MASTER

# Dynamisch reconfigureerbare multiprocessor systemen

Wijnen, C.R.P.

*Award date:*
1988

# Dynamisch reconfigureerbare multiprocessor systemen

C.R.P. Wijnen

FI/FIV 88-13

Augustus 1988

Verslag van het afstudeeronderzoek, verricht van september 1987 tot augustus 1988, in de vakgroep Fysische Informatica van de faculteit der Technische Natuurkunde van de Technische Universiteit Eindhoven, onder leiding van ir. G.J.W. van Dijk en prof. dr. A.J. van der Wal.

## Samenvatting.

Het is essentieel dat processoren in een multiprocessor systeem efficiënt met elkaar kunnen communiceren om een probleem gezamelijk op te kunnen lossen. De communicatie kan om redenen van uitbreidbaarheid en efficiëntie van het systeem het best uitgevoerd worden door middel van point to point verbindingen tussen de processoren. Omdat een processor slechts een beperkt aantal interfaces heeft voor zulke point to point verbindingen is er een limiet in het aantal processoren dat direct aan een processor gekoppeld kan worden. Gebruikt men meer processoren dan deze limiet, dan is het noodzakelijk om boodschappen tussen twee processoren die niet direct met elkaar verbonden zijn, via andere processoren te laten lopen. Dit gaat ten koste van de efficiëntie van het multiprocessor systeem. Het is echter met behulp van zogenaamde crossbars mogelijk, om de verbindingen tussen de processoren te verleggen.

Tijdens het afstudeeronderzoek is een architectuur ontwikkeld waarmee het mogelijk is processoren in een multiprocessor configuratie met elkaar te verbinden. De processoren kunnen op basis van transputer links met elkaar communiceren. Het is mogelijk om het netwerk dynamisch te reconfigureren. Dit wil zeggen dat tijdens het uitvoeren van programma's verbindingen tussen de processoren omgelegd kunnen worden. Het is essentieel dat er geen data transport plaats vindt wanneer verbindingen gereconfigureerd worden. Om dit te kunnen garanderen is tijdens het onderzoek een protocol ontwikkeld.

## CONTENTS

## INTRODUCTION

In tightly coupled multiprocessor systems efficient communication between the processors working in parallel on one problem is essential. Communication can be realized by means of a shared bus or shared memory. Increasing the number of processors implies an increase in communication between the processors. Because the communication bandwidth of such systems is limited, the number of processors that can use the same bus or the shared memory is limited. Using point to point communication links the bandwidth of the system increases proportionally as more processors are added. Point to point connections have however the major disadvantage that only a limited number of processors can communicate directly with a certain processor. If more processors are added, messages have to pass through other processors before reaching their destination. This implies the use of routing algorithms, causing an extra overhead for the system, and thus performance degradation [1]. To maintain the efficiency of the system, the routing of the messages should be carried out by independent hardware. Instead of passing the messages over fixed links through a series of intermediate processors, the connections between the processors are reconfigured, thus making direct communication between processors possible.

Another problem in the design of a distributed multiprocessor system is the definition of the interconnection network topology. Three different types of interconnection network topologies can be distinguished, viz. fixed, statically reconfigurable, and dynamically reconfigurable. A fixed network topology cannot be changed at all. Therefore, only a limited number of problems can be executed efficiently on such a system. In order to execute a much wider class of problems efficiently, a system is statically reconfigurable, i.e. the topology of the network can be adapted to the problem before runtime. The need for dynamic reconfiguration arises for reasons of wide applicability. The network topology can be changed during runtime.

In Sec. I, we describe the design of an interconnection network based on transputer links, for a multiprocessor system. In Sec. II, a protocol to control the dynamic reconfiguration of the network is described. In Sec. III a communication manager that hides details of the communication via the network for the user, is described. Conclusions are summarized in Sec. IV.

## I. A RECONFIGURABLE INTERCONNECTION NETWORK

To maintain the efficiency of the system, the routing of messages should be carried out by independent hardware. Therefore it must be possible to configure the interconnection network topology. For complete reconfigurability, a system must offer connectivity between any two processors, that means that any processor should be able to communicate with any other processor. In this section we describe an interconnection network which has this property. The basic element in the interconnection network, a switch, is described in Sec. I A. Switches can be cascaded to offer extendibility of the system. In Sec. I B we discuss the cascading of the switches.

### A. The switch

Starting point of our reconfigurable link switch design is the IMS C004 crossbar switch circuit [2], since it provides one of the most up to date circuits available.

The crossbar is capable of simultaneously connecting 32 input to 32 output links. The INMOS link protocol [3] requires bi-directional links. Hence the crossbar can set up paths between 32 communication links. From here on we will adopt the convention that a link is bi-directional.

The crossbar is programmed via a special link, viz. the configuration link. This link is directly connected to a special processor, the controller. The controller establishes paths between the 32 links by sending commands via the configuration link, thus programming the crossbar.

The controller receives information to program the crossbar via one of its other links. These links can for instance be connected to other crossbar controllers or to a central host processor [4]. The host processor is able to transmit connection messages to the controller. The interconnection network can be changed during runtime, but the host has no means to check whether for instance data is transported via the interconnection network. Such a system can support static reconfiguration.

Dynamic reconfigurability implies that the topology of the interconnection network can be changed when two processors that are not directly connected, want to communicate with each other in order to exchange data. Therefore, a controller must be able to receive a request message to set up a path from each processor. A possible configuration is to connect the controller to its crossbar by means of <u>one</u> link. The controller is able to connect itself to each of the processors so it can interrogate them. If a processor wants to transmit a message to another processor, the controller can connect the two by sending commands via the configuration link to the crossbar.

After a direct connection between two processors has been established, the controller of the switch cannot communicate with these processors as long as it does not break the path. This implies that both processors are unable to request the controller to set up a different connection. Also, the processors cannot send a message to the controller to indicate that the data transfer is finished. In this configuration, it cannot be guaranteed that a connection between two processors is only broken when there is no data being transmitted via that connection. This is a highly undesirable situation, and therefore the controller must be able to communicate with the processors via a second link.

A second crossbar can be used to connect all processors with a second link to the controller. In principle, it is possible to use the same crossbar as used for the connections between processors, but in order to maximize the number of processors directly connected to one switch, a second crossbar is used (Fig. 1).

The _data crossbar_, is solely used to interconnect processors in order to exchange data. The link of a processor connected to the data crossbar is called a _data link_. The connection between two processors is called a _data path_. The other crossbar, the _polling crossbar_, is used to exchange control messages between processor and controller. The controller cannot be connected to all processors simultaneously. Therefore, it connects itself consecutively to all the processors. The consecutive interrogation of the processors by the controller is called _polling_. The link of a processor connected to the polling crossbar is called the _polling link_.

The controller and the two crossbars connected to the controller form the basic element for the interconnection network, and will be referred to as the <u>switch</u>.

A controller of a switch can investigate to which link of the polling crossbar a processor is connected. The controller is unable to determine via which link a processor is connected to the data crossbar. It is assumed here that if the data link of a system is connected to a certain link of the data crossbar that then its polling link is connected to the corresponding number of the polling crossbar. A polling and a data link connected in this way will be called a <u>communication line</u>.

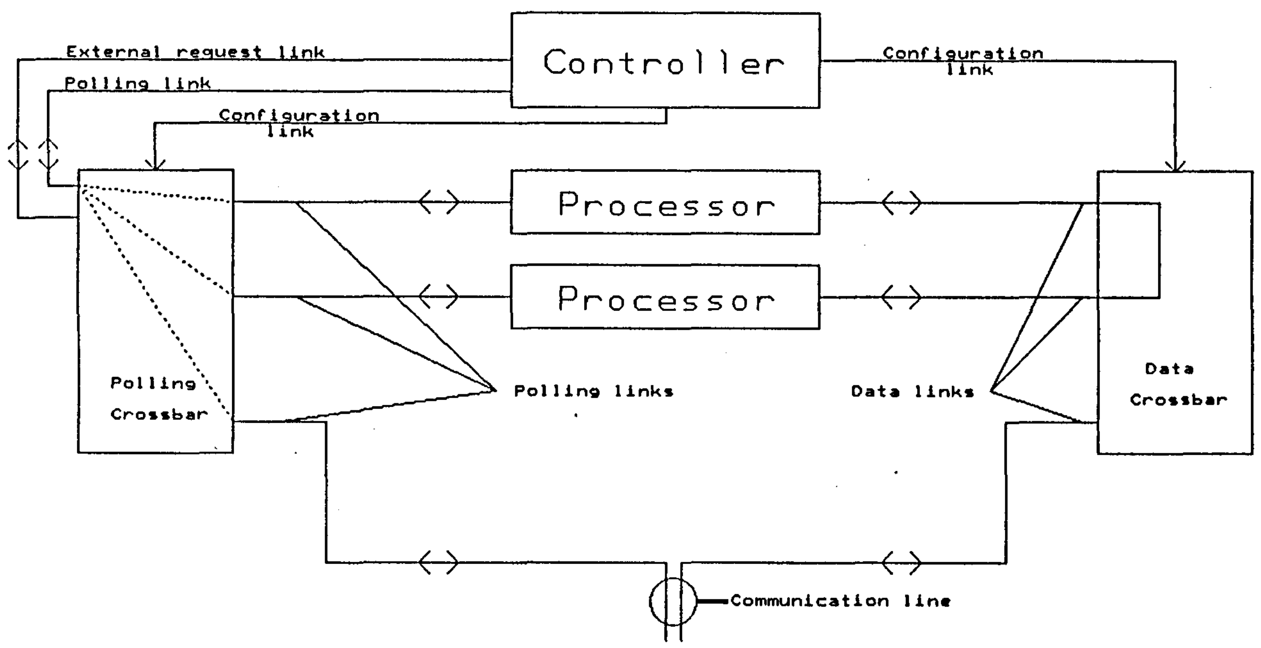

Fig. 1. A switch to which two processors are connected. The switch consists of the controller, the polling crossbar, the data crossbar and the links between the controller and these crossbars. The processors are connected to the switch by means of a communication line, i.e. a polling link and a data link.

## B. Distributed switching network

The switch described in Sec. I A is able to reconfigure a network with at most 31 processors connected. To extend this network, switches will be cascaded.

A connection between the controllers of two switches must exist in order to exchange control messages. The connection can be realized using a link, connecting the polling crossbars of both switches. In this way the number of switches connected to one switch is limited by the size of the polling crossbar. A direct connection between controllers is possible but the number of switches connected to one switch would be much lower because of the limited number of link interfaces of a controller.

To connect processors of two different switches, at least one link connecting both data crossbars is needed. In this way any processor is able to communicate with any other processor but it is possible that a connection can only be realized after another connection is broken. It is more efficient to use more links between the data crossbars. If sufficient connections [2] are used, all possible connections between the processors can be realized without breaking another connection. Such a network is called a non-blocking interconnection network [5].

A controller cannot determine via which data link a switch is connected to the data crossbar. Therefore the connections between switches are realized by means of communication lines (see Sec. I A).

The controller of a switch transmits a control message to another switch in the following way: The transmitting controller programs the polling crossbar of the switch such that its polling link is connected to the polling link of the communication line by means of which the two switches are connected. When the controller of the other switch polls the communication line, a direct connection between the two controllers is established and the control message is transmitted.

Problems occur when for example, three controllers simultaneously want to transmit control messages in a cyclic way (e.g. A to B, B to C, and C to A). The systems are blocked forever because each controller waits until it is polled by the other controller, which in turn is waiting to be polled.

The use of a second link between a controller and the polling crossbar of a switch, could avoid such blocking (Fig. 1). One of the links, the <u>external request link</u>, is solely used for the passing of messages to other switches. The other link, the polling link, is used for the polling function and the passing of control messages to the processors that are connected to the switch.

## II. A PROTOCOL FOR DYNAMIC RECONFIGURATION OF THE INTERCONNECTION NETWORK

When the network topology is changed, it is essential to ensure that there is no data transfer taking place via the links involved in the reconfiguration. A data transport is controlled by the processors connected via the data path. The switch, which has to reconfigure the network, cannot directly control the data transport. However, the switch must be able to communicate with the processors in order to guarantee that the data transport has been finished when the network is reconfigured.

We first describe a protocol to control the reconfiguration of the interconnection network. Implementation details of the protocol are discussed in Sec. II B. In Sec. II C we describe routing between switches, and an example of communication is described in Sec. II D.

### A. Control of the network reconfiguration

A processor that wants to transmit a message to another processor has to request the switch to establish a direct connection between the two processors: a data path. With each data path a mastership is associated. The configuration of the network is controlled by the following rules:

(1) A switch creates a mastership of a data path after it has established a direct connection between two

processors. The mastership is passed to the processor that requested the path in order to transmit a message. The processor that has the mastership will be referred to as the <u>master processor</u>.

(2) A switch requests the mastership of a data path from the master processor, when a request is received to break the data path. When no data transport is taking place via the path, mastership is passed to the switch by the processor. The switch then destroys the mastership and breaks the data path.

(3) A switch may use lines that are not involved in data paths. A switch that wants to use a line that is involved in a data path, has to break that data path first. If the master processor is connected to another switch, the request to break the path must be sent to that switch.

When a data path is set up, two different situations can be distinguished:

(1) The requesting processor and the destination processor are connected to the same switch. This switch establishes a direct connection between the two processors and creates a mastership of the data path. The mastership is passed to the requesting processor.

(2) The requesting processor and the destination processor are connected to different switches. The switch, to which the requesting processor is connected, will be referred to as the <u>requesting switch</u>; the switch,

to which the destination processor is connected, will be referred to as the destination switch.

The requesting switch looks up in its routing table via which communication line the destination switch can be reached. It sends a request, via that line, to connect the destination processor to the communication line. The destination switch receives the request via the communication line and establishes the connection between the destination processor and that line. After that, the destination switch sends a message via the same line to the requesting switch, to indicate that the connection is established. After the message is received by the requesting switch, it connects the communication line to the requesting processor. The requesting switch then creates a mastership of the data path and passes it to the requesting processor.

It should be noted that it is possible to set up a connection via a switch that is neither connected to the requesting processor nor to the destination processor. Such an intermediate switch receives via a communication line a request to connect the destination processor to that communication line. Because the destination processor is not directly connected to the intermediate switch, the switch looks up in its routing table via which communication line the destination switch can be reached. It sends a request to the destination switch, via that line, to connect the destination processor to that line. After the confirmation of the connection is received, the intermediate switch connects the line to the line via which the request was received. The switch then sends a message to the requesting switch to indicate that the connection is established.

Upon establishing a connection between two processors, the possibility exists that a switch has to use communication lines that are already in use by other data paths. These data paths must be broken first, before these communication lines can be used. Two different situations can be distinguished when a switch wants to break a data path:

(1) The master processor is directly connected to the switch.

(2) The master processor is connected to a different switch.

In the first situation, the switch requests the processor to pass the mastership. The processor passes the mastership to the switch after the completion of a data message transfer. After the mastership has been passed to the switch, it is guaranteed that data transfer over this path is blocked. Hence, the switch may destroy the mastership and break the connection.

In the second situation, the switch has to send a request to break the path to the switch to which the master is connected, the <u>master switch</u>. The switch helped setting up the path and therefore knows via which communication line it has to send the request. The master switch requests the mastership from the master processor, and breaks the path after it received the mastership. The master switch sends back a message to indicate that the path is broken to the other switch. The other switch now may use the lines of the former path.

A path can be set up via an intermediate switch. Such an intermediate switch has to pass the request to break the path to the master switch. When the switch receives the message from the master switch that the path is broken, it breaks its part of the former path and passes the message to the switch that requested the breaking.

The possibility exists that a switch sends a request to the master switch to break a data path when the path already is broken, e.g. when an intermediate switch requested the master switch to break the path.
A switch that receives such a request transmits back a message that indicates that the path is broken.

## **B. Implementation details**

The protocol described in Sec. II A will be implemented on the switching network described in Sec. I. In this section, we describe the implementation details of a polling mechanism, and the implementation of the establishing and breaking of connections between processors.

### 1. Polling

The communication lines consist of a polling link and a data link. Control messages are sent via the polling link and data messages are sent via the data link.

The processors and neighbor switches, which will be referred to as systems, are connected to the polling crossbar of a switch by means of the polling link of their communication line (see Fig. 1). The controller of a switch consecutively interrogates the systems whether they want to transmit a control message. Therefore, the controller connects itself to the system by configuring the polling crossbar in such a way that the polling link of the controller is connected to the polling link of the system. The controller and the system are now able to exchange control messages.

## 2. Establishing a connection

A processor that wants to transmit a data message to another processor, sends a request to set up a connection when it is polled by the controller of the switch to which the processor is connected. Also the destination address, which consist of a switch number, a processor number, and a process number, is exchanged. Two different situations can be distinguished when establishing a connection between two processors:

(1) The destination processor is connected to the same switch as the processor that requested to set up the connection. The controller of the switch programs the data crossbar to connect the data link of the destination processor to the data link of the requesting processor. The controller creates a mastership of the data path and passes it to the requesting processor via its polling link, the polling crossbar and the polling link of the processor.

(2) The destination processor and the requesting processor are connected to different switches. The controller of the requesting switch looks up in its routing table via which communication line the destination switch can be reached. The polling crossbar is configured in such a way that the external request link of the controller and the polling link of that communication line are connected. When the controller of the destination switch polls that line, the two controllers exchange the destination address. The controller of the

destination switch configures the data crossbar such that the data link of the destination processor and the data link of the communication line, via which the request was received, are connected. The controller of the destination switch transmits a message via the communication line that confirms this connection. Therefore, it configures the polling crossbar of the destination switch in such a way that its external request link and the polling link of the communication line are connected. After receiving the confirmation message, the controller of the requesting switch configures the data crossbar such that the data link of the communication line and the data link of the requesting processor are connected. The controller creates a mastership of the data path between the two processors. The controller connects its polling link to the polling link of the requesting processor by programming the polling crossbar of the switch and passes the mastership of the data path to that processor.

It is possible that the destination switch is reached via an intermediate switch. This intermediate switch receives the request to connect the destination processor via a communication line. It passes the request to the destination switch in the same way as the requesting switch. After receiving the confirmation message, the intermediate switch connects the data link of the communication line via which the destination switch can be reached, to the data link of the communication line via which the request was received. It sends a message to confirm the connection to the requesting switch.

## 3. Breaking a connection

When a new connection between two processors is requested, the possibility exists that a switch has to use data links that are already in use by other data paths. The switch has to break the path before it can use the data links. It is also possible that the switch receives a request to break a path from another switch.

The mastership of a data path must be destroyed before the connection may be broken. In order to break a path, two different situations can be distinguished:

(1) The master processor is connected to the switch that has to break the data path. The controller of the switch connects itself via its polling link, the polling crossbar and the polling link of the processor, to the master processor, and requests the mastership. The processor passes the mastership to the switch after the completion of a data message transfer. The controller of the switch may now destroy the mastership of the data path and break the connection.

When the request to break the path was received from another switch, the controller has to send a message to confirm that the path is broken, to the controller of that other switch. Therefore, the controller connects its external request link to the polling link of the communication line via which the request was received, and sends the confirmation message.

(2) The master processor is connected to a different switch. The controller of the switch that has to break the path, helped setting up the path and therefore knows, via which communication line it can reach the master switch. The controller connects its external request link to the polling link of that communication line. When it is polled by the controller of the switch connected to the other end of the communication line, it sends a request to break the path. It waits until it receives a message that indicates that the path is broken.

## C. Routing between switches

A switch has to know via which line a specific switch can be reached. Therefore each switch builds its routing table by exchanging information with neighbor switches.

A switch cannot determine whether its routing table is completed because it does not know the size of the network. It is also possible that the topology of the switching network is changed, e.g. when switches are added or when lines fail. It is therefore essential that the routing table has the property of consistency, even when the table is built or altered. In the following, an algorithm for building a routing table, which has the property of consistency, will be described.

Each switch has to build its own sinktree, a tree which represents the set of optimal routes from the switch to any other switches [6]. The tree is built by exchanging routing information with neighbor switches. If two routes are of the same length, only one is chosen.

The sinktree contains the information that is needed to reach any switch in the network. The routing table is built by exchanging the sinktrees between the neighbor switches. Only the branches of the trees that do not contain the switch itself are used to build the table. Thus, the routing table also contains alternative routes that do not pass through the switch itself. A sinktree is altered when either a failing line or a new switch is detected. Whenever one sinktree is altered, sinktrees are exchanged between neighbor switches in order to update the routing tables and the sinktrees of all switches in the network.

## D. Example of communication

As an example of a communication one can consider the network outlined in Fig. 2. Processor P1 and P2 are connected to the switch S1 that is controlled by controller C1. Processor P3 and P4 are connected to the switch S2 that is controlled by controller C2. The switches are connected by one communication line.

Processor P1 is currently transmitting data messages to processor P3. The path between P1 and P3 consists of the

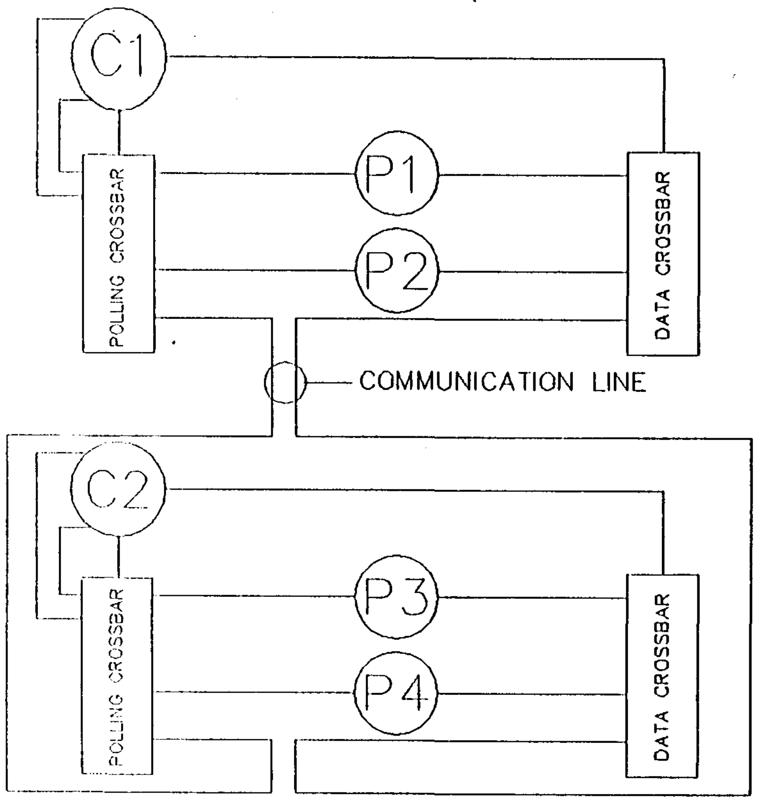

Fig. 2. An example of an interconnection network consisting of two switches by means of which four processors are connected. Processors P1 and P2 are connected to the switch controlled by controller C1. Processors P3 and P4 are connected to the switch controlled by controller C2.

data link of P1, the data crossbar of switch S1, the communication line, the data crossbar of switch S2, and the data link of P3. P1 is master of the data path between processors P1 and P3. Processor P4 wants to transmit a message to processor P2. Therefore, P4 transmits a request to set up a path to processor P2, when P4 is polled by controller C2. Controller C2 looks up in its routing table via which line it can reach the switch, to which processor P2 is connected. The data link of the communication line is already in use by the data path between processors P1 and P3. To use the data link, controller C2 has to ask controller C1 to break the data path. Therefore, C2 connects its external request link to the polling link of the communication line and sends the request to break the path at the moment that C2 is polled by controller C1. C1 sends a request to P1, via its own polling link and the polling link of P1, to pass the mastership. As soon as processor P1 finishes its data transport to P3, it passes the mastership to C1. Controller C1 destroys the mastership and breaks the path. It informs this to controller C2 via its external request link, the polling link of the communication line and the extern request link of C2. Controller C2 sends a request to connect P2 to the data link of the communication line to C1. Controller C1 connects the data link of processor P2 to the data link of the communication line and informs controller C2. C2 then connects the data link of processor P4 to the data link of the communication line. C2 creates a mastership of the data path between P4 and P2, and passes the mastership to processor P4. P4 can now transmit data messages to P2.

## III. COMMUNICATION MANAGER

A user of the multiprocessor is not interested in the way processes communicate with each other. It is sufficient to know that a process can communicate with any other process either running on the same or on a different processor. To hide the details of the communication for the user, each processor has a communication manager. This communication manager is downloaded to the processor when the controller of the switch polls the processor for the first time. The communication manager takes care of the communication between processes running on different processors and between processes running on the same processor (Fig. 3).

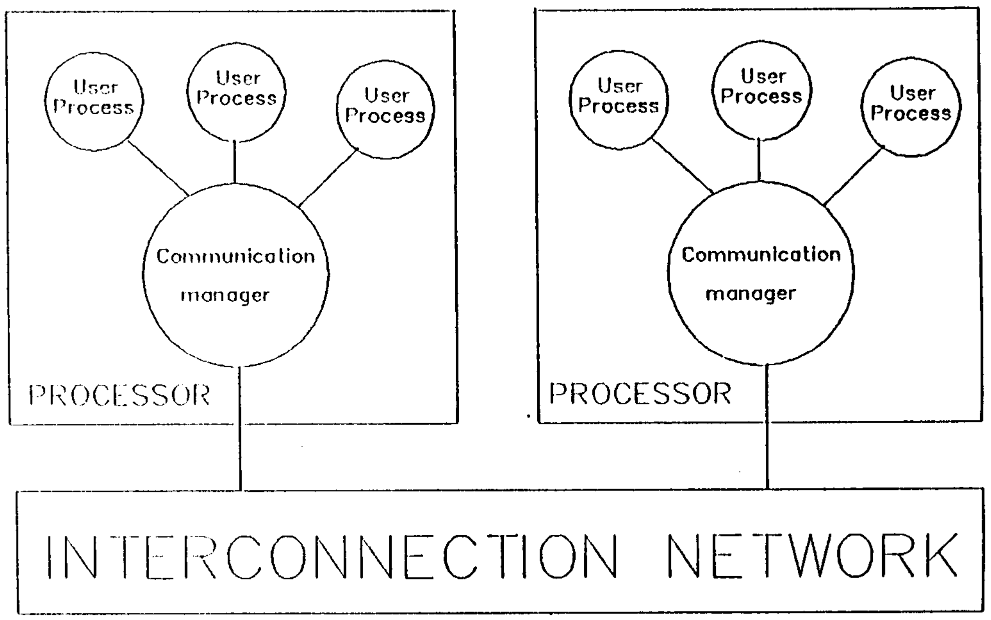

Fig. 3. Two processors connected via the interconnection network. Both processors are loaded with a communication manager. The user processes are able to communicate with each other via these managers.

A message consists of a header and a data field. The header contains the destination address that consists of a switch number, a processor number, and a process number. The data field contains the actual message. Instead of transmitting the message directly to a destination process, a user process passes the message to the communication manager.

The communication manager examines the header of the message in order to determine whether the destination process resides either on the same processor or on a different processor. If the destination process is located on the same processor, the communication manager passes the message directly to the destination process. If the destination process resides on a different processor, the communication manager sends a request to establish a direct connection between the two processors, to the controller. After the data path has been set up (see Sec. II B 2), the controller of the switch creates a mastership of the path and passes it to the communication manager. The communication manager transmits the data message to the communication manager of the destination processor. This communication manager passes the message to the destination process.

## IV. CONCLUDING REMARKS

In this paper, we describe an interconnection network that can be reconfigured dynamically using crossbar switches. Efficient communication between processors is achieved because at any time each processor can be directly connected to any other processor in the network. Thus message routing, which causes a performance degradation of the system, becomes unnecessary. The control of the configuration of the network topology is distributed. The network can be extended by cascading switches. It has been shown that the protocol to control the dynamic reconfiguration of the network topology guarantees that no connections between processors are broken unless no data is transmitted via that connection. The protocol provides for fault tolerance when communication lines between switches fail.

The interrogation of processors connected to different switches is done in parallel. This decreases the time that a processor has to wait until a request to establish a connection can be passed to a controller. As long as switches do not need each other to establish a connection, the set up of different paths can be done in parallel also.

A difficulty however with parallel polling is to obtain fairness, implying that all processors should have equal opportunity to be connected to a destination processor. The present protocol provides for fairness between processors connected to the same switch because the switch polls the processors in a cyclic way. Because a switch can only transmit a request to establish a connection to another switch when it is polled by that switch, also fairness between processors connected to different switches is provided by our protocol.

Presently the switch is being developed and will be available at end of 1988. The investigation of the exact properties of the interconnection network and the implementation of the protocol will be the subject of further study.

## REFERENCES

[1] P.K. Das, D.Q.M. Fay, "Dynamically reconfigurable multi-transputer systems", Microprocessing and microprogramming, Vol 23, pp. 247-252, 1988.

[2] INMOS ltd., "IMS C004 programmable link switch", Preliminary data sheet, April, 1987.

[3] INMOS ltd. "The transputer implementation of occam", Technical note 21.

[4] T. Äijänen, "Distributed interconnection of a reconfigurable multicomputer system", Microprocessing and microprogramming, Vol 23, pp. 243-246, 1988.

[5] G.J. Lipovski, M. Malek, Parallel computing, John Wiley & Sons, Inc., 1987.

[6] A. S. Tanenbaum, Computer networks, Prentice-Hall, Inc., pp. 205-212 ,1981.