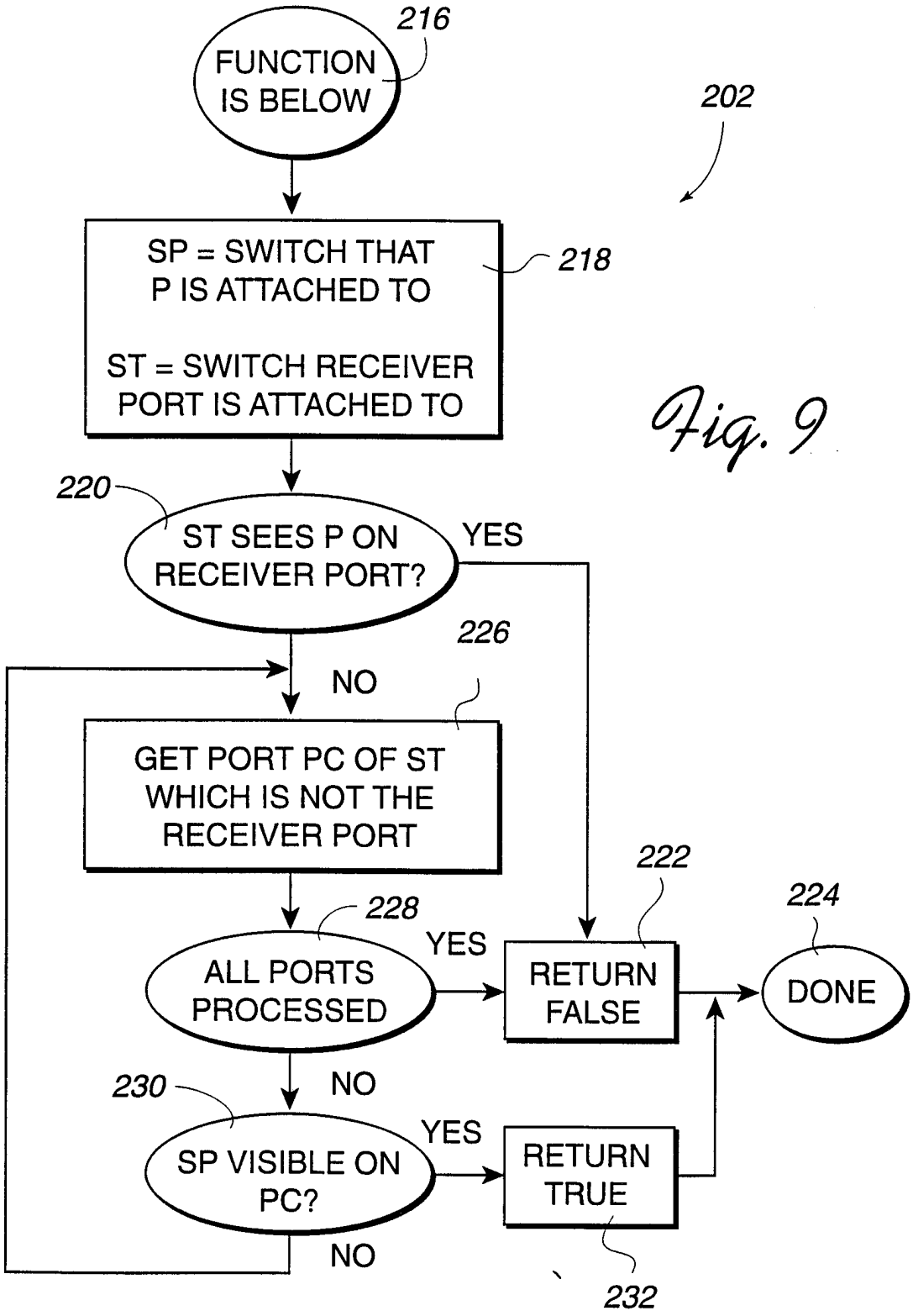
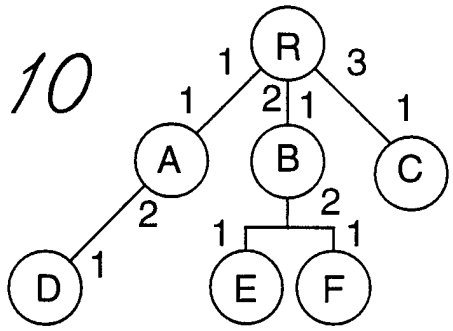

FUNCTION IS BELOW
216
202
SP = SWITCH THAT P IS ATTACHED TO
ST = SWITCH RECEIVER PORT IS ATTACHED TO
218
220
ST SEES P ON RECEIVER PORT?
YES
NO
226
GET PORT PC OF ST WHICH IS NOT THE RECEIVER PORT
228
ALL PORTS PROCESSED
YES
NO
222
RETURN FALSE
224
DONE
230
SP VISIBLE ON PC?
YES
NO
RETURN TRUE
232

Fig. 9

Fig. 10



DEVICE LIST
R
F
C
D
B
E

Fig. 11a

SCAN LIST
R1 = A,D
R2 = B,E,F
R3 = C
A1 = R
A2 = D
B1 = R
B2 = E,F
C1 = R
F1 = R,B,E
D1 = A,R

Fig. 11b

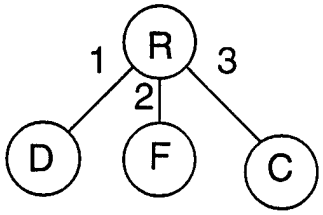


Fig. 12a

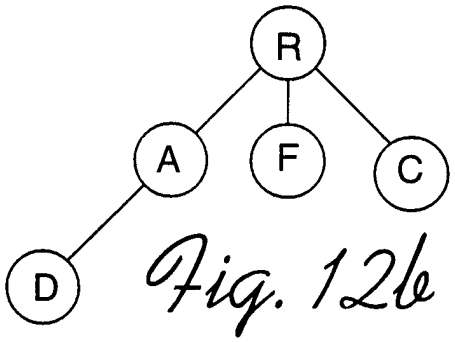


Fig. 12b

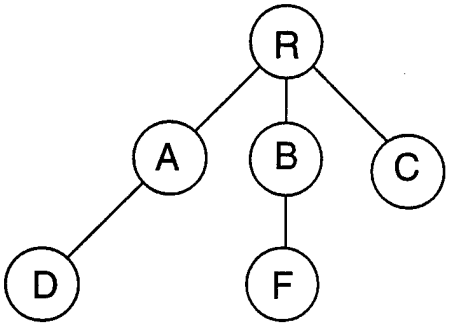


Fig. 12c

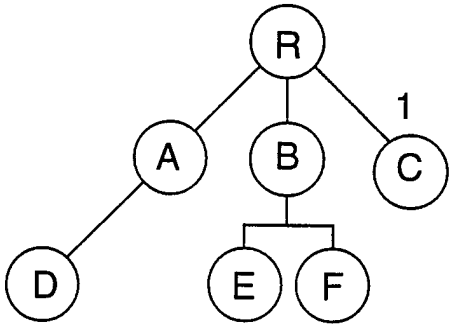


Fig. 12d

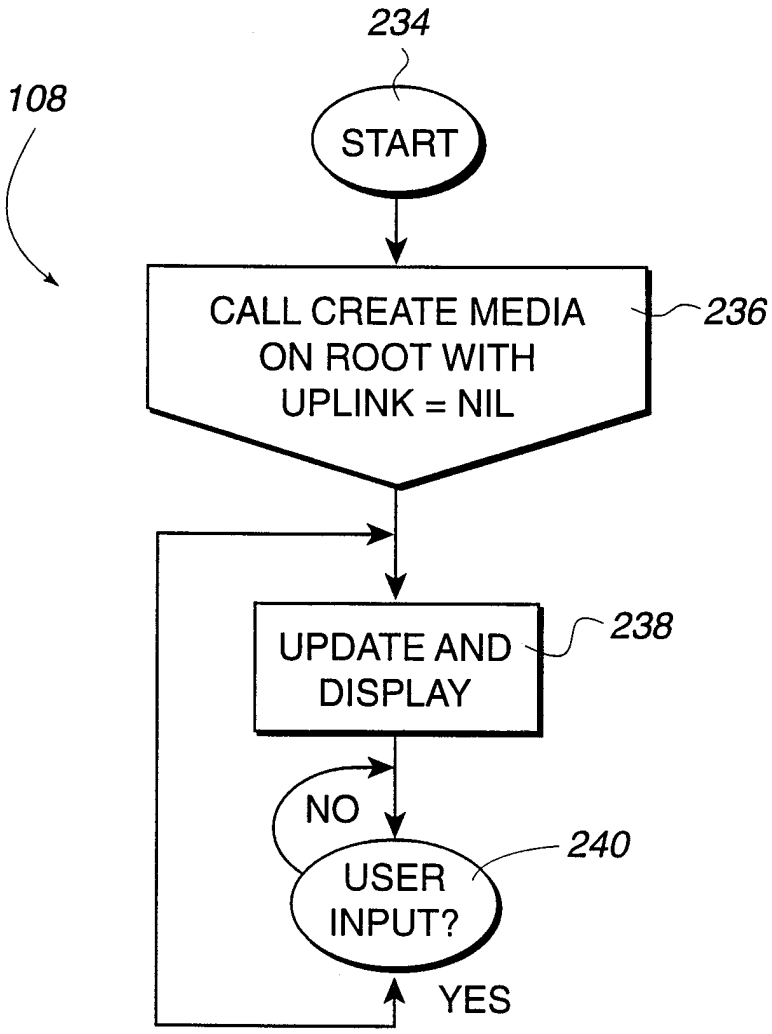

234
108
START
CALL CREATE MEDIA ON ROOT WITH UPLINK = NIL
236
UPDATE AND DISPLAY
238
NO
USER INPUT?
240
YES

Fig. 13

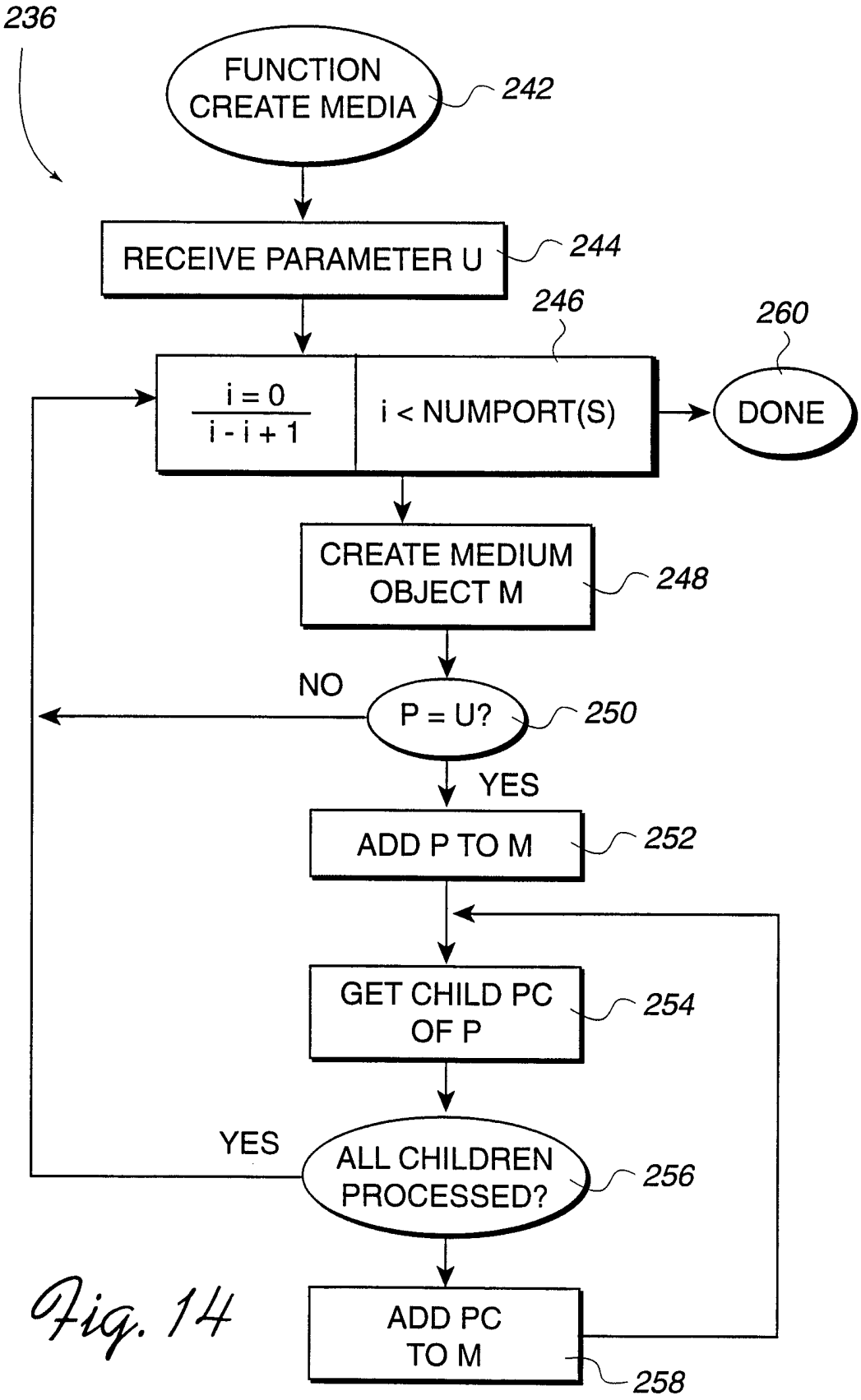

236
FUNCTION CREATE MEDIA
242
RECEIVE PARAMETER U
244
246
260
i = 0
i - i + 1
i < NUMPORT(S)
DONE
CREATE MEDIUM OBJECT M
248
NO
P = U?
250
YES
ADD P TO M
252
GET CHILD PC OF P
254
YES
ALL CHILDREN PROCESSED?
256
ADD PC TO M
258

Fig. 14

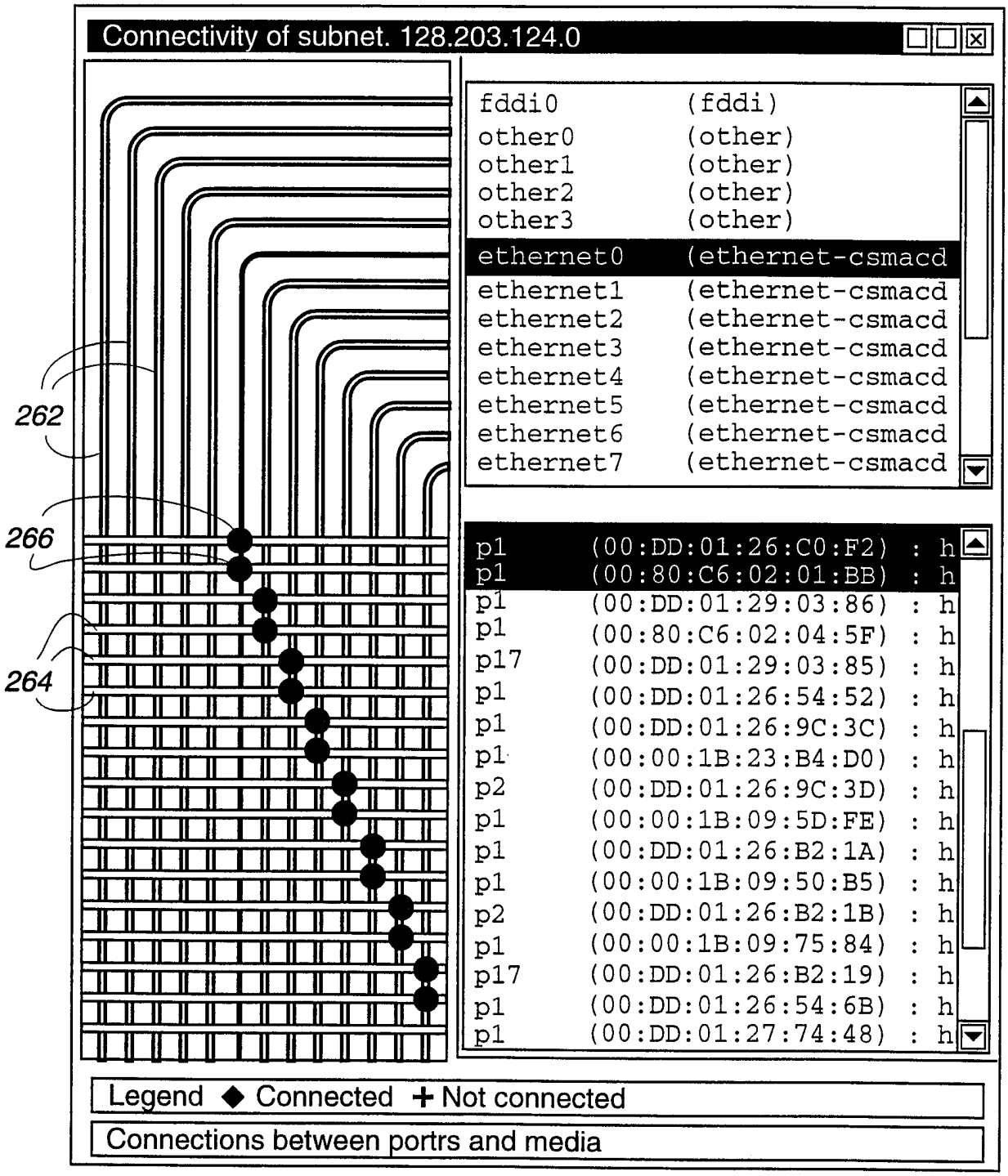

Connectivity of subnet. 128.203.124.0
fddi0 (fddi)
other0 (other)
other1 (other)
other2 (other)
other3 (other)
ethernet0 (ethernet-csmacd
ethernet1 (ethernet-csmacd
ethernet2 (ethernet-csmacd
ethernet3 (ethernet-csmacd
ethernet4 (ethernet-csmacd
ethernet5 (ethernet-csmacd
ethernet6 (ethernet-csmacd
ethernet7 (ethernet-csmacd
262
266
264
p1 (00:DD:01:26:C0:F2) : h
p1 (00:80:C6:02:01:BB) : h
p1 (00:DD:01:29:03:86) : h
p1 (00:80:C6:02:04:5F) : h
p17 (00:DD:01:29:03:85) : h
p1 (00:DD:01:26:54:52) : h
p1 (00:DD:01:26:9C:3C) : h
p1 (00:00:1B:23:B4:D0) : h
p2 (00:DD:01:26:9C:3D) : h
p1 (00:00:1B:09:5D:FE) : h
p1 (00:DD:01:26:B2:1A) : h
p1 (00:00:1B:09:50:B5) : h
p2 (00:DD:01:26:B2:1B) : h
p1 (00:00:1B:09:75:84) : h
p17 (00:DD:01:26:B2:19) : h
p1 (00:DD:01:26:54:6B) : h
p1 (00:DD:01:27:74:48) : h
Legend ◆ Connected + Not connected
Connections between portrs and media

Fig. 15