The Real-Time Channel Administration Protocol

Anindo Banerjea
email: banerjea@tenet.berkeley.edu
Bruce A. Mah
email: bmah@tenet.berkeley.edu

The Tenet Group
Computer Science Division
University of California, Berkeley
and
International Computer Science Institute
Berkeley, California

ABSTRACT

The Real-time Channel Administration Protocol (RCAP) provides control and administration services for the Tenet real-time protocol suite, a connection-oriented suite of network and transport layer protocols for real-time communication. RCAP performs per-channel reservation of network resources based on worst-case analysis to provide hard guarantees on delay, jitter, and packet loss bounds. It uses a hierarchical approach to provide these guarantees across a heterogeneous internetwork environment.

In this paper, we outline our assumptions and approaches to real-time communication. We then describe the service provided by RCAP, the protocol itself, our plans for implementation, and current status of our research.

1. Introduction

The Real-time Channel Administration Protocol (RCAP) [BanMah91][Low91] provides control services for the Tenet real-time protocol suite, which consists of the Real-time Internet Protocol (RTIP) [VerZha91], the Real-time Message Transport Protocol (RMTP) and the Continuous Media Transport Protocol (CMTP) [WolMor91]. The protocol suite is intended to provide packet based data delivery services with guaranteed delay and jitter (delay variance) bounds, bandwidth guarantees, and bounded packet loss.

We use the term *real-time* to denote network services which provide such guarantees, especially guarantees on delay and jitter bounds. Applications which require such services include digital video and audio, interactive systems, and remote control systems. Current networks based on packet-switching provide no such guarantees. Networks based on

This research was supported by the National Science Foundation and the Defense Advanced Research Projects Agency (DARPA) under Cooperative Agreement NCR-8919038 with the Corporation for National Research Initiatives, by AT&T Bell Laboratories, Hitachi, Ltd., Hitachi America, Ltd., the University of California under a MICRO grant, and the International Computer Science Institute. Bruce A. Mah was also supported by a National Science Foundation Graduate Fellowship. The views and conclusions contained in this document are those of the authors, and should not be interpreted as representing official policies, either expressed or implied, of the U.S. Government or any of the sponsoring organizations.

circuit-switching do provide these guarantees, but at the expense of blocking resources regardless of whether they are being used or not. The Tenet real-time protocol suite is designed to provide guarantees by reserving resources for channels which require guarantees, but making the resources available to non-real-time traffic if they are not actually being used at the time.

RTIP is a connection-oriented network layer protocol which provides delivery of fixed size packets over a simplex channel, with client-specified bounds on packet delay, jitter, and loss rate, at a guaranteed bandwidth which is also chosen by the client. RMTP is a simple transport protocol which uses RTIP as a underlying network layer protocol to provide delivery of arbitrarily-sized messages with guarantees on delay, jitter, loss rate, and bandwidth. CMTP is a more complex transport layer protocol that provides a stream-like interface for continuous media clients that generate data at regular intervals, such as a digital video source. CMTP also uses RTIP as an underlying layer, and provides guarantees similar to RMTP.

RCAP provides the services of channel establishment, channel teardown, and status inquiry for these delivery protocols. It manages resources at nodes along the path of a channel, and provides admission control, to enable the network to provide end-to-end performance guarantees.

Designing and developing RCAP as a separate protocol has the advantages of making both RCAP and the data delivery protocols simpler and easier to debug. This division into control and data delivery portions also allows us to make changes in one without necessarily requiring changes in the other.

2. Assumptions

The Berkeley approach to real-time communication [FerVer89] makes some assumptions about the network, which must be satisfied in order for the guarantees to be met.

1. The network consists of nodes interconnected by logical links, which are either physical links (with bounded latencies) or subnetworks (see Figure 1). The delays across the subnetworks must also be boundable.
2. The nodes are store-and-forward nodes. The logical links may be non-store-and-forward networks (such as circuit-switched networks) provided their total delay can be bounded.
3. The loss rate on the physical links (due to noise) is low enough to be negligible. RCAP provides means of bounding the loss rate due to buffer overflow.
4. To provide jitter guarantees, either it must be possible to bound the jitter on each logical link, or the clocks on the nodes must be at least loosely synchronized.

The Berkeley approach can be used to provide real-time communication on any network which satisfies the above assumptions.

The first assumption may seem rather restrictive, because we seem to be assuming that the delays on the subnetworks are already bounded. However, RCAP provides the mechanism to bound the delays on the subnetworks as well. It is useful to think of the network as consisting of a hierarchy of different levels. The top level network is level 1; in the rest of the paper we refer to this level as the internetwork level, since it is potentially composed of a number of subnetworks linked together by gateways. These subnetworks can be considered level 2, and may in turn be composed of lower level networks, to an arbitrary depth of levels. RCAP provides a mechanism to recursively apply the approach on subnetworks to bound their delays, by structuring its control messages in a hierarchic way to model the hierarchic nature of the network. Once the delay on the subnetwork has been bounded, this bound can then be used to provide an end-to-end bound on the delay in the higher level network.

3. Approach

The Berkeley approach to real-time communication is connection-oriented and involves resource reservation during connection establishment. Connection establishment consists of a control message originating from the sending client and making one round trip along the path of a simplex channel.

During the forward pass the best possible resources are reserved for the channel. The lowest possible delay and jitter, sufficient bandwidth, and a corresponding amount of buffer space are reserved at each node at this time. This means that the network does its best to see if the channel can be established. However, it also means that the resources reserved during this stage are likely to be far better than the requirements of the channel.

The reverse pass allows the network to relax the reservations depending on the client's performance requirements and the actual performance attained by the reservations made on the forward pass. For example, if the end-to-end delay achieved by the forward pass is lower than the client's requirement, the delay reservations can be relaxed on the reverse pass. Jitter reservations are also relaxed, and buffer space may be released. This relaxation is needed to correct the over-allocation of resources during the forward pass.

The Berkeley approach proposes multi-class Earliest Due Date (EDD) as the scheduling mechanism of choice for the level 1 network nodes. This has many advantages. EDD is very amenable to fine grain resource allocation. Bandwidth and delay are decoupled as resources, so that low delay channels do not necessarily also require high bandwidth allocation. The scheduling mechanism is also amenable to worst-case analysis, so that hard guarantees can be made about delay, jitter, and packet loss bounds.

In addition, EDD allows the coexistence of channels having probabilistic guarantees (statistical channels) and best-effort traffic on the same network with channels having hard guarantees (deterministic channels). This allows higher utilization of the network resources, in addition to providing more flexible price versus performance choices for clients.

The subnetworks can use EDD scheduling or any other mechanism, as long as the delay across the subnetworks can be bounded. This means that the network may be heterogeneous, with subnetworks based, for example, on circuit-switching or Asynchronous Transfer Mode (ATM). All that matters is that there should be some mechanism to bound the delay on the subnetwork.

This bounding usually involves an establishment phase and resource reservation. We believe it is a good idea to use the same establishment process for subnetwork levels as for the internetwork level. By doing so, we can preserve the single round-trip establishment scheme and extend it to an arbitrary number of levels of subnetworks. These goals are achieved through the hierarchic structure of RCAP control messages.

4. RCAP Service Description

We now define the services that RCAP must provide to the client. We do this in the context of a simple client-server model, where the client asks the local RCAP entity to provide channels and to perform actions on those channels.

RCAP performs channel administration functions (setup, teardown, and status inquiry) for the data delivery protocols in the Tenet real-time protocol suite. The data delivery protocols are in the network layer (RTIP) and transport layer (RMTP and CMTP).

RCAP exports a number of primitive functions to client programs:

establish_request — The sending client invokes this primitive to request a channel. The client provides its performance requirements, traffic characteristics, and addressing information. RCAP returns a unique channel identifier if the request succeeds.

status — The sender can request information about the state of a channel by invoking the *status* primitive.

register — A receiving client uses the *register* primitive to indicate to RCAP that it is ready to receive connections on a given port.

receive_request — This primitive, when invoked by the receiving client, causes the receiving client to wait until an establishment request arrives from a sending client. It then returns the establishment information from that request, allowing the receiver to accept or deny the request.

accept — The receiving client invokes the *accept* primitive to indicate acceptance of an establishment request.

deny — The receiving client uses the *deny* primitive to reject an establishment request.

unregister — This primitive is used by the receiving client to indicate that it will no longer be accepting requests on a port.

close — Either the sending or receiving client can invoke the *close* primitive to close a channel and release the resources of that channel.

5. Protocol Description

To provide the services described above, RCAP entities on each node in the network need to communicate using control messages. These must be structured hierarchically to capture the hierarchic nature of the network. We now describe our protocol in the above context and elucidate with an example.

The RCAP entities in the network interact by exchanging RCAP control messages. At each node along the path of a channel, the local RCAP entity communicates with the network layer (RTIP) entity to reserve or release resources; at the channel endpoints, RCAP communicates in a similar fashion with the appropriate transport layer entity.

RCAP entities exchange a number of different control messages:

`establish_request` — This control message contains channel parameters provided by the sending client. This message causes RCAP entities along the channel path to reserve resources for the channel, if possible.

`establish_accept` — The `establish_accept` control message indicates that a channel was successfully established. It propagates along the reverse path from the receiver to the sender and causes the RCAP entities in the nodes to confirm, and possibly relax, their resource reservations.

`establish_denied` — This control message indicates that a channel could not be established. It propagates backwards along the channel path towards the sending client and causes the RCAP entities to release resources allocated to the channel.

`status_request` — Sent forward along the channel's path, this message collects information from each node about its status and resource allocation.

`status_report` — This `status_report` message returns the status information from a `status_request` message to the sending client.

`close_request_forward` — This message is a request from the sending client to close the channel. The channel's resources are deallocated.

`close_request_reverse` — This message is a request from the receiving client to close the channel. The channel's resources are deallocated. The effects of the two `close_request` messages are very similar; they are distinguished in order to associate an implicit direction (forward or backward) with each type of control message.

RCAP is designed for use in an internetwork; it uses a hierarchical view in which the links and nodes in a subnetwork are abstracted into a single logical link in the internetwork. Such an approach allows RCAP to utilize the characteristics peculiar to an individual network in order to provide guarantees, yet hide the underlying details of the networks whenever possible. The RCAP control messages reflect this hierarchical view when applicable.

This hierarchical approach is most clearly seen in the `establish_request` RCAP control messages generated by the *establish_request* primitive (see Figure 2). Each message is composed of various records:

Header Record — The Header Record (HR) contains end-to-end parameters for the transport layer entities. There is exactly one HR in each `establish_request` message.

Network Subheader Record — A Network Subheader Record (NSR) marks the start of the records for a network. Each NSR contains end-to-end network-dependent parameters for its associated network and is followed in the message by zero or more Establishment Records.

Establishment Record — An Establishment Record (ER) carries network-dependent, per-node, local parameters.

The `establish_request` control message is created by the RCAP entity on the sending host; the message initially contains only an HR followed by an NSR for the inter-network level. As the control message travels along the channel path, records are added and deleted. Upon the entrance to a lower-level network, the RCAP entity on the entering router adds an NSR for the network; ERs appropriate to that network are then added by the nodes in that network. When exiting a network, the RCAP entity on the exiting router saves the NSR and ERs collected in that network and summarizes the information they contain into a new ER for the next higher-level network. In this way, end-to-end information about lower-level networks is abstracted for higher-level networks.

Figure 3 illustrates a sample `establish_request` message about to exit a network and enter another, as shown. The RCAP establishment message in (i) has traversed the first two networks; the establishment information for those networks has been summarized in $ER_{inet,I}$ and $ER_{inet,II}$. The progress of the message across the third network is recorded in NSR_{III}, $ER_{III,\,1}$, and $ER_{III,\,2}$. Upon arrival at the exiting router of the third network, $ER_{III,\,3}$ is added to describe the last hop as shown in (ii). (iii) shows the entire path through network III summarized in $ER_{inet,III}$. Before entering network 4, NSR_{IV} is added to the message, as shown in (iv).

The `establish_accept` message, sent by the *accept* primitive to confirm resource reservations and indicate acceptance of a channel, has a similar hierarchical structure. ERs are removed from the message by the individual nodes as they confirm the resource reservations. Strings of records that were removed and summarized on the forward pass are processed and reattached to the `establish_accept` message using the state stored in the routers.

Some control messages, such as `establish_denied` and the two `close_request` messages, do not need such a hierarchical structure, as they can convey all necessary information without needing to refer to network-specific parameters. Accordingly, they take on a much simpler structure, containing only the channel identifier and the action to be performed.

The `status_request` and `status_report` control messages have a structure similar to that of the `establish_request` control message in that they are all composed of records which are added or removed by the nodes along the channel path. The major difference is that while the channel establishment process abstracts details of lower-level networks, the status reporting process requires these details to be made available to the client process. Therefore, the status information from the nodes is not summarized or abstracted in any way.

RCAP does not depend on any particular delivery mechanism for its control messages. An ideal implementation would utilize some service providing in-order, reliable message delivery.

6. Implementation Plans

The Tenet Group's implementation plans involve two stages. At first, the protocol suite will be implemented on a local testbed with a small number of nodes and a simple topology. After some experience with this version, the protocols will be ported to the Experimental University Network (XUNET II).

6.1. Local Testbed

The first implementation is targeted for a local testbed with three workstations connected by FDDI links. The FDDI subnetworks, being one hop subnetworks, can be regarded as physical links for the purposes of the protocol. Thus, the links between the nodes are all simple physical links with bounded delays. Only the queueing delays in the nodes are variable and need to be bounded by RCAP. This provides the simplest environment in which to build and test the protocols.

6.2. XUNET II

The Tenet Group plans to implement the protocol suite on the Experimental University Network (XUNET II) in collaboration with AT&T Bell Laboratories. XUNET II will provide a heterogeneous topology [Fer91]. The backbone will be an Asynchronous Transfer Mode (ATM) based wide area network, consisting of XUNET II switches connected by T3 (45 Mb/s) links. At the periphery of the backbone network will be routers, with one or more FDDI rings attached to each router, and host computers attached to the rings.

Thus a typical path on XUNET II can be viewed as composed of the following logical links:

1. Host to router (subnetwork - FDDI)
2. Router to router (subnetwork - ATM backbone)
3. Router to host (subnetwork - FDDI)

As before, the FDDI subnetworks can be treated as physical links for the purposes of the protocol. However, the ATM backbone needs to be treated in a hierarchical manner.

7. Status

As of the date of this writing, the design of the protocol suite is complete and implementation for the local testbed is in progress. The RCAP software has been partially written. Software for RTIP and RMTP has been written and is being tested. The CMTP software is in the process of being written.

At this time, the long-haul fibers for XUNET II are in place and rudimentary routers have been deployed. Switches and more advanced routers are being tested, and are expected to be installed by the end of 1991.

8. Summary

RCAP provides the functions of connection establishment, teardown, and status reporting for the real-time data delivery protocols of the Tenet real-time protocol suite. The protocols in the suite provide data delivery with hard guarantees on the delay and jitter of the packets. Packet loss due to buffer overflow can be eliminated or bounded by correct calculation of buffer requirements.

Separating the control functions from the data delivery functions gives the advantage of simpler design for each of the protocols and a certain amount of insulation, preventing changes in the implementation of one set of functions from adversely affecting the other functions.

The hierarchical nature of RCAP makes it a very flexible protocol that can work in a large variety of environments with little modification. It preserves a single round-trip establishment scheme in an internetwork composed of gateways linking together a number of heterogeneous subnetworks.

The approach is based on worst-case analysis. It allows the provision of hard guarantees with very general assumptions about the underlying network. For applications that do not require hard guarantees, multi-class EDD allows statistical channels and best-effort traffic on the same network as deterministic channels, with correspondingly higher utilization of network resources.

9. References

[BanMah91] A. Banerjea and B. Mah, ''The Design of a Real-Time Channel Administration Protocol,'' unpublished report, University of California at Berkeley and International Computer Science Institute, Berkeley, California, May 1991.

[FerVer89] D. Ferrari and D. Verma, ''A Scheme for Real-Time Channel Establishment in Wide-Area Networks,'' Report TR-89-036, International Computer Science Institute, Berkeley, California, May 1989.

[Fer91] D. Ferrari, ''Tailoring the Tenet Real-Time Protocols to XUNET II,'' unpublished report, University of California at Berkeley and International Computer Science Institute, Berkeley, California, May 1991.

[Low91] C. Lowery, ''Protocols for Providing Performance Guarantees in a Packet-Switching Internet,'' Report TR-91-002, International Computer Science Institute, Berkeley, California, January 1991.

[VerZha91] D. Verma and H. Zhang, ''Design Documents for RTIP/RMTP,'' unpublished report, University of California at Berkeley and International Computer Science Institute, Berkeley, California, May 1991.

[WolMor91] B. Wolfinger and M. Moran, ''A Continuous Media Data Transport Service and Protocol for Real-Time Communication in High Speed Networks,'' *Proc. 2nd Intl. Workshop on Network and Operating System Support for Digital Audio and Video,* Heidelberg (November 1991).

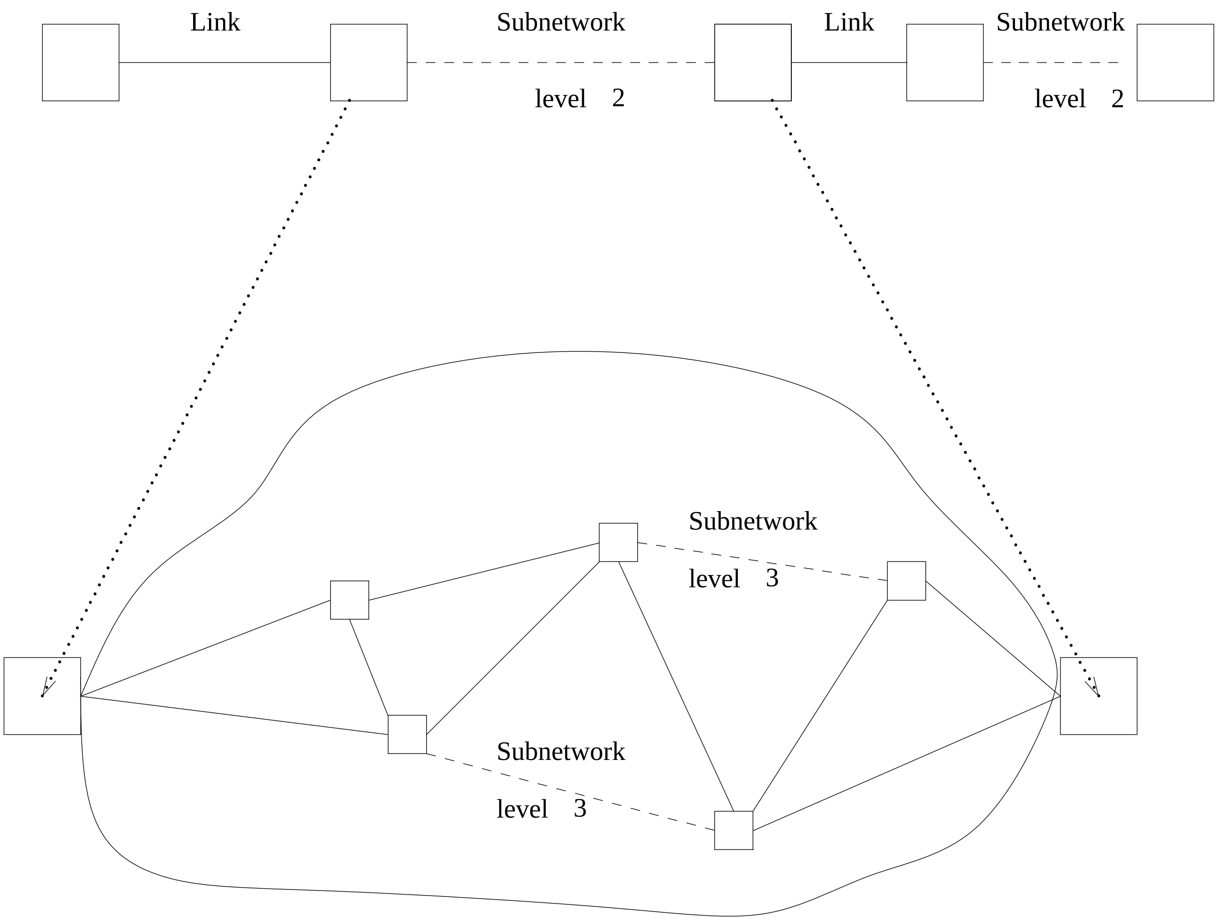

Figure 1. Hierarchic view of the internetwork. The network can be viewed as composed of nodes joined together by links which are either physical links or subnetworks.

The internetwork in this example consists of two physical links and two level 2 subnetworks. One of the level 2 subnetworks is shown in greater detail. It composed of physical links and two level 3 subnetworks.

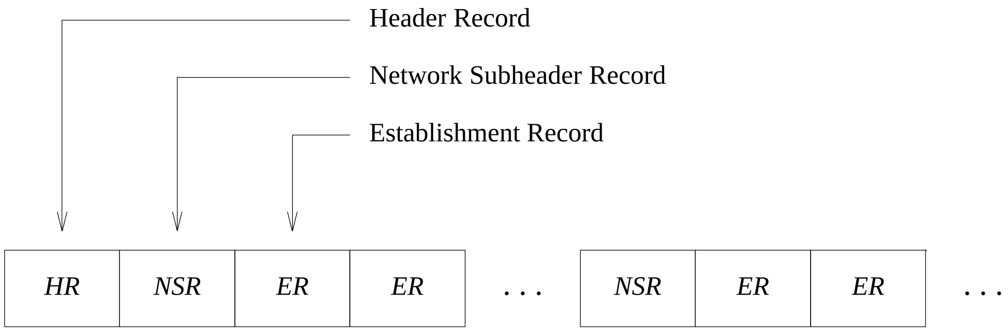

Figure 2. Structure of an `establish_request` message. The start of the message is to the left. The Establishment Records (ERs) following a Network Subheader Record (NSR) contain establishment information on nodes within the network associated with the NSR.

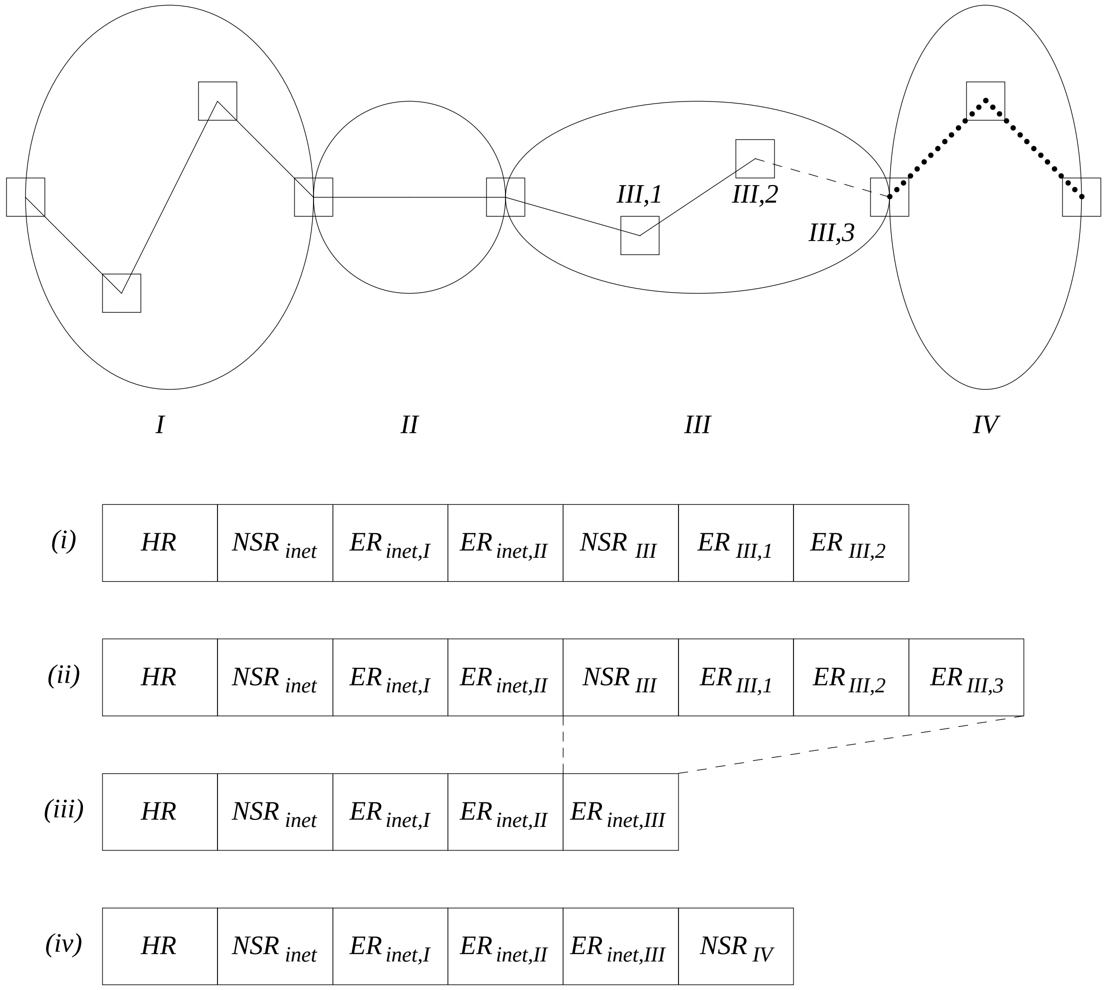

Figure 3. An `establish_request` control message at various points along a path through networks *I*, *II*, *III*, and *IV*. Ellipses represent networks and squares represent nodes along the path. A node in two adjacent networks is a router, which considered to be a part of both networks.

Within the representations of the establishment messages, *HR* denotes a header record. NSR_i represents a Network Subheader Record; with *i* indicating the applicable network (the subscript *inet* indicates that the NSR is for the internetwork level). $ER_{i,j}$ stands for an establishment record for node *j* in the network *i*. Note that in higher levels of the hierarchy, $ER_{i,j}$ can contain router-to-router establishment information for the subnetwork preceding node *j* in network *i*.

(*i*) shows the message just after leaving node *III*, 2. (*ii*) shows the message just after arrival at node *III*, 3; the real-time establishment information computed over the last hop through network *III* has been added in $ER_{III,\,3}$. In (*iii*), the establishment information for network *III* has been summarized in $ER_{inet,III}$. (*iv*) shows the message just prior to entering network *IV*; NSR_{IV} has been attached with the router-to-router parameters for network *IV*.