In *4th International Symposium on Artificial Intelligence*, pp. 371-377, 1991.

A Connectionist Method for Adaptive Real-Time Network Routing

Kelly C. McDonald
Tony R. Martinez
Douglas M. Campbell

Department of Computer Science
Brigham Young University, Provo, Utah 84602

Abstract

This paper proposes a connectionist mechanism to support adaptive real-time routing in computer networks. In particular, an Adaptive Self-Organizing Concurrent System (ASOCS) model is used as the basic network router. ASOCS are connectionist models which achieve learning and processing in a parallel and self-organizing fashion. By exploiting parallel processing the ASOCS network router addresses the increased speed and complexity in computer networks. By using the ASOCS adaptive learning paradigm, a network router can utilize more flexible routing algorithms.

1. Introduction

As networks increase in speed and routing complexity, new types of processors are required to compute routing decisions. In addition, network routers must accommodate and adapt to rapid changes in network topologies.

This paper proposes a connectionist mechanism to support adaptive real-time routing in computer networks. In particular, an Adaptive Self-Organizing Concurrent System (ASOCS) model is used as the basic network router. ASOCS [Martinez, 1986;1991] is class of models which seek computational efficiency as highly parallel and adaptive connectionist logic systems that map boolean input values to output values. An ASOCS can infer routing decisions from routing rules that have been expressed as boolean input variables.

We propose an ASOCS as the central computing element in a network router. Because of the parallel and adaptive nature of ASOCS, such a router addresses the problems of increasing speed, complexity, and change.

In section 2 we review current routing requirements and problems associated with current routing solutions. In section 3 we present the ASOCS model. In section 4 we propose a router with an ASOCS unit as its central computing mechanism. Section 5 demonstrates that this architecture can compute the necessary routing decisions in an efficient fashion.

2.0 Network Routing

Computer networks typically exchange data between computer systems in small units called packets, transmitted through a communications facility consisting of communications lines, local networks, and network routers. A network router switches a packet between network segments to deliver the packet to its final destination. A router typically uses a general purpose micro-processor to compute the route for a data packet based on the destination address in the packet's header.

2.1 The Need for Forwarding Performance

Communications transmission rates continue to increase. As little as ten years ago, much of data communications traffic utilized transmission rates below 10,000 bits per second (Tanenbaum, 1988). Todays local area networks typically operate at 10 megabits per second (Ethernet), with 100 megabit per second equipment (FDDI) coming into widespread use. Wide area networks currently operate at speeds of one megabit per second, with recent availability of 45 megabits per second transmission rates. The wide bandwidth of optical fiber cabling promises gigabit transmission speeds in the future (Stallings, 1987).

Routers that operate with Ethernet networks switch packets at less than the theoretical packet limit of 14,880 packets per second (Benhamou, 1984). This theoretical limit, for minimum size packets, means that a packet can arrive at the router interface every 51 microseconds. Thus, a packet must be saved in memory for more lengthy processing.

The 100 megabit FDDI networks increase the theoretical packet limit of the network segment to over 400,000 packets per second (Franceus, 1990). This means that minimum length packets can arrive at the router interface on an FDDI network every 6.4 microseconds. A 20 MHZ Intel 386 routing processor could only execute 15-20 instructions in that time (Intel, 1988). A processing element is needed that can make routing decisions while the packet is being received without resorting to memory buffering. Such an element would remove the memory buffering delays.

2.2 The Problem of Complexity

Network routers typically base their routing decisions on the destination address embedded in the arriving data packet. The router compares this destination address against an internal database of routing information. The internal database typically contains static entries from the network administrator and dynamic entries from routing information exchanged through a routing protocol. If the destination address is in the database, then the packet is either sent directly to its destination or forwarded to another router. If the destination address is not found, then the packet is either discarded or delivered to a default router that contains a more complete database. To keep search time short, the number of entries in the database is limited.

Recently, it has been proposed that the destination address is insufficient to properly utilize networking facilities. "Type-of-Service" routing has been proposed to augment routing decisions (Proteon, 1989). Type-of-Service is a rule based routing mechanism whereby boolean statements govern the decisions for packet routing. For example, the packet transit time may be just as important as the ability to deliver the packet to its destination.

Rules dealing with packet security, content, or size can influence routing decisions. A processing element that can process many routing rules in a parallel fashion permits the implementation of new routing paradigms.

2.3 The Need for Adaptability

The sheer size of todays computer networks creates difficulties. Figure 1 shows alternate links outside of a strict hierarchical relationship between networks. These alternate links, necessary for improved performance and fault tolerance, add to the complexity of network routing. Currently, routers exchange this information through special protocols to synchronize their routing databases.

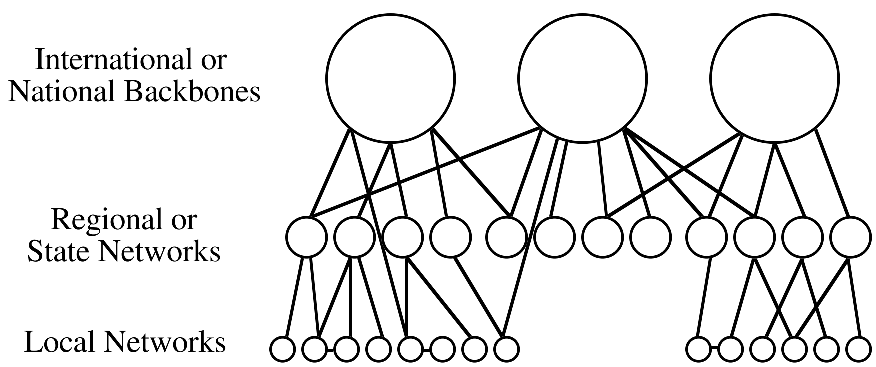

Figure 1 - Model of Internet

The routing information for the current Internet consists of a large distributed database (Comer, 1988). As the Internet grows, more human intervention is necessary to reflect network changes in the routing database. These changes increase in frequency as the network grows. Human error during updates are one of the major causes of inconsistencies in the routing database. Inconsistencies cause routing loops and prevent reliable operation. A processing element that can easily adapt to changes in the network would help alleviate human intervention for changes to the routing database.

An address recognition technique has recently been used by bridges in local area networks (Backes, 1988) to address this need for adaptability. The bridge "learns" the location of stations by observing the source address as a data packet arrives. In this way a bridge creates its own routing database that describes the network topology.

Such adaptive learning algorithms have not been applied outside local area network bridging because they are unable to deal with redundant paths between routers. An extension of the bridge learning algorithm for more diverse network topologies has been proposed (Sincoskie, 1988). A network router is needed that can deal with complex routing or bridging algorithms. We introduce an ASOCS implementation of a network router in section 6 that uses such extended bridge algorithms.

3.0 A Brief Description of ASOCS

ASOCS is a parallel computational model, utilizing digital logic, that accepts a vector of boolean inputs and maps them to a vector of boolean outputs, based on a previously learned input to output mapping (Martinez, 1986:1991).

3.1 The Instance Set

The boolean mapping that is learned by the ASOCS unit is called the "instance". An instance is a group of input variables and an associated output variable. The input variables are written on the left side and the output variable is written on the right side of an inference rule. For example:

(1) $A\ B'\ C\ E \rightarrow Z_1$

(2) $B\ C'\ F\ G' \rightarrow Z_2'$

Instance (1) says that if A, C, and E are true and B is false, then Z_1 must become true. Instance (2) says that if B and F are true and C and G are false, then Z_2 becomes false.

Instances are partial in nature; they are not equivalences or implications. In particular we cannot infer from:

(3) $A \rightarrow Z_1$

that when A is false, Z_1 must also be false.

Instances can be mutually inconsistent. The instances $A \rightarrow Z_1$ and $A \rightarrow Z_1'$ are inconsistent; they output contradicting values when A is true. By accepting instances incrementally, ASOCS can rectify inconsistencies. The ASOCS unit gives precedence to the portion of a new instance that conflicts with any old instance. A consistent group of instances is called an "instance set".

3.2 Instance Combination

ASOCS instances support the three boolean operations: "and", "or", and "negate".

The "and" operator is supported by combining variables in a single instance. For example, the following instance "ands" the variables A, B, C', and D:

(4) $A\ B\ C'\ D \rightarrow Z_1$

This instance is read, "A and B and not C and D implies Z_1."

The "or" operation is supported by creating an instance with each variable represented singly. Therefore, A "or" B implies Z_1 is supported by creating the two instances:

(5) $A \rightarrow Z_1$

(6) $B \rightarrow Z_1$

The "negate" operation of a variable is supported by creating an instance and complementing the variable:

(7) $A \rightarrow Z_1$

(8) $A' \rightarrow Z_2$

One powerful feature of ASOCS is its ability to "generalize" an output with an incomplete set of input variables. For example, if instances have been learned that imply that Z_1 should be asserted true, the ASOCS system may also assert Z_1 for future input that is similar, though not the same as a learned instance.

3.3 Operation of an ASOCS Unit

There are a number of defined ASOCS models (Martinez, 1986:1991), each differing in their architecture and learning algorithms. An ASOCS unit is composed of simple logic gates that performs the mapping of input to output. For example, if an application requires a correspondence of the boolean input "1011101" with the boolean output "01101", an ASOCS model can make this arbitrary association. ASOCS adapts its internal logic array when new instances are presented for learning so that it can reproduce the output and still accommodate previous instances. Unlike a random access memory, an ASOCS unit may recognize patterns in a distributed manner. The instances need not be explicitly stored in a memory cell; instead the pattern association can be stored in the structure and interconnections between the simple logic nodes. ASOCS combines the flexibility of more traditional software based pattern matching algorithms with the speed and robustness of digital logic circuitry. Figure 2 shows a block diagram of an ASOCS unit. The vector (X_1-X_n) is the input to the system and the vector (Z_1-Z_m) is the output from the system.

An ASOCS unit has two modes of operation: adaptation and execution. In adaptation (learning) mode, boolean patterns of input values (incoming instances) are presented with the desired output values. The ASOCS unit adapts its internal logic to reproduce the desired output mapping when the same or similar patterns of input values are presented during the execution mode. In execution mode, the ASOCS unit acts like a parallel digital logic circuit.

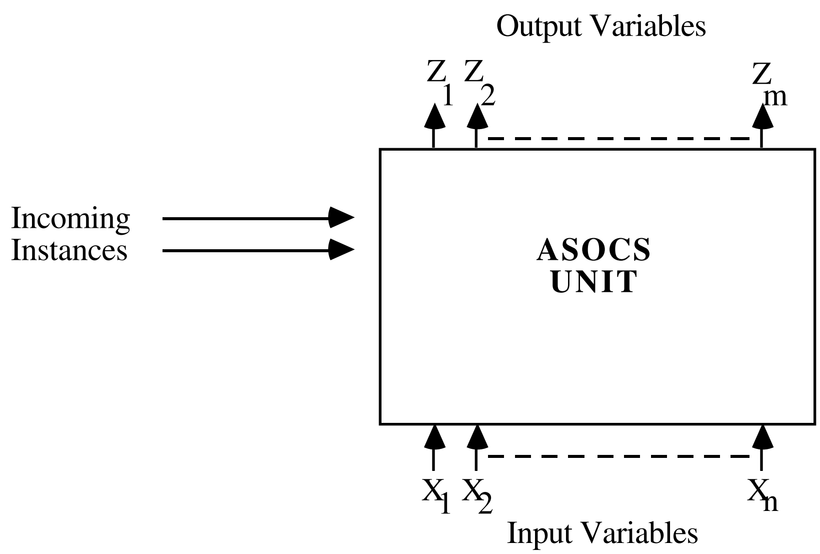

Figure 2 - Black Box ASOCS Unit

A set of input values is presented as an input vector to the ASOCS unit and an output pattern is produced as an output vector consistent with the values that were previously learned during the adaptation phase. The time required for output presentation depends on the number of instances learned by the ASOCS system. Typically, the time delay required to generate the output variable is O(log(n)), where n represents the number of nodes utilized to learn the instance set in the ASOCS system. The reader is referred to (Martinez, 1986) for an in-depth discussion of the origin of the instance set and the ASOCS model.

4.0 Network Routing with the Instance Set

Figure 3 represents a model of a network router, receiving and transmitting data packets on lines 1 through line n. Routing operation is controlled by routing parameters that govern the re-transmission of packets as they are received. Routing parameters may be static or dynamic. Static parameters affect all packets and are input to the router on the control input line. Dynamic parameters affect a particular set of packets. Dynamic parameters are exchanged between routers in data packets, using a special routing protocol.

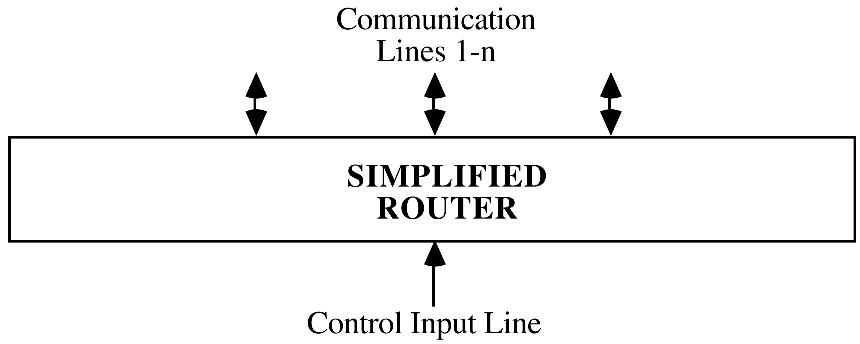

Figure 3 - Block Diagram of Network Router

In an ASOCS router, an instance represents a routing rule. The input variables of the ASOCS instance represent boolean routing parameters and the output variable represents eligibility for transmission on a given communication line.

4.1 Routing Rules as Instance Variables

Sincoskie and Cotton (Sincoskie, 1988) have extended the learning bridge algorithm to networks with packet cycles or loops. As in the original learning bridge algorithm (Backes, 1988), a router observes the source address in each incoming packet and remembers on which interface it arrived. If the address is later observed as a destination address, then the router forward the packet on the interface on which the address was earlier observed as a source address.

Instances, as abstract representations of boolean rules represent a variety of real-world information. In this paper we let the input variables of an instance (the left side) represent the various bits of the network address. For example, in a network of four bit network addresses (0000-1111), the instance variables A through D represent the address. Thus, network address: "1001" is represented as:

$$A\ B'\ C'\ D \tag{1}$$

in an instance.

If a packet arrived on communications line 1, and contained "1001" as the source address, then we output Z_1 (corresponding to line 1) to indicate that this is the proper return path to "1001". Therefore, the following instance captures the routing information to route subsequent network traffic to "1001":

$$A\ B'\ C'\ D \rightarrow Z_1 \tag{2}$$

4.2 Application to the Internet

This instance set definition can represent routing information for more complex networks than the simple router of figure 3. The Internet currently utilizes a thirty- two bit address of which up to twenty-four bits can be designated as a network address. The remaining bits assign host addresses. It is not necessary to have a route to every host address; usually, a designated router routes all packets to hosts within a given network address. To represent Internet network addresses as instance variables we require:

Class A network addresses - 8 variables. (A-H)
Class B network addresses - 16 variables. (A-P)
Class C network addresses - 24 variables. (A-X)

An instance set can have instances representing any combination of the Internet network classes because an instance need not contain every variable. Table 1 shows instance sets for three different communication lines for a hypothetical Internet router. Internet addresses are converted to binary representation to generate the instances.

Communication Line	Arriving Internet Address	Instance
1	128.110.0.0	1000000001101110 -> Z_1
	128.213.0.0	1000000011010101 -> Z_1
	12.0.0.0	00001100 -> Z_1
2	10.0.0.0	00001010 -> Z_2
	130.5.0.0	1000001000000101 -> Z_2
	130.4.0.0	1000001000000100 -> Z_2
3	192.24.15.0	110000000001100000001111 -> Z_3
	128.160.0.0	1000000010100000 -> Z_3

Table 1 - Instance Creation Examples

Note that Boolean digits have been used instead of letters in this table to conserve space in the instance description. A "1" indicates a positive variable and a "0" indicates a negative variable.

4.3 Scaling Up to Larger Internets

A problem arises when multiple routers in a network must all adapt to source addresses. If a router observes a packet, but discards it because of a lack of routing information, then subsequent cannot adapt to the source-address. To overcome this problem, "beacon" packets are transmitted by each host when it initializes its connection to the network, with the destination address set to some special address value. These packets are re-transmitted by network routers on each interface except the one on which the beacon packet arrived. A computer connected to the network can also transmit a beacon packet after some timeout period, thus adapting all of the routers in the network to its location.

The beacon packet can be implemented by adding a global address to each router in the network. For example, if a router has three interfaces and the address "1111" has been selected as the beacon packet address, the following instances would be created:

(3) 1111 -> Z_1

(4) 1111 -> Z_2

(5) 1111 -> Z_3

A host simply sends a packet with the destination address "1111"; the router will transmit it on all interfaces. When a beacon packet is received it will be transmitted on all lines except the one on which it arrived. To prevent loops with beacon packets, the router tests the source address to determine if it was previously learned with the same communications line on which the beacon packet is now arriving. If it is the same communications line, then the beacon packet is discarded (it has previously passed through this router). If, however, the communications line has changed, then it must be processed and re-transmitted to all other communications lines because it represents a change in the network topology.

As long as the packet latency for each link is similar, such a load sharing mechanism can effectively improve throughput for communicating hosts. This algorithm provides dynamic load sharing on multiple paths because it automatically adapts to several different paths for a given host.

5.0 Example of Networking Routing

This section shows an example of network routing using the instance set to store routing information. The example utilizes a router that interconnects four communications lines. For the sake of simplicity, we restrict the network address to four bits, yielding a potential address space of 0000-1111. Figure 4 diagrams the example network. Packets are sent from one host to another by specifying a source and destination network address.

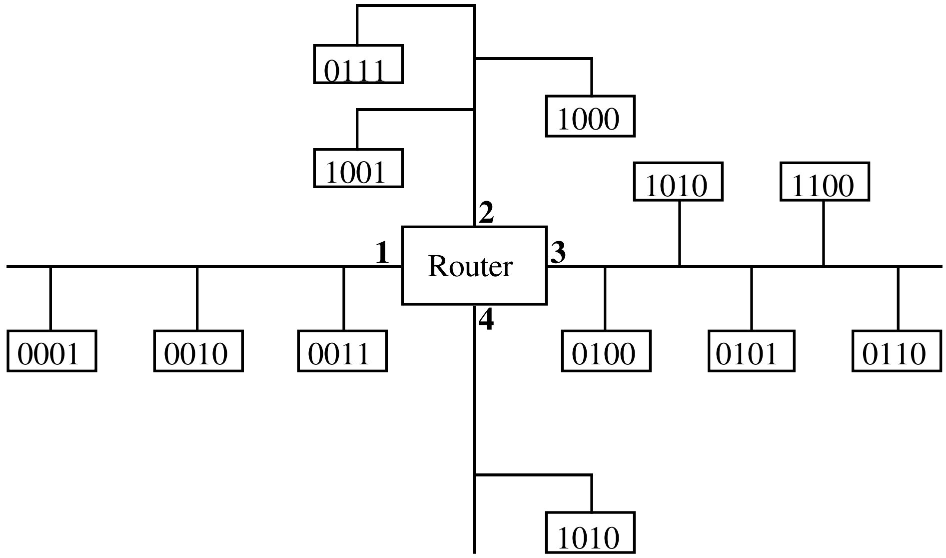

Figure 4 - Example Network

5.1 Initial Instance Sets

Initially, the instance set for each communications line contains no instances. The router dynamically builds the instance set as packets arrive. The router goes through adaptation and execution for each arriving packet.

5.2 Instance Set Addition

Assume the following packets arrive at the router:

Source Address	Dest. Address	Instance Created	Disposition of this Packet
"0010" (line 1)	"1000"	0010->Z_1	Discard
"1010" (line 4)	"1000"	1010->Z_4	Discard
"1000" (line 2)	"1011"	1000->Z_2	Discard
"0010" (line 1)	"1000"	0010->Z_1	(duplicate) Forward on line 2

The first column represents the source address of the incoming packet which becomes the input variables of a corresponding instance. These input variables create the instance of the third column. The instance is presented to the ASOCS unit during the learning (adaptation) phase. The destination address is the boolean input to the ASOCS unit during execution mode. In this example, packets are discarded if no instance matches the destination address. Because no packet had yet arrived with source address of "1000", the first two packets could not be routed. The third packet created the instance that matches address "1000". This instance allowed the fourth packet to be properly routed.

The instance sets that are associated with each line are now:

Line 1:	0010->Z_1
Line 2:	1000->Z_2
Line 4:	1010->Z_4

5.3 Instance Set Adaptation

Assume the initial instance sets as follows.

Line 1:	0010->Z_1
	1000->Z_2
Line 4:	1010->Z_4

Then suppose the following packets are received by the router:

Source Address	Dest. Address	Instance Created	Disposition of this Packet
"1000" (line 1)	"1010"	1000->Z_1	Forward on line 4
"1010" (line 4)	"1000"	1010->Z_4 (no change)	Forward on line 1

Note that the first of these two packets creates an instance that supersedes the earlier instance describing address "1000" for line 1. The next packet destined for address "1000" takes advantage of the adaptation and utilizes the new routing information.

The modified instance sets are:

Line 1:	0010->Z_1
	1000->Z_1
Line 4:	1010->Z_4

5.4 Type of Service Routing

In addition to address routing, variables can indicate "type of service" for a routing system. For example, a boolean variable for packet security can be concatenated to source address based variables to control the routing of confidential data over secured networks. A host might have two network connections, one to an open network and one to a secure network. The host controls which network is used by setting the packet security variable.

5.5 Routing Generalization

Adaptive connectionist routing permits *route generalization.* How should the system route messages for which no previous path has been learned. ASOCS seeks to minimize instances by finding critical variables which can discriminate without use of the entire input. When a message is to be forwarded to a location which the forwarding node has not previously seen, the system can use discovered critical variables of the location address to "guess" which path to forward the message on.

6.0 An Architecture for an ASOCS Router

Figure 5 shows the various components of an ASOCS router.

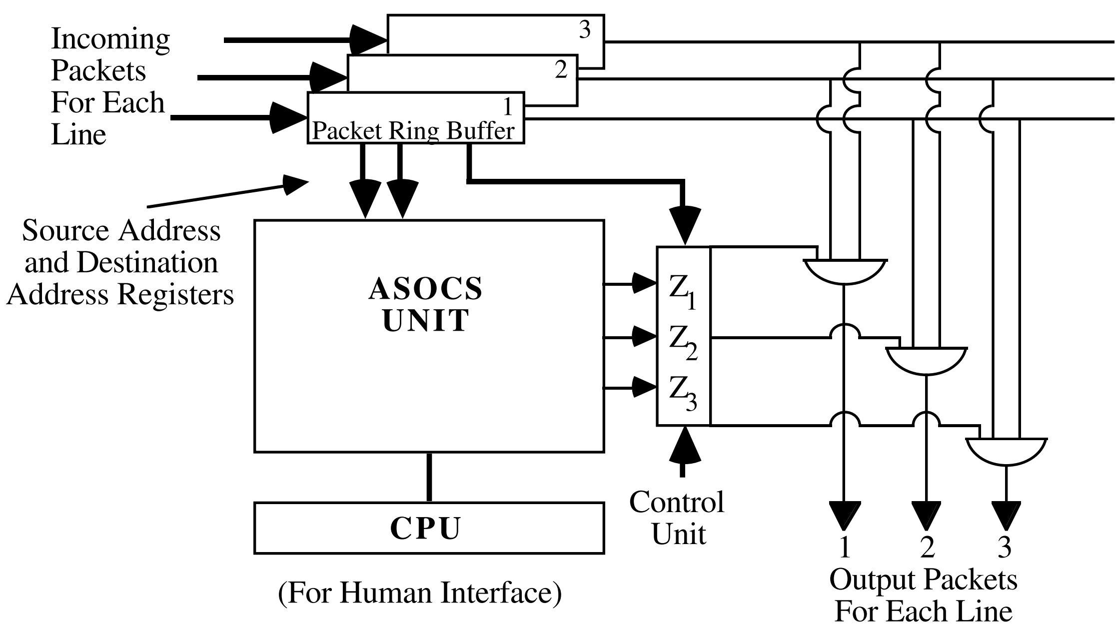

Figure 5 - Model of ASOCS Router

As packets arrive on the input lines, they are received in a high speed ring buffer. During the time that the packet is being placed in the ring buffer, the source-address, and destination-address fields become accessible through special registers and can be sent to the ASOCS unit. It is not necessary for the entire packet to be received before the ASOCS unit performs its function.

After the source and destination address at the front of the packet have arrived, the ring buffer generates an interrupt to the control unit. The control unit directs the ASOCS unit to: a) adapt to the source-address and then b) generate routing information for the arriving packet. However, if the packet is to be checked for CRC errors, then it can not be re-transmitted until after it has completely arrived, If the error rates are sufficiently low, the CRC checks could be skipped, permitting re-transmission of the incoming packet to begin on the appropriate output line before the complete packet had arrived.

The ASOCS unit accomplishes different tasks during adaptation mode execution mode.

1. During the adaptation mode, the source-address register is presented to the ASOCS unit. The source address becomes the input variables while the communications line on which the packet arrived becomes the output variable of the instance. The ASOCS unit learns this list of variables along with the correct output ("instance").
2. During the execution mode, the destination-address is presented to the ASOCS unit. The output vector which the ASOCS unit generates is passed to the control unit and stored in a special routing register in the ring buffer for each packet. This special routing register is used as routing information during packet transmission.

The control unit regulates packet transmission. It inspects each ring buffer and selects the first packet on the ring that has its routing register set. The control unit determines on which output line to transmit the packet by using as a selection mask the routing information that the ASOCS unit generated. After a packet is transmitted, the space in the ring buffer is released.

The CPU is a traditional processor with an interactive interface. The network administrator uses the interactive interface to enter commands or query the status of the ASOCS router.

When the packet is received, the routing components must be fast enough to act on the fields in the ring buffer while the packet is being received, if learning is to be done while data is arriving.

6.1 Architectural Considerations

Unlike traditional table-lookup mechanisms an ASOCS collection of routing rules is processed in parallel. During the execution mode, it takes an upper bound of *O(log(n))* time to process an instance set, where *n* is the number of nodes used to represent the instance set.

The following represents the algorithm to process each incoming packet with the traditional learning-bridge algorithm: (Sincoskie, 1988)

```
loop do
        receive packet;
        add <packet.source-addr,input_subnet> to route-list;
        lookup <packet.dest-addr,output subnet> in route-list;
        if <packet.dest-addr,*> "is not element of" route-list
                then
                        {transmit packet on all subnetworks except
                        input_subnet;} *
        else if output_subnet ≠ input_subnet then
                        transmit packet on subnetwork output_subnet;
        end if
end loop
```

* (This step could utilize the generalization features of ASOCS to make a "best guess", rather than flooding the network with unnecessary packets.)

The ASOCS router performs the same basic function as the above algorithm, but does so with the adaption and execution modes of the ASOCS unit. Therefore, a highly computation-intensive algorithm is replaced with the parallelism and generalization of an ASOCS unit.

7.0 Conclusion and Further Research

We build a network router using a boolean logic paradigm called the instance set to capture routing information necessary. An example shows that the instance set can correctly store and retrieve information to operate a network router. An ASOCS router is proposed to address the three network problems of increasing speed, increasing complexity, and the increasing need for adaptability. The ability of an ASOCS router to generalize or "best guess" a routing decision with incomplete information is a useful alternative to such other techniques as network flooding to determine a route. Future work with adaptive routing involves a component simulation of the ASOCS router to determine performance characteristics.

8.0 Bibliography

Backes, F. (1988). Transparent Bridges for Interconnection of IEEE 802 LANS, *IEEE Network*, **2**, 5-9.

Benhamou, E. (1988). Integrating Bridges and Routers in a Large Internetwork, *IEEE Network*, **2**, 65-71.

Comer, D.E. (1988). *Internetworking with TCP/IP: Principles, Protocols, and Architecture.*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.

Franceus, Paul. (18 May 1990). University of Maryland, Discussion of FDDI attributes via electronic newsgroup.

Intel Corporation Literature Dept. (1988). *Microsystem Components Handbook*, Intel Corporation, Santa Clara, California.

Martinez, T.R. (1986). *Adaptive Self-Organizing Logic Networks*, PhD Dissertation, University of California Los Angeles, Los Angeles, California.

Martinez, T.R. and Vidal, J.J. (1988). Adaptive Parallel Logic Networks, *Journal of Parallel and Distributed Computing*, **5**, 26-58.

Martinez, T.R. (1990). Adaptive Self-Organizing Concurrent Systems, *Progress in Neural Networks*, **vol. 1**, Ch. 5, 105-126.

Martinez, T.R. and Campbell, D. M., (1991). A Self-Adjusting Dynamic Logic Module, *Journal of Parallel and Distributed Computing*, **11,** 303-313.

Martinez, T. R. and Campbell, D. M., (1991). A Self-Organizing Binary Decision Tree for Incrementally Defined Rule Based Systems, *IEEE Transactions on System, Man, and Cybernetics,* vol. **21**.

Proteon Inc. (1989). OSPF: The New Dynamic Routing Protocol Standard, Techview.

Sincoskie, W.D. and Cotton, C.J. (1988). Extended Bridge Algorithms for Large Networks, *IEEE Network*, **2**, 16-23.

Stallings, W. (1987). *Local Networks*, Macmillan Publishing Company, New York, New York.

Tanenbaum, A.S. (1988). *Computer Networks*, Prentice-Hall, Inc., Englewood Cliffs, New Jersey.