Supporting Network Management through Declaratively Specified Data Visualizations [1]

Mariano P. Consens
consens@db.toronto.edu

Masum Z. Hasan
zmhasan@db.toronto.edu

Computer Systems Research Institute
University of Toronto
Toronto, Canada M5S 1A1

Abstract

The complexity of managing and controlling large heterogeneous networks requires the availability of management stations equipped with sophisticated tools. A fundamental feature of an advanced network management station is the capability to present to the human manager a comprehensible picture of the relevant scenarios. The tools present at the station must allow the manager specify management functionalities in a graceful manner. But the overwhelming volume and complexity of the information involved in network management poses a major challenge.

In this paper we show how the **Hy**$^+$ *visual database system* fulfills the information presentation and specification requirements of advanced network management stations. A visual database system is capable of manipulating data visualizations through visually expressed queries. The **Hy**$^+$ system provides a uniform framework for *hygraph* based data visualizations, queries, and their results. Visual queries, expressed in the *GraphLog* language, are interpreted as patterns that match existing visualizations and create new ones. A prime example of the functionality supported by the system consists of describing a query that *filters* a large visualization to retain the portion that is of interest for the network manager in the context of a particular task.

Examples of data visualizations that are relevant for network management are the network topology at different levels of abstraction, the presentation of network configuration information, and the display of management information bases and their history traces. Employing a fault management scenario as motivation, we demonstrate how the **Hy**$^+$ system deals with all of the above visualizations and several of their combinations, as well as entirely new ones generated through ad-hoc visual queries. With the aid of this system a network manager can also specify network management functionalities as declarative GraphLog queries, for example, define alerts.

Keywords: Network Management, Management Information Databases, Data Visualization, Visual Database Systems, GraphLog Visual Query Language, **Hy**$^+$ System.

[1] Appears in Proceedings of the IFIP/IEEE Third International Symposium on Integrated Network Management, Elsevier North Holland, 1993.

1 Introduction

Managing a large heterogeneous network is a considerably complex task. The proposed network management standards for TCP/IP based Internets [RM, CFSD, MR] and the ISO-OSI network management standard [ISOa, ISOc, ISOb] are attempts by the standard organizations to reduce the complexity involved. The manager-agent paradigm, where the manager keeps the global picture of the network, has been proposed in both standards. The management station is the control center of the network. One of the vital responsibilities of the manager is to maintain a global database containing all the management related data. The data itself is distributed over the network and accessible by the manager through the agents. Conceptually, we can think of this distributed database as one single database residing at the management station.

The complexity of managing, operating and controlling networks is reduced by systematically imposing different levels of abstraction onto the network. By doing so, the functional management can be performed in a meaningful and structured way. Network objects can be abstracted into higher level entities based on different criteria set by the network manager, for example, geographical sites, such as countries, cities, buildings, floors, rooms etc., or administrative responsibilities within the network, e.g., domains. Network objects can also be abstracted into entities that can be derived by the ISO/OSI protocol layering. Based on the protocol layering we can distinguish between two different views of the topology map of the network: the *logical map* which corresponds to the network layer, and the *physical map* which corresponds to the datalink and physical layers. The network addressing mechanism (e.g., IP) defines a logical or network layer map of the network. Devices like bridges, repeaters and cables, together with the datalink and physical layer protocol objects define a physical layer map. Logical and physical layer maps can be abstracted into maps corresponding to geographical and administrative views. The clustering of network elements according to the views of the network at different levels of abstraction is emphasized in [KMG88], [Val91].

The limitation in the size of the presentation media makes it difficult to visualize the status of the entire network. It might be impossible to squeeze a large and complex network with hundreds and thousands of devices and complex interconnections into one single screen, even if we use very sophisticated layout algorithms. When a problem occurs in the network, it would be helpful to see only the relevant areas of the network. It would be nice to be able to specify what we want in a simple and declarative way, filter out the problem area and visualize it in terms of the topology of the network and the views at different levels of abstraction.

In this paper we describe a database system for network management stations that helps a network manager manipulate not just data, but the associated visualizations as well. With the aid of this system a network manager can also specify network management functions as declarative data manipulation statements, for example, define alerts. The $\mathbf{Hy}^+$ system supports queries involving recursion and incorporates deductive capabilities. The user interface of the system enables network managers and operators to browse and edit the visualizations of databases, queries and query results in a uniform way.

The rest of the paper is organized as follows. An overview of the $\mathbf{Hy}^+$ visual database system is given in Section 2. In Section 3 we describe the structure of an example network management database, and how we query and filter the initial database to produce different visualizations of the network information. We present an application of the proposed technology in the context of a realistic fault management scenario in Section 4. We conclude in Section 5.

2 Overview of the Hy$^+$ System

In this section, we present an overview of the **Hy**$^+$ system developed at the University of Toronto in the context of its application within the CORDS project [SBF$^+$92].

The **Hy**$^+$ system provides extensive support for query visualization, visualizing the input instance, and visualizing the output instance in several different modes. The visualizations manipulated by the system are labeled graphs and *hygraphs*.

Many databases can be naturally viewed as graphs. A relational database can be represented by a directed multigraph having an edge labeled $r(c_1, \ldots, c_k)$ from a node labeled $(a_1, \ldots, a_i)$ to a node labeled $(b_1, \ldots, b_j)$ corresponding to each tuple $(a_1, \ldots, a_i, b_1, \ldots, b_j,\ c_1, \ldots, c_k)$ of each relation r in the database. Consider, for example, the following tuples: *connects(gateway('geol.gw'), ethernet(ether8), ip, csmacd)* and *connects(gateway('geol.gw'), tokenring(tokenring1), ip, tokenring)*. The first tuple stores the following information: gateway *geol.gw* is connected (*connects*) to the ethernet segment *ether8* and protocols IP over CSMA/CD is run over the link. Figure 1 shows the graph representation of these tuples.

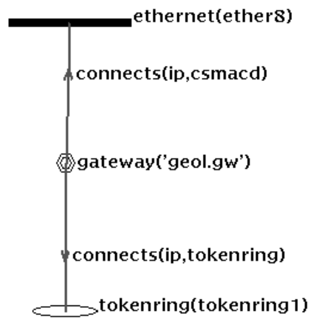

Figure 1: Visualizing tuples.

Hygraphs, defined in [Con92], are a hybrid between Harel's higraphs [Har88] and directed hypergraphs. A hygraph extends the notion of a graph by incorporating *blobs* in addition to edges. A blob relates a containing node with a set of contained nodes, and is diagrammatically represented as a region associated with the container node that encloses the contained nodes. Figure 2 shows a blob where the subnet (*blob node*) contains other network nodes. This containment relationship (*blob relation* or edge) *subnet_contains* is depicted as a box. The outer box may contain more than one blob relation box or other blob nodes deeply nested. Extending the representation to hygraphs allows varying levels of abstraction in the display of hierarchical data (e.g., by interactively hiding and showing blob contents) and provides a flexible mechanism for clustering information.

The **Hy**$^+$ system has browsers with extensive facilities for editing hygraphs (e.g., copy, cut and paste; panning and zooming; executing several layout algorithms; moving hygraphs to and from files; editing of node and edge labels; etc.). These hygraph browsers can present the user with options to start browsing the structure being visualized in hypermedia-like fashion.

Figure 3 contains two **Hy**$^+$ views of the same visualization of an example database used in the remaining of the paper. Window Database 1.1 presents an overall view (with no labels displayed) of a graph containing several hundred objects and relationships. The schema of the database is displayed in a pane of Database 1.1 that lists the colors assigned to the edges corresponding to each

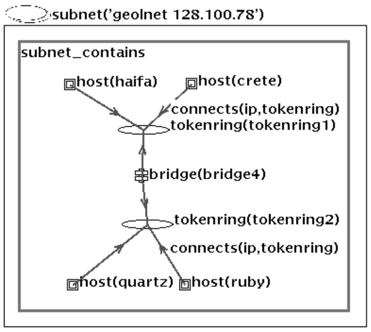

Figure 2: Visualizing a Hygraph.

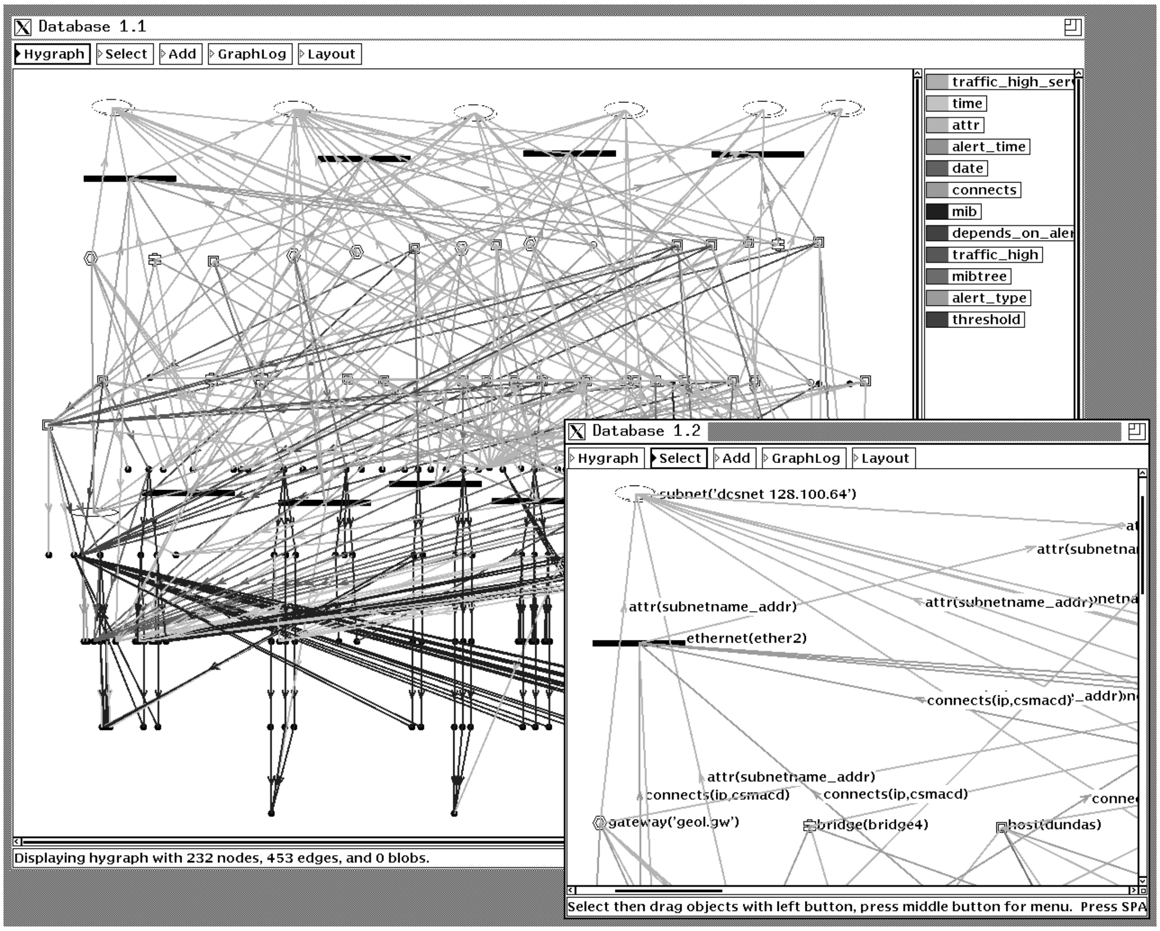

Figure 3: Browsing the example database.

of the relations in the adjacent visualization. Color-coding relations constitutes a very primitive, but visually appealing, example of tailoring the visualizations to the semantics of the applications. Another instance of the customization of the visualizations supported by the system is the selection of different icons for nodes based on the properties of the database objects represented by the nodes. Window Database 1.2 contains a zoomed-in view of the top-left portion of the graph (with labels displayed). The information contained in the example database has been partly acquired using the programs described in [Has92].

The visual queries supported by the **Hy**$^+$ system are expressions of the GraphLog query language [CM90a]. GraphLog queries are graph patterns with nodes labeled by sequences of variables and constants, and whose edges are labeled by *path regular expressions* on relations. The query evaluation process consists of finding in the database all instances of the given pattern and for each such instance performing some action, such as defining new edges, blobs or nodes in the database graph (**Define** mode), or extracting from the database the instances of the pattern (**Show** mode). GraphLog has higher expressive power than SQL; in particular, it can express, with no need for recursion, queries that involve computing transitive closures or similar graph traversal operations. The language is also capable of expressing first order aggregate queries (an example involving maximizing is given later in the paper) as well as aggregation along path traversals (e.g., shortest path queries)[CM90b]. Precise theoretical characterizations of the expressive power of GraphLog and of its computational complexity can be found in the references cited above.

We will now give simple examples of GraphLog queries and show how the visual pattern that a user wants to see can be specified in GraphLog. If the required visualization is not in the database, it is defined using the define mode of the query and added to the database. Later this visualization can be filtered out of the database using the show mode of the query. Consider the visualization of the database shown in Figure 3. This database contains enough information to create different visualizations required for a network management station. For example, we can create visualizations of the topology of the network at different levels of abstractions. To simplify the example we shall work on a small portion of the example database shown in Figure 3, so that the structure of the database is easily comprehensible. The portion of the database is shown in Window Window 1 of Figure 4. We want to create from this database, for example, the visualization shown in Figure 2. Query def in Window 2 of Figure 4 is executed in define mode (note the defineGraphLog box). This query states that any node N1 that has a attr(subnetname_addr) edge to the node subnet('geolnet 128.100.78') should be clustered together and blob edge called subnet_contains be created. The crossed out edge labeled ^attr(type) (representing negation) excludes the gateways from being part of the resulting subnet. Note that the blob subnet_contains has been drawn thick. Any blob or edge that is drawn thick in the define mode is created and added to the database. The resulting subnet blob can be filtered out and visualized through the query show executed in show mode (note the showGraphLog box). The thick edges are the ones we want to see.

The **Hy**$^+$ visual query system is implemented as a front-end that can communicate with multiple back-ends for the actual evaluation of the queries. The front-end, written in Smalltalk, includes the user interface of the system. Currently, there are two back-end query processors; one uses the LDL deductive database system developed at MCC [NS89], while the other is implemented in Prolog [Fuk91].

The system has the ability to invoke external programs that, for instance, browse an object being represented by a node in one of the graphs displayed by the system. This feature, which is part of the overall **Hy**$^+$ goal of providing an open architecture, can be used to support network management stations that require the co-existence of several browsers provided by different tools. The user can navigate, in hypermedia style, through the information contained in management information bases. The **Hy**$^+$ visualizations are used as overviews to locate information and then invoke third-party browsers to display the contents associated with the relevant objects. An important advantage of this approach over a purely navigational one, is the ability to use the convenience and expressive power of GraphLog patterns to retrieve the objects of interest, instead of attempting an often impractical brute force search.

The **Hy**$^+$ system has similarities to the heterogeneous network management systems proposed

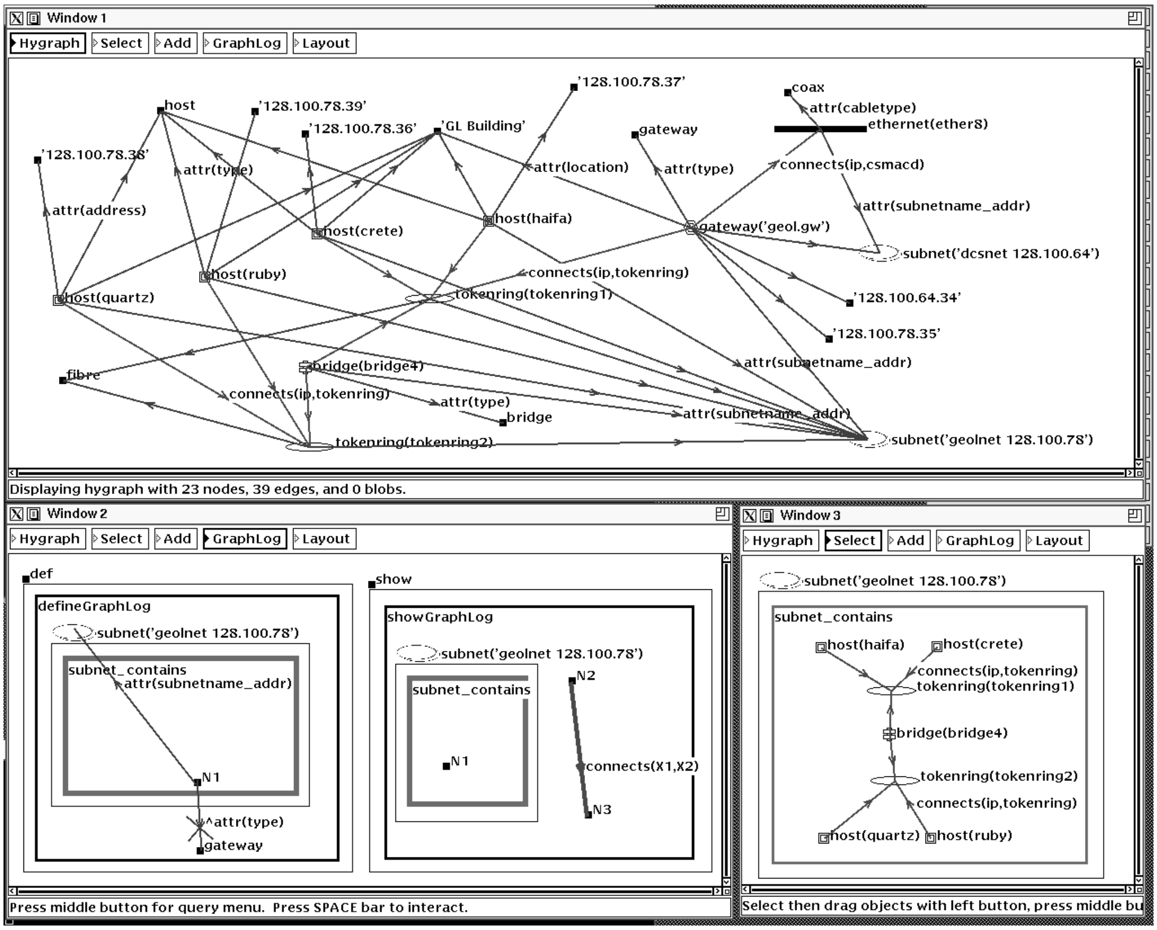

Figure 4: Example GraphLog queries and result.

in [WS89, DSWY91] and to the active database approach in [WSY91], where network management functions are specified as data manipulation statements. These systems differ from $\mathbf{Hy}^+$ in the emphasis given by the latter to the manipulation of database visualizations as an integral part of a toolset for network management stations. Another important difference is the expressive power of the GraphLog query language. SQL-based systems are not capable of expressing transitive closure queries, while these are directly supported in the GraphLog query language in a very intuitive way. Transitive closure queries are necessary when we want to retrieve network management information in terms of the connectivity and topology of the network.

3 Visualizing the Network Database

In this section we describe what constitutes a network database and how we obtain different views of the network using the querying capabilities provided by the $\mathbf{Hy}^+$ system.

A central database that stores all the information associated with the different views of the network and the management subsystems, is a vital part of the network management station. This conceptual global database contains detailed information about the topology of the network (configuration data) and protocol related MIB objects [MR] distributed over the network. The schema for each different MIB (private, enterprise or experimental), as well as historical MIB

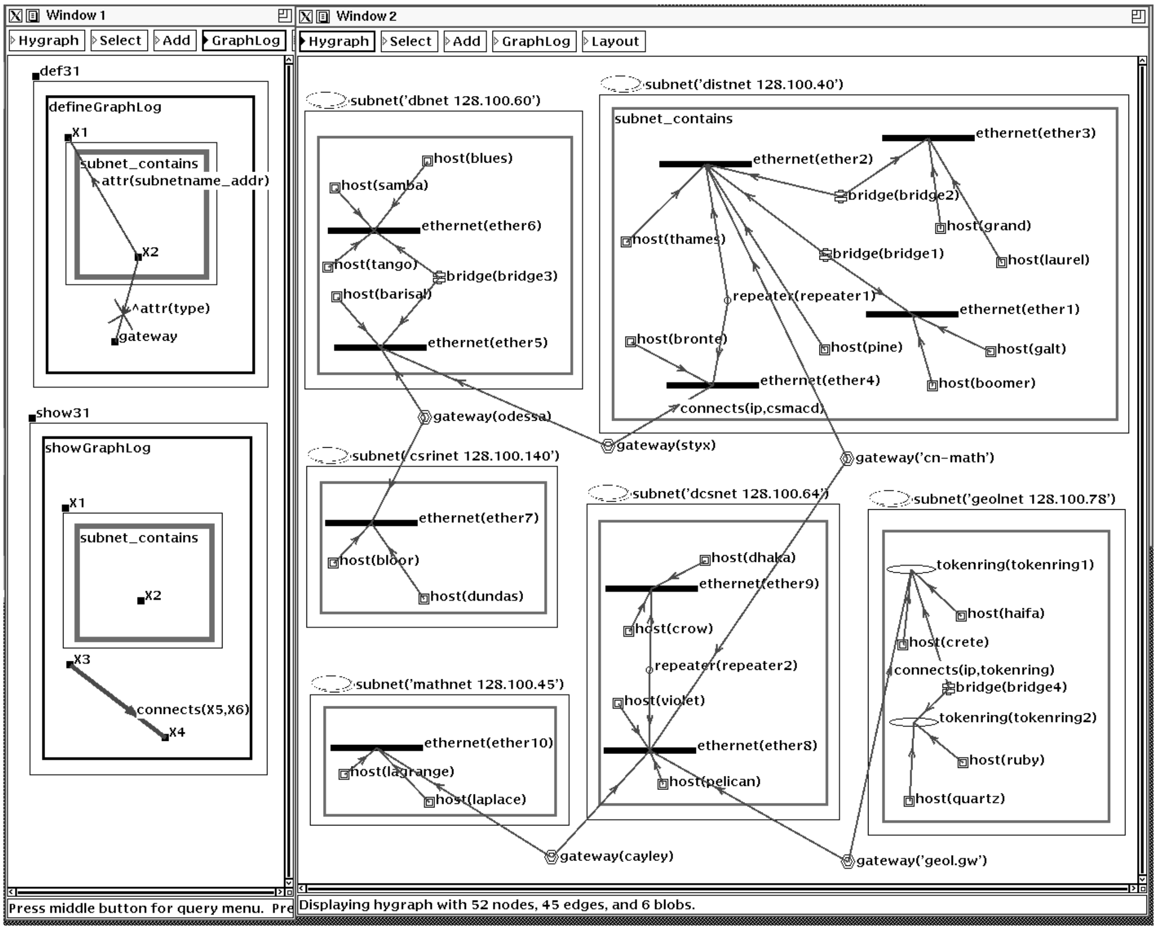

Figure 5: Defining subnets over the physical network topology.

objects, or traces, are also stored in the central database.

Using the sample network database shown in Figure 3, we will now show how the physical and logical maps can be created. By executing the queries shown in Window 1 of Figure 5 on the full database we get the physical topology map shown in Window 2 of the same figure.

The database is transformed using queries to obtain yet another different view of the network: the logical network layer map. Queries def41 and def42 in Window 1 of Figure 6 define a new relationship (nlayer) between the hosts, the gateways and the subnet logical objects. In other words, these queries express that only hosts and gateways are shown at the network layer map. The logical map shown in Window 2 is obtained using the filter query show41 in Window 1.

We can browse the current values and the history traces of MIB objects. For example, the query in Window 1 of Figure 7 displays (in Window 2) the history trace of MIB objects for the host named boomer. This involves filtering the values as well as the timestamps of the traces. The edge labeled with the regular expression mib(_)+ (shown dashed to emphasize that the edge in the query graph matches a path in the input graph) is used to return the entire MIB subtree for boomer.

Administrative and geographical views of the network can be created and displayed in a similar way.

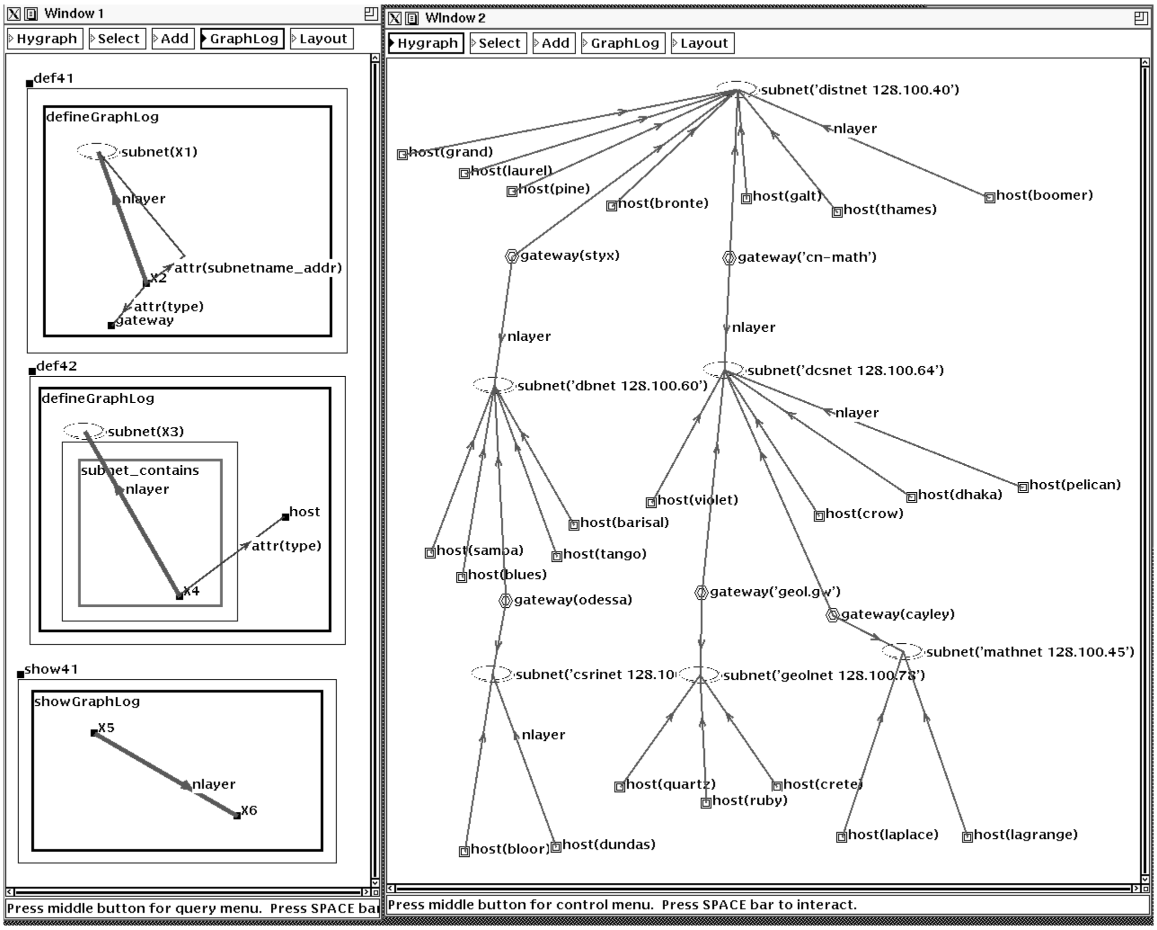

Figure 6: Defining and displaying the logical network layer map.

4 Case Study: Fault Management

In this section we describe how the **Hy$^+$** system can be used in the context of a fault management scenario. The following detailed example is meant to emphasize, in the context of one typical network management activity, how the use of powerful and flexible visualization tools can contribute to the understanding of the complex information required for successfully completing the task at hand.

Fault management is a complex task. Due to the layering structure of protocol software, a fault caused at one level may show up at other levels. Additionally, the combination of protocol software incorporating mechanisms to recover from problems together with the hiding of lower level problems to the higher levels, contributes to make fault analysis and diagnosis difficult. For example, a connection failure or a long response time can be caused by congestion or it can be the result of a hardware problem. This is not to say that protocol software should not try to recover, but that fault analysis and diagnosis requires sophisticated tools. It is quite helpful to visualize the problem area and to be able to deduce possible causes by exploiting the capabilities of a powerful database querying mechanism combined with an inference engine.

To find out the exact cause of a problem we need to proceed in a structured manner. The fault diagnosis process incorporates the following steps: observing the symptoms, developing a

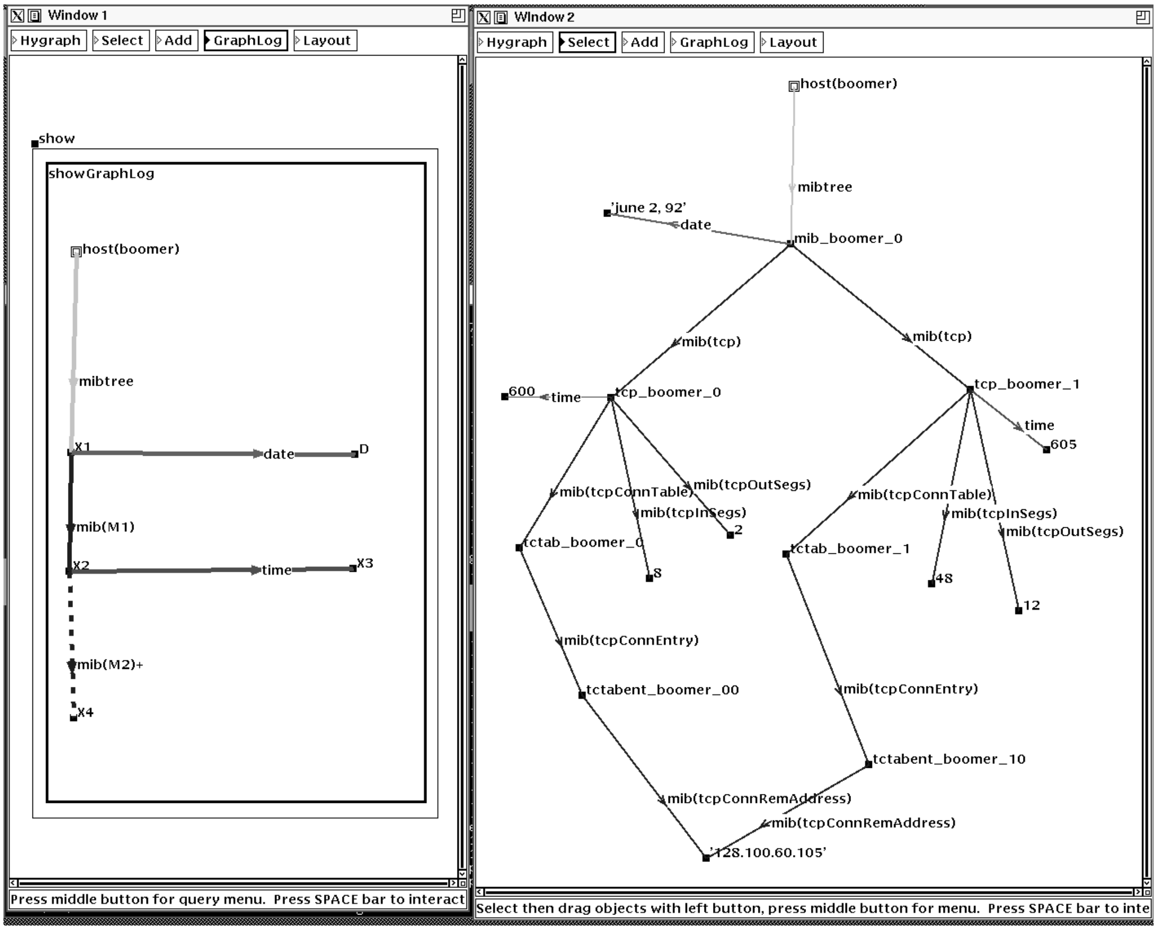

Figure 7: History trace of MIB objects for boomer.

hypothesis on what could be the cause of a symptom, testing the hypothesis, and if the hypothesis fails, iterating the above steps until a conclusion is reached.

4.1 Defining and Observing Problem Symptoms

Before we can perform fault analysis and diagnosis, we have to find out what kind of network behavior we can consider as problem symptoms. Defining potential problem symptoms is very difficult without a priori knowledge of the behavior of the network. Symptoms can be defined in terms of statistics or the expected behavior of the network. In [GL91], it is observed that a temporal cycle exists in the traffic distribution of a network. Therefore, a suitable approach consists of defining problem symptoms in terms of traffic patterns. For example, during high traffic periods, it is natural that network accesses are slowed down. But if the delays reach a certain threshold (the threshold being a function of period and traffic pattern) or the utilization of the network drops during a high traffic period, then this may indicate a possible problem symptom. Henceforth, we require access to statistics about the behavior of the network (e.g., utilization, traffic patterns, etc.).

The information about high volumes of traffic can be obtained by monitoring interesting points of the network. Traffic patterns among individual hosts can be established by monitoring the

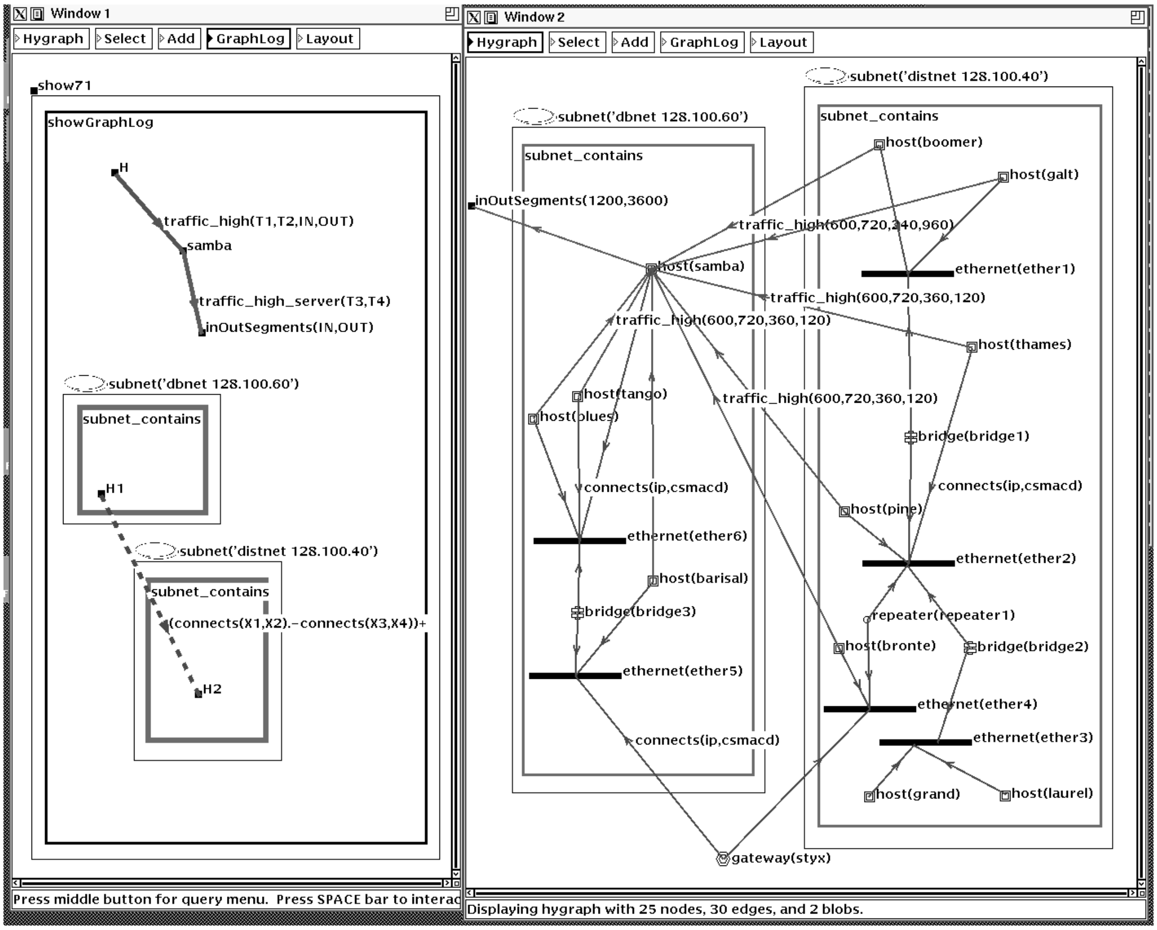

Figure 8: Traffic information displayed against the topology map.

tcpInSegs and tcpOutSegs TCP MIB variables, assuming that the only activity that the clients perform is accessing the server by establishing TCP links. Current SNMP TCP MIB variables do not provide us with traffic information on a per TCP link basis. However, given SNMP's widespread usage and simplicity, we will use SNMP MIB objects in our example. Traffic patterns can also be established using the procedure mentioned in [GL91].

The traffic information between two subnets (dbnet and distnet) taken from the example network in Figure 3) is shown against the topology map in Figure 8 The server samba located in the subnet dbnet is accessed by clients, in the same as well as in other subnets. The edge labeled traffic_high_server(600,720) from the node labeled samba to the node labeled inOutSegments(1200,3600) tells the expected number of incoming and outgoing TCP segments from the server samba between 10 a.m (600 minutes from midnight) and 12 p.m. (720 minutes from midnight). Other edges labeled traffic_high(Time1, Time2, In, Out) between individual hosts and samba indicate the (In, Out) expected values of high volumes of traffic between the server and the hosts during the periods from Time1 to Time2.

The query show71 in Window 1 of Figure 8 is used to filter the physical map to display the contents of the two subnets of interest, 'dbnet 128.100.60' and 'distnet 128.100.40', together with the traffic_high_server and traffic_high edges. In this way, high traffic spots are visualized directly against the topology, instead of presenting them as tables or rules. If we were dealing with a large

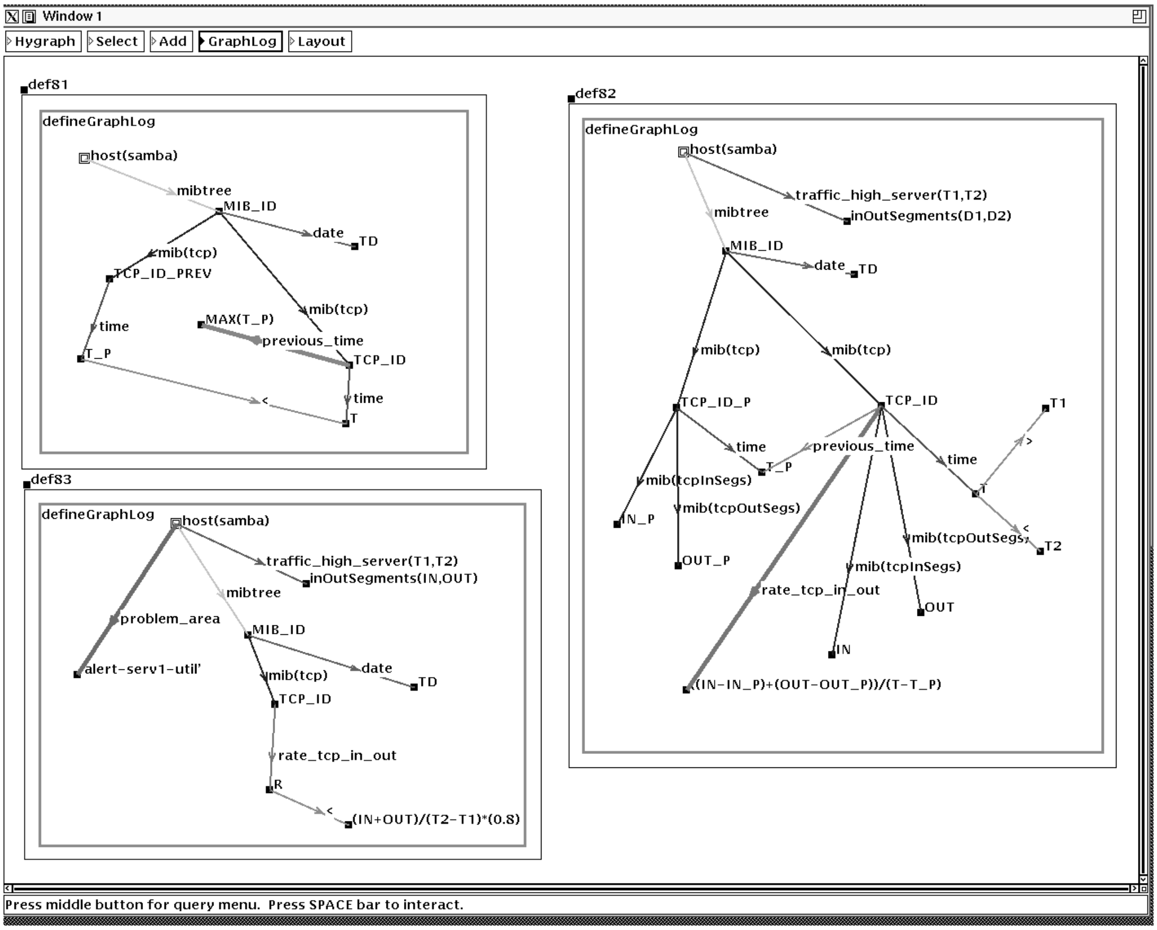

Figure 9: Defining an alert for possible problem symptoms.

network, filtering out all the subnets we are interested in, produces a more understandable and less cluttered map. The resulting display (Window 2) helps the human network manager to understand the traffic pattern directly in terms of the topology of the network.

Now, assume that we monitor the utilization of the server samba. For this purpose we poll the tcpInSegs and tcpOutSegs TCP objects of the server. The values returned are kept in the database as history traces.

An alert can be generated if the server utilization falls below a certain threshold in a particular period of time, which in this case coincides with an expected high traffic period. If the alert goes off, it indicates a possible problem symptom. The queries in Figure 9 define the alert alert-serv1-util. Query define81 finds out the time (pointed by the distinguished edge previous_time) when the previous poll of samba was taken. Query define82 computes the rate of tcpInSegs and tcpOutSegs traffic in the latest poll interval during high traffic time. Query define83 checks whether the rate computed in the previous query falls 80% below the expected traffic kept on the traffic_high_server edge of samba. If the condition is satisfied, an edge called problem-area is created from the node labeled samba to the node labeled with the name of the alert.

The queries define82 and define83 should be interpreted as two (active database) rules (see [Day88, BM91, Cer92]) that are triggered on non-empty retrieval events and perform the action of adding the distinguished edges. These queries defining an alert shows how relevant network management functions can be specified as declarative GraphLog queries. We are considering the addition of features to the system that address temporal monitoring of database changes similar to the ones proposed in [WSY91].

4.2 Diagnosing a Fault

Let us consider that the alert named alert-serv1-util (the one we defined in the previous subsection) has gone off and the problem area is the server samba.

Our first hypothesis is that the cause is *congestion* at nearby gateways (routers). It is possible that other sources of traffic are placing an unexpected demand on the gateways through which traffic to the server passes. As a consequence the packets destined for the server are being excessively discarded.

To prove this hypothesis we check the gateways through which packets addressed to the server are likely to flow. While investigating for congestion we are interested in focusing on the portion of the logical map of the network that is related to the alert alert-serv1-util. For each host that uses the server samba, and is therefore affected by the alert alert-serv1-util, an edge called depends-on-alert is created from the host to the node containing the value alert-serv1-util.

When we are presented with the logical map, we can easily identify the gateways that should be checked for congestion. In addition, the gateways can be identified automatically by querying and highlighting them. The queries and the resulting network layer map are shown in Figure 10, with the potentially congested gateway styx highlighted. Note that focusing on the physical map is not necessary at this stage, it might complicate the analysis procedure.

Query define91 in Window 1 of Figure 10 creates a direct edge called snc(gateway(G)) between two subnets S1 and S2 with the gateway information carried along the edge. Query define92 finds the *simple paths* [MW89] between high traffic nodes and highlights the gateways traversed along the paths.

As a result of the filter query show91 in Window 1, window Window 2 shows the subset of the logical map of the network that depends on the alert alert-serv1-util, with a loop labeled highl around the gateways we are interested in highlighting. Note that the depends_on_alert edges are not shown as thick in the query show91, indicating that we are not interested in showing these edges in the resulting visualization, thus reducing cluttering. Filtering out the problem area through a powerful querying mechanism as shown above, definitely helps.

After identifying the gateways we check for congestion. The query define93 in Window 3 checks the current value of the ifOutDiscards variable of the interfaces MIB object for the highlighted gateways. If the value exceeds the specified threshold, then the distinguished edge called congestion is created.

If no congestion is detected at the gateway styx, we have to come up with a second hypothesis. For now, assume that there is indeed congestion at the gateway styx. The source of the congestion could be the clients of the server samba, or it could be originated somewhere else. To find out the source of the congestion we check whether the hosts using the server are placing the expected traffic to the server samba. As discussed earlier, the expected traffic patterns between the server and its clients have been established before. We can use a query to check whether the sum of the current values of tcpOutSegs for all the clients is 20% less or more than the sum of the expected values that are kept on the traffic_high edges.

If the answer to the above query is no, we may hypothesize that other sources are causing the congestion, since the clients of samba are generating traffic as expected. We could continue tracing the source of the congestion by looking at the other hosts in the network. This can be done by querying the current tcpConnTable for those hosts and creating visualizations to help locate the

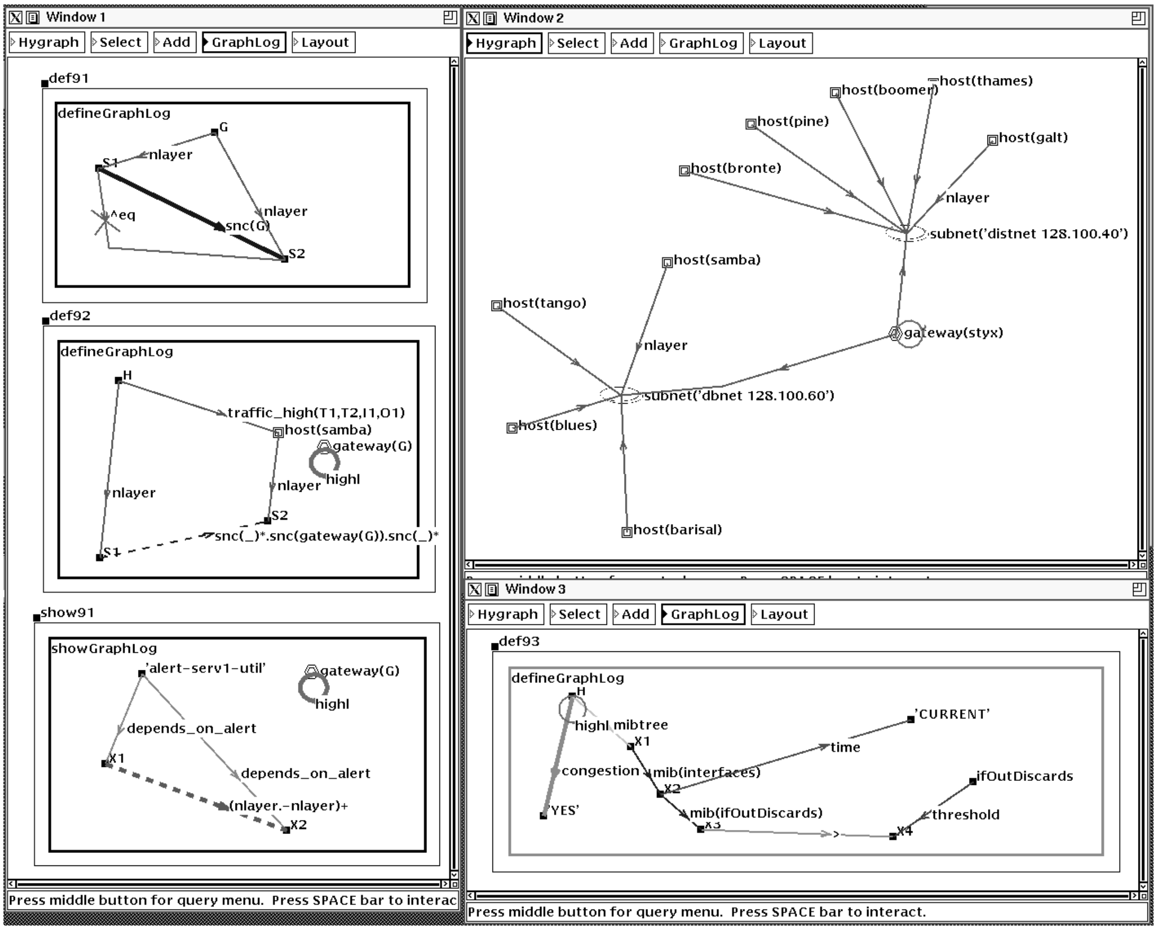

Figure 10: Highlighting congested gateways in the logical map.

TCP links that have routes through the congested gateway styx.

Assume that no congestion was detected, but it was found that the clients of samba are not generating the expected traffic to the server. Our second hypothesis is that the cause of the alert is due to a *hardware problem*. In this case the map is brought up showing physical details like repeaters, bridges, datalink layer protocols (CSMA/CD, tokenring), cable types (coax, fiber), and so on. Before looking for the physical causes of the problem, we superimpose the currently active TCP links on the portion of the physical map that depends on the alert.

Query define101 in Window 1 of Figure 11 can be interpreted as follows: create an edge called tcp_link between the hosts H that depend on the alert alert-serv1-util and the server samba, if the IP address found in the tcpConnRemAddress of H's TCP connection table matches the address of samba (which is stored in the attribute attr(address) of the configuration database). Query show101 in Window 1 produces, as a result, the visualization in window Window 2 showing the portion of the physical topology map that depends on the alert, together with the previously defined tcp_link edges superimposed on the map.

The visualization of TCP links allows us to pin-point the problem area at the portion of the network beyond ether4. The problem could be at the repeater repeater1, or at the ethernet segments ether2. This follows from noticing that no TCP connection originates at ether1 or ether2. The SNMP manager could not access the current TCP objects from that part of the network.

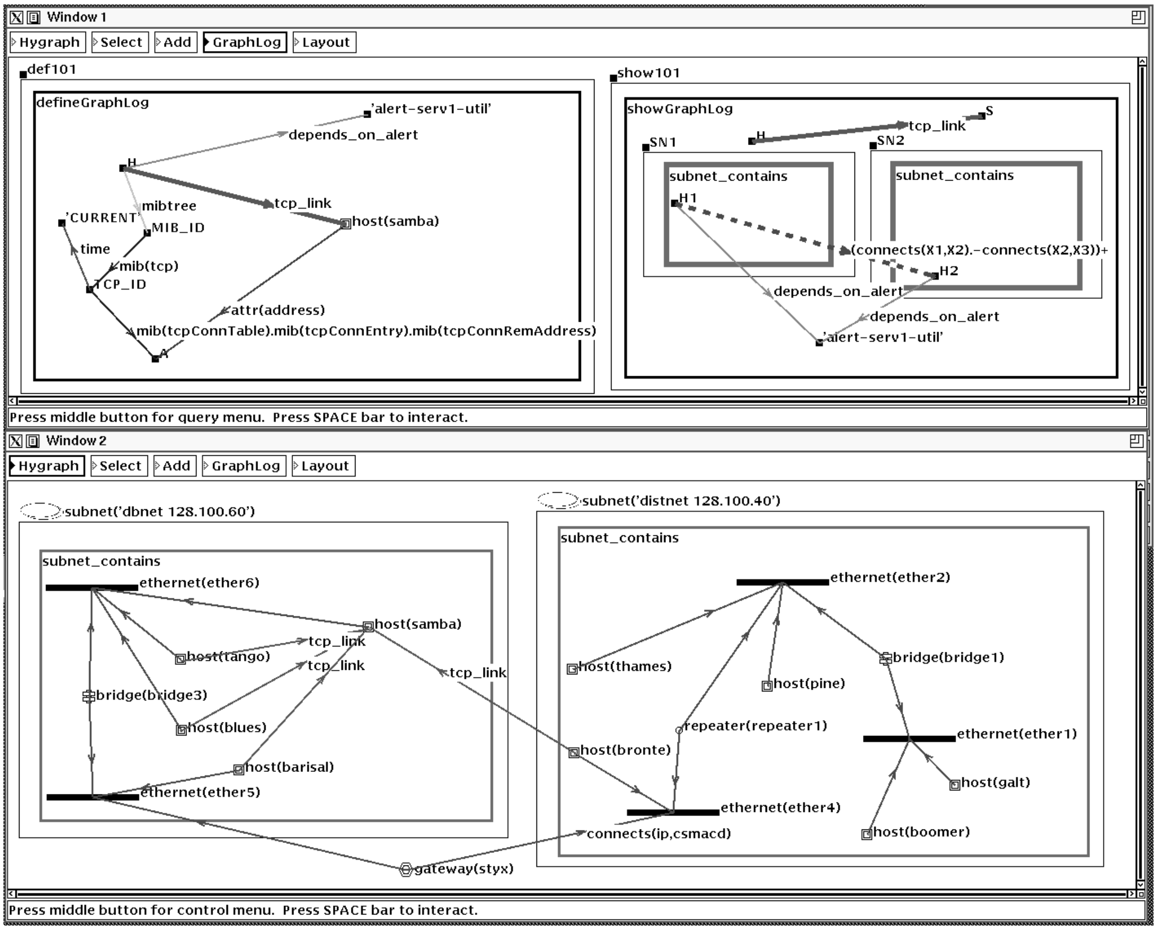

Figure 11: TCP links superimposed on the physical topology map.

The example shows how, when isolating faults in a network, it is advantageous to view network maps at different abstract levels, while proceeding in a structured manner to pin-point the problem area. If we were limited to work only with the entire physical map of a large network, the complexity of the fault analysis and diagnosis procedure would increase significantly. This is just one particular instance that illustrates how the capabilities of the **Hy**$^+$ system can be exploited by human network managers.

5 Conclusions

The **Hy**$^+$ visual database system described in the paper constitutes a powerful and flexible tool for visualizing the information required to manage large heterogeneous networks. Below we summarize the main characteristics of the system.

- A consistent hygraph-based framework for dealing with a wide variety of network related data visualizations is provided.
- Data visualizations are manipulated by means of visually expressed queries. Visual queries, expressed in the GraphLog query language, are interpreted as patterns matched against ex-

isting visualizations to either create new visualizations or to filter existing visualizations to retain only their relevant portions.

- The expressive power of the GraphLog query language is adequate for network management applications. Several examples of transitive closure queries were presented and extensively motivated in the previous sections (and these queries cannot be expressed in SQL).

An extensive fault management example has been used to illustrate how the capabilities of the system can be exploited by human network managers.

Future research is focused on two areas. At the back-end level we are concerned with the incorporation of heterogeneous and active database features to support the integration of network management services. The **Hy**$^+$ approach of translating GraphLog patterns into an underlying target database language constitutes a flexible architecture for the incorporation of these additions (like the contemplated addition of a translation incorporating SNMP queries). At the front-end, we are extending the presentation capabilities of the **Hy**$^+$ system to incorporate several domain specific visualizations and corresponding layout algorithms that profit from knowledge of the application domain.

Acknowledgments

We would like to thank Alberto Mendelzon for his fruitful suggestions and support. We are also indebted to the other members of the **Hy**$^+$ development group: Frank Eigler, Yanni Jew, Emanuel Noik, Dimitra Vista, and Annie Yeung. The CORDS project at the IBM Canada Centre for Advanced Studies, headed by Jacob Slonim, provided a stimulating research environment.

The first author was supported by an IBM Canada Research Fellowship and by PEDECIBA (United Nations Program for the Development of Basic Sciences in Uruguay).

References

[BM91] Catriel Beeri and Tova Milo. A model for active object oriented database. In *Proceedings of the 17th International Conference on Very Large Data Bases*, pages 337–349, September 1991.

[Cer92] Stefano Ceri. A declarative approach to active databases. In *Proceedings of the Data Engineering*, pages 452–456, 1992.

[CFSD] J. D. Case, M. S. Fedor, M. L. Schoffstal, and J. R. Davin. A simple network management protocol (SNMP). RFC 1157, SNMP Research, Performance Systems International, MIT Laboratory for Computer Science, May 1990.

[CM90a] Mariano Consens and Alberto Mendelzon. GraphLog: a visual formalism for real life recursion. In *Proceedings of the Ninth ACM SIGACT-SIGMOD Symposium on Principles of Database Systems*, pages 404–416, 1990.

[CM90b] Mariano Consens and Alberto Mendelzon. Low complexity aggregation in GraphLog and Datalog. In *Proceedings of the Third International Conference on Database Theory, Lecture Notes in Computer Science Nr. 470*, pages 379–394. Springer-Verlag, 1990. A revised version has been accepted for publication in TCS.

[Con92] Mariano P. Consens. Visual manipulations of database visualizations. PhD Thesis Proposal, 1992.

[Day88] U. Dayal. Active database management systems. In *Proceedings of the third International Conference on Management of Data*, pages 225–234, June 1988.

[DSWY91] A. Dupuy, S. Sengupta, O. Wolfson, and Y. Yemini. NETMATE: A network management environment. *IEEE Network Magazine*, pages 35–43, October 1991.

[Fuk91] Milan Fukar. Translating GraphLog into Prolog. Technical report, Center for Advanced Studies IBM Canada Limited, October 1991.

[GL91] Mario Gerla and Ying-Dar Lin. Network management using database discovery tools. *IEEE proceedings on Distributed Computing Systems 1991*, pages 378–385, 1991.

[Har88] David Harel. On visual formalisms. *Communications of the ACM*, 31(5):514–530, 1988.

[Has92] Masum Z. Hasan. Topology and device discovery in an Internet environment. Draft Report, 1992.

[ISOa] ISO 7498, information processing systems, open systems interconnection, basic reference model part 4, OSI management framework. International Organization for Standardization, October 1986.

[ISOb] ISO 9596, information processing systems, open systems interconnection, management information protocol specification, common management information protocol. International Organization for Standardization, November 1990.

[ISOc] ISO/IEC DIS 10165-1, 10165-2 and 10165-4, information processing systems, open systems interconnection, structure of management information, parts 1, 2 and 4. International Organization for Standardization, 1990.

[KMG88] G. Kar, B. Madden, and R. S. Gilbert. Heuristic layout algorithms for network management presentation services. *IEEE Network Magazine*, pages 29–36, November 1988.

[MR] K. McCloghrie and M. T. Rose. Management information base for network management of TCP/IP based internets - MIB-II. RFC 1213, Hughes LAN Systems, Performance Systems International, March, 1991.

[MW89] A.O. Mendelzon and P.T. Wood. Finding regular simple paths in graph databases. In *Proc. 15th International Conference on Very Large Data Bases*, pages 185–194, 1989.

[NS89] Shamim Naqvi and Tsur Shalom. *A logical language for data and knowledge bases.* Computer Science Press, New York, 1989.

[RM] M. Rose and K. McCloghrie. Structure and indentification of management information for TCP/IP-based internets. RFC 1155, Performance Systems International, Hughes LAN Systems, May, 1990.

[SBF+92] J. Slonim, M. Bauer, P. Finnigan, P. Larson, A. Mendelzon, R. McBride, T. Teorey, Y. Yemini, and S. Yemini. Distributed programming environments: Challenges. In *Proceedings of the IFIP TC6/WG6.4 International Workshop on Open Distributed Processing.* North-Holland, October 1992.

[Val91] R. F. Valta. Design concepts for a global network management database. *Integrated Network Management, II*, pages 777–788, 1991.

[WS89] U. S. Warrier and Carl A. Sunshine. A platform for heterogeneous interconnection network management. In B. Meandzija and J. Westcott, editors, *Integrated Network Management, I, IFIP*, chapter 1, pages 13–24. North-Holland, 1989.

[WSY91] O. Wolfson, S. Sengupta, and Y. Yemini. Managing communication networks by monitoring databases. *IEEE Transactions on Software Engineering*, 17(9):944–953, September 1991.