Event-driven network topology monitoring function

by W. Chao
W. Tsun

This paper discusses the design of a topology management application. The application monitors the topology of point-to-point networks in which each network node is required to contain only local trunk information. To build and continuously update the topology map, information must be collected from network nodes. The challenge is to do this in real time without adversely consuming network bandwidth. By defining a clear set of conditions to determine whether polling or event monitoring should be used, this design makes it possible to realize the advantage of monitoring network topology by processing events. With the use of event monitoring, first, consumption of network bandwidth by network management traffic can be significantly reduced compared to a pure polling approach, and second, the topology map represents a continuous topology history.

One of the challenges of network management is how to help a network operator conceptualize the layout of the network and call attention to adverse status changes in a fast and accurate manner. A modern network could be a collection of voice, video, or data processing devices connected by communication lines. As the network complexity increases, it becomes more difficult to conceptualize how the various pieces are laid out. One way to help an operator visualize the layout is to display a topology map showing each node and the connections that exist among them. To this end, a prototype[1] has been built using virtual-world technology to provide three-dimensional input and output capabilities to facilitate the management tasks.

This topology map must reflect the actual network topology and its current status in real time as accurately as possible so that an operator can respond quickly to possible network trouble spots. This requirement is most evident in current and future broadband networks where an outage could impact many end-to-end connections. In addition, each connection may suffer the loss of large amounts of information before the source can be suspended or an alternate path can be established to reroute traffic.

To build and continuously update the topology map, information must be collected from each network node. The challenge is to obtain the information in real time without adversely consuming network bandwidth. The simple polling approach to gather network information results in a severe problem by wasting network bandwidth.[2] This problem is especially a concern in wide area networks (WANs), where recurring transmission line expenses directly affect the cost of network operation. Bandwidth used by network management traffic comes at the expense of user bandwidth. Even in broadband networks, such as asynchronous transfer mode (ATM) networks, minimizing management bandwidth usage is a concern. It is evident in the ATM Forum's requirement for ILMI (interim local management interface)[3] traffic to be less than 1 percent of line capacity. Since topology monitoring is but one part of the overall

 0018-8670/96/$5.00 © 1996 IBM

management function, it is necessary to minimize traffic generated by the topology application.

This paper presents a design for a topology management application (TMA) that monitors point-to-point networks in which each node contains only information concerning its neighboring nodes. This design simplifies the implementation of the node and thus reduces its costs. Existing monitoring applications for point-to-point networks utilize either polling or event monitoring. This paper introduces a method of interchangeably using polling- and event-monitoring modes to maintain an accurate and up-to-date topology map while minimizing network management traffic. The design favors event monitoring and resorts to polling monitoring only when conditions for event monitoring are not satisfied.

In a point-to-point network, typically seen in a WAN, one link or trunk directly connects two network nodes. The trunk connection established between two nodes is not shared with other nodes. Because of transmission line costs, each node has a relatively small number of trunk connections. In some point-to-point network architectures, each node maintains an updated copy of the entire network topology. Therefore, topology monitoring can be performed by collecting information from just one network node or two to verify the integrity of the information. One example is a feature of IBM NetView* called Advanced Peer-to-Peer Networking* Topology and Accounting Manager (APPNTAM).[4] The agent application in a network node is able to provide a manager with the entire APPN backbone from its point of view. The APPNTAM exploits event monitoring, but unlike the design presented in this paper, it requires each network node to have a full topology.

Other types of networks have a different architecture and therefore require a different monitoring application. For instance, stations within a local area network (LAN) segment, Ethernet, or token ring are all connected to one another through shared transmission media. Another example is based on Transmission Control Protocol/Internet Protocol (TCP/IP) networks. Using IP-based services, a monitoring application can discover and present a topology map showing IP subnets connected by IP gateways. IP hosts within an IP subnet can also be shown. However, the actual connectivity among all IP hosts within a subnet is generally not available to the application, unless the application is designed to understand the underlying subnet technology. Therefore, these two architectures do not lend themselves to the design discussed in this paper.

In this paper, the management environment consisting of the managed network, the management station, and related application programs is shown first. Then the management information required at the network nodes is explained. In the next section, polling and event monitoring are defined and compared. Then conditions for switching between polling and event monitoring are presented. After that, details on how event monitoring works are discussed. Finally, some implementation details and future extensions are given.

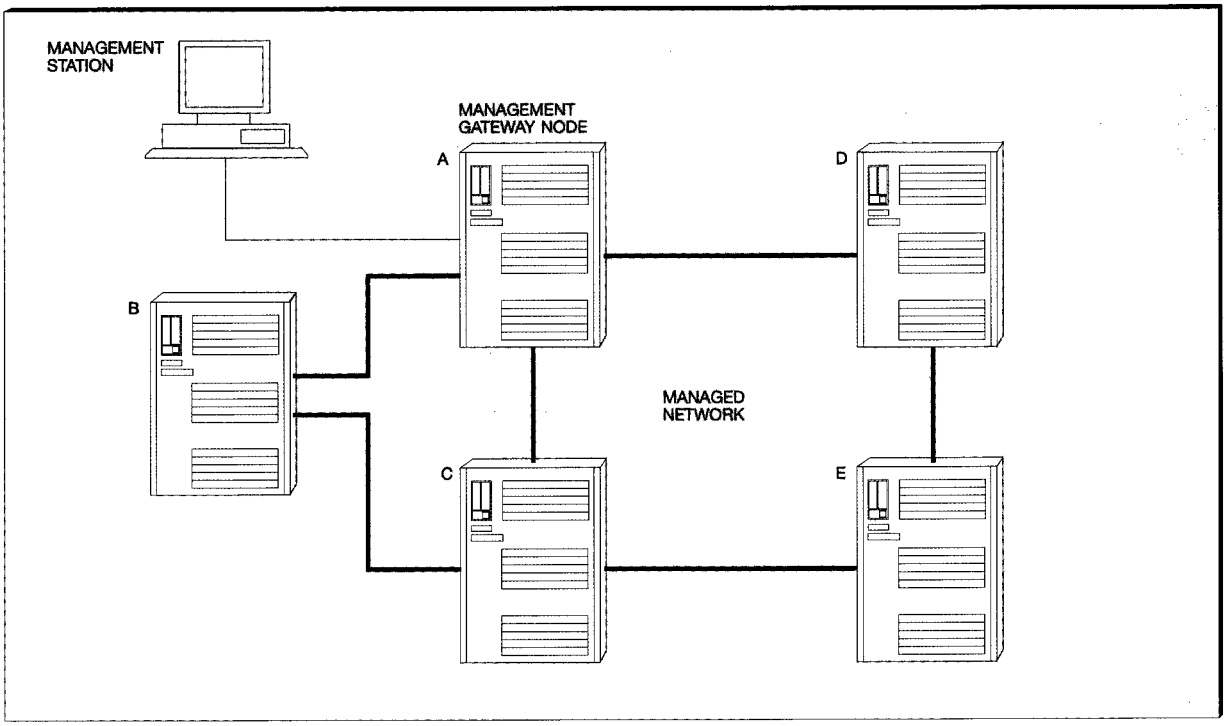
Management environment

The management environment consists of a management station (MS) and a managed network. The management station provides management platform services such as a graphical user interface (GUI), database, and communication support. The TMA is an application program running on the management station. It uses platform services to perform the following: process network operator commands, communicate with the managed network, retrieve management information, present the topology in graphical form, and display network status information. This information can be made available to other management functions such as accounting, connection, and performance. On the basis of the functions it performs, the topology management application is also referred to as *manager* in this document.

The managed network consists of nodes and trunks connecting the nodes. Each node in the network has a special management application called an *agent* running locally. The main functions supported by an agent are as follows: maintain real-time management information related to the networking functions of the node, respond to a manager's request for information retrieval, and emit unsolicited notifications when defined events such as a trunk outage occur.

The relationship between a manager and an agent is similar to a client/server relationship in which the manager is the client and the agent is the server. In most cases, a server is activated first and requires no client information or client presence to become operational. Once it is operational, it waits to provide services to its clients. A client is usually activated with some information about the server and cannot perform meaningful tasks until a server is available. The agent application is typically activated as part of the

Figure 1 Management environment



startup procedure of the network node. However, it may be stopped and reactivated at any time independent of the manager. It also does not require any information about the manager. The job of the agent is to make a set of supported management information, also called the management information base (MIB), available to the manager upon request. Configuration information such as network node addresses is required by the manager to establish a connection with each agent. If a manager is interested in subscribing to unsolicited notifications, it must inform each agent of the manager's address and specify the type of information of interest. It should terminate its subscription when the node is no longer of interest to avoid unwanted notifications from entering the network.

A manager needs to establish a connection with each agent in order to exchange information. Since the manager is external to the network, one of the nodes must be assigned as the gateway. All communication between manager and agents must go through the gateway (see Figure 1). The connection between the manager and the gateway need not be direct. (This extension is not within the scope of this paper.) If the gateway fails, another node may be activated to take over. However, only one gateway is in a network at any one time for a given manager. This condition does not preclude another management station from using the same or different node as its gateway.

Figure 1 shows how a manager relies on the gateway node and the managed network to exchange information with its agents. If any part of the managed network fails or if the gateway node becomes dysfunctional, communication between the manager and parts or all of the network may be cut off. These situations need to be taken into account when designing a TMA.

Management information and topology map

As stated earlier, in a point-to-point network, the network topology information available at each node can vary from the entire network topology to only local connectivity, depending on the architectural design of the network. The TMA discussed in this pa-

Table 1 Example of local connectivity information in a node

Trunk Endpoint	Operation Status	Partner Endpoint
t1	enabled	(n2, t1)
t2	enabled	(n3, t1)
t3	disabled	(n4, t1)
t4	disabled	—

per is designed to work with network nodes that are only required to have local connectivity information. At each node, the agent needs to maintain management information related to each of its trunk endpoints. A trunk endpoint is a resource that can be connected to the trunk endpoint of another node to form a trunk connection. When a trunk connection is established, the two trunk endpoints are said to be in partnership with each other. Each trunk endpoint must contain the following information: its own name, its node name, the trunk endpoint name of the partner, the node name of the partner, and the operational state. At any given time, a trunk endpoint may have valid, invalid, or no partner information. Invalid information occurs when the alleged partner no longer exists or it does not reveal the alleged connection. The operational state can be either *enabled* or *disabled*. A trunk endpoint without a partner is defined to be disabled. However, an enabled trunk endpoint must have a partner. Local connectivity is the collection of all trunk endpoints for a given node. Using the notation (n, t) to denote a trunk endpoint t in node n, Table 1 shows an example of local connectivity information.

A topology map is a collection of resource information presented to the user in graphical form. Resources include nodes, trunk connections, and their status. The status of a node is either *manageable* or *unmanageable*, depending on whether the agent of the node is responding to the manager's management requests. The existence of trunk connections cannot be obtained by merely querying the agents, since agents are only aware of local connectivity as described in the previous section. Because trunk endpoints that form trunk connections are located at different nodes, it is possible that the local connectivity information retrieved from these nodes is inconsistent. For instance, if a trunk connection between two trunk endpoints fails to function normally, one of the trunk endpoints may be removed, or it may form a new trunk connection with a different trunk endpoint. In either case, the partner information in the other node will be incorrect. Also, it is possible that two trunk endpoints have different operational states. This possibility occurs when each end detects the failure of the trunk connection at a different time.

When an inconsistency is detected in local connectivity information retrieved from different nodes, the following rules are used by the manager to resolve the conflict. A trunk connection is created in the topology map between two manageable nodes if there is a trunk endpoint at each node with each other endpoint as partner. If the operational state of these two trunk endpoints is inconsistent, the resulting status of the trunk connection will be set to disabled. In other words, the trunk connection in a topology map will be set to "disabled" as long as one of the trunk endpoints is reported as disabled. Only when both trunk endpoints are enabled is a trunk connection set to "enabled."

Figure 2 shows an example of a network with the local connectivity information of each manageable node and the resulting topology map created by the manager. For the purpose of illustration, the trunk endpoints within a node are shown together with the topology map. In the actual implementation, the topology map may contain only nodes and trunk connections between nodes, and the trunk endpoints within a node can be shown in a separate window.

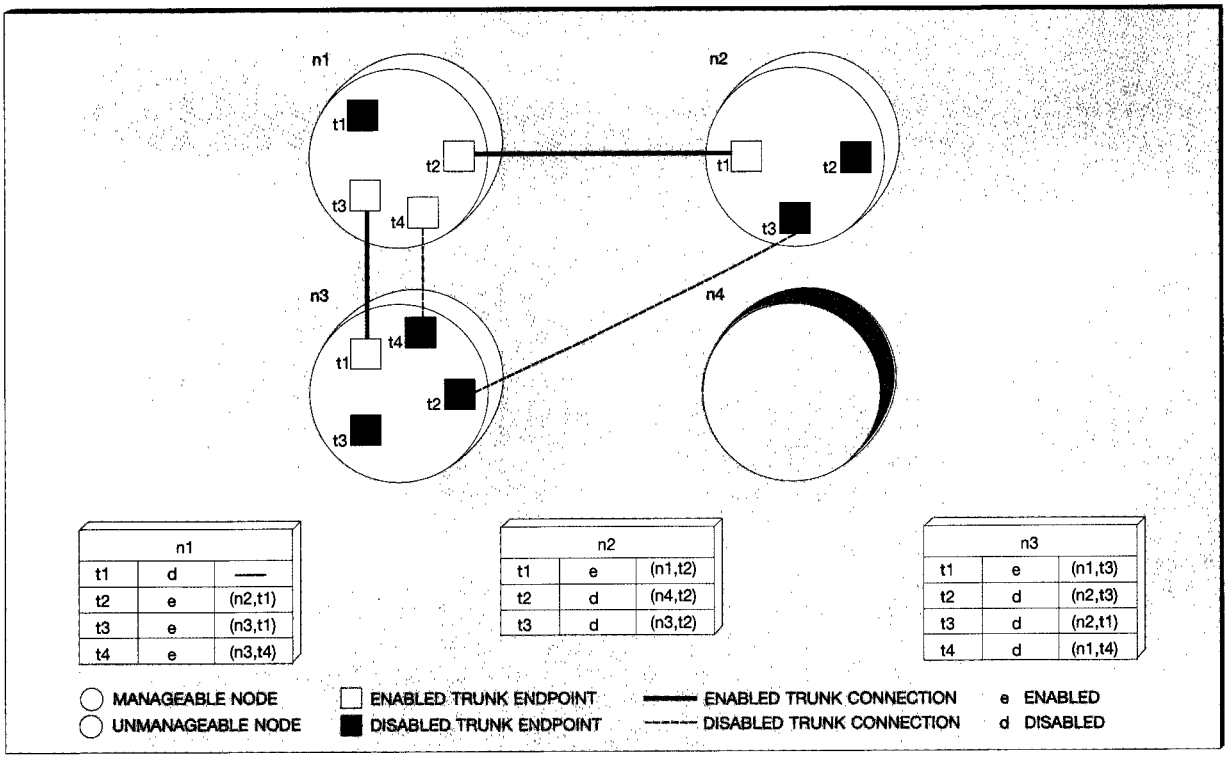
Polling monitoring and event monitoring

In order to define polling monitoring, a snapshot must first be introduced. A snapshot action refers to the task performed by the manager to obtain a best effort topology of the network. First, a request for local connectivity information is sent to all nodes currently configured in the network. The returned information is processed and the resultant topology map is created.

Polling monitoring is an approach that monitors network topology by periodically performing a snapshot. This mode is the simplest to implement for both manager and agent. The trade-off is that the accuracy of the topology map depends on the granularity of the polling interval. As the polling frequency increases, the accuracy increases. At the same time, more network bandwidth is consumed by management traffic. Therefore, a practical limit is imposed on the snapshot frequency.

A second approach is to have the manager perform a snapshot to create an initial topology map. There-

Figure 2 Topology map and local connectivity information



after, the agent emits an event whenever changes to local connectivity are detected. The manager updates the topology map based on received events. Should an agent in a reachable node abort without notice, it will be detected by the agent liveliness mechanism that is explained within a subsection of the next section. This method will be referred to as *event monitoring*.

Polling versus event monitoring. Event monitoring has the following advantages over polling monitoring:

1. The manager need not perform snapshots as frequently as required in polling monitoring while maintaining a more accurate topology map. Management traffic is reduced because unnecessary snapshots are not performed. In addition, the bandwidth consumption used to transport a topology change event is significantly less than that required by a network-wide snapshot request or response.
2. A continuous network topology history may be maintained as opposed to a sequence of samples. In event monitoring, if no event is received, the network experiences no change. In polling monitoring, if consecutive snapshots result in two identical topology maps, it cannot be asserted that a network change has not occurred. This information may be critical to other management functions (e.g., accounting) that are inherently continuous.

In event monitoring, it is critical for all topology change events to be received by the manager. A management protocol can provide reliable transport of events once sent, as described in Reference 5. However, there is no guarantee that all the agents will always be able to report local connectivity changes. Consider the case when an agent application aborts. If the networking functions of the node continue to operate and all the trunk endpoints remain enabled, their partner endpoints in adjacent nodes will not detect any failure. Hence, no topology events are emitted from the adjacent nodes. The manager will not be made aware of the absent agent until another

snapshot is performed. Before the snapshot, the local connectivity is subject to change. A possible scenario is that a new trunk endpoint is created in the agentless node and subsequently forms a trunk connection with a new node. Since no event is emitted and the new node will not be contacted by the manager as to where to send information, the topology map will not be updated to include the new trunk connection and the new node. Subsequently, any other nodes activated and connected to the new node will not be discovered either. The inconsistency between the map and the actual network topology that should be seen by the manager will not be corrected until another snapshot is performed, which will not happen if the manager is waiting for events. It is as though a blind spot exists in the network. Thus, event monitoring may not be feasible at all times.

Switching between polling and event monitoring. In order to take advantage of event monitoring and ensure the correctness of the resulting topology map, it is necessary to switch between polling and event monitoring.

When topology monitoring is started, a snapshot is performed to create an initial topology map. The manager then evaluates whether conditions are conducive to event monitoring. If not, polling monitoring will be used, and snapshot actions are performed periodically to create topology maps. At the end of each snapshot, the conditions for event monitoring are reevaluated.

How event monitoring works

Once in event monitoring, the manager will update the topology map based on received events to maintain the topology history. In addition, the manager needs to constantly determine if event monitoring should be continued. If not, the monitoring mode must return to polling. In this section we discuss the following:

- Definition of reachability
- Conditions for event monitoring
- A mechanism called agent liveliness query
- Kinds of events that are reported to the manager by agents
- How these reported events are used by the manager

Definition of reachability. A node is *reachable* if there is an *enabled* path between the manager and the node; otherwise, it is *unreachable*. Note that this definition is independent of the agent.

Conditions for event monitoring. At the end of a snapshot, the status of each node is either manageable or unmanageable, depending on whether the local connectivity of the node is successfully retrieved. A node is *manageable* if both of the following statements are true:

1. The node is reachable.
2. The agent in the node is actively running.

Otherwise, a node is *unmanageable*. Since all the connections between manager and agents use paths that go through the management gateway, a node unreachable from the gateway is also unreachable from the manager. In cases where statement 1 is false, these nodes are disconnected from the manager, and no further update from them is possible until they become connected again. That part of the topology map will be shown with markings that indicate that the information could be out-of-date. In cases where statement 2 is false, an incorrect topology map would be created if event monitoring is used. A scenario was given in the last section to illustrate the problem. As long as there is one such unmanageable node in the network, it is possible that the manager will lose events from that node without being aware of the situation. Therefore, both of the following conditions have to be true for the manager to use event monitoring:

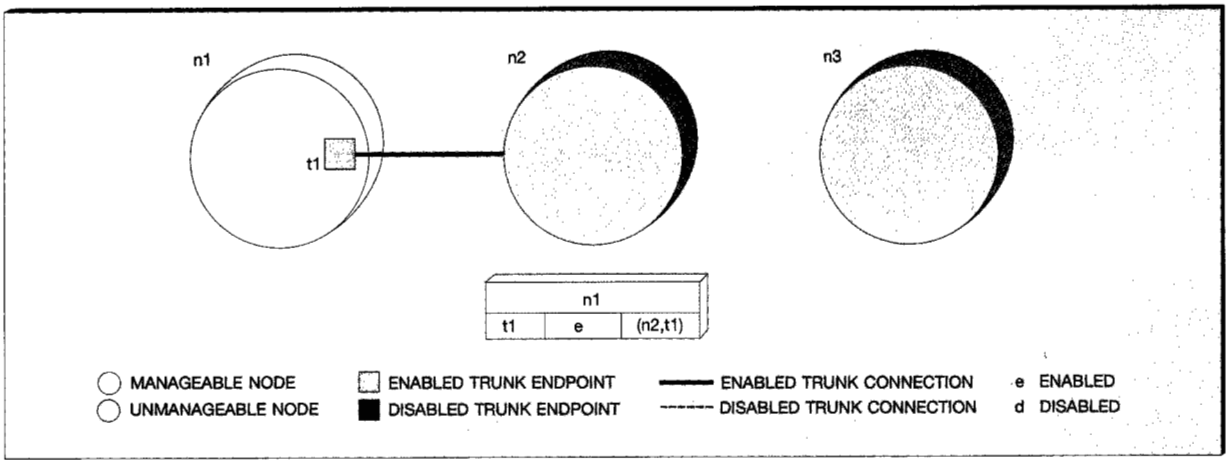
Condition 1. The gateway node is manageable.

Condition 2. All the unmanageable nodes are unreachable.

It is trivial to determine whether condition 1 is met. But if there is any unmanageable node in the network, how can a manager determine whether an unmanageable node is unmanageable due to statement 1 or statement 2 being false? The answer lies in the local connectivity information of the neighboring nodes. Consider the following test when all local connectivity information has been collected.

1. An unmanageable node m and a manageable node n are given.
2. If there exists a trunk endpoint t in node n such that (n, t) is enabled and has a partner trunk endpoint in node m, it follows that node m is reachable from the manager.
3. Thus node m violates condition 2.

Figure 3 Detecting a reachable but unmanageable node



Note that this test is insufficient for identifying all reachable but unmanageable nodes. Consider an example shown in Figure 3. In this network, node n1 is the gateway node for the manager and is manageable. The local connectivity information of n1 can be retrieved by the manager. Nodes n2 and n3 are both unmanageable because their agents are not active; therefore, the local connectivity information of these nodes is not available to the manager. The resulting topology map of a snapshot will look like the picture in Figure 3. With use of the above test, node n2 will be recognized by the manager as reachable because an enabled trunk connection exists between node n1 and n2. If an enabled trunk exists between node n2 and n3, node n3 is also reachable from the manager. However, this trunk will not be detected by the manager because the local connectivity information of neither node n2 nor n3 is available to the manager. But it can be shown that given a manageable gateway node, at least one of the reachable but unmanageable nodes can be detected by using the above test if any exist in the network.

Figure 2 shows a topology map that satisfies event-monitoring conditions if node n4 is not the gateway. Note that even though node n2 has a trunk endpoint whose partner is n4, it is disabled; therefore, it does not violate condition 2.

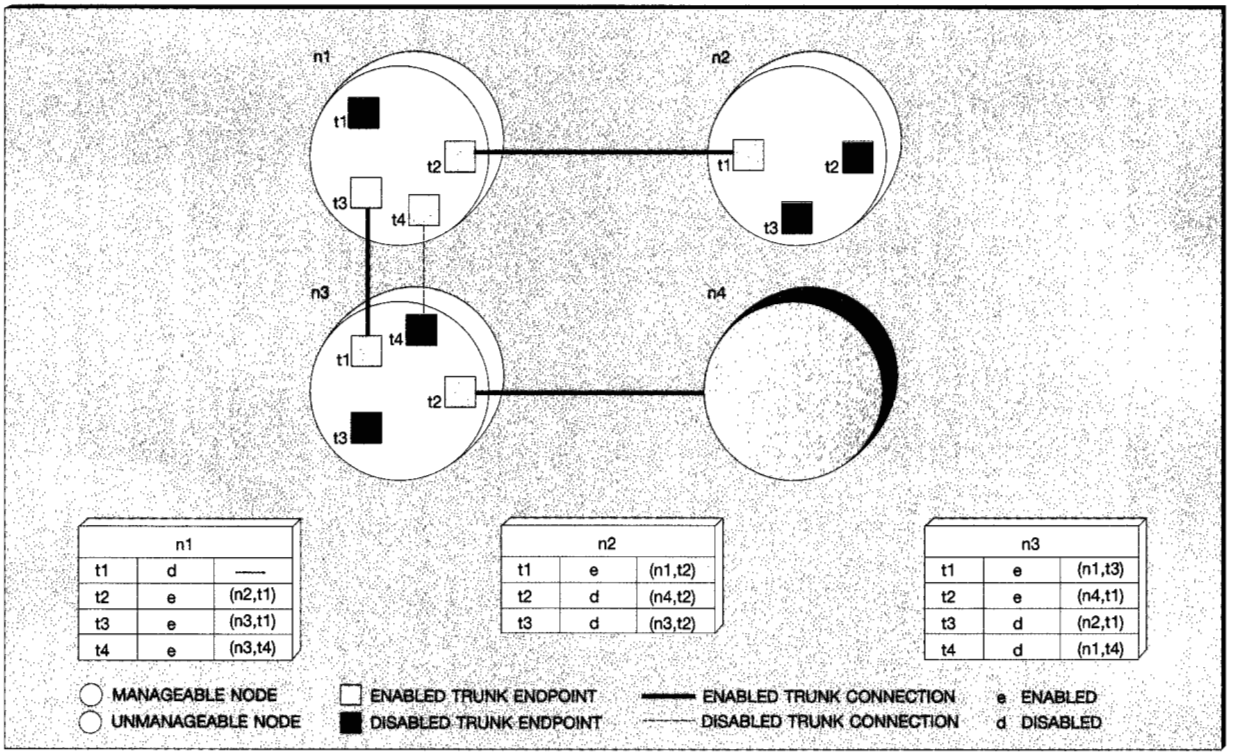
Figure 4 shows a topology map created by a snapshot. Node n1 is the gateway and node n4 is unmanageable but reachable from the manager through node n3. Event-monitoring conditions are not satisfied; therefore, polling monitoring must be used. Note that in this case it is still useful to display a trunk connection between node n3 and n4 even though it cannot be verified with the agent in n4.

In summary, given that the gateway node is manageable, all agents of manageable nodes are active, and all unmanageable nodes are not reachable (i.e., disconnected from the manager), event monitoring should be used. If any agent of the manageable nodes becomes inactive or if any unmanageable nodes become reachable but with an inactive agent, polling monitoring should be used.

Agent liveliness query. While in event monitoring, the manager needs all agents residing in manageable nodes to be continuously present. Otherwise, if any agent aborts, or aborts and reactivates, there exists a period of time when a reachable unmanageable node exists, and events may be lost. It is possible to design a protocol among agents so that agent failure could be detected and reported by other agents. However, the lack of standard protocol among agents and the complexity to the agent makes this approach less attractive. This topology management application chooses an alternative solution that is simpler and equally robust.

When event monitoring is started, the manager periodically queries the agent liveliness from all currently manageable nodes. In each query, the manager polls the agent start time or anything that is updated each time the agent is restarted. For exam-

Figure 4 A topology map where event monitoring is not used



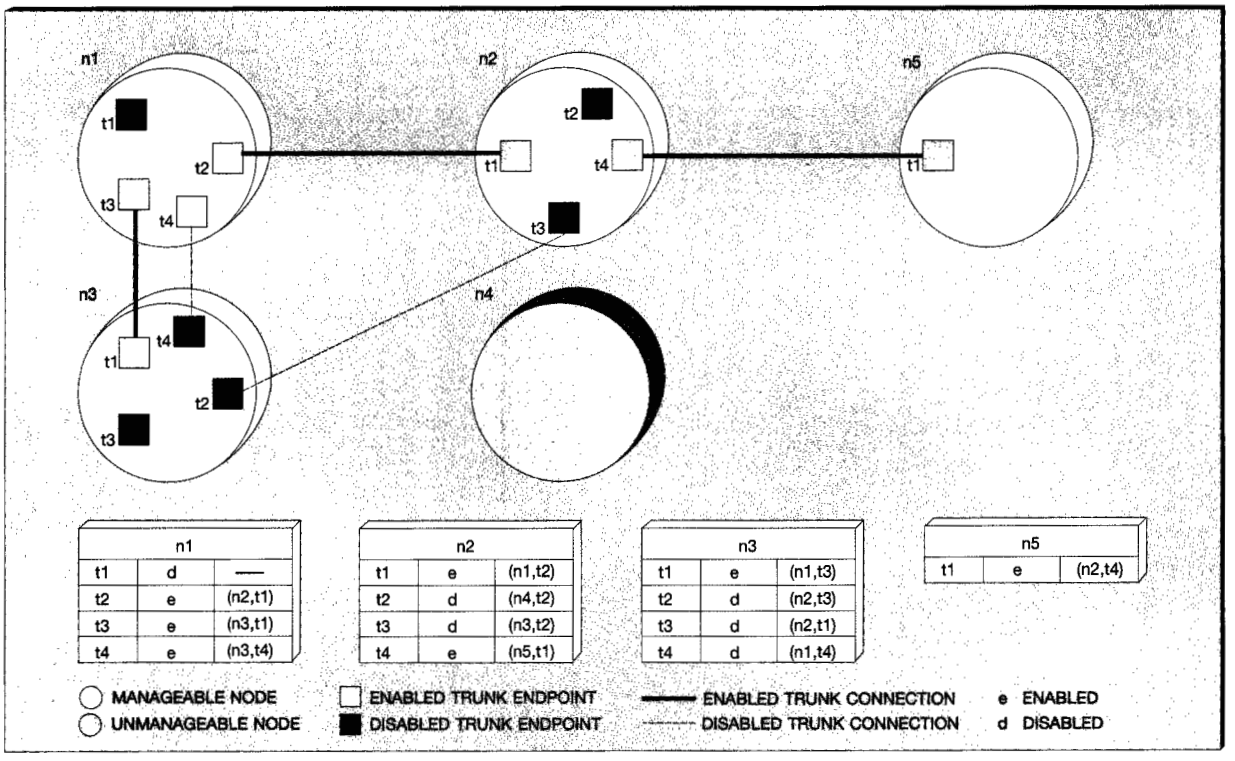
ple, a local time stamp indicating the agent activation time or a counter that keeps the number of times activated will serve this purpose. In cases where the agent fails to function normally but does not abort, e.g., when an event cannot be forwarded, the start time must also be updated to indicate the interruption of normal agent function. If any of the manageable nodes do not respond to the query, or if the returned value in any response is different from that of the previous query, switching to polling monitoring is made immediately by performing a snapshot.

This mechanism allows the manager to ensure correctness while in event monitoring. The additional cost compared to pure event monitoring is the expense of polling the agent liveliness information. However, there is still a significant saving in network management traffic compared to a pure polling method. Consider the following:

1. The information for each query response is considerably less than the local connectivity information. There is only one small piece of information regardless of how many trunk endpoints a node contains.
2. Only currently manageable nodes are queried. The saving is directly related to the percentage of unmanageable nodes.
3. The frequency of agent liveliness queries need not be as high as that for local connectivity queries when the agent applications are relatively stable.
4. The above reasons become more significant as the size of the network increases.

It should be noted that if nodes are designed in such a way that the networking function terminates whenever the agent aborts, an unmanageable node is always unreachable from the manager. Therefore, the conditions for event monitoring are always met, and the liveliness query mechanism is not needed at all. However, it may be impractical to design management functions based on this assumption, since it is unlikely that real products would sacrifice network availability for a simpler management function design.

Figure 5 Topology map before nodes become disconnected



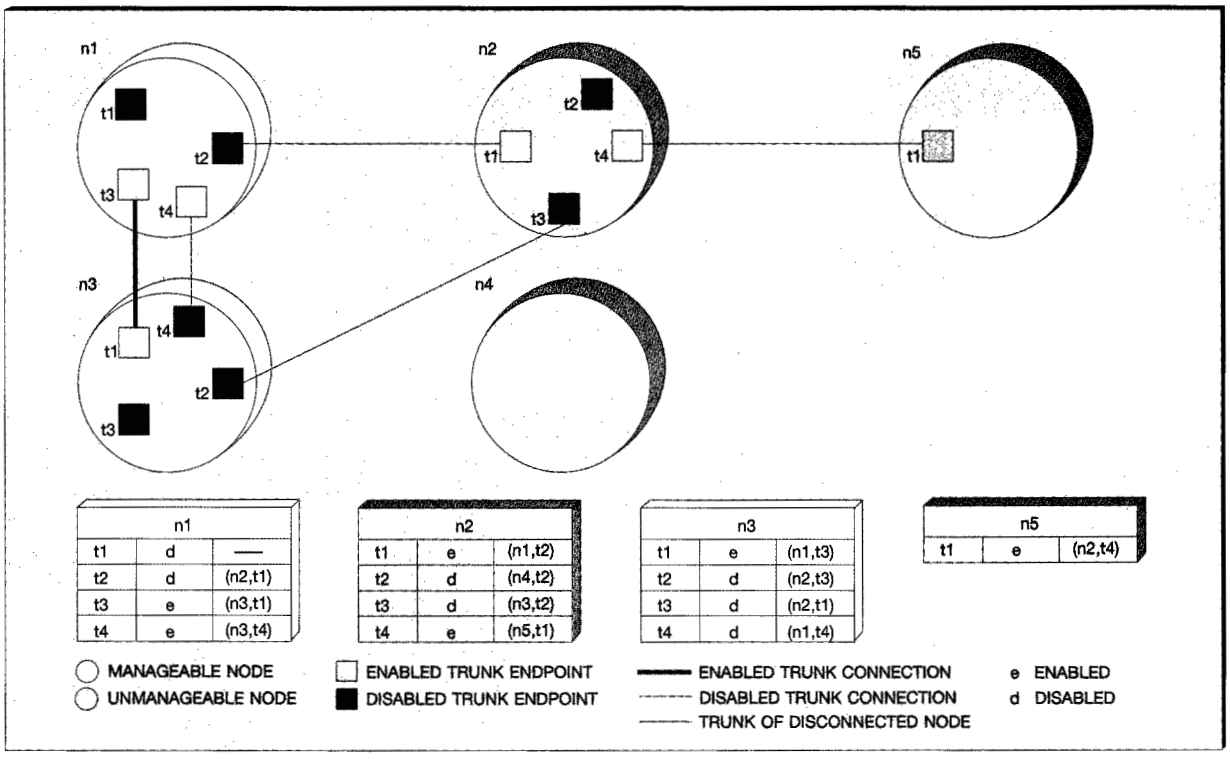
Trunk endpoint events. In event monitoring, the manager will only update the topology map when events are received. These four trunk endpoint events are of interest: *create, delete, enable*, and *disable*. To simplify the design of the agent, a trunk endpoint is created with disabled status and no partner. Under these conditions, a create event does not affect the set of existing trunk connections present in the network. When it forms a trunk connection with another trunk endpoint, an enable event carrying the partner information is emitted. A trunk endpoint should only allow deletion when its status is disabled; therefore, a disable event will always precede a delete event. When a delete event is received, the trunk endpoint is removed. If a trunk connection is anchored on one end by this trunk endpoint, it is removed from the topology map.

The processing of create and delete events is straightforward. Enable and disable events, however, require additional processing. A disable event may result from a failed trunk connection. This failure may cause a network partition. An enable event may reconnect a partition or be an indication that a new node has joined the network. A disable event is processed as follows:

> Upon receipt, check the operational status of the corresponding trunk connection. If status is enabled, change it to disabled.

Because the connection between the management station and each agent goes through the gateway, an agent must be reachable from the gateway in order to communicate with the manager. Each time a trunk connection changes from enabled to disabled, a reachability tree, using the shortest path tree algorithm[6] with the gateway node as root, is computed to determine whether any node has become disconnected from the manager. All disconnected nodes that are currently manageable are marked as unmanageable because they are not reachable from the manager. The trunk connections between two disconnected nodes or between one manageable node and one disconnected node will still be displayed in their state just prior to being disconnected. A dif-

Figure 6 Topology map after nodes become disconnected



ferent color or style is used to indicate the information is out-of-date. Updates to that part will not be possible until at least one of the disconnected nodes becomes connected again.

To illustrate a trunk endpoint disabled event that results in a network partition, consider the example shown in Figure 5. It is the same network as shown in Figure 2 with the following additions: a manageable node n5, two enabled trunk endpoints (n5,t1) and (n2,t4), and a resulting trunk connection between n2 and n5. The local connectivity information from each manageable node and the resulting topology map from a snapshot are shown in the figure. Since event monitoring conditions are met, the manager will be in event-monitoring mode.

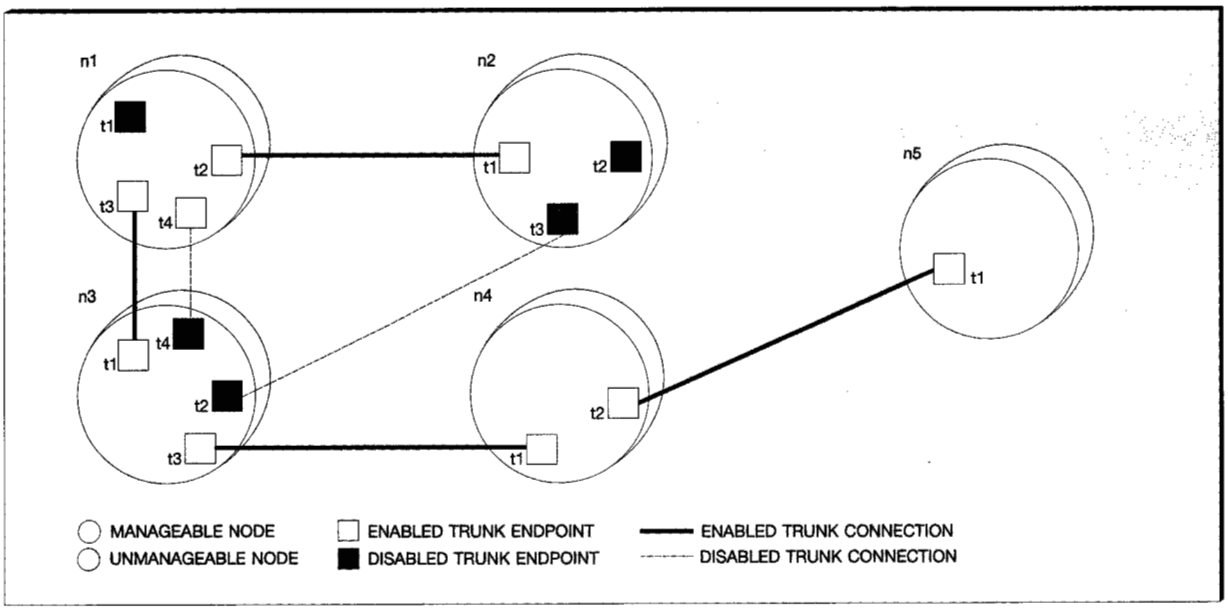
Given that the picture is the current topology map and that event monitoring is in effect, a trunk endpoint disabled event from (n1,t2) is then received by the manager. The manager will proceed as follows. First, the status of the trunk connection between (n1,t2) and (n2,t1) is set to disabled. Then the reachability tree is computed using the gateway node n1 as root and the status of all trunk connections in the network. Node n2 and n5 are then identified as disconnected nodes, and their status is set to unmanageable. Any trunk connection with one of the endpoints in n2 or n5 will be marked differently in the map to indicate its status may be out-of-date. The resulting topology map is shown in Figure 6. The connectivity information of node n2 and n5 maintained by the manager is now obsolete.

Enable events are processed as follows:

> Upon receipt, verify the partner information carried in the event by checking the status of the partner trunk endpoint. If the partner exists and has consistent operational status and partner information, a trunk connection with enabled status is created if one does not already exist.

Discovering nodes. If the partner information in the enable event indicates that the partner trunk endpoint is currently in an unmanageable node or a new

Figure 7 A topology map after discovery procedure is performed



node, a discovery procedure is started from that node. First, the local connectivity information is retrieved from the new or unmanageable node. If there exists a trunk endpoint in the explored node with enabled status and its partner trunk endpoint is in another new or unmanageable node, continue exploration from that node. This procedure is applied recursively to discover all the unmanageable or new nodes that have just become manageable as a direct result of the enable event.

Given the network topology map in Figure 2, where node n1 is the gateway and node n4 is unmanageable, Figure 7 shows a resulting topology map after discovery has completed due to an enable event. The scenario is as follows:

1. While n4 is unmanageable, a new node n5 is activated. A trunk connection is established between n4 and n5.
2. The agent in n4 reactivates. Once n4 becomes reachable, it can become manageable.
3. The trunk connection between endpoint (n4, t1) and (n3, t3) is activated. As a result, the operational status of (n3, t3) changes from disabled to enabled with partner (n4, t1). An enable event is emitted for (n3, t3) and forwarded to the manager from node n3.
4. When the manager receives the event, it first changes the display of the trunk endpoint (n3, t3) to reflect enabled status. Since the partner of (n3, t3) is (n4, t1), which is currently unmanageable, the manager explores n4 to retrieve its local connectivity information. Given that the agent in n4 is active, it confirms its trunk connection with n3 and uncovers another enabled trunk endpoint (n4, t2) with partner (n5, t1), which is in a new node n5.
5. The manager continues by exploring node n5. Discovery ends after finding no additional enabled trunk endpoints in n5. Upon completion, the topology map in Figure 2 is changed to the one in Figure 7.

Implementation and future extension

The topology management function outlined in this paper has been implemented as an application that runs on top of the NetView for AIX* (Advanced Interactive Executive*) management platform.[7] It uses Common Management Information Protocol (CMIP) over TCP/IP (CMOT)[8] as the protocol for exchanging management information between manager and agents. This design is not restricted to any specific management protocol. However, it does depend on an event-forwarding mechanism as defined in Inter-

national Organization for Standardization (ISO) management standards.[5] Some potential enhancements include:

1. Supporting multiple management gateways for a single manager. Having multiple attachment points into the network will reduce the chances of losing contact between manager and agents.
2. Extending the local connectivity information supported by agents to include information related to physical connectivity, such as local and remote port number, associated with a trunk endpoint. In some environments where a trunk connection between two nodes is actually established through an end-to-end connection of another network, this information would make it possible to display the integrated topology of both networks.

Conclusion

Though polling for agent liveliness information is still needed during event monitoring when unmanageable nodes are detected, the added cost in terms of network traffic is relatively low compared with polling for local connectivity information in polling monitoring. This is true both in terms of the required frequency and the amount of information for each poll. As the network size increases, the advantages of topology monitoring based on this design become even more significant.

*Trademark or registered trademark of International Business Machines Corporation.

Cited references

1. L. Crutcher and A. Lazar, "Management and Control for Giant Gigabit Networks," *IEEE Communications Magazine* 7, No. 6, 62–71 (November 1993).
2. M. Rose, *The Simple Book*, Prentice-Hall Inc., Englewood Cliffs, NJ (1991), p. 76.
3. *ATM User-Network Interface Specification Version 3.1,* ATM Forum, Worldwide Headquarters, 480 San Antonio Road, Suite 100, Mountain View, CA 94040-1219 (1994).
4. *APPNTAM Feature Implementation Guide Version 2 Release 4,* SC31-7050, IBM Corporation (January 1994), pp. 2 and 85; available through IBM branch offices.
5. *Information Technology—Open Systems Interconnection—Event Report Management Function*, CCITT Recommendation X.734 (1992)|ISO/IEC 10164-5:1992, International Telecommunication Union, Geneva, and International Organization for Standardization, Geneva.
6. D. Bertsekas and R. Gallager, *Data Network*, Prentice-Hall, Inc., Englewood Cliffs, NJ (1987), p. 322.
7. *NetView for AIX, Programmer's Guide Version 3*, SC31-6238, IBM Corporation (September 1994); available through IBM branch offices.
8. M. Rose, *ISO Presentation Services on Top of TCP/IP-Based Internets*, RFC 1085, IETF (December 1988); see RFC 1085 on URL http://www.es.net/html-stuff/rfcs.html.

Accepted for publication August 10, 1995.

Wei Chao *Hewlett-Packard Corporation, Video Communications Division, P.O. Box 700713, San Jose, California 95170-0713 (electronic mail: cwchao@vid.hp.com).* Dr. Chao received a Ph.D. degree in electrical engineering from Polytechnic University in 1990. He joined IBM in August 1990 in Networking Systems Architecture, working on the design of network management architectures. From 1992 to March 1994 he was a team leader for the IBM Nways™ Switch Manager. He then worked on management applications for multimedia networking systems. He recently joined the Hewlett-Packard Corporation.

William Tsun *IBM Networking Hardware Division, P.O. Box 12195, Research Triangle Park, North Carolina 27709 (electronic mail: wtsun@vnet.ibm.com).* Mr. Tsun is a staff programmer. He received a B.S. degree in electrical engineering from Brigham Young University in 1987 and an M.S. in computer engineering from Syracuse University in 1992. He joined IBM in August 1987 in Independent Systems Verification, working on test tools for the low-end System/390® processors. From 1992 to 1995, he worked on the IBM Nways Switch Manager.

Reprint Order No. G321-5593.