

A shared network system and a method for operating the shared network system

Background of the Invention

- 5 The present invention relates to a shared network system and a method for operating the shared network system.

Data are normally stored in a computer network comprising computer nodes by storing the data in memory units of the computer nodes or memory units connected to the computer nodes. In recent years, data is made to store in the computer network by looping the data through the computer nodes of the network.

10

Data stored in the computer network by looping therein is known as persistent data. Using the technique of data persistence in the computer network, wide-area storage devices like giant-sized disks or Inter-Processor Communication systems can be implemented effectively.

15

Inter-processor communication in a distributed computer environment using persistent data can be improved as data are now immediately available to any processors that are free to perform computations. An implementation of an inter-processor communication in a distributed computer environment using persistent data can be seen in the Wavelength Disk Drive (WDD) by CANARIE as disclosed in [1].

20

The WDD of CANARIE comprises a physical fiber ring and a plurality of WDD nodes or ring servers along the fiber ring. Application computers or ring clients are connected to the WDD nodes. When a process running in one of the ring clients has been completed, the end-result of the process is injected into the fiber ring via the WDD node and is circulated in the fiber ring. Another process on another ring client that needs the end-result for further processing can then retrieve the end-result from the fiber ring.

25

30

Data circulated or stored in the WDD are associated with the physical address of the next WDD node along the fiber ring in order for the data to be forwarded to the next WDD node. At the next WDD node, the data are re-associated with the physical address of another WDD node along the fiber ring in order for the data to be forwarded to said
5 WDD node. Thus, the circulation of data in the WDD is achieved by forwarding the data to each subsequent WDD node of the fiber ring by associating and re-associating the physical address of each subsequent WDD node to the data. Therefore, in order for data to be circulated or stored in the WDD, the physical address of the next WDD node which the data is to be forwarded to must be known. And in the event when any one of the
10 WDD nodes along the path the data is circulated is disconnected or faulty, the WDD would fail. Furthermore, the WDD has no provisions for the ring clients to control the amount of data circulating in the WDD nor the lifetime of the data in the WDD. There is also no provision in the WDD for the ring clients to find out information about the data available in the WDD without actually retrieving the data.

Summary of the Invention

The invention provides a shared network system and a method for operating the shared network system according to the features of the independent claims. Preferred
20 embodiments of the invention are defined in the dependent claims.

The shared network system according to the invention includes a plurality of computer nodes and connections between the computer nodes. In the shared network system, a plurality of first computer nodes are adapted to define a plurality of looping paths, with
25 each of the looping paths traversing at least two of the first computer nodes. Data can be stored in the shared network system by circulating in one of the looping paths.

Each looping path is associated with a logical address which is unique within the shared network system and is independent from the physical address of any of the computer
30 nodes in the shared network system.

The shared network system further includes at least one second computer node which is able to inject data into one of the looping paths and is also able to retrieve data from one of the looping paths. When data is to be injected into one of the looping paths by the second computer node, the logical address of the looping path the data is to be injected into is determined, and is associated with the data to be injected into the looping path. When data is to be retrieved from one of the looping paths by the second computer node, the logical address of the looping path the data is to be retrieved from is determined.

In the shared network system according to the invention, data can be stored in the network system by circulating the data in one of the looping paths. In other words, data are said to persist in the network system. The second computer node is able store data in the network system, or to retrieve or read data which are stored in the network system.

The second computer node is capable of managing the attributes of the looping paths in the network system, and is also capable of managing the data circulating in the looping paths. According to the invention, the network system may comprise more than one second computer node, which allows a more efficient and effective manner of managing the looping paths in the network system and the data circulating therein.

In addition, the shared network system may also function as a transmission network for transmitting data from one second computer node to another second computer node which may be far apart via the plurality of first computer nodes.

As mentioned above, a unique logical address is associated with each looping path in the shared network system, and accordingly, data can be injected by the second computer node (sending node) into the looping path in the network system by associating the logical address of the looping path with the data to be injected therein. Therefore, logical addressing independent of the physical location or address of the receivers (the same computer node for injecting data into the looping path or other second computer nodes in the shared network system) can be supported according to the invention. In other words,

the data to be injected does not need to have a destination address to a specific physical destination such as a second computer node as a receiving node, (but only the logical address of a looping path is needed), and can still be forwarded to the looping path in the shared network system to be circulated therein.

5

Similarly, a receiving node (second computer node) can retrieve the data circulating in the looping path based on the logical address of the looping path (which is also the destination address associated with the desired data). It should be noted that the receiving node can retrieve the data in the looping path immediately after the sending node has injected the data into the looping path, or at a later time when the sending node is disconnected from the shared network system.

10

The shared network system according to the invention has the following advantages:

15

1. Asynchronous communication between a sending node (the second computer node which injects data into the looping path) and a receiving node (the second computer node which retrieves data from the looping path).

20

Both the sending node and the receiving node can inject and retrieve data into and from the looping path at different times. In other words, the receiving node does not need to be connected to the shared network system at a specific time in order to receive data from the sending node. Instead, the receiving node can connect to the network system any time after the sending node has injected or sent the data into the looping path and still be able to retrieve the data sent by the sending node from the looping path.

25

Such an asynchronous system according to the invention can be used in many different types of applications which involve dynamic sets of clients, producers/consumers, agents or service providers where the senders do not have to be synchronized with the receivers or even have any knowledge about the receivers.

30

2. Connectivity to the shared network system. Both the receiving node and the sending node do not need to be connected to the shared network system all the time. After the sending node has sent the data into the looping path, the sending node may disconnect from the shared network system. The receiving node may, at a later time,
5 connect to the shared network system and retrieve the data which are stored in the looping path by the sending node.

3. Mobility of end-users. End-user (sending or receiving node) can move from one location of the shared network system to another location, and will still be able to access
10 the data in the looping path of the shared network system by connecting to the network system as a second computer node.

4. Simplified group communications. Network complexity and overheads of doing m-to-m communications is reduced to simple groups of end-users on the same looping
15 paths, thereby extending the features of the looping paths to the end-hosts and host-based software object frameworks.

It should be noted that the second computer nodes of the shared network system are not limited to only injecting and retrieving data into and from the looping paths. The second
20 computer nodes may also be capable of other data operations such as the creating or removing looping paths in the shared network system.

The plurality of first computer nodes of the shared network system are preferably computer nodes of a core network, and the at least one second computer node is a
25 computer node of an access network. The computer nodes of the access network are connected to the core network via at least one of the first computer nodes. The said one of the first computer nodes which connects the access network to the core network allows the at least one second computer node of the access network to access at least one of the looping paths in the core network.

As can be seen, since the first computer nodes are in the core network, the looping paths defined by the first computer nodes are also formed within the core network.

The system according to the invention may comprise more than one access network being
5 connected to the core network. This allows second computer nodes from different access networks to be able to store data in a common system, i.e. in the core network, and also to access the commonly stored data through the ~~second~~ at least one of the first computer nodes which interconnects the access networks and the core network. The access network may be, for example, a local area network (LAN). Therefore, the shared
10 network system may correspond to a plurality of LANs which are connected to a common high-speed and high-bandwidth network for storing and transmitting data between the LANs.

According to an embodiment of the invention, the at least one second computer node of
15 the access network is adapted to deliver data to be injected into one of the looping paths in the core network using a multicast protocol mechanism. The multicast protocol mechanism is a simple delivery mechanism that allows the delivery of data from a computer node to a logical address without knowing the physical location/address of any receiving computer nodes. It is also possible that the data is delivered by specifying a
20 specific destination address (which in this case would be the physical address of the at least one of the first computer nodes which connects the access network to the core network). The said one of the first computer nodes would then perform network address translation to deliver the data to the looping path.

25 Internet Protocol (IP) multicast mechanism is preferably used as the multicast protocol mechanism for delivering data to be injected into one of the looping paths to the core network from the second computer node in the access network. The advantage of using IP multicast protocol is that multicast support can be provided efficiently at the network layer since IP networks has been the widely deployed networks in many computer
30 network systems. However, it is also possible to use other multicast protocol mechanisms such as ATM multicast or application-level multicast (e.g P2P multicast).

Similarly, the at least one second computer node of the access network is adapted to deliver data retrieved from one of the looping paths using a multicast protocol mechanism. Preferably, IP multicast mechanism is used as the multicast protocol
5 mechanism for delivering data retrieved from one of the looping paths to the second computer node in the access network. The advantages of using the multicast protocol mechanism and the IP multicast mechanism and their alternatives thereof are already described above.

10 The core network of the shared network system according to the invention may be a wired network such as using Ethernet cables, an optical network or a wireless network such as wireless LAN using radio frequencies.

According to an embodiment of the invention, the computer nodes of the core network
15 support the Generalized Multi-protocol Label Switching (GMPLS). It should however be noted that the core network can be implemented by any computer nodes which support connection oriented services such as Asynchronous Transfer Mode (ATM), Frame-Relay, Multi-protocol Label Switching (MPLS), etc, so that looping paths can be defined by the computer nodes of the core network.

20 According to an embodiment of the invention, the core network is an optical network. An optical network has the advantage of a high bandwidth capacity compared to a wired or wireless network.

25 With Dense Wavelength Division Multiplexing (DWDM) technique in optical technologies [2], network bandwidth of an optical network can be in the range of Terabits per second [3],[4]. This large bandwidth increases the storage capacity of the network and also enables data to be injected into and retrieved from the looping paths of the optical network almost instantaneously.

Another advantage of using an optical network is that data in optical fibers travels at about the speed of light. Therefore, data stored in the looping paths defined in the optical network can be delivered to computer nodes of the access networks requesting them almost instantaneously even when the access networks are thousands of kilometers apart.

5

When the optical network is implemented as the core network in the shared network system according to the invention, the storage (or loop) capacity and data access speed of the core network is high and the circulation delay of the data in the looping paths is low.

10 According to an embodiment of the invention, the access network of the shared network system is an optical network or a wired network.

The advantages of an optical network is already described above. A wired network, such as an Ethernet network, as the access network allows a low-cost implementation of the
15 shared network system according to the invention, and also allows existing access networks to be connected to the core network.

It should be noted that the access network may also be implemented as a wireless
20 network.

20

The computer nodes of the core network are preferably routers or switches. As the basic function of the computer nodes of the core network is to forward data to a next node along the looping path, routers/switches is a low-cost and low-complexity implementation of the computer nodes in the core network.

25

It should however be noted that the computer nodes may also be computer systems, network hubs or servers. It is also possible to have looping paths implemented along a path having computer systems acting as an overlay of computer nodes over the underlying network of routers/switches.

30

The computer nodes of the access network and the computer node which connects the access network to the core network each include an input/output (I/O) module. The I/O modules of the computer nodes of the access network inject data into and retrieve data from the core network by interacting with the I/O module of the computer node which connects the access network to the core network.

According to an embodiment of the invention, the I/O modules of the computer nodes of the access network interact with the I/O module of the computer node which connects the access network to the core network based on an I/O protocol according to a transport layer protocol in the Open Systems Interconnection (OSI) reference model.

This I/O protocol is adapted to allow the I/O modules of the computer nodes of the access network to signal to the I/O module of the computer node which connects the access network to the core network in order to instruct the computer node which connects the access network to core network what operations the computer nodes of the access network want to perform, for example, creating a looping path, injecting or retrieving data into or from a looping path, etc. The interaction between the I/O modules using the I/O protocol can be considered to be the control plane of the I/O modules.

It should be noted that the I/O protocol may also be implemented according to other layer protocols in the OSI model, such as the network layer (Layer 3) protocol, similar to the Internet Control Message Protocol (ICMP).

In a further embodiment of the invention, the input/output protocol of the I/O modules is implemented based on the User Datagram Protocol (UDP). The implementation of the I/O protocol based on the UDP allows the I/O protocol to be implemented as a lightweight protocol running over the unreliable data transport layer in the Open Systems Interconnection (OSI) reference model. This allows existing networks to be connected to the core network as access networks according to the invention, without modifying the network stack structure of the computer nodes of the existing networks. It should be

noted that the I/O protocol of the I/O modules can also be implemented based on the Transmission Control Protocol (TCP) in an alternative embodiment.

When a data operation is requested by a computer node of the access network, the data operation request is issued by the I/O module of the computer node of the access network to the I/O module of the computer node which connects the access network to the core network. According to an embodiment of the invention, the I/O module of the computer node of the access network is adapted to re-issue the data operation request after a predefined time.

As the I/O protocol is adapted to be implemented according to the transport layer protocol which is unreliable, the re-issuing of the data operation request by the I/O module of the computer node of the access network can increase the probability the I/O module of the computer node which connects the access network to the core network receiving the request as the earlier data operation request may be lost.

When data is required to be retrieved from the core network by the computer node of the access network, the I/O module of the computer node which connects the access network to the core network is responsible for filtering and retrieving the desired data from one of the looping paths in the core network. The filtering of the desired data may be performed using a technique similar to IP (Internet Protocol) address lookup as described in [5] on the packet prefix (for example, the content or packet identifier) of the desired data.

According to an embodiment of the invention, the I/O module of the computer node which connects the access network to the core network is adapted to retrieve data from one of the looping paths of the core network by performing a prefix matching of the data based on a Multibit Trie data structure.

The multibit trie data structure may be based on [6] which provides good hardware implementation. The prefix matching of the data for filtering or retrieving desired data from the core network may also be performed using other techniques such as a sequential

search technique. In the sequential search technique, the search data structure is an array having the prefix of desired data. The search process comprises comparing the prefix of the data with the prefix entries in the array to retrieve the desired data.

5 Another technique for retrieving desired data is by using a hashing approach. As an example of a hashing approach, the prefix of the data to be retrieved is hashed to serve as the index to a hash table. By matching the data in the looping path with the entries in the hash table, the desired data can be retrieved.

10 According to an embodiment of the invention, the shared network system further includes a loop data manager. The loop data manager is adapted to remove data from the looping paths of the core network, wherein the data to be removed by the loop data manager either has an expired lifetime or has been selected to be removed from one of the looping paths, for example, by a computer node of the access network. The loop data manager
15 performs data removal from the looping path as a background process, and hence, frees the computer nodes of the access network from unnecessary processing overheads. In addition, the loop data manager also ensures efficient data removal and minimal loop access, since data removal from the looping paths is now performed centrally.

20 It should however be noted that data removal can also be performed by the I/O modules of the computer nodes of the access network, without using the loop data manager.

According to an embodiment of the invention, the plurality of first computer nodes in the shared network system are adapted to define the plurality of looping paths based on an
25 extended Generalized Multiprotocol Label Switching protocol (GMPLS). An example of defining the plurality of looping paths based on an extended GMPLS is described in [8]. The advantages of using GMPLS are:

(1) Since GMPLS has been designed as a single standard control plane supporting
30 multiple types of switching, by leveraging on this generalized framework defined by GMPLS, data persistency can be implemented on many different types of

networks with first computer nodes having various multiplexing and switching capabilities.

(2) Only one first computer node in the core network would suffice to function as a connecting node (with the GMPLS extension capabilities) for connecting the access network(s) to the core network. The other first computer nodes in the core network can participate in the operation of the looping paths with standard GMPLS.

(3) Data forwarding in GMPLS is very efficient since it is done close to the physical layer of the network.

(4) Quality of Service (QoS) mechanisms (e.g bandwidth reservation, traffic prioritization) already available in GMPLS can be leveraged to provide QoS for the looping paths.

It is also possible for the plurality of first computer nodes to define the plurality of looping paths based on the Asynchronous Transfer Mode (ATM), Frame-Relay or the Multi-protocol Label Switching (MPLS).

Although the above embodiments are described referring to the shared network system, these embodiments are also applicable to the method for operating shared network system according to the invention.

The method for operating the shared network system comprises defining a plurality of looping paths by a plurality of first computer nodes, wherein each of the looping paths traverses at least two of the plurality of first computer nodes, wherein data can be circulated in each of the looping paths, associating each of the looping paths with a logical address which is unique within the network system and which is independent from the physical address of any of the computer nodes. In the case of injecting data into one of the looping paths by at least one second computer node which is adapted to inject data into one of the looping paths, determining the logical address of the looping path the data should be injected into, and associating the data to be injected into the looping path with

the logical address. In the case of retrieving data from one of the looping paths by at least one second computer node which is adapted to retrieve data from one of the looping paths, determining the logical address of the looping path the data should be retrieved from.

5

In an embodiment of the method of the invention, loop-information data which comprises information on all the looping paths defined in the network system is circulated in at least one of the looping paths. Such loop-information data may include the set of logical addresses defined for the looping paths, which looping paths are currently in use, and their corresponding loop capacities. The looping path having the loop-information data circulated therein is called a system loop. The use of the system loop provides information which allows an efficient management of the looping paths in the core network.

10

15 In a further embodiment of the invention, content-information which comprises information on the content stored in the looping paths defined in the network system is circulated in at least one of the looping paths.

20

The features and advantages of the present invention will become apparent from the following description and the appended claims, taken in conjunction with the accompanying drawings in which like parts or elements are denoted by like reference numbers.

Brief Description of the Figures

25

Illustrative embodiments of the invention are explained below with reference to the drawings, in which

30

Figure 1 shows an architecture of a shared network system according an embodiment of the invention.

Figure 2 shows a simplified illustration of the shared network system of Figure 1.

Figure 3 shows an interface between a computer node of the access network and a computer node which connects the access network to the core network according to an
5 embodiment of the invention.

Figure 4 shows a generic data packet which is to be circulated in a looping path of the core network according to an embodiment of the invention.

10 Figure 5 shows an example of a Packet Request Tree.

Figure 6(a) shows a node when the *tap_loop* function is called, according to an embodiment of the invention.

15 Figure 6(b) shows a node when the *open_loop* function is called, according to an embodiment of the invention.

Figure 6(c) shows a node when the *close_loop* function is called, according to an embodiment of the invention.

20

Figure 7 shows a flow diagram which illustrates the function *read_loop_map* according to an embodiment of the invention.

Figure 8 shows a flow diagram which illustrates the function *read_loop_info* according to
25 an embodiment of the invention.

Figure 9 shows a flow diagram which illustrates the function *add_loop* according to an embodiment of the invention.

30 Figure 10 shows a flow diagram which illustrates the function *write* according to an embodiment of the invention.

Figure 11 shows a flow diagram which illustrates the function *read_content_map* according to an embodiment of the invention.

5 Figure 12 shows a flow diagram which illustrates the function *read_content_info* according to an embodiment of the invention.

Figure 13 shows a flow diagram which illustrates the function *read* according to an embodiment of the invention.

10

Figure 14 shows a flow diagram which illustrates the function *listen* according to an embodiment of the invention.

Figure 15 shows a flow diagram which illustrates the function *take* according to an embodiment of the invention.

15

Figure 16 shows a flow diagram which illustrates the function *remove* according to an embodiment of the invention.

Figure 17 shows a flow diagram which illustrates the function *remove_loop* according an embodiment of to the invention.

20

Detailed Description of the Embodiments of the Invention

A computer network in very general terms comprises a plurality of computer nodes which are linked together to allow exchange of information.

25

Computer nodes may be personal computers, printers, servers, hubs, routers, switches or any programmable devices and allows sharing of resources by devices connected to the computer network.

30

A Local Area Network (LAN) is a group of computers and associated devices connected together in a localized area, for example within an office building, and typically share the resources of a server. The computers and associated devices of the LAN are the computer nodes. The connections between the computer nodes in the LAN may be wired or wireless links.

Wired LAN comprises computer nodes which are connected to one another using physical cables, such as Ethernet cables. Wireless LAN comprises computer nodes which are connected to one another using wireless means such as radio frequency. An example of a wireless LAN is IEEE 802.11, which is the proposed standard of Institute of Electrical and Electronics Engineers (IEEE).

A computer network may also be implemented in an optical network. In the optical network, computer nodes are interconnected using optical fibers. As optical fibers have high data capacity and is able to transmit data at a high speed, optical networks are normally implemented as a Wide Area Network (WAN). A WAN covers a large geographical area and often uses physical transmission lines such as telephone lines, optical fibers, etc.

Service Providers (SP) may allow one or more LANs to be connected to the WAN, giving rise to an internet allowing transportation of data from one computer in a LAN to another computer in another LAN.

Figure 1 shows an architecture of a shared network system according to an embodiment of the invention.

The shared network system 100 as shown in Fig.1 comprises a core network 101 and two access networks 102. The core network 101 comprises interconnected computer nodes, which are preferably routers (or switches) 103 and Data-In-Network (DIN) nodes 104, with the DIN nodes 104 arranged at the edge of the core network 101. DIN nodes 104 are specialized nodes which are modified or enhanced routers capable of managing the

attributes of looping paths in the core network 101 and the data circulating in the looping paths, in addition to usual routing functions. The DIN nodes 104 can also be described as DIN-capable Network Elements (NE)

- 5 The access networks 102 also comprise computer nodes called hosts 105, which are preferably computer systems.

It should be noted that the core network 101 may further comprise other computer devices such as computer systems or network hubs in addition to the routers 103.

- 10 Similarly, the access network 102 may further comprise other computer devices such as routers/switches, printers or servers in addition to the computer systems.

The access networks 102 are each connected to the core network 101 via a DIN node 104.

- 15 A looping path 106 is shown as a bold line in the core network 101, and traverses some of the DIN nodes 104 and the routers 103. It should also be noted that the looping path 106 also traverses some of the DIN nodes 104 which are connected with the access networks 102.

- 20 Data, in the form of data packets, can be injected by one of the hosts 105 in the access networks 102 into the looping path 106 of the core network 101 to be circulated therein. The injected data is said to be stored in the core network 101. The stored data can be retrieved from the looping path 106 by another (or the same) host 105 in the same or another access network 102. The injecting and retrieval of data into and from the looping
- 25 path 106 in the core network 101 are performed via the DIN nodes 104 which connect the access networks 102 to the core network 101 and also form part of the looping path 106 the data is stored in.

- 30 Although only one looping path 106 is shown in Fig.1, it should be understood that more than one looping paths 106 can be established in the core network 101. Each looping path 106 should traverse at least one DIN node 104.

Similarly, the shared network system 100 may comprise only one access network 102 or more than two access networks 102. Each of the access networks 102 is connected to the core network 101 via a DIN node 104.

5

Figure 2 shows a simplified illustration of the shared network system 100 of Fig.1.

The core network 101 as shown in Fig.2 includes a plurality of looping paths 106. All the looping paths 106 traverse through the DIN nodes 104. Each access network 102 comprising hosts 105 is connected to the core network 101 via one DIN node 104.

10

The shared network system 100 also further includes a loop data manager 107. The functions of the loop data manager 107 will be described later.

15

According to the invention, Internet Protocol (IP) multicast communication is used in the access network 102 where logical addressing independent of the physical location of the receiving hosts is supported.

Specifically, a set of logical or IP multicast address is reserved and allocated for the use of the shared network system 100, with each logical address or IP multicast address being unique. Each looping path 106 defined in the core network 101 is identified by a loop identifier *loopId* with each loop identifier being associated with a logical or IP multicast address in the set. According to the invention, data to be stored in the core network 101 are assigned a IP multicast address as the destination address, and are subsequently forwarded using IP multicast mechanism to the core network 101 to be circulated in the looping path 106 identified by the *loopId* which is associated with the corresponding logical or IP multicast address.

20

25

It can be seen that the destination address of the data need not be addressed to a specific destination or physical address of any computer nodes or hosts. When a host (or hosts)

30

wants to retrieve data from the core network 101, the host may retrieve the data by specifying the IP multicast address associated with the looping path.

5 An example can be used to illustrate the concept of IP multicast (logical) addressing in the DIN system according to the invention. In the example, a sending host has some data to be injected into the core network at time T_1 . The sending host does not or need not specify any specific receiving host(s) to forward the data to, but uses a pre-allocated IP multicast address as the destination address to forward the data to a looping path in the core network, the looping path having a *loopId* which is associated with the pre-allocated
10 IP multicast address.

The data are delivered using standard IP multicast delivery mechanisms from the sending host to the respective DIN node in the access network, and are mapped by the DIN node to the corresponding looping path in the core network to be circulated therein.

15 At time T_2 , which is a time after T_1 and when the sending host may still be connected to the core network or has been disconnected from the core network, a receiving host may retrieve the data circulating in the looping path by specifying the IP multicast address associated with the looping path the data is stored in. The DIN node of the access
20 network of the receiving host retrieves the data from the looping path, and the retrieved data is delivered to the receiving host using standard IP multicast delivery mechanisms.

It can be seen from the above example that forwarding of data from the sending host in the access network to the looping path in the core network, and from the looping path in
25 the core network to the receiving host in the access network are performed entirely in the network layer. Also, since IP multicast is used only in the access networks 102, IP multicast trees set up can be limited in scope to a single or only a few administrative domains.

30 Three types of looping paths can be defined in the shared network system 100 according to an embodiment of the invention: system loop, content loop and control loop.

The system loop is identified by a predefined IP multicast address and the data circulated therein contain information on the set of reserved IP multicast addresses or *loopIds*. For example, the data in the system loop provides information on which looping paths 106 in the core network 101 are currently in use, and the corresponding capacities of the looping paths 106.

The content loop contains data injected by the hosts 105 of the access networks 102. The control loop contains data which provides information about the data in the content loop.

Therefore, the looping paths 106 shown in Fig.2 may comprise the system loop, the content loops and the control loops.

Figure 3 shows an interface between a host 105 of the access network 102 and a DIN node 104.

The host 105 comprises an application module 301 and a host input/output (I/O) module 302. The DIN node 104 comprises a DIN input/output (I/O) module 303, a data forwarding module 304 and a data recovery module 305.

The application module 301 communicates with the host I/O module 302 via a communication link 310. The application module 301 may send a command to the host I/O module 302 via the communication link 310 to request to inject or retrieve data from the core network 101. The host I/O module 302 interacts or communicates the request to inject or retrieve data with the DIN I/O module 303 via a communication link 311. The DIN I/O module 303 in turn communicates with the data forwarding module 304 via a communication link 312, and instructs the data forwarding module 304 to inject or retrieve data into or from a corresponding looping path 106 in the core network 101.

When the host 105 has communicated to the DIN node 104 to inject data into the core network 101, the data to be injected is sent from the host I/O module 302 to the DIN I/O

module 303 via a data link 313. The data is in turn sent to the data forwarding module 304 via a data link 314. The data forwarding module 304 upon receiving the data from the DIN I/O module 303 injects the data into the corresponding looping path 106 in the core network 101 via a data link 315.

5

Similarly, when the host 105 has communicated to the DIN node 104 to retrieve from the core network 101, the data (to be retrieved) is retrieved from the corresponding looping path 106 in the core network 101 via the data link 315 by the data forwarding module 304. The retrieved data is sent to the DIN I/O module 303 via the data link 314. The
10 retrieved data is in turn sent to the host I/O module 302 via the data link 313.

The data forwarding module 304 should preferably be implemented using an extended Generalized Multiprotocol Label Switching (GMPLS), as this protocols supports not only the switching of data packets at each computer node, but also supports lambda switching,
15 fiber switching and time-division multiplexing in a computer network which connects the computer nodes using optical fibers. However, the data forwarding module 304 may also be implemented using any protocols which support connection oriented services such as Asynchronous Transfer Mode (ATM), Frame-Relay, Multi-protocol Label Switching (MPLS), etc.

20

The communication between the application module 301 and the host I/O module 302, and between the DIN I/O module 303 and the data forwarding module 304 will be described later.

25 The host I/O module 302 communicates with the DIN I/O module 303 via the communication link 311 using an I/O protocol. The I/O protocol will also be described in detail later.

The data recovery module 305 may be used to protect the data circulating in the looping
30 paths 106. When the host 105 has communicated to the DIN node 104 to inject data into the core network 101, the data to be injected is sent from the host I/O module 302 to the

DIN I/O module via a data link 313. The data is in turn sent to the data recovery module 305 via a data link 316. The data recovery module 305 encodes the data for protection purposes, for example using redundancy algorithms. The encoded data is then sent from the data recovery module 305 to the data forwarding module 304 via a data link 317 to be
5 injected into the core network 101. The data forwarding module 304 upon receiving the data from the data recovery module 305 injects the data into the corresponding looping path 106 in the core network via a data link 315.

10 In the event that data circulating in the looping path 106 of the core network 101 is lost or is corrupted, the data recovery module 305 may reconstruct the lost data, for example using redundancy algorithms, and send the reconstructed data to the DIN I/O module 303 via the data link 316. The reconstructed data is in turn sent to the host I/O module 302 via the data link 313.

15 It should be noted that the various modules 301, 302, 303, 304, 305 shown in Fig.3 in both the host 105 and the DIN node 104 only serve to illustrate the functions/operations performed by the host 105 and DIN node 104. The exact implementation of the host 105 and DIN node 104 may differ from Fig.3.

20 In addition, the various communication links 310, 311, 312 and the data links 313, 314, 315, 316, 317 only serve to illustrate the communication and data flow between the various modules of the host 105 and the DIN node 104. Specifically, the communication between the various modules via the communication links 310, 311, 312 represents the control plane of the I/O modules 302, 303, and the data forwarding between the various
25 modules via the data links 313, 314, 315, 316, 317 represents the data forwarding plane of the I/O modules. It should be noted that the exact physical implementation of the communication and data links may differ from Fig.3.

30 Figure 4 shows a generic data packet which is to be circulated in the looping path 106 of the core network 101.

The data packet 400 comprises various fields including IP Header field 401, Content Identifier field 402, Packet Identifier field 403, Timestamp field 404, Lifetime field 405 and Payload field 406.

- 5 The Content Identifier field 402 is used to identify the content of the data packet 400. For example, if the data packet 400 is part of the contents of a file, the file ID of that file may be indicated in the Content Identifier field 402.

- 10 The Packet Identifier field 403 is used to specify the data packet sequence number of the data packet 400. When the size of the content of the file is greater than the Maximum Transfer Unit (MTU) of the corresponding network, the content is packetized and the data packet sequence number is contained in this Packet Identifier field 403 of the data packet 400 and also the ensuing data packets.

- 15 The Timestamp field 404 indicates the time when the data packet 400 is created. Data packets 400 having the same content should have the same value in the Timestamp field 404.

- 20 The Lifetime field 405 indicates the length of time that the data packet 400 should circulate in the looping path 106 of the core network 101. Data packets 400 having the same content should have the same value in the Lifetime field 405.

- 25 Certain information about the looping paths setup in the core network 101 and content of data circulating in the looping paths 106 are shared by circulating data containing such information in special data packets in predefined loops in the core network 101.

A summary of the types of loops and data packets used in the core network 101 is shown in Table 1.

Loop Identifier	Content Identifier	Number of Packets	Description
SYSTEM_INFO_LOOP	MAP_SIZE	1	Contains the size of the loop map in terms of number of LOOP_MAP

			packets, also used as a write lock for the loop map.
SYSTEM_INFO_LOOP	LOOP_MAP	Multiple	Contains the set of IP multicast addresses for the use of the shared network system, which address is used or unused and the capacity of the loop allocated to that address if used.
SYSTEM_INFO_LOOP	<loopId>	1	Contains the allocated capacity and description of the loop identified by <loopId>.
<loopId>	<contentId>	1 or Multiple	Contains the content identified by <contentId>.
<loopId>+1	LOOP_CAPACITY	1	Contains information about the loop identified by <loopId> in terms of the allocated capacity and remaining capacity of the loop.
<loopId>+1	MAP_SIZE	1	Contains the size of the content map in terms of the number of CONTENT_MAP packets, also used as a write lock for the content map.
<loopId>+1	CONTENT_MAP	Multiple	Contains information about the content on <loopId> in terms of the content identifier, size and validity of the content.
<loopId>+1	<contentId>	1	Contains the size and description of the content identified by <contentId>.

Table 1

The SYSTEM_INFO_LOOP is a predefined loop identifier which is usually created with system information packets during initialization of the shared network system 100.

5

The MAP_SIZE, LOOP_MAP, LOOP_CAPACITY and CONTENT_MAP are predefined Content Identifiers 402 for special data packets in the shared network system 100.

10 The <loopId>+1 is the control loop for the loop identified by <loopId>.

The following sections highlight the various features of the shared network system 100 according to the invention.

15 **I/O protocol**

The I/O protocol between the host I/O module 302 and the DIN I/O module 303 enables the host 105 to signal to the DIN node 104 the operations to perform. Such operations include, but not limited to, creating a looping path, removing a looping path, injecting data and retrieving data.

The I/O protocol is preferably a lightweight protocol running over unreliable protocols according to the transport layer protocol in the OSI reference model. Examples of such protocols include User Datagram Protocol (UDP) and Transport Control Protocol (TCP).

The I/O protocol maps each operation of the host/DIN node to a request/response pair. Since either the request or the response can be lost over the unreliable protocols in the transport layer, the host 105 initiating the request should preferably re-issue the request after a predefined time.

A summary of the request/response of the I/O protocol which may be used in the shared network system 100 is shown in Table 2.

Request	Response	Description
READ_LOOP_MAP_REQ	READ_LOOP_MAP_ACK	End-host signals to DIN Node that it wants the <i>loop_map</i> . DIN Node acknowledges request. DIN Node filters and sends MAP_SIZE, LOOP_MAP packets from SYSTEM_INFO_LOOP.
STOP_LOOP_MAP_REQ	STOP_LOOP_MAP_ACK	End-host signals to DIN Node that it wants to stop receiving the <i>loop_map</i> . DIN Node acknowledges request. DIN Node stops filtering and sending MAP_SIZE, LOOP_MAP packets from SYSTEM_INFO_LOOP.
READ_LOOP_INFO_REQ	READ_LOOP_INFO_ACK	End-host signals to DIN Node that it wants the <i>loop_info</i> for a particular loop identified by <i><loopId></i> . DIN Node acknowledges request. DIN Node filters and sends <i><loopId></i> packet from SYSTEM_INFO_LOOP.
STOP_LOOP_INFO_REQ	STOP_LOOP_INFO_ACK	End-host signals to DIN Node that it wants to stop receiving the <i>loop_info</i> . DIN Node acknowledges request. DIN Node stops filtering and sending

		<loopId> packet from SYSTEM_INFO_LOOP.
ADD_LOOP_REQ	ADD_LOOP_SUCCESS or ADD_LOOP_FAIL	End-host signals to DIN Node that it wants to add a DIN loop. DIN Node returns success or failure depending on whether the loop was successfully added to the system.
WRITE_REQ	WRITE_SUCCESS or WRITE_FAIL	End-host signals to DIN Node that it wants to write to a DIN loop. DIN Node returns success or failure depending on whether data packets from the end-host were successfully written to the loop.
READ_CONTENT_MAP_REQ	READ_CONTENT_MAP_ACK	End-host signals to DIN Node that it wants the <i>content_map</i> for a particular loop identified by <loopId>. DIN Node acknowledges request. DIN Node filters and sends MAP_SIZE, CONTENT_MAP packets from <loopId>+1.
STOP_CONTENT_MAP_REQ	STOP_CONTENT_MAP_ACK	End-host signals to DIN Node that it wants to stop receiving the <i>content_map</i> . DIN Node acknowledges request. DIN Node stops filtering and sending MAP_SIZE, CONTENT_MAP packets from <loopId>+1.
READ_CONTENT_INFO_REQ	READ_CONTENT_INFO_ACK	End-host signals to DIN Node that it wants the <i>content_info</i> for a particular content identified by <contentId>. DIN Node acknowledges request. DIN Node filters and sends <contentId> packet from <loopId>+1.
STOP_CONTENT_INFO_REQ	STOP_CONTENT_INFO_ACK	End-host signals to DIN Node that it wants to stop receiving the <i>content_info</i> . DIN Node acknowledges request. DIN Node stops filtering and sending <contentId> packet from <loopId>+1.
READ_REQ	READ_ACK	End-host signals to DIN Node that it wants to read content corresponding to <contentId_filter>. DIN Node acknowledges request. DIN Node filters and sends packets corresponding to <contentId_filter> from <loopId>.
STOP_READ_REQ	STOP_READ_ACK	End-host signals to DIN Node that it wants to stop receiving content corresponding to <contentId_filter>. DIN Node acknowledges request. DIN Node stops filtering and sending packets corresponding to <contentId_filter> from <loopId>.
LISTEN_REQ	LISTEN_ACK	End-host signals to DIN Node that it wants to listen for the arrival of new content on <loopId>. DIN Node

		acknowledges request. DIN Node filters and sends <contentId> packet from <loopId>+1 when new content corresponding to <contentId> is written to <loopId>.
STOP_LISTEN_REQ	STOP_LISTEN_ACK	End-host signals to DIN Node that it wants to stop listening for the arrival of new content on <loopId>. DIN Node acknowledges request. DIN Node stops filtering and sending <contentId> packet from <loopId>+1 when new content corresponding to <contentId> is written to <loopId>.
TAKE_REQ	TAKE_ACK	End-host signals to DIN Node that it wants to take content corresponding to <contentId_filter>. DIN Node acknowledges request. DIN Node marks the content as invalid in the <i>content_map</i> and removes the <contentId> packet(s) corresponding to <contentId_filter> from <loopId>+1. DIN Node filters and sends packets corresponding to <contentId_filter> from <loopId>.
STOP_TAKE_REQ	STOP_TAKE_ACK	End-host signals to DIN Node that it wants to stop receiving content corresponding to <contentId_filter>. DIN Node acknowledges request. DIN Node stops filtering and sending packets corresponding to <contentId_filter> from <loopId>.
REMOVE_REQ	REMOVE_ACK	End-host signals to DIN Node that it wants to remove content corresponding to <contentId_filter>. DIN Node acknowledges request. DIN Node marks the content as invalid in the <i>content_map</i> and removes the <contentId> packet(s) corresponding to <contentId_filter> from <loopId>+1.
REMOVE_LOOP_REQ	REMOVE_LOOP_ACK	End-host signals to DIN Node that it wants to remove a DIN loop. DIN Node acknowledges request.

Table 2**Packet Request Tree**

- 5 The DIN I/O module 303 accesses data in the looping paths 106 through the data forwarding module 304. When a request is made by the host 105 to retrieve data from a particular looping path 106, the data forwarding module 304 retrieves the data packets circulating in that looping path 106 to the DIN I/O module 303. The DIN I/O module

303 is responsible for filtering the requested data packets from the data packets retrieved by the data forwarding module 304.

5 The filtering of the data packets may be done on part of the Content Identifier 402 and the Packet Identifier 403 of the data packets. Accordingly, prefix matching algorithms similar to those used for IP address lookup [5] may be used here. One such algorithm that provides fast lookup and is suitable for hardware implementation [6] is based on a tree structure, known as a multibit trie structure.

10 The Content Identifier 402 and the Packet Identifier 403 of the data packets may be represented as nodes of a multibit trie, thus forming a Packet Request Tree (PRT). The PRT is made specific to each looping path, and has nodes representing the Content Identifiers 402 and/or the Packet Identifiers 403 of the data packets requested by the host 105 to be retrieved.

15 In an example, the Content Identifier field 402 and the Packet Identifier field 403 are each 16 bits long. Assuming there are three requests for data packets in a looping path 106 as follows:

20 Request 1: Content Identifier = xFF*
 Request 2: Content Identifier = xF011
 Request 3: Content Identifier = x1000,
 Packet Identifier = x8A26

25 Figure 5 shows the PRT 500 representing the above three requests.

The PRT 500 of Fig.5 comprises a plurality of nodes 501, including three end nodes 502, 503, 504. The first end node 502 represents the Request 1, the second end node 503 represents the Request 2, and the third end node 504 represents the Request 3.

30

To filter the requested data packets from the data packets retrieved by the data forwarding module 304, each of the data packets retrieved by the data forwarding module 304 is made to traverse the PRT 500. Data packets having their corresponding Content Identifiers 402 and/or Packet Identifiers 403 match the end nodes of the PRT 500 are
 5 subsequently retrieved by the DIN I/O module 303 as the requested data packets.

Reliable Transportation of Data Packets

When a request for reading data packets is issued by the host 105, the read request is
 10 added to the PRT 500 as an end node. The request for reading data packets will persist until a request to stop reading the data packets is issued by the host 105. In other words, the end node in the PRT 500 corresponding to the read request will not be removed until a request to stop reading the data packets is issued.

15 This means that the requested data packets are continually sent to the host 105 requesting the data packets, as the requested data packets continue to circulate in the corresponding looping path and is retrieved by the corresponding DIN node.

Such duplicate or multiple requested data packets decrease the probability that the
 20 requesting host 105 is not able to receive the requested data packets (e.g. to re-create the content) due to packet loss since packet loss is highly probable in the unreliable multicast protocol used for data delivery.

In the control loop *loopId+1*, data packets containing information about the size of the
 25 content in the content loop *loopId* can be retrieved by the host 105. From the *loopId* content loop, the host 105 then retrieves the content in the form of data packet(s) having the Content Identifier corresponding to the requested content. With a knowledge of the size of the content and from sequencing the Packet Identifier of the received data packets, the entire content from the content loop can be re-created by the host 105. Once the
 30 entire content has been created, the host 105 issues a request to stop reading the data packets from the looping path (content loop).

Alternatively, the host 105 may issue the request to stop reading the data packets before the entire content is created. In this case, the host 105 issues a new read request for the missing data packets only. This is similar to the negative acknowledgement (NAK) schemes used in some reliable multicast protocols such as PGM [7]. In these schemes, the DIN node do not acknowledge (ACK) request messages, but only acknowledge (NAK) the missing messages. This improves the scalability of such reliable multicast protocols by preventing message implosion at the host 105.

Request Suppression

Request suppression relates to reducing the number of requests sent by the host 105 to the DIN node 104 for the same content. Request suppression can prevent request implosion at the DIN node 104 when a very popular content is being distributed over the core network 101.

Before sending a request by the host 105, the host 105 “senses” for the content in the core network 101 by listening on the IP multicast address corresponding to the looping path 106 containing the content desired to be requested. If the desired content is already being sent from the core network 101, the host can then receive the content directly from the core network 101 without sending a request for the content to the DIN node 104.

If the desired content is not available to be retrieved from the core network 101, the host 105 waits for a time duration, which can be randomly chosen from a predefined time ~~duration~~ interval, before determining if the desired content is available, and subsequently sends a request for the desired content when it is not available. If during the time duration of waiting, another host sends a request for the same desired content, the host 105 then “suppresses” its own request for the desired content.

Hosts 105 determine whether other hosts 105 are sending request for the desired content in the content loop *loopId* by requiring the other hosts to broadcast their requests in the

control loop *loopId*+1 (which is the control IP multicast address for the content loop *loopId*).

- 5 Such request suppression mechanism described above is similar to the NAK suppression used in the PGM type of reliable multicast protocols [7].

Loop Data Manager

- 10 The loop data manager 107 as shown in Fig.2 is responsible for physically removing data packets from the looping paths 106 in the core network 101. Such data packets are removed by the loop data manager 107 because they either have expired lifetime or has been marked as invalid in their corresponding content map because they have been explicitly taken or removed by hosts 105.

- 15 The loop data manager 107 determines the data packets to be removed by reading the content map of each content loop and creates a PRT for data packets which have been marked as invalid in the content map for the loop. For each content loop, the loop data manager 107 removes the data packets if they corresponds to the end nodes of the PRT.

- 20 The loop data manager 107 also checks the Lifetime field 404 and the Timestamp field 405 (refer to Fig.4) of the data packets. If the difference between the current time and the value of the Timestamp field 405 is greater than the value of the Lifetime field 404, the data packets are removed.

- 25 When all the content loops have been processed by the loop data manager 107, the loop data manager 107 then updates the loop capacity and the content map of the content loops.

Listening

30

A host 105 can “listen” on a looping path 106 and be informed when new data packets are injected into the looping path 106.

Whenever a new content is placed in the loop *loopId*, a control data packet containing
 5 information about this new content will be created in the control loop for *loopId*, that is in *loopId+1*. If there is a host listening on this control loop *loopId+1*, this control data packet will be sent onto the access network 102 by the DIN node 104 to notify the listening host.

10 **Information Sharing**

Information services, also known as directory services, are an essential part of many systems. These services are typically provided using information or directory servers. For scalability across a wide-area, multiple servers with replicated information are often
 15 used. This gives rise to the problem of maintaining consistency with replicated information.

Information services can be provided in the shared network system 100 according to the invention by using the looping paths 106 in the core network 101 to share information
 20 among the hosts 105 distributed over the access networks 102.

Since only one copy of the information circulates in the looping path 106, consistency control of the information is not necessary. However, when more than one hosts 105 want to update the information circulating in the looping path 106, concurrency control
 25 can be implemented using write-lock packets. Accordingly, only host 105 which managed to retrieve the specific write-lock packet is able to update the information.

The write-lock packets may be special data packets injected in the looping paths 106. Alternatively, data packets already circulating in the looping paths 106 may be used as
 30 the write-lock packets. Examples of such data packets include the data packets in the

system loop or control loops having MAP_SIZE as the value of the Content Identifier field.

Interface between the application module and the host I/O module

5

The application module 301 can call the following functions onto the host I/O module 302 to perform specialized operations.

The operation of the functions from the application module 301 is summarized below.

10

read_loop_map(loop_map) retrieves information about the set of loops that can be created in the DIN network into *loop_map*. The *loop_map* contains a set of *loopIds* and their allocation status. If the *loopId* has been allocated, that is, a DIN loop with *loopId* has been added to the core network, the *loop_map* will also indicate the storage capacity of the loop.

15

read_loop_info(loopId, loop_info) retrieves information specific to the loop with *loopId*. This information pertains to the storage capacity and a description (if available) of the loop.

20

add_loop(loopId, loop_size, route, description) creates a DIN loop identified by *loopId* and with storage capacity of *loop_size*. The route parameter, if it is not NULL, specifies the list of DIN Nodes/routers that the DIN loop should pass through. The *description* parameter, if it is not NULL, is added to the information about the loop which can be retrieved with *read_loop_info*. Since the capacity of the DIN network is finite, it may be not be possible to create the loop in the DIN network. In this case, *add_loop* should return with a fail status.

25

write(loopId, buffer, buffer_size, contentId, lifetime, description) injects data in *buffer* of size *buffer_size* into loop with *loopId*. If *buffer_size* is greater than the MTU of the network, the data will be packetized. *contentId* specifies the value of the *Content*

30

Identifier field to be inserted in the packet(s) corresponding to the data. *lifetime* specifies the value of the *Lifetime* field to be inserted in the packet(s). See Section 3, Packet Format. The *description* parameter, if it is not NULL, is added to the information about the content which can be retrieved with *read_content_info*. Since the capacity of the loop
 5 is finite, it may not be possible to inject the required data into the loop. In this case, *write* should return with a fail status.

read_content_map(loopId, content_map) retrieves information about all the content in the loop with *loopId* into *content_map*. The *content_map* contains a set of *contentIds* and the
 10 sizes of their corresponding content.

read_content_info(loopId, contentId, content_info) retrieves information specific to the content with *contentId* found on loop identified by *loopId*. The information pertains to the size and a description (if available) of the content.
 15

listen(loopId, contentId, content_info) waits for the arrival of new content on the loop with *loopId*. When new content arrives on the loop, *listen* returns with information about the content in *contentId* and *content_info*.

read(loopId, buffer, buffer_size, contentId_filter) retrieves data from loop with *loopId* into *buffer* of size *buffer_size*. If *contentId_filter* is NULL, then all the packets on the loop will be retrieved. Otherwise, only the packets with content identifier field matching the *contentId_filter* will be retrieved.
 20

take(loopId, buffer, buffer_size, contentId_filter) is the same as *read* except that the retrieved data is also removed from the loop.
 25

remove(loopId, contentId_filter) is the same as *take* except that data is not returned to the end-host.
 30

remove_loop(loopId) removes the loop with *loopId* from the DIN network. All data in the loop will be removed and will no longer be accessible by the end-hosts.

5 The functions described above are only some of the possible functions the application module 301 may call onto the host I/O module. A detailed explanation of each of the above functions will be given later using flow diagrams.

10 It should be noted that the above described functions are not exhaustive, and it is possible to define further functions for the host I/O module 302 to perform operations not described above.

Interface between the DIN I/O module and the data forwarding module

15 Similarly, the DIN I/O module 303 can call the following functions onto the data forwarding module 304 to perform specialized operations.

The operation of the functions from the DIN I/O module 304 is summarized below.

20 *add_loop(loopId, loop_size, route)* creates a DIN loop identified by *loopId* and with storage capacity of *loop_size*. The *route* parameter, if it is not NULL, specifies the list of DIN Nodes/routers that the DIN loop should pass through. Since the capacity of the DIN network is finite, it may not be possible to create the loop in the DIN network. In this case, *add_loop* should return with a fail status.

25 *tap_loop(loopId)* taps the DIN loop identified by *loopId*. When a loop with *loopId* is tapped, a copy of the packets with *loopId* is made and passed to the calling function. Figure 6(a) illustrates a *tap_loop*.

30 *open_loop(loopId)* opens the DIN loop identified by *loopId*. When a loop with *loopId* is opened, packets with *loopId* are passed to the calling function and packets from the calling function are forwarded onto the loop. Figure 6(b) illustrates an *open_loop*.

close_loop(loopId) closes the DIN loop identified by *loopId*. When a loop with *loopId* is closed, packets with *loopId* are no longer copied nor passed to the calling function. Figure 6(c) illustrates a *close_loop*.

5

remove_loop(loopId) removes the DIN loop identified by *loopId*. When a loop with *loopId* is removed, the path, for example Label Switched Path (LSP), corresponding to the loop will be torn down and all packets with *loopId* will be discarded.

10 **Illustration of the functions called by the application module 301.**

- *read_loop_map*

Figure 7 shows a flow diagram between the host I/O module 302, the DIN I/O module 15 303 and the data forwarding module 304 when the function *read_loop_map* is called by the application module 301.

In block 700, the application module 301 calls *read_loop_map* to the host I/O module 302 via communication link 310. The host I/O module 302 sends 20 READ_LOOP_MAP_REQ (block 701) to DIN I/O module 303 via communication link 311. The DIN I/O module 303 sends READ_LOOP_MAP_ACK to the host I/O module 302 via communication link 311.

The DIN I/O module 303 determines if SYSTEM_INFO_LOOP PRT exists (block 702). 25 If SYSTEM_INFO_LOOP PRT exists, the DIN I/O module 303 adds MAP_SIZE and LOOP_MAP nodes to the PRT (block 703). Else, the DIN I/O module 303 creates the SYSTEM_INFO_LOOP PRT (block 704) with MAP_SIZE and LOOP_MAP nodes. The DIN I/O module 303 calls *tap_loop*(SYSTEM_INFO_LOOP) to the data forwarding module 304 via communication link 312.

30

The data forwarding module 304 copies and sends packets (block 705) on SYSTEM_INFO_LOOP to the DIN I/O module 303 via data link 314. The DIN I/O

module 303 filters the packets by traversing SYSTEM_INFO_LOOP PRT, and sends MAP_SIZE and LOOP_MAP packets (block 706) to the host I/O module 302 via data link 313.

5 The host I/O module 302 creates *loop_map* from the packets received from data link 313 (block 707), and sends *loop_map* to the application module 301 via communication link 310. The host I/O module 302 also sends STOP_LOOP_MAP_REQ to the DIN I/O module 303 via communication link 311.

10 The DIN I/O module 303 send STOP_LOOP_MAP_ACK to the host I/O module 302 via communication link 311, and also removes MAP_SIZE and LOOP_MAP nodes from the SYSTEM_INFO_LOOP PRT (block 708). It then calls *close_loop*(SYSTEM_INFO_LOOP) onto the data forwarding module via communication link 312.

15 The data forwarding module stops copying and sending packets on SYSTEM_INFO_LOOP to the data link 314 (block 709).

- *read_loop_info*

20 Figure 8 shows a flow diagram between the host I/O module 302, the DIN I/O module 303 and the data forwarding module 304 when the function *read_loop_info* is called by the application module 301.

25 In block 710, the application module 301 calls *read_loop_info* to the host I/O module 302 via communication link 310. The host I/O module 302 sends READ_LOOP_INFO_REQ (block 711) to DIN I/O module 303 via communication link 311. The DIN I/O module 303 sends READ_LOOP_INFO_ACK to the host I/O module 302 via communication link 311.

30 The DIN I/O module 303 determines if SYSTEM_INFO_LOOP PRT exists (block 712). If SYSTEM_INFO_LOOP PRT exists, the DIN I/O module 303 adds *<loopId>* node to

the PRT (block 713). Else, the DIN I/O module 303 creates the SYSTEM_INFO_LOOP PRT (block 714) with *<loopId>* node. The DIN I/O module 303 calls *tap_loop*(SYSTEM_INFO_LOOP) to the data forwarding module 304 via communication link 312.

5

The data forwarding module 304 copies and sends packets (block 715) on SYSTEM_INFO_LOOP to the DIN I/O module 303 via data link 314. The DIN I/O module 303 filters the packets by traversing SYSTEM_INFO_LOOP PRT, and sends *<loopId>* packet (block 716) to the host I/O module 302 via data link 313.

10

The host I/O module 302 creates *loop_info* from the packets received from data link 313 (block 717), and sends *loop_info* to the application module 301 via communication link 310. The host I/O module 302 also sends STOP_LOOP_INFO_REQ(*<loopId>*) to the DIN I/O module 303 via communication link 311.

15

The DIN I/O module 303 send STOP_LOOP_INFO_ACK to the host I/O module 302 via communication link 311, and also removes *<loopId>* node from the SYSTEM_INFO_LOOP PRT (block 718). It then calls *close_loop*(SYSTEM_INFO_LOOP) onto the data forwarding module via communication link 312.

20

The data forwarding module stops copying and sending packets on SYSTEM_INFO_LOOP to the data link 314 (block 719).

25 - *add_loop*

Figure 9 shows a flow diagram between the host I/O module 302, the DIN I/O module 303 and the data forwarding module 304 when the function *add_loop* is called by the application module 301.

30

In block 720, the application module 301 calls *add_loop* to the host I/O module 302 via communication link 310. The host I/O module 302 sends *ADD_LOOP_REQ(<loopId>, <loop_size>,...)* (block 721) to DIN I/O module 303 via communication link 311.

- 5 The DIN I/O module 303 determines if *SYSTEM_INFO_LOOP* PRT exists (block 722). If *SYSTEM_INFO_LOOP* PRT exists, the DIN I/O module 303 adds *MAP_SIZE* node to the PRT (block 723). Else, the DIN I/O module 303 creates the *SYSTEM_INFO_LOOP* PRT (block 724) with *MAP_SIZE* node. The DIN I/O module 303 calls *open_loop(SYSTEM_INFO_LOOP)* to the data forwarding module 304 via
10 communication link 312.

The data forwarding module 304 sends packets on *SYSTEM_INFO_LOOP* to the DIN I/O module 303 via data link 314, and/or sends packets received from the DIN I/O module 303 via data link 314 into *SYSTEM_INFO_LOOP* (block 725). The DIN I/O
15 module 303 filters the packets received from the data forwarding module 304 by traversing *SYSTEM_INFO_LOOP* PRT (block 726), and keeps the *MAP_SIZE* packet while sending all the other packets unchanged to the data forwarding module 304 via data link 314. The DIN I/O module 303 calls *add_loop(<loopId>, <loop_size>,...)* onto the data forwarding module 304 via communication link 312.

20 The data forwarding module 304 creates a (DIN) loop in the core network with Forward Equivalence Class (FEC) equivalent to *<loopId>* and storage capacity of *<loop_size>* (block 727). The data forwarding module 304 returns a success or failure status on the loop creation to the DIN I/O module 303 via communication link 312.

25 The DIN I/O module 303 tests the success/failure status (block 728). If the DIN I/O module 303 receives a failure status via the communication link 312, it sends *ADD_LOOP_FAIL* to the host I/O module (not shown) via communication link 311 and sends *MAP_SIZE* packet to the data forwarding module 304 via data link 314. It then
30 calls *close_loop(SYSTEM_INFO_LOOP)* to the data forwarding module 304 via communication link 312. If a success status is received via the communication link 312,

the DIN I/O module 303 calls *add_loop*(<loopId>+1, LOOP_CTRL_SIZE,...) to the data forwarding module 304 via communication link 312.

5 Upon receiving *close_loop*(SYSTEM_INFO_LOOP) from the DIN I/O module 303 via communication link 312, the data forwarding module 304 stops sending and receiving packets (block 729) from the data link 314 onto SYSTEM_INFO_LOOP. Similarly, upon receiving *add_loop*(<loopId>+1, LOOP_CTRL_SIZE,...) from the DIN I/O module 303 via the communication link 312, the data forwarding module 304 creates a loop in the core network (block 730) with FEC equivalent to (<loopId>+1) and storage
10 capacity of LOOP_CTRL_SIZE. The data forwarding module 304 then returns a success or failure status to the DIN I/O module 303 via communication link 312.

The DIN I/O module 303 again tests the success/failure status of the loop creation (block 731). If a failure status is received via the communication link 312, the DIN I/O module
15 303 sends ADD_LOOP_FAIL to the host I/O module 302 via communication link 311. It also calls *remove_loop*(<loopId>) via communication link 312, sends MAP_SIZE packet via data link 314 and calls *close_loop*(SYSTEM_LOOP_INFO) via communication link 312 to the data forwarding module 304.

20 If a success status is received via the communication link 312, the DIN I/O module 303 performs the following operations:

- adds LOOP_MAP node to SYSTEM_INFO_LOOP PRT,
- filters packets by traversing SYSTEM_INFO_LOOP PRT,
- 25 - marks LOOP_MAP packet corresponding to <loopId> as used and sends LOOP_MAP packet to data forwarding module 304 via data link 314,
- sends all other packets unchanged via data link 314,
- creates <loopId> info packet and sends it to data forwarding module 304 via data link 314,
- 30 - sends MAP_SIZE packet via data link 314,

- calls *close_loop*(SYSTEM_LOOP_INFO) onto data forwarding module via communication link 312,
- calls *open_loop*(<loopId>+1) on P9,
- creates MAP_SIZE packet(size=0) and LOOP_CAPACITY
5 packet(loop_capacity=<loop_size>, remaining_loop_capacity=<loop_size>) and sends it to data forwarding module 304 via data link 314,
- calls *close_loop*(<loopId>+1) via communication link 312, and
- sends ADD_LOOP_SUCCESS to host I/O module 302 via communication link 311.

10

The host I/O module 302 returns a success status to the application module 301 via communication link 310 if ADD_LOOP_SUCCESS is received. When ADD_LOOP_FAIL is received, the host I/O module 302 returns a failure status to the application module 301 via communication link 310 (block 732).

15

If the data forwarding module 304 receives a *remove_loop*<loopId> from the DIN I/O module 303 via communication link 312, the data forwarding module 304 tears down the path of <loopId>. If the data forwarding module 304 receives *close_loop*(SYSTEM_LOOP_INFO), it stops sending and receiving packets from data
20 link 314 into SYSTEM_INFO_LOOP. If the data forwarding module 304 receives *open_loop*(<loopId>+1), it sends packets received from data link 314 into (<loopId>+1). If the data forwarding module 304 receives *close_loop*(<loopId>+1), it stops sending and receiving packets from data link 314 into (<loopId>+1) (block 733).

25

- *write*

Figure 10 shows a flow diagram between the host I/O module 302, the DIN I/O module 303 and the data forwarding module 304 when the function *write* is called by the application module 301.

30

In block 740, the application module 301 calls *write* to the host I/O module 302 via communication link 310. The host I/O module 302 sends *WRITE_REQ(<loopId>, <buffer_size>, <contentId>, <description>)* (block 741) to DIN I/O module 303 via communication link 311.

5

The DIN I/O module 303 determines if (*<loopId>+1*) PRT exists (block 742). If (*<loopId>+1*) PRT exists, the DIN I/O module 303 adds *LOOP_CAPACITY* node to the PRT (block 743). Else, the DIN I/O module 303 creates the (*<loopId>+1*) PRT (block 744) with *LOOP_CAPACITY* node. The DIN I/O module 303 calls

10

open_loop(<loopId>+1) to the data forwarding module 304 via communication link 312.

The data forwarding module 304 sends packets on (*<loopId>+1*) to the DIN I/O module 303 via data link 314, and/or sends packets received from the DIN I/O module 303 via data link 314 into (*<loopId>+1*) (block 745). The DIN I/O module 303 filters the

15

packets received from the data forwarding module 304 by traversing (*<loopId>+1*) PRT (block 746), and keeps the *LOOP_CAPACITY* packet while sending all the other packets unchanged to the data forwarding module 304 via data link 314.

If

20

<loop_remaining_capacity> >= <buffer_size>,

the DIN I/O module 303 updates *LOOP_CAPACITY* packet as

25

<loop_remaining_capacity> = <loop_remaining_capacity>-<buffer_size>.

The DIN I/O module 303 then sends the updated *LOOP_CAPACITY* packet to the data forwarding module 304 via data link 314, and sends *WRITE_SUCCESS* to host I/O module 302 via communication link 311. Else, the DIN I/O module 303 sends

30

WRITE_FAIL to the host I/O module 302 via communication link 311 and sends *LOOP_CAPACITY* packet to the data forwarding module 304 via data link 314. The

DIN I/O module 303 then calls *close_loop(<loopId>+1)* to the data forwarding module 304 via communication link 312.

5 The data forwarding module 304 upon receiving the *close_loop(<loopId>+1)* function stops sending and receiving packets from the data link 314 into (*<loopId>+1*) (block 747).

10 The host I/O module 302 determines whether WRITE_SUCCESS is received (block 748). When the host I/O module 302 receives a WRITE_SUCCESS from the DIN I/O module 304, it packetizes data if necessary in *<buffer>* with *<loopId>* in destination address field, *<contentId>* in CONTENT_IDENTIFIER field, *<lifetime>* in LIFETIME field, current timestamp in TIMESTAMP field and packet sequence number in PACKET_IDENTIFIER field of packet, and sends the data packets to the DIN I/O module 303 via data link 313. When the host I/O module 302 receives WRITE_FAIL
15 form the DIN I/O module 304, it returns a failure status to the application module 301 via communication link 310 (block 749).

20 The DIN I/O module 303 receives the data packets from the host I/O module 302 via data link 313 and forwards them to the data recovery module 305 via data link 316 where they are encoded, and subsequently sent to the data forwarding module 304 via data link 317 (block 750). The data forwarding module 304 receives the data packets via data link 317 and forward them into the core network via data link 315 (block 751).

25 The DIN I/O module 303 checks if all the packets in *<buffer>* are received (block 752). If not all the packets in *<buffer>* are received, it sends a WRITE_FAIL(*lost_packets*) to the host I/O module 302 via communication link 311. The host I/O module 302 upon receiving the WRITE_FAIL(*lost_packets*) re-sends *lost_packets* to the DIN I/O module 303 via data link 313 (block 753).

30 When all the packets in *<buffer>* are received, the DIN I/O module 303 checks whether (*<loopId>+1*) PRT exists (block 754). If (*<loopId>+1*) PRT exists, the DIN I/O module

303 adds MAP_SIZE node to (*<loopId>+1*) PRT (block 755). Else, the DIN I/O module 303 creates (*<loopId>+1*) PRT with MAP_SIZE node (block 756).

The DIN I/O module 303 further performs the following:

- 5 - calls *open_loop(<loopId>+1)* to data forwarding module 304 via communication link 312,
- filters packets by traversing (*<loopId>+1*) PRT and keeps MAP_SIZE packet while sending all other packets unchanged into data link 314,
- creates CONTENT_MAP packet(*content_identifier=<contentId>*,
10 *size=<buffer_size>*) and sends it to the data forwarding module 304 via data link 314,
- creates *<contentId>* info packet and sends it to the data forwarding module 304 via data link 314,
- if (*listen_state(<loopId>)* == TRUE), sends *<contentId>* info packet to the
15 host I/O module 302 via data link 313,
- updates MAP_SIZE packet(*size=size+1*) and sends it to the data forwarding module 304 via data link 314,
- calls *close_loop(<loopId>+1)* to the data forwarding module 304 via communication link 312, and
- 20 - sends a WRITE_SUCCESS to the host I/O module 302 via communication link 311.

The host I/O module 302 returns a success status to the application module 301 via communication link 310 if WRITE_SUCCESS is received (block 757).

25

If the data forwarding module 304 receives the *open_loop(<loopId>+1)* function from the DIN I/O module 303 via communication link 312, it sends packets on (*<loopId>+1*) to data link 314 and packets received from data link 314 into (*<loopId>+1*). If the data
30 forwarding module 304 receives *close_loop(<loopId>+1)* function from the DIN I/O

module 303 via communication link 312, it stops sending and receiving packets from data link 314 into (*<loopId>+1*) (block 758).

- *read_content_map*

5

Figure 11 shows a flow diagram between the host I/O module 302, the DIN I/O module 303 and the data forwarding module 304 when the function *read_content_map* is called by the application module 301.

10

In block 760, the application module 301 calls *read_content_map* to the host I/O module 302 via communication link 310. The host I/O module 302 sends READ_CONTENT_MAP_REQ (block 761) to DIN I/O module 303 via communication link 311. The DIN I/O module 303 sends READ_CONTENT_MAP_ACK to the host I/O module 302 via communication link 311.

15

The DIN I/O module 303 determines if (*<loopId>+1*) PRT exists (block 762). If (*<loopId>+1*) PRT exists, the DIN I/O module 303 adds MAP_SIZE and CONTENT_MAP nodes to the PRT (block 763). Else, the DIN I/O module 303 creates the (*<loopId>+1*) PRT (block 764) with MAP_SIZE and CONTENT_MAP nodes. The

20

DIN I/O module 303 calls *tap_loop*(*<loopId>+1*) to the data forwarding module 304 via communication link 312.

The data forwarding module 304 copies and sends packets (block 765) on (*<loopId>+1*) to the DIN I/O module 303 via data link 314. The DIN I/O module 303 filters the packets by traversing (*<loopId>+1*) PRT, and sends MAP_SIZE and CONTENT_MAP packets (block 766) to the host I/O module 302 via data link 313.

25

The host I/O module 302 creates *content_map* from the packets received from data link 313 (block 767), and sends *content_map* to the application module 301 via communication link 310. The host I/O module 302 also sends

30

STOP_CONTENT_MAP_REQ(<loopId>) to the DIN I/O module 303 via communication link 311.

5 The DIN I/O module 303 send STOP_CONTENT_MAP_ACK to the host I/O module 302 via communication link 311, and also removes MAP_SIZE and CONTENT_MAP nodes from the (<loopId>+1) PRT (block 768). It then calls *close_loop*(<loopId>+1) onto the data forwarding module via communication link 312.

10 The data forwarding module stops copying and sending packets on (<loopId>+1) to the data link 314 (block 769).

- *read_content_info*

15 Figure 12 shows a flow diagram between the host I/O module 302, the DIN I/O module 303 and the data forwarding module 304 when the function *read_content_info* is called by the application module 301.

20 In block 770, the application module 301 calls *read_content_info* to the host I/O module 302 via communication link 310. The host I/O module 302 sends READ_CONTENT_INFO_REQ(<loopId>, <contentId>) (block 771) to DIN I/O module 303 via communication link 311. The DIN I/O module 303 sends READ_CONTENT_INFO_ACK to the host I/O module 302 via communication link 311.

25 The DIN I/O module 303 determines if (<loopId>+1) PRT exists (block 772). If (<loopId>+1) PRT exists, the DIN I/O module 303 adds <contentId> node to the PRT (block 773). Else, the DIN I/O module 303 creates the (<loopId>+1) PRT (block 774) with <contentId> node. The DIN I/O module 303 calls *tap_loop*(<loopId>+1) to the data forwarding module 304 via communication link 312.

30

The data forwarding module 304 copies and sends packets (block 775) on (<loopId>+1) to the DIN I/O module 303 via data link 314. The DIN I/O module 303 filters the packets

by traversing (*<loopId>+1*) PRT, and sends *<contentId>* packet (block 776) to the host I/O module 302 via data link 313.

The host I/O module 302 creates *content_info* from the packets received from data link 313 (block 777), and sends *content_info* to the application module 301 via communication link 310. The host I/O module 302 also sends STOP_CONTENT_INFO_REQ(*<loopId>*, *<contentId>*) to the DIN I/O module 303 via communication link 311.

The DIN I/O module 303 send STOP_CONTENT_INFO_ACK to the host I/O module 302 via communication link 311, and also removes *<contentId>* node from the (*<loopId>+1*) PRT (block 778). It then calls *close_loop(<loopId>+1)* onto the data forwarding module via communication link 312.

The data forwarding module stops copying and sending packets on (*<loopId>+1*) to the data link 314 (block 779).

- *read*

Figure 13 shows a flow diagram between the host I/O module 302, the DIN I/O module 303 and the data forwarding module 304 when the function *read* is called by the application module 301.

In block 780, the application module 301 calls *read* to the host I/O module 302 via communication link 310. The host I/O module 302 sends READ_REQ(*<loopId>*, *<contentId_filter>*) (block 781) to DIN I/O module 303 via communication link 311. The DIN I/O module 303 sends READ_ACK to the host I/O module 302 via communication link 311.

The DIN I/O module 303 determines if (*<loopId>+1*) PRT exists (block 782). If (*<loopId>+1*) PRT exists, the DIN I/O module 303 adds *<contentId_filter>* node(s) to the PRT (block 783). Else, the DIN I/O module 303 creates the (*<loopId>+1*) PRT

(block 784) with *<contentId_filter>* node(s). The DIN I/O module 303 calls *tap_loop(<loopId>+1)* to the data forwarding module 304 via communication link 312.

5 The DIN I/O module 303 also determines if (*<loopId>*) PRT exists (block 785). If (*<loopId>*) PRT exists, the DIN I/O module 303 adds *<contentId_filter>* node(s) to the PRT (block 786). Else, the DIN I/O module 303 creates the (*<loopId>*) PRT (block 787) with *<contentId_filter>* node(s). The DIN I/O module 303 calls *tap_loop(<loopId>)* to the data forwarding module 304 via communication link 312.

10 The data forwarding module 304 copies and sends packets (block 788) on (*<loopId>+1*) and *<loopId>* to the DIN I/O module 303 via data link 314. The DIN I/O module 303 filters the packets by traversing (*<loopId>+1*) PRT and *<loopId>* PRT, and sends *<contentId_filter>* packets (block 789) to the host I/O module 302 via data link 313.

15 The host I/O module 302 creates *content* from the packets received from data link 313 (block 790), and sends *content* in *<buffer>* to the application module 301 via communication link 310. The host I/O module 302 also sends *STOP_READ_REQ(<loopId>, <contentId_filter>)* to the DIN I/O module 303 via communication link 311.

20 The DIN I/O module 303 send *STOP_READ_ACK* to the host I/O module 302 via communication link 311, and also removes *<contentId_filter>* nodes from the (*<loopId>+1*) PRT and *<loopId>* PRT (block 791). It then calls *close_loop(<loopId>+1)* and *close_loop(<loopId>)* onto the data forwarding module via communication link 312.

25

The data forwarding module stops copying and sending packets on (*<loopId>+1*) and *<loopId>* to the data link 314 (block 792).

- *listen*

30

Figure 14 shows a flow diagram between the host I/O module 302, the DIN I/O module 303 and the data forwarding module 304 when the function *listen* is called by the application module 301.

5 In block 800, the application module 301 calls *listen* to the host I/O module 302 via communication link 310. The host I/O module 302 sends LISTEN_REQ (block 801) to DIN I/O module 303 via communication link 311. The DIN I/O module 303 sends LISTEN_ACK to the host I/O module 302 via communication link 311.

10 The DIN I/O module 303 sets listen_state(<loopId>) to be TRUE (block 802).

If the host I/O module 302 receives <contentId> info packet from the DIN I/O module 303 via data link 313, it then sends *content_info* in <contentId>, <content_info> to the application module 301 via communication link 310, and sends
15 STOP_LISTEN_REQ(<loopId>) to the DIN I/O module 303 via communication link 311 (block 803).

The DIN I/O module 303 upon receiving the STOP_LISTEN_REQ(<loopId>) sends STOP_LISTEN_ACK to the host I/O module 302 via communication link 311, and sets
20 listen_state(<loopId>) to be FALSE (block 804).

- *take*

Figure 15 shows a flow diagram between the host I/O module 302, the DIN I/O module
25 303 and the data forwarding module 304 when the function *take* is called by the application module 301.

In block 810, the application module 301 calls *take* to the host I/O module 302 via communication link 310. The host I/O module 302 sends TAKE_REQ(<loopId>, <contentId_filter>) (block 811) to DIN I/O module 303 via communication link 311.
30 The DIN I/O module 303 sends TAKE_ACK to the host I/O module 302 via communication link 311.

The DIN I/O module 303 determines if (*<loopId>+1*) PRT exists (block 812). If (*<loopId>+1*) PRT exists, the DIN I/O module 303 adds MAP_SIZE node to the PRT (block 813). Else, the DIN I/O module 303 creates the (*<loopId>+1*) PRT (block 814)
 5 with MAP_SIZE node. The DIN I/O module 303 calls *open_loop(<loopId>+1)* to the data forwarding module 304 via communication link 312.

The data forwarding module 304 sends packets on (*<loopId>+1*) to the DIN I/O module 303 via data link 314 and sends packets received from the data link 314 into
 10 (*<loopId>+1*) (block 815). The DIN I/O module 303 filters the packets by traversing (*<loopId>+1*) PRT, and keeps MAP_SIZE packet while sending all other packets unchanged via the data link 314 (block 816) to the data forwarding module 304.

The DIN I/O module further performs the following:

- 15
- Adds CONTENT_MAP and *<contentId_filter>* nodes to (*<loopId>+1*) PRT,
 - Filters the packets by traversing the (*<loopId>+1*) PRT,
 - Marks CONTENT_MAP packet(s) corresponding to *<contentId_filter>* as invalid and sends them to the data forwarding module 304 via data link 314,
 - 20 - Sends *<contentId>* packet(s) to the host I/O module 302 via data link 313,
 - Sends MAP_SIZE packet to the data forwarding module 304 via data link 314, and
 - Calls *close_loop(<loopId>+1)* to the data forwarding module via communication link 312.

25 The data forwarding module 304 upon receiving the *close_loop(<loopId>+1)* function stops sending and receiving packets from the data link 314 into (*<loopId>+1*) (block 817).

30 The DIN I/O module 303 then determines if (*<loopId>*) PRT exists (block 818). If (*<loopId>*) PRT exists, the DIN I/O module 303 adds *<contentId_filter>* node(s) to the

PRT (block 819). Else, the DIN I/O module 303 creates the (*<loopId>*) PRT (block 820) with *<contentId_filter>* nodes. The DIN I/O module 303 calls *tap_loop(<loopId>)* to the data forwarding module 304 via communication link 312.

5 The data forwarding module 304 sends packets on (*<loopId>*) to the DIN I/O module 303 via data link 314 and sends packets received from the data link 314 into (*<loopId>*) (block 821). The DIN I/O module 303 filters the packets by traversing (*<loopId>*) PRT, and sends *<contentId_filter>* packet (block 822) to the host I/O module 302 via data link 313.

10 The host I/O module 302 creates *content* from the packets received from data link 313 (block 823), and sends *content* in *<buffer>* to the application module 301 via communication link 310. The host I/O module 302 also sends *STOP_TAKE_REQ(<loopId>, <contentId_filter>)* to the DIN I/O module 303 via communication link 311.

15 The DIN I/O module 303 sends *STOP_TAKE_ACK* to the host I/O module 302 via communication link 311, and also removes *<contentId_filter>* node(s) from the (*<loopId>*) PRT (block 824). It then calls *close_loop(<loopId>)* onto the data forwarding module via communication link 312.

20 The data forwarding module stops copying and sending packets on (*<loopId>*) to the data link 314 (block 825).

- *remove*

25 Figure 16 shows a flow diagram between the host I/O module 302, the DIN I/O module 303 and the data forwarding module 304 when the function *remove* is called by the application module 301.

30 In block 830, the application module 301 calls *remove* to the host I/O module 302 via communication link 310. The host I/O module 302 sends *REMOVE_REQ(<loopId>,*

<contentId_filter>) (block 831) to DIN I/O module 303 via communication link 311. The DIN I/O module 303 sends REMOVE_ACK to the host I/O module 302 via communication link 311.

- 5 The DIN I/O module 303 determines if (*<loopId>+1*) PRT exists (block 832). If (*<loopId>+1*) PRT exists, the DIN I/O module 303 adds MAP_SIZE node to the PRT (block 833). Else, the DIN I/O module 303 creates the (*<loopId>+1*) PRT (block 834) with MAP_SIZE node. The DIN I/O module 303 calls *open_loop(<loopId>+1)* to the data forwarding module 304 via communication link 312.

10

The data forwarding module 304 sends packets on (*<loopId>+1*) to the DIN I/O module 303 via data link 314 and sends packets received from the data link 314 into (*<loopId>+1*) (block 835). The DIN I/O module 303 filters the packets by traversing (*<loopId>+1*) PRT, and keeps MAP_SIZE packet while sending all other packets
15 unchanged to the data forwarding module 304 via data link 314 (block 836).

The DIN I/O module further performs the following:

- Adds CONTENT_MAP and *<contentId_filter>* node(s) to (*<loopId>+1*)
20 PRT,
- Filters the packets by traversing the (*<loopId>+1*) PRT,
- Marks CONTENT_MAP packet(s) corresponding to *<contentId_filter>* as invalid and sends them to the data forwarding module 304 via data link 314,
- Removes *<contentId_filter>* packet(s),
- 25 - Sends MAP_SIZE packet to the data forwarding module 304 via data link 314, and
- Calls *close_loop(<loopId>+1)* to the data forwarding module via communication link 312.

The data forwarding module 304 upon receiving the *close_loop*(<loopId>+1) function stops sending and receiving packets from the data link 314 into (<loopId>+1) (block 837).

5 - *remove_loop*

Figure 17 shows a flow diagram between the host I/O module 302, the DIN I/O module 303 and the data forwarding module 304 when the function *remove_loop* is called by the application module 301.

10

In block 840, the application module 301 calls *remove* to the host I/O module 302 via communication link 310. The host I/O module 302 sends REMOVE_LOOP_REQ(<loopId>) (block 841) to DIN I/O module 303 via communication link 311. The DIN I/O module 303 sends REMOVE_LOOP_ACK to the
15 host I/O module 302 via communication link 311.

20

The DIN I/O module 303 determines if SYSTEM_INFO_LOOP PRT exists (block 842). If SYSTEM_INFO_LOOP PRT exists, the DIN I/O module 303 adds MAP_SIZE node to the PRT (block 843). Else, the DIN I/O module 303 creates the
SYSTEM_INFO_LOOP PRT (block 844) with MAP_SIZE node. The DIN I/O module 303 calls *open_loop*(SYSTEM_INFO_LOOP) to the data forwarding module 304 via communication link 312.

25

The data forwarding module 304 sends packets on SYSTEM_INFO_LOOP to the DIN I/O module 303 via data link 314 (block 845). The DIN I/O module 303 filters the packets by traversing SYSTEM_INFO_LOOP PRT, and keeps MAP_SIZE packet ~~which~~ while sending all other packets unchanged to the data forwarding module 304 via data link 314 (block 846).

30

The DIN I/O module further performs the following:

- Adds LOOP_MAP and <loopId> nodes to SYSTEM_INFO_LOOP PRT,

- Filters the packets by traversing the SYSTEM_INFO_LOOP PRT,
- Marks LOOP_MAP packet corresponding to *<loopId>* as unused,
- Removes *<loopId>* packet,
- Calls *remove_loop(<loopId>+1)* to the data forwarding module via communication link 312, and
- Calls *remove_loop(<loopId>)* to the data forwarding module via communication link 312.

The data forwarding module 304 upon receiving the *remove_loop(<loopId>+1)* and *remove_loop(<loopId>)* functions tears down *<loopId>+1* and *<loopId>* loops (or more specifically Label Switched Paths (LSP)) (block 847).

The DIN I/O module 303 further sends MAP_SIZE packet via data link 314, and calls *close_loop(SYSTEM_INFO_LOOP)* via communication link 312 to the data forwarding module 304 (block 848).

The data forwarding module 304 upon receiving the *close_loop(SYSTEM_INFO_LOOP)* function stops sending and receiving packets from the data link 314 into (*<loopId>+1*) (block 849).

The following references are cited in this document:

[1] CANARIE Inc., *Wavelength Disk Drives*, <http://www.ccc.on.ca/wdd/>

5 [2] Light Reading, “Wavelength Division Multiplexing(WDM)”, 2001

[3] <http://www.lucent.com/press/0601/010628.bla.html>

[4] Alcatel Press Release, 7 September 2000, <http://www.cid.alcatel.com>

10

[5] Miguel A. Ruiz-Sanchez, Ernst W. Biersack and Walid Dabbous, “Survey and Taxonomy of IP Address Lookup Algorithms”, IEEE Network, Vol. 15 No. 2, pp. 8-23, March/April 2001

15 [6] David E. Taylor et. al. , “Scalable IP Lookup for Programmable Routers”, IEEE Infocom 2002

[7] Jim Gemmell et. al. , “The PGM Reliable Multicast Protocol”, IEEE Network Special Issue on "Multicasting: An Enabling Technology", Vol. 17, No. 1, pp. 16-22, Jan/Feb 2003

20

[8] US Patent Application 10/618,170

What is claimed:

1. A shared network system comprising a plurality of computer nodes and connections between the computer nodes, wherein

5 - a plurality of first computer nodes in the shared network system are adapted to define a plurality of looping paths such that data can be circulated therein, wherein each of the looping paths traverses at least two of the plurality of first computer nodes, wherein each of the looping paths is associated with a logical address which is unique within the network system and which is independent
10 from the physical address of any of the computer nodes;

- at least one second computer node being adapted to inject data into and retrieve data from one of the looping paths, such that

○ in the case of injecting data into one of the looping paths, the logical address of the looping path the data should be injected into is
15 determined and is associated with the data to be injected into the looping path; and

○ in the case of retrieving data from one of the looping paths, the logical address of the looping path the data should be retrieved from is determined.

20 2. The shared network system according to claim 1, wherein the plurality of first computer nodes are computer nodes of a core network, and wherein the at least one second computer node is a computer node of an access network, and wherein the computer node of the access network is connected to the core network via at least one of
25 the first computer nodes so as to access at least one of the looping paths through the said one of the first computer nodes.

3. The shared network system according to claim 2, wherein the at least one second computer node of the access network is adapted to deliver data to be injected into one of
30 the looping paths using a multicast protocol mechanism.

4. The shared network system according to claim 3, wherein the at least one second computer node of the access network is adapted to deliver data to be injected into one of the looping paths using the Internet Protocol (IP) multicast mechanism.

5 5. The shared network system according to claim 2 or 3, wherein the at least one second computer node of the access network is adapted to deliver data retrieved from one of the looping paths using a multicast protocol mechanism.

6. The shared network system according to claim 5, wherein the at least one second
10 computer node of the access network is adapted to deliver data retrieved from one of the looping paths using the Internet Protocol (IP) multicast mechanism.

7. The shared network system according to any one of claims 2 to 6, wherein the core network is an optical network.

15

8. The shared network system according to any one of claims 2 to 7, wherein the access network is an optical network or a wired network.

9. The shared network system according to any one of claims 2 to 8, wherein the
20 computer nodes of the core network are routers or switches.

10. The shared network system according to any one of claims 2 to 9, wherein the computer nodes of the access network and the computer node which connects the access network to the core network each comprises an input/output module, wherein the
25 input/output modules of said computer nodes interact with each other to inject and retrieve data into and from the looping path in the core network.

11. The shared network system according to claim 10, wherein the input/output modules of the computer nodes of the access network are adapted to interact with the
30 input/output module of the computer node which connects the access network to the core

network based on an input/output protocol according to a transport layer protocol in the Open Systems Interconnection (OSI) reference model.

12. The shared network system according to claim 11, wherein the input/output
5 modules of the computer nodes of the access network and the computer node which connects the access network to the core network are adapted to implement the input/output protocol based on the User Datagram Protocol (UDP).

13. The shared network system according to any one of claims 10 to 12, wherein
10 when a data operation request is issued by the input/output module of one of the computer nodes of the access network to the input/output module of the computer node which connects the access network to the core network, the input/output module of the computer node of the access network is adapted to re-issue the data operation request after a predetermined time.

14. The shared network system according to any one of claims 10 to 13, wherein the
15 input/output module of the computer node which connects the access network to the core network is adapted to retrieve desired data from one of the looping paths in the core network by performing a prefix matching of the data in the looping path using a Multibit
20 Trie data structure.

15. The shared network system according to any one of claims 2 to 14, wherein the
shared network system further comprises a loop data manager being adapted to remove
data from the looping path when the lifetime of the data has expired or when the data has
25 been selected to be removed from the looping path by a computer node of the access network.

16. The shared network system according to any one of claims 1 to 15, wherein the
plurality of first computer nodes in the shared network system are adapted to define the
30 plurality of looping paths based on the extended Generalized Multiprotocol Label Switching protocol.

17. A method for operating a shared network system comprising a plurality of computer nodes and connections between the computer nodes,

the method comprising

- 5 - defining a plurality of looping paths such that data can be circulated therein by a plurality of first computer nodes in the shared network system, wherein each of the looping paths traverses at least two of the plurality of first computer nodes;
- 10 - associating each of the looping paths with a logical address which is unique within the network system and which is independent from the physical address of any of the computer nodes;
- 15 - in the case of injecting data into one of the looping paths by at least one second computer node which is adapted to inject data into one of the looping paths, determining the logical address of the looping path the data should be injected into, and associating the data to be injected into the looping path with the logical address;
- 20 - in the case of retrieving data from one of the looping paths by at least one second computer node which is adapted to retrieve data from one of the looping paths, determining the logical address of the looping path the data should be retrieved from.

18. The method for operating a shared network system according to claim 17, further comprising circulating loop-information data in at least one of the looping paths, wherein the loop-information data comprises information on the looping paths defined in the shared network system.

19. The method for operating a shared network system according to claim 17, further comprising circulating content-information data in at least one of the looping paths, wherein the content-information data comprises information on the content stored in the looping paths of the shared network system.