Ring

The Software Tools Subsystem Network Utility
Version 1.0

Roy J. Mongiovi

School of Information and Computer Science
Georgia Institute of Technology
Atlanta, Georgia 30332

April, 1983

TABLE OF CONTENTS

Ring

Ring

Introduction

Ring is a distributed request server for the Software Tools Subsystem which uses PRIMENET to communicate between nodes in a distributed ring. It performs simple system functions such as keeping the time of day synchronized on all the machines in the ring, as well as accepting user requests for services. It validates all requests it receives, which ensures that a devious user cannot create his own Ring server and transmit invalid requests to the other Ring processes.

One copy of the Ring process executes on each of the systems in the ring. Each process establishes two virtual circuits (a transmit and a receive circuit) with the next and previous systems, where next and previous are defined by the system names in lexically sorted order. As systems are brought up and down, the ring dynamically reconstructs itself to maintain that ordering. A user who wishes to make a request of the ring connects to the Ring process on his own system and transmits his request. That Ring process reformats the request and transmits it around the ring where it is eventually seen and acted upon by the Ring process to which it was addressed.

Validation

There are two distinct types of connection request validation performed by Ring. The first is the validation of virtual circuits connecting each of the Ring processes in the ring, and the second is the validation of a virtual circuit connection from a user to the Ring process. These two types of validation are distinguished by the fact that ring connections are normally between two systems, while user connections are restricted to the same system (that is, a user is not allowed to connect to a Ring process on another system).

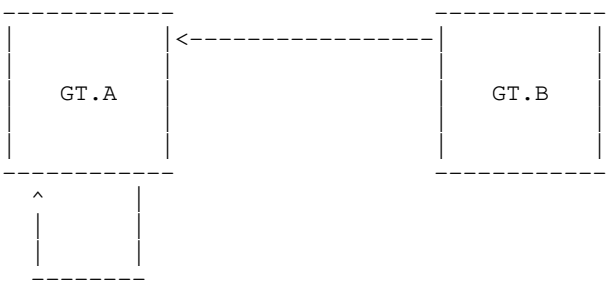
Validation is made difficult by the fact that it is impossible to determine the user name (or any other information) of the process on the other end of a virtual circuit. Information may be returned only for virtual circuits on the current system, and even then only for known virtual circuits. As we shall see, it is possible to find the user name of the process on the other end of a circuit given certain restrictions. In fact, the entire purpose of user validation is to determine the user name and process id of the process on the other end of a virtual circuit.

Ring Connections

When a Ring process attempts to break into a previously existing ring (i.e. when a system has been down and is being brought up), and when a system that was in the ring has gone down, the new connections must be validated before they are accepted as coming from a Ring process. It would be very simple if a user name (such as SYSTEM) could be checked, but as has already been mentioned it is impossible to determine the user name on the other end of a virtual circuit that is on another system. The only piece of information that can be used for validation that is assured by the PRIMENET routines is the fact that a port can be assigned by only one process. Using this fact, together with the assumptions that the Ring process will be started at boot time, will immediately assign its ports, and will never relinquish those ports as long as the system is up, it is possible to validate ring connections. Note that this assumes that Ring will never fail on a hardware/software error, a rather stringent requirement. Should Ring ever fail and unassign the validation port while the system is up, it would be possible for another user process to assign that port and become the Ring process for that system.

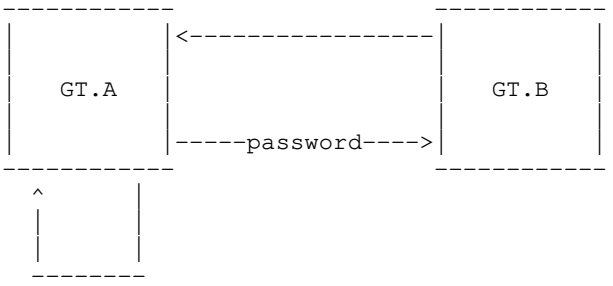
When a Ring process begins execution, the first thing it does is assign three ports: a ring port, a validation port, and a user port. These ports are never unassigned. It then determines all system names, sorts them, and begins attempting to connect to an already existing ring starting with the next system (in the sorted list). Should it be the first Ring process, it will eventually connect to itself and establish the initial degenerate ring. Validation of that connection proceeds as follows:

When a Ring process detects a connection request to its ring port, it accepts it provisionally and then attempts to validate it.



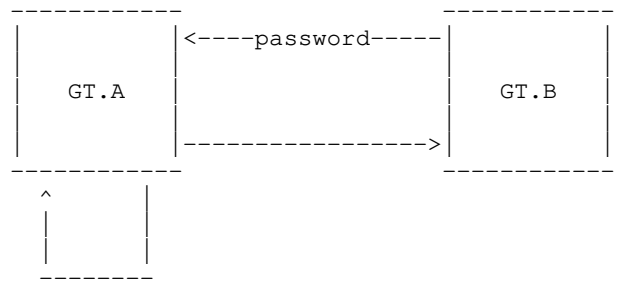
1. The new Ring process makes a connection request.

The Ring process makes a connection request to the validation port on the system from which the ring connection was received. When that connection is accepted, it generates a random number password and transmits it to the validation circuit.



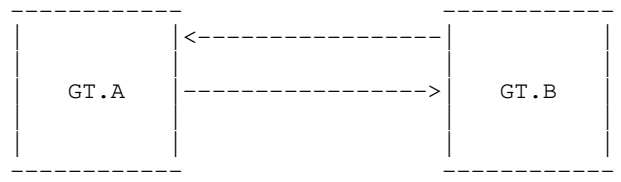
2. The validation password is transmitted.

If the ring connection is indeed valid, then the validation connection is to the same process that issued the ring connection. The password is then received and retransmitted to the ring circuit.



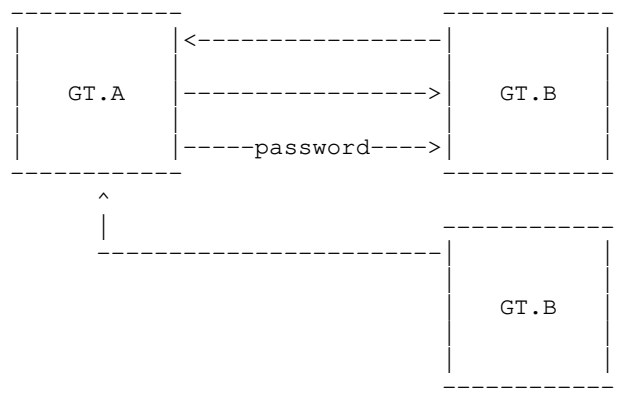
3. The response password is retransmitted.

The Ring process that is validating the connection receives that password on the circuit that is being validated, compares it with the password that was transmitted, and validates the circuit.



4. The new ring connections are established.

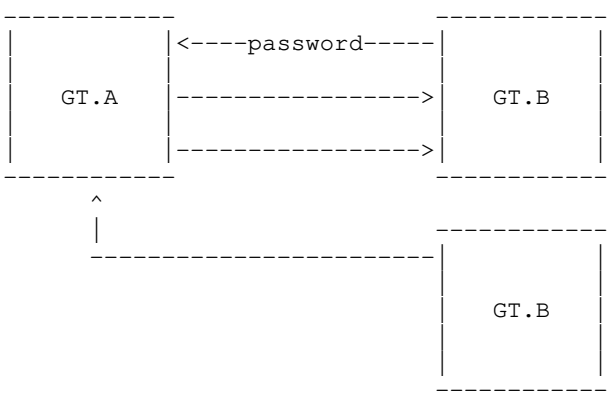
If the ring connection is from a pretender, then the validation connection is to the actual Ring process on that system, the pretender cannot receive the password, and the ring connection is not validated.



5. The false Ring process cannot receive the password.

When the actual Ring process receives the password, it transmits it through the already validated ring circuits, and when the

validating process receives it from that circuit (and not the circuit being validated) it knows that the connection attempt is not valid and clears the connection.



6. The password is received from the existing ring.

User Connections

When a user connection is received, the Ring process must determine the user name and process id of the process making the connection request in order to ensure the validity of any requests that the process may make. It is not good enough to have the user process transmit this information since that process could easily fabricate it. The ability to identify the user process hinges on the following ideas: it is possible to determine the virtual circuit numbers of all allocated virtual circuits open on a system, user connections must be from the same system as the Ring process that they are connected to, and user connections are accepted and identified one at a time.

To identify a user connection, the Ring process obtains a list of all open virtual circuits on the current system. This list is scanned to find all circuits that are to the user port, which have been accepted, and which are not the process id of the Ring process. The list of existing user connections is then scanned, and the corresponding entries in the list of virtual circuits are marked as known. Since user connections are accepted one at a time, there will be exactly one virtual circuit that was not marked as known, and that is the virtual circuit corresponding to the newly accepted user connection. The user name of that process is determined using a system call, and the connection is added to the list of known virtual circuits.

Ring Requests

All operations performed by Ring are initiated by request packets which are passed around the ring connections. Each packet has the same size and consists of two parts: a fixed identification header, and a variable argument array. The header consists of a flag that indicates whether the packet is a request or a response, source and destination addresses, a count of the number of Ring processes that have seen the packet, a process id and unique identifier to indicate what process created the packet, and the Ring request command/status words. The format of the variable argument array depends of the value of the command word in the packet header.

Ring requests are passed around the ring, from receive connection to transmit connection, until they are received by the system to which they are addressed or the number of Ring processes that have seen them is greater than the number of systems in the ring. A packet destination with all bits set (-1) is received by all Ring processes in the ring. When the request packet is performed or destroyed, it is transformed into a response packet which is transmitted to the system that created the request.

Internal Requests

When a new ring is established, as well as when an existing ring is changed because one or more systems have come up or gone down, a special request packet is transmitted around the ring. This packet, the INITIALIZE request, has two purposes. First, it is used to count the number of Ring processes that are actually in the ring. PRIMENET provides a status call which returns the number of systems configured in the network, but they may not all be running Ring. As the INITIALIZE packet goes around the ring, each Ring process increments a counter in the packet. When the request arrives back at the Ring process that created it, an INITIALIZE response packet is created which contains the number of systems that saw the original request. This response packet is then used by each Ring process to set the actual number of systems in the ring. The second purpose of the INITIALIZE request is to determine who is to set the time of day on all systems initially. Normally, the time of day is set by the first (in lexically sorted order) system that is running Ring. However, should that system be the one that caused the ring to change (i.e. it just entered the ring), it is assumed not to know the correct time, and the next system which was in the ring previously should set the time. As the INITIALIZE response is transmitted around the ring, a state variable is transmitted along with it. This variable starts as 0, when the system that is supposed to set the time of day sees the packet, it sets the state to 1 if it just entered the ring and does not know the time of day, and 2 if it does know the time of day. If the state is 1, then the next system that does know the time of day sets the state to 2 and then sets the time of day on all systems.

Each hour on the hour, the Ring process that is first in lexically sorted order transmits the current time of day to all other systems in the ring. Although this is not necessary for orderly system operation, it does make sense for each processor in a distributed system to have the same time of day.

User Requests

Currently, four kinds of user requests are implemented by Ring: a BROADCAST request which allows a PRIMOS message to be sent on all systems in the ring, an EXECUTE request which starts up a SWT phantom on a particular system in the ring, a TERMINATE request which allows one or all of the Ring processes to be stopped and re-executed (so that a new version of the Ring process may be brought up), and a SETTIME request that allows the time to be reset on all systems in the ring.

To make a user request, a user process first connects to the user port of the Ring process which is executing on its system. When the connection has been accepted, the user transmits the request and begins waiting for a response. When the Ring process has received the request and checked its validity, it transmits a status code to indicate that the operation has been initiated or that an error has been encountered back to the user process. The user process receives this status code, and if it indicates that the request has been initiated begins waiting for a completion response. When the Ring request has been completed (successfully or not), the Ring process will transmit a final status code to the user process. The user process then examines the returned status and clears the connection.

BROADCAST. The BROADCAST user request consists of three parts: the BROADCAST request word, a three word user name of the user who is to receive the message (zero if all users), and a Software Tools string which is to be broadcast.

EXECUTE. The EXECUTE user request also consists of three parts: the EXECUTE request word, a three word system name of the system on which the phantom is to be executed (zero if all systems), and a Software Tools string which is the command line to be executed.

TERMINATE. The TERMINATE user request consists of two parts: the TERMINATE request word, and a three word system name of the system which is to be terminated (zero if all systems). Because it is impossible to determine when a transmitted message has been received, the TERMINATE request actually occurs in two stages. After the user's TERMINATE request has been processed and the status response has been transmitted, an internal request (SHUTDOWN) is transmitted around the ring. It is this request which actually causes the selected Ring process(es) to terminate, thus allowing time for the user process to receive its status.

SETTIME. The SETTIME user request consists of two parts: the SETTIME request word, and a five word block which contains

the month, day, year, hour, and minute to which the current time is to be set.

Future Requests

Ring is intended to handle simple requests by itself. A simple request is defined as one which would require no more than one request and response packet to perform. In the future, it is envisioned that complex requests such as remote execution of commands and remote file handling will be performed by a helper phantom which the Ring process will create and which will then be connected directly to the requesting user. Ring can also be used to moderate interprocess communication by allocating ports and controlling access to those ports. This will allow two or more user processes to communicate without requiring fixed port numbers which may be used by other user processes with which communication is not desired.

The major drawback with this scheme of creating helper phantoms is the relatively large amount of time required to create a phantom. In fact, when PRIME itself decided to replace the old FAM (the File Access Manager) with a new version which uses SLAVE$ helper phantoms, it was necessary to special-case the SLAVE$ phantoms so that they would start up more quickly.

PRIMENET Problems

During the development of Ring, only one significant error was found, and that was in the PRIMENET documentation. However, quite a bit of code in Ring is devoted to determining information that should most likely be available directly from the PRIMENET subroutines. Several enhancements to the existing routines come easily to mind.

Errors

The only problem with PRIMENET that may be classified as an error is in the documentation for the message transmission subroutine X$TRAN. The following information about the return status codes (taken directly from the PRIMENET manual) is not correct:

> The codes that may be returned in status by a call to X$TRAN appear below:
>
> XS$CMP The transmit is complete. The message has been copied out of the sender's buffer and transmission is initiated. (A transmit status of complete means only that PRIMENET

will attempt to deliver the message. Applications requiring assured delivery must implement their own end to end acknowledgement.)

XS$IP The transmit is in progress. status will be further updated by the completion or failure of the operation.

XS$BVC The calling process does not control the virtual circuit specified in vcid.

XS$MEM Temporary PRIMENET congestion prevents the acceptance of the request at this time.

XS$MAX The maximum number of transmits simultaneously in progress over a single virtual circuit has been exceeded. This request to initiate another transmission is denied.

XS$RST The virtual circuit has been reset. The status of this operation is unknown and no further attempts will be made to complete it.

XS$CLR The virtual circuit has been cleared. See the virtual circuit status array for the clearing cause.

XS$ILL The transmit operation is illegal because a circuit connection request or a clear request is pending. This is the result of attempting transmission over an "almost-open" or "almost-closed" circuit.

The description of status codes XS$CMP, XS$MEM, and XS$MAX seems to indicate that once a transmit operation is in progress it must either complete or return an error code. In fact, this is not the case. If too many transmit requests have been issued on a virtual circuit, the status code remains XS$IP until enough receives have been performed to allow the transmit to take place. In its example programs, the PRIMENET manual gives a subroutine which is called after a transmit to wait until the transmit status is not "in progress". In ratfor, this subroutine is essentially:

```
subroutine complete(status)
integer status
```

```
while (status == XS$IP)
   call x$wait(1)
return
end
```

The real difficulty with the documentation is with an application like Ring, when only one system is in the ring. In this case the ring is a loop back to that one system, and the Ring process is talking to itself. If the wait loop given above is used in this case, the Ring process will never receive any of the transmissions that have been made, and space will never become available for the new transmit. In other words, the status will stay XS$IP forever.

Enhancements

X$GVVC. The PRIMENET subroutine call X$GVVC may be used to pass control of a virtual circuit to another process. This would be very useful to Ring when a complex user request requires that a helper process be phantomed, except for the fact that it can only be used to pass a connection to another process on the same system. To be truly useful, it must be possible to pass a connection to any system.

X$STAT. The X$STAT PRIMENET subroutine can be used to determine virtual circuit information about circuits only on the current system. It would be extremely useful if it could return information about circuits on any system. Then it could return the system name and virtual circuit id of the other end of a connection, and it would be possible to find the user name of the owner of the other end of a virtual circuit easily.

X$TRAN. The X$TRAN subroutine call is documented as not informing the transmitting process that the reception has been completed. This is extremely annoying because it means that it is impossible to transmit a response code to a user process, wait until that process has received the code, and then clear the virtual circuit. Saying that "applications requiring assured delivery must implement their own end-to-end acknowledgement" is certainly the easy way out, but it leaves much to be desired. More importantly, it assumes that the processes on both ends of a circuit are intelligent enough to perform an end-to-end acknowledgement. Ring cannot assume that the user process is going to acknowledge that it has received the response since the user program is not under its control. Neither can Ring allow a user connection to remain long past the completion of the user request if no acknowledgement takes place. Ring solves the problem by keeping the time of day when the last activity on a circuit took place, and clearing a circuit when it has been inactive for a sufficiently long period of time.

Bibliography

PRIMENET Guide, DOC3710-190, Second Edition, by Peter A. Neilson, Prime Computer, Incorporated, 500 Old Connecticut Path, Framingham, Massachusetts 01701.

Software Tools Subsystem User's Guide, April 1982, by T. Allen Akin, Terrell L. Countryman, Perry B. Flinn, Daniel H. Forsyth, Jr., Jeanette T. Myers, and Peter N. Wan, School of Information and Computer Science, Georgia Institute of Technology, Atlanta, Georgia 30332.

Appendix

The following is a trace of Ring operating on two systems. The text which is **boldfaced** is commentary, not part of the trace itself.

System GT.A	System GT.B
Ring is brought up on GT.A	
Wednesday, April 6, 1983 3:53 PM	
Attempting connection to GT.B Attempting connection to GT.C Attempting connection to GT.D Attempting connection to GT.E Attempting connection to GT.A Connection received from GT.A Connection received from GT.A Validated transmission to GT.A Validated reception from GT.A Degenerate ring initialized	
The ring is initialized	
	Ring is brought up on GT.B
	Wednesday, April 6, 1983 3:54 PM
	Attempting connection to GT.C Attempting connection to GT.D Attempting connection to GT.E Attempting connection to GT.A
GT.A receives a connection	
Connection received from GT.B	
	GT.B receives the validation connection request
	Connection received from GT.A Validated transmission to GT.A
	New connection validated
New connection validated **Previous connection cleared**	

```
Validated reception from GT.B
Attempting connection to GT.B

                                        GT.B receives a connection

                                      Connection received from GT.A

   GT.A receives a validation
      connection request

Connection received from GT.B
Validated transmission to GT.B

   New connection validated

                                        New connection validated
                                        INITIALIZE request created

                                      Validated reception from GT.A
                                      Transmitted INITIALIZE request

INITIALIZE request received

                                      Created INITIALIZE response

   Initial time set

Transmitted SYNCHRONIZE request at 15:55 on 04/06/83

                                      Synchronized at 15:55 on 04/06/83

                                        New ring is initialized

                                        User issues a BROADCAST

                                      Connection received from GT.B
                                      Connection received from ROY (29)
                                      User request made for ROY (29)

   Roy is not logged on

*** Unknown addressee.
Message broadcast to user ROY

                                      this is a test.
                                      Message broadcast to user ROY

                                        User issues EXECUTE on ALL
```

```
                                    Connection received from GT.B
                                    Connection received from ROY (29)
                                    User request made for ROY (29)

Phantom (58) created for user ROY

                                    Phantom (63) created for user ROY
```

Time is set on the hour

```
Transmitted SYNCHRONIZE request at 16:00 on 04/06/83

                                    Synchronized at 16:00 on 04/06/83
```

User issues EXECUTE on GT.A

```
                                    Connection received from GT.B
                                    Connection received from ROY (29)
                                    User request made for ROY (29)

Phantom (59) created for user ROY
```

4 users issue BROADCASTs

```
                                    Connection received from GT.B
                                    Connection received from ROY (59)
                                    Connection received from GT.B
                                    Connection received from ROY (56)
                                    Connection received from GT.B
                                    Connection received from ROY (63)
                                    User request made for ROY (59)

*** Unknown addressee.
Message broadcast to user ROY

                                    message 4
                                    Message broadcast to user ROY
                                    User request made for ROY (63)

*** Unknown addressee.
Message broadcast to user ROY

                                    message 2
                                    Message broadcast to user ROY
                                    User request made for ROY (56)

*** Unknown addressee.
Message broadcast to user ROY

                                    message 3
                                    Message broadcast to user ROY
                                    Connection received from GT.B
```

```
                                    Connection received from ROY (61)
                                    User request made for ROY (61)

*** Unknown addressee.
Message broadcast to user ROY

                                    message 1
                                    Message broadcast to user ROY
```

User issues TERMINATE

```
                                    Connection received from GT.B
                                    Connection received from ROY (29)
                                    User request made for ROY (29)

TERMINATE request received
```

User receives the response

```
                                    TERMINATE request received
                                    SHUTDOWN request transmitted

Ring SHUTDOWN initiated
Shutdown complete

                                    Ring SHUTDOWN initiated
                                    Shutdown complete
```