Faculty of Computing and Information Technology

Department of Robotics and Digital Technology

Technical Report 93/15

The Caroline Project: Service and Protocol Description

J. W. Breen

February 18, 1994

Enquiries:-

Technical Report Coordinator
Robotics and Digital Technology
Monash University
Clayton VIC 3168
Australia

tr.coord@rdt.monash.edu.au   +61 3 565 3402

Contents

List of Tables

List of Figures

Abstract

The Caroline Network is an experimental packet switch network being developed in the Department of Robotics & Digital Technology at Monash University to experiment with switching fabrics and packet protocols. Among the features of the Caroline network are: a bus-based switching fabric, with fixed-length packets travelling between switching nodes on Virtual Circuits; a very light-weight, but robust packet protocol; a routing system for packets which uses a series of virtual circuits between switching nodes. A flooding algorithm is used to determine the optimal inter-node path at the time of the establishment of a Virtual Circuit. The Caroline network provides a connection-oriented unreliable packet delivery service, capable of efficiently supporting a wide range of user services, including those of continuous and variable bit-rates. This paper defines the services of the Caroline network and the inital packet protocol. The two sections can stand alone as the respective service and protocol "standards" of the Caroline network.

Keywords

packet switch, network, switching fabric, flooding, service, protocol

Chapter 1

Introduction

The Caroline Network is an experimental packet switch network being developed in the Department of Robotics & Digital Technology at Monash University. Among the features of the Caroline network are:

1. a bus-based switching fabric, with fixed-length packets travelling between switching nodes on Virtual Circuits;

2. a very light-weight, but robust packet protocol;

3. a routing system for packets which uses a series of virtual circuits between switching nodes. A flooding algorithm is used to determine the optimal inter-node path at the time of the establishment of a Virtual Circuit.

The Caroline network provides a connection-oriented unreliable but sequenced packet delivery service, capable of efficiently supporting a wide range of user services, including those of continuous and variable bit-rates.

The Caroline network is being developed to experiment with switching fabrics and packet protocols. In particular the use of multiple links between switching nodes, and a flooding mechanism for determining the route of a Virtual Circuit are novel aspects of the network.

This paper defines the services of the Caroline network and the inital packet protocol. The two sections can stand alone as the respective service and protocol "standards" of the Caroline network.

Chapter 2

Caroline Service Definition

2.1 Introduction

This section outlines the abstract service definition of the Caroline network in terms of the traditional OSI nomenclature (see ISO TR 8509 and ITU/TSS X.210). It treats the Caroline network as a service layer providing services to a user layer.

It is not intended that user interfaces to the Caroline network implement the primitives of this service interface. Indeed, it is expected that many user interfaces will take the form of hardware/software modules which combine the functions of the Caroline Terminal Node with user protocol and/or Adaptation Layer implementations. The purpose of the service description is to provide an abstract interface model which can separate the interface aspects of the network from the working of the network's internal protocol.

2.2 Definitions and Abbreviations

For the purposes of the document, the following definitions and abbreviations have been used:

- Calling CS user

  A Caroline Service user that initiates a Caroline connection establishment request.

- Called CS user

  A Caroline Service user with whom a calling CS user wishes to establish a Caroline-connection.

- CPL - Caroline Packet Layer
- CS - Caroline Service
- CC - Caroline Connection
- CSAP - Caroline Service Access Point
- CSAPID - Caroline Service Access Point Identifier
- CSDU - Caroline Service Data Unit

2.3 Features of the Caroline service

The Caroline Service has the following features:

1. The means to establish a CC with another CS user for the purpose of exchanging CSDUs. More than one CC can exist between the same pair of CS users.
2. The means of transferring CSDUs on a CC. The transfer of CSDUs is transparent. Note that the transfer of CSDUs is unreliable, in that a CSDU may be corrupted, misdirected or lost without prior or post notification. All CSDUs are of the same fixed length (15 octets).
3. maintenance of sequence between CSDUs
4. The unconditional, and therefore possible destructive, release of CC.

Note that there is no expedited data option, error detection or recovery, flow control or Quality of Service selection.

2.4 Model of the Caroline Service and Connection

2.4.1 Model of the Caroline Service

This document uses the abstract model for a layer service (ISO TR 8509). The primitives are abstract representations of of CSAP interactions. They are solely descriptive and do not represent a specification for implementation.

2.4.2 Model of a Caroline Connection

The operation of a CC is modelled in an abstract way by a pair of queues linking the two CSAPs. There is a queue for each direction of information flow. Each CC is modelled by a separate queue, as depicted in Figure 2.1.

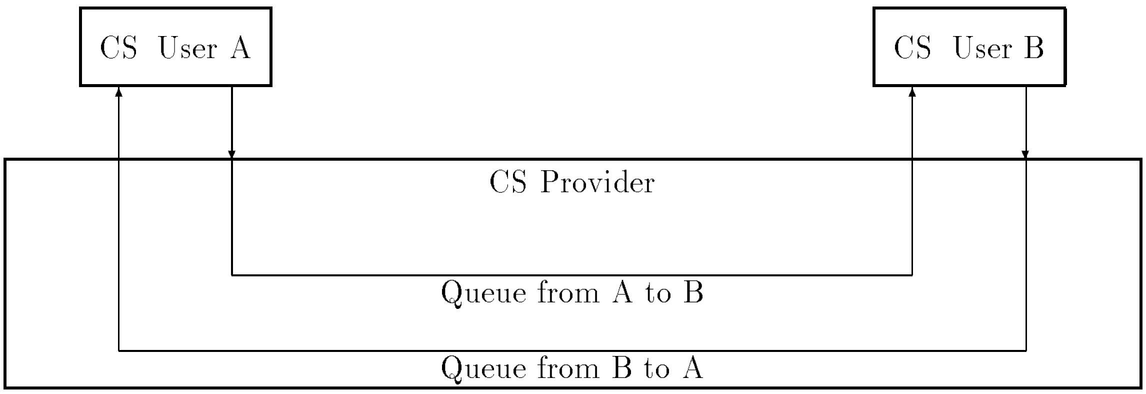

Figure 2.1: Abstract Model of a Caroline Connection

The ability of a TS user to add objects to the queue will be determined by the behaviour of the TS provider removing objects from the queue and transmitting them over the connection.

The objects which may be placed in the queue by the CS user are:

1. connect objects (each representing all the parameters contained in a C-CONNECT request primitive.

2. octets of normal data, in 15-octet CSDUs.

3. disconnect objects (each representing all the parameters contained in a C-DISCONNECT request primitive.)

The only objects placed in the queue by the CS provider are disconnect objects.

2.5 Service Phases and Primitives

The Caroline Service is provided in three phases: CC establishment, data transfer and CC release.

2.5.1 Connection Establishment

The CC establishment phase has the following primitives:

1. C-CONNECT.request (Called Address, Calling Address)

2. C-CONNECT.indication (Called Address, Calling Address)

3. C-CONNECT.confirm (Called Address, Calling Address)

Note that there is no response primitive; called CSAPs are deemed to be always prepared to accept a CC establishment. A User layer which does not wish a particular CC to be maintained may disconnect immediately. Figures 2.2 and 2.3 represent the primitive interactions in the Connection Establishment Phase.

Note also that while there may be multiple connections between CSAP pairs, only one connection can be in the process of being set up at a time.

In the case of "connection collision", i.e. each end of a potential connection issuing overlapping C-CONNECT.request primitives, two separate connections may be established.

2.5.2 Data Transfer

The data transfer phase has the following primitives:

1. C-DATA.request (CS user-data)

2. C-DATA.indication (CS user-data)

Figure 2.4 represents the primitive interaction in the Data Phase.

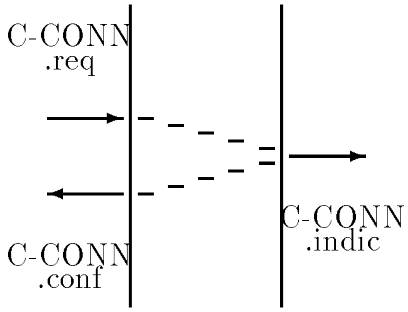

Figure 2.2: Successful Connection Establishment

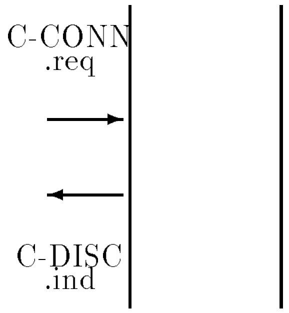

Figure 2.3: Rejection of Connection by CS Provider

2.5.3 Connection Release

The CC release phase has the following primitives:

1. C-DISCONNECT.request
2. C-DISCONNECT.indication (Disconnect reason)

Figures 2.5, 2.6, 2.7 and 2.8 represent the primitive interactions in the Release Phase.

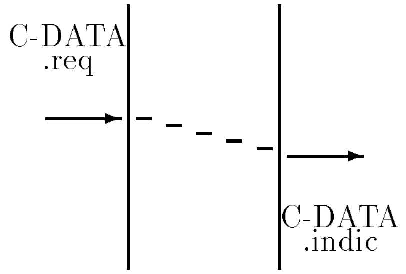

Figure 2.4: Data Transfer

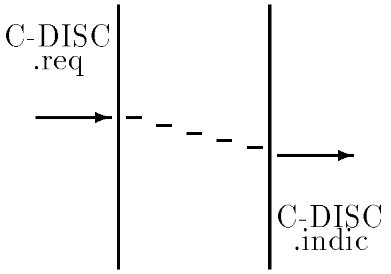

Figure 2.5: CC Release Initiated by CS User

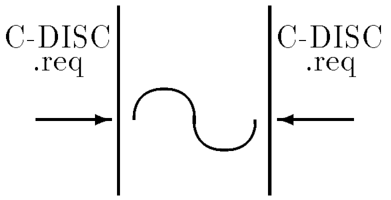

Figure 2.6: CC Release initiated by both CS Users

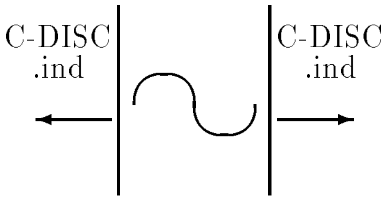

Figure 2.7: CC Release initiated by CS Provider

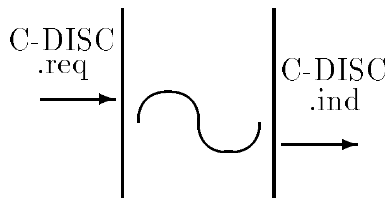

Figure 2.8: CC Release initiated by CS User and CS Provider

Chapter 3

Caroline Protocol Definition

3.1 Introduction

This section describes the Caroline Packet Protocol in terms of its frame/packet structure, and its elements of procedure. The latter has been done in the form of a Finite State Machine. The style adopted for the FSM is that used in ISO 8802/2 (IEEE Std 802.2), which is generally considered to be more easily followed and implementable than the CCITT/ISO style.

The protocol specification is based in part on the narrative description contained in Chapter 9 (Flooding) in the 1991 Caroline Project Report[1], with subsequent revisions proposed by members of the Caroline Project team.

3.2 Protocol Overview

The Caroline Network is connection oriented. All data packets are passed though the network following a path established at the time of setting up a connection.

The path of a Caroline connection will be between two Terminal Nodes via one or more Transit Nodes (see below.) Paths through a Transit Node are characterized by four elements: the incoming link number, the Virtual Circuit Number (VCN) on the incoming link, the outgoing link, the VCN on the outgoing link. The switching hardware in Transit Nodes copies data packets from an incoming link to an outgoing link and modifies the VCN field, according to tables set up at the time of the establishment of the VC. Each Transit Node has a processor which sets up the VCN/Link tables, and which communicates with other nodes via VCN zero, which is reserved on each link for this purpose.

Note that each Caroline connection is Full-Duplex, and is implemented by a pair of Simplex paths through one or more Transit Nodes. The VC Numbers used on a particular link for the two parts of the connection may be different.

Paths are established within the Caroline Network by a Flooding technique. A Terminal Node wishing to establish a connection transmits a Connection Request control packet to the adjacent Transit Node requesting a Virtual Circuit to a specified Terminal Node. The Transit Node will copy the request to all adjacent nodes, except the ones

[1] an unpublished final-year project report in the Bachelor of Computing (Digital Technology).

from which Connection Request packets have already been received in the course of the current connection establishment. When the target Terminal Node receives a Connection Request packet it will reply with a Connection Accepted packet which retraces the path taken by the Request packet. All other Terminal Nodes will return a Connection Rejected packet.

As the Caroline Network structure may be a mesh, it is possible that a Transit node will receive more than one Connection Request packet as part of the establishment of a particular connection, and thus several Connection Accepted packets may result. Each Connection Accepted packet has a Connection Difficulty field which is incremented by each Transit Node through which it passes according to the nodes perception of the load on the links used.

Once a Transit Node passes on Connection Request packets to adjacent nodes, it will expect to receive responses from all those nodes, either Connection Rejected packets or Connection Accepted packets. Once it has received responses from all nodes, it will choose the link which delivered the Connection Accepted packet with the lowest value in the Connection Difficulty field as being the downstream component of the connection, and reject the pending connection on all the other links which delivered Connection Accepted packets by returning Connection Reject packets. It will pass a Connection Accepted packet with an updated Connection Difficulty field to all upstream nodes which delivered Connection Request packets in the course of the current connection establishment.

3.3 Caroline Addresses

The Caroline Network employs 6-octet addresses, which occur in the CREQ PDUs. Addresses are unique, and no internal address structure or hierarchy is used or implied. The address is of a CSAP, and a Terminal Node may have an unlimited number of addresses associated with it, although at most 31 CSAPs can be in use at any instant, this being the currnet upper limit on the number of connections available on each inter-node link.

3.4 Node Types

For the sake of convenience, and as a result of the quite different roles they play in a Caroline network, nodes have been divided into two types:

1. Terminal Nodes. These are the user access points to the network. They alone can initiate Call Request (CREQ) and Call Accept (CACC) PDUs, and they do not participate in the flooding mechanism. By definition, they cannot provide an onward connection to another Node. Typically a Terminal Node will be implemented in a special card attached to a Transit Node, or as a hardware/software module in an end-system.

2. Transit Nodes. This is the basic element in the Caroline packet-switching fabric. It cannot originate or be the destination of connections. As DATA PDU switching in these nodes is carried out in dedicated hardware, the role of the controlling processor and software in Transit Nodes is to establish, maintain and release virtual circuits.

This is achieved by the various control PDUs being actioned by the node processor, which in turn maintains the Link/VCN tables of the node.

3.5 Caroline PDU Structure

All PDUs in the Caroline Network are 16 octets in length. The common structure is as in Figure 3.1.

Octet 0	Bits 5-7	Reserved for future use
	Bits 0-4	Virtual Circuit (VC) Number
Octets 1-15		Dependant on PDU type

Figure 3.1: Common Caroline PDU Structure

DATA PDUs will have values from 1 to 31 in the Virtual Circuit field. All other PDU types have this field set to zero, as they are control packets which will be routed to the processor in each node for analysis and action.

All PDUs other than the DATA PDU will have a one-octet PDU type filed in Octet 1. The possible values of this field are as in Figure 3.2.

CREQ	Connection Request	00000001
CACC	Connection Accepted	00000010
CREJ	Connection Rejected	00000011
CCLO	Connection Closed	00000100
LSTA	Link Status	00000101
CACT	Connection Activated	00000110

Figure 3.2: Caroline PDU Types

3.6 Caroline PDU Types

1. DATA PDU

   The DATA PDU has the format as shown in Figure 3.3:

   A DATA PDU is originated by a Terminal Node, and transmitted to another Terminal Node via one or more Transit Nodes. The Virtual Circuit number is of significance only to each particular inter-node link.

2. LSTA PDU (Link Status)

Octet 0	Bits 5-7	Reserved for future use
	Bits 0-4	Virtual Circuit (VC) Number
Octets 1-15		User Data

Figure 3.3: DATA PDU Structure

Octet 0	Bits 5-7	Reserved for future use
	Bits 0-4	0
Octet 1		LSTA (00000101)
Octets 2-5		VC Status
Octets 6-13		Reserved for future use
Octet 14-15		CheckSum

Figure 3.4: LSTA PDU Structure

The LSTA PDU has the format as shown in Figure 3.4:

The LSTA PDU is transmitted regularly between adjacent nodes of all types on every inter-node link. Failure to receive successive LSTA PDUs will result in a node declaring the loss of that link, and proceeding to terminate all Virtual Circuits passing through that link.

The 4-octet VC Status field consists of a bitmap describing the transmitting node's record of the status of every VC assigned to the link. The field will be regarded as a 32-bit bit-string with one bit reflecting the status of each VC thus: [31 30 .... 2 1 X]. A 0-bit indicates the VC is idle, and a 1-bit indicates the VC is active.

The 2-octet checksum will be generated as follows:

(a) octet 14 shall contain the octet-by-octet exclusive-or of octet 2 through octet 13.

(b) octet 15 shall contain the sum of the products of each of octets 1 through 13 with its ordinal octet position, modulo 255. (Example: 1∗octet1 + 2∗octet2 + ...)

On receipt of a LSTA PDU with a valid checksum, a node will compare the VC Status field with its own record of VC status. Any VC marked as idle in the PDU but active in the node will be released, and the connection closed by transmission of a CCLO PDU on the corresponding VC.

3. CREQ (Connection Request) PDU

The CREQ PDU has the format as shown in Figure 3.5:

Octet 0	Bits 5-7	Reserved for future use
	Bits 0-4	0
Octet 1		CREQ (00000001)
Octet 2	Bits 5-7	Reserved for future use
	Bits 0-4	Reference VCN (incoming link)
Octet 3		Unused
Octets 4-9		Called CSAP Address
Octets 10-15		Calling CSAP Address

Figure 3.5: CREQ PDU Structure

The CREQ PDU is created by a Terminal Node as a result of a C-CONNECT.request primitive, and is transmitted to the adjacent Transit node. The Transit node will transmit the PDU to adjacent Nodes according to the Caroline network Flooding rules.

Upon transmission of CREQ PDU, the transmitting Node will set the Reference VCN field to a vacant outgoing VCN on that link. If a VC is subsequently established on that link, the Reference VCN will become the active outgoing VCN.

4. CACC (Connection Accepted) PDU

   The CACC PDU has the format as shown in Figure 3.6:

Octet 0	Bits 5-7	Reserved for future use
	Bits 0-4	0
Octet 1		CACC (00000010)
Octet 2	Bits 5-7	Reserved for future use
	Bits 0-4	Reference VCN (outgoing)
Octet 3	Bits 5-7	Reserved for future use
	Bits 0-4	Reference VCN (incoming)
Octet 4		CDM: Connection Difficulty Metric
Octets 5-15		Reserved for future use

Figure 3.6: CACC PDU Structure

The CACC PDU is created by a Terminal Node upon receipt of a CREQ PDU bearing a CSAP address associated with that Node. It is transmitted initially to the adjacent Transit Node, and eventually one CACC PDU is returned to the originating Terminal Node.

The CDM field will initially be set to zero. Each Transit Node passing on a CACC PDU may increment the field by the amount associated with the current measured activity on the associated inter-node link. The maximum value of the CDM field is 255.

5. CACT (Connection Activated) PDU

   The CACT PDU has the format as shown in Figure 3.7:

Octet 0	Bits 5-7	Reserved for future use
	Bits 0-4	0
Octet 1		CACT (00000110)
Octet 2	Bits 5-7	Reserved for future use
	Bits 0-4	Reference VCN
Octets 3-15		Reserved for future use

Figure 3.7: CACT PDU Structure

   The CACT PDU is created by the Terminal Node initiating a connection when it receives a CACC PDU. The PDU travels to the destination Terminal Node with the purpose of informing that node that the connection establishment has been completed.

6. CREJ (Connection Rejected) PDU

   The CREJ PDU has the format as shown in Figure 3.8:

Octet 0	Bits 5-7	Reserved for future use
	Bits 0-4	0
Octet 1		CREJ (00000011)
Octet 2	Bits 5-7	Reserved for future use
	Bits 0-4	Reference VCN
Octets 3-15		Reserved for future use

Figure 3.8: CREJ PDU Structure

   The CREJ PDU is created in the following circumstances:

   (a) by a Terminal Node when it receives a CREQ PDU with a CSAP address which is not associated with that node;

   (b) by a Terminal Node when it receives a CREQ PDU with an appropriate CSAP address, but does not have a free VC to include in a CACC PDU.

   (c) by a Transit Node when it receives a CREQ PDU, but does not have a free VC to establish the reverse path.

(d) by a Transit Node upon receipt of a CREJ PDU on the last link to report after a Flood, in which all links have returned CREJ PDUs.

(e) by a Transit Node in response to the CACC PDUs received during a Flood which arrived on links that were not selected to be the successful downstream link of the connection.

(f) by a Transit Node upon expiry of the Flood Timer (T5). It will be sent to all nodes which provided CREQ PDUs, and all nodes which responded with PDUs other than CREJs.

7. CCLO (Connection Closed) PDU

The CCLO PDU has the format as shown in Figure 3.9:

Octet 0	Bits 5-7	Reserved for future use
	Bits 0-4	0
Octet 1		CCLO (00000100)
Octet 2	Bits 5-7	Reserved for future use
	Bits 0-4	Reference VCN
Octets 3		Reserved for future use
Octet 4		Disconnection Reason (for further study)
Octets 5-15		Reserved for future use

Figure 3.9: CCLO PDU Structure

The CCLO PDU is created in the following circumstances:

(a) by a Terminal Node upon receipt of a C-DISCONNECT.request primitive. In this case it is transmitted to the adjacent Transit Node, and from thence eventually to the peer Terminal Node in the connection.

(b) by a Transit Node when it receives a valid LSTA PDU which indicates that it has one more VCs which should be inactivated.

(c) by a Transit Node following a LSTA Receive Timer (T4) timeout. The CCLO PDU is sent on all the VCs which passed through the failed link.

3.7 Timers

3.7.1 Description of Timers

1. T1 Source Connection Establishment Timer

There is one instance of this Timer for each connection in the process of being established by a Terminal Node. This timer is initialized by a Terminal Node when it transmits a CREQ PDU to begin the establishment of a connection. It is stopped on the receipt of a CACC PDU. Expiry of the timer indicates the failure of the connection attempt.

2. T2 (Unused)

3. T3 LSTA Transmit Timer

   There is one instance of this Timer for each node. This timer is initialized by a node each time it completes transmission of LSTA PDUs on all its links to adjacent nodes. On expiry, a further round of LSTA PDUs is transmitted, and the Timer reinitialized.

4. T4 LSTA Receive Timer

   There is one instance of this Timer for each link recognized by a node. This Timer is initialized on expiry, or on the receipt of a valid LSTA packet on the link. After N4 successive expiries of the Timer, the link is declared inactive, and any active VCs associated with the are terminated.

5. T5 Flood Timer

   There is one instance of this Timer for each Flood active in a Transit Node. The Timer is intialized when the Flood is performed, and terminated when all links involved in the Flood have returned appropriate PDUs. On expiry, the Flood process is deemed to have failed somewhere down-stream of this node, and one or more CREJ PDUs are sent as a result.

3.7.2 Timer Values

The following abbreviations are used:

MNT	the maximum number of Transit Nodes in a Connection
UBQD	the upper bound of the queuing delay on a PDU
PTD	PDU Transmit Delay - the transmission time for a PDU

1. T1 Source Connection Establishment Timer

   The optimum value of this timer will be dependant on network size. In practice, it can be the same as the T5 Timer, as a flood failing anywhere in the network will result in a failed connection attempt.

   The minimum value of the Timer is:

   T5 (i.e. $2 * \mathrm{MNT} * (\mathrm{UBDQ} + \mathrm{PTD})$)

2. T2 (Unused)

3. T3 LSTA Transmit Timer

   As this is a watchdog timer, it can have a value in the order of seconds. Too high a value will result in a delay in detecting a link has failed, or in the recognition of the availability of a new link or node. Too short a value will result in a highr load on the node processor and links.

4. T4 LSTA Receive Timer

   This timer should be set to a value of:

   T3 + UBDQ + PTD

5. T5 Flood Timer

   This timer is also dependant on network size, and should be set to the maximum time in which a CACC PDU could be reasonably expected to arrive.

   The minimum value of the Timer is:

   $2 * \mathrm{MNT} * (\mathrm{UBQD} + \mathrm{PTD})$

3.8 State, Event and Action Descriptions

NB: A State Machine will operate for each connection which has been established or is in the process of establishment. The Machine will be identified uniquely by the VCN in use (established connections), the Reference VCN (during connection establishment), the CSAPID, or an appropriate combination of these.

3.8.1 State Descriptions - Terminal Nodes

- IDLE. The connection component is in disconnected mode. It can accept a CREQ PDU from a remote node or, at the request of the service access point user, initiate a CREQ PDU transmission into the network.
- AWAIT_CALL_ACCEPT. The connection component has transmitted a CREQ PDU, and is waiting for a reply.
- AWAIT_ACTIVATION. The connection component has received one or more CREQ PDUs and transmitted CACC PDUs in reply. It is waiting for the arrival of either a CACT PDU or a DATA PDU, either of which will signal the completion of the connection.
- DATA. A packet connection exists between the local and remote CSAPs. Sending and reception of data and control PDUs can proceed.

3.8.2 State Descriptions - Transit Nodes

- IDLE. No flood is in progress for the indicated address pair or reference VCN.
- DATA. A connection exists between a pair of incoming and outgoing virtual circuits. Sending and reception of data and control PDUs can proceed.
- FLOODING. The node has received a CREQ PDU for this address pair, and passed it over at least one link to at least one neighbouring node. Not all replies have arrived.

3.8.3 Event Descriptions - Terminal Nodes

- CONNECT_REQUEST. The user has requested that a connection be established with a remote CSAP. Note that while there may be multiple connections between CSAPs, only one VC can be in the process of being established at a time from each end-point.
- DATA_REQUEST. The user has requested that a data unit be sent to a remote CSAP.
- DISCONNECT_REQUEST. The user has requested that the connection to the remote CSAP be terminated.
- RECEIVE_CREQ_PDU. A remote CSAP has sent a CREQ PDU.
- RECEIVE_CACC_PDU. A remote CSAP has sent a CACC PDU.
- RECEIVE_CREJ_PDU. A CREJ PDU has been received, usually indicating that a requested connection cannot be established on that link.
- RECEIVE_CCLO_PDU. A CCLO PDU has been received indicating that the connection on the indicated VC no longer exists.
- RECEIVE_CACT_PDU. A CACT PDU has been received indicating that the connection establishment has been completed.
- RECEIVE_DATA_PDU. A DATA PDU has been received.
- T1_TIMER_EXPIRED. The VC source establishment timer has expired.

3.8.4 Event Descriptions - Transit Nodes

- RECEIVE_DATA_PDU. A neighbouring node has sent a DATA PDU.
- RECEIVE_CREQ_PDU. A neighbouring node has sent a CREQ PDU.
- RECEIVE_CACC_PDU. A neighbouring node has sent a CACC PDU.
- RECEIVE_CREJ_PDU. A neighbouring node has sent a CREJ PDU.
- RECEIVE_CCLO_PDU. A neighbouring node has sent a CCLO PDU.
- RECEIVE_CACT_PDU. A neighbouring node has sent a CACT PDU.
- T5_TIMER_EXPIRED. The flood timer has expired.

3.8.5 Event Descriptions - Both Node Types

- RECEIVE_LSTA_PDU. A neighbouring node has sent a LSTA PDU.
- T3_TIMER_EXPIRED. The link status transmit timer has expired.
- T4_TIMER_EXPIRED. The link status receive timer has expired.

3.8.6 Action Descriptions - Terminal Nodes

- SEND_CREQ_PDU. Transmit a CREQ PDU.
- SEND_CACC_PDU. Transmit a CACC PDU.
- SEND_CREJ_PDU. Transmit a CREJ PDU.
- SEND_CCLO_PDU. Transmit a CCLO PDU.
- SEND_CACT_PDU. Transmit a CACT PDU.
- START_T1_TIMER. Start the VC source establishment Timer from zero.
- CONNECT_CONFIRM. Inform the user that the requested connection is now available.
- CONNECT_INDICATION. Inform the user that a connection has been opened from a remote user.
- DISCONNECT_INDICATION. Inform the user that the connection is no longer available.
- REPORT_FAULT. Report the details of this state/event to the control plane.
- DATA_INDICATION. The data unit from the received DATA PDU is passed to the user.
- SEND_DATA_PDU. Send a DATA PDU with the data unit supplied by the user with the DATA_REQUEST.
- STOP_T1_TIMER. Stop the VC source establishment timer.

3.8.7 Action Descriptions - Transit Nodes

- FLOOD. Send a CREQ PDU on all out-going links, except those from which CREQs have been received during this flood.
- SEND_CREJ_PDU. Send a CREJ PDU on the appropriate link to indicate that a VC cannot be established on this link. Mark the VC as idle.
- SEND_CACC_PDU. Send a CACC PDU to indicate that a VC requested in a previous CREQ PDU can be established.
- SEND_CCLO_PDU. Send a CCLO PDU to indicate that a particular VC on a link is now inactive.
- SEND_CACT_PDU. Pass on the CACT PDU to the indicated virtual circuit.
- SEND_DATA_PDU. Pass on the DATA PDU to the indicated virtual circuit.
- START_T5_TIMER. Start the T5 (Flood) Timer from zero.

- CLEAR_VC. Mark this VC as idle, and responded if it was included in a flood.
- MAKE_BEST. Mark the VC on which the CACC PDU arrived as the currently best candidate for VC establishment
- FLOOD_UPDATE. Update the flooding table to reflect the arrival of the most recent PDU.
- STOP_T5_TIMER. Stop the flood timer.

3.8.8 Action Descriptions - Both Nodes Types

- CHECK_VC. Compare the bit-map of active VCs in the (valid) LSTA PDU with the internal node tables. Release any VCs which are active, but which the LSTA indicates are idle.
- SEND_LSTA_PDU. Transmit a LSTA PDU into the network (Terminal Node), or on all links to adjacent nodes (Transit Node).
- CLEAR_RETRY_COUNTER. Set the (N4) RETRY_COUNTER associated with the T4 timer to zero.
- LINK_SHUT_DOWN. A total failure of the link to the network. Invoke DISCONNECT_INDICATIONs for each active virtual circuit (Terminal Nodes only). Mark all VCs idle on the failed link. Send CCLO PDUs on all corresponding VCs on other links (Transit Nodes only).
- RESET_T3_TIMER. Restart the T3 (LSTA Transmit) Timer from zero.
- RESET_T4_TIMER. Restart the T4 (LSTA Receive) Timer from zero.

3.9 Protocol Finite-State Tables

The following tables define the Caroline Connection Establishment and Link Status protocols in the form of Finite-State Tables.

STATE TRANSITION TABLE - TERMINAL NODE			
Current State	Event	Action	Next State
IDLE	CONNECT_REQUEST (Inactive address)	SEND_CREQ_PDU START_T1_TIMER	AWAIT_ CACC_PDU
IDLE	CONNECT_REQUEST (Active address)	DISCONNECT_ INDICATION	IDLE
IDLE	RECEIVE_CREQ_PDU (Non-local address)	SEND_CREJ_PDU	IDLE
IDLE	RECEIVE_CREQ_PDU (This Address - VC available)	SEND_CACC_PDU	AWAIT_ ACTIVATION
IDLE	RECEIVE_CREQ_PDU (No VC available)	SEND_CREJ_PDU REPORT_FAULT	IDLE
IDLE	All Other Events	REPORT_FAULT	IDLE
AWAIT_ CACC_PDU	RECEIVE_CACC_PDU	CONNECT_CONFIRM SEND_CACT_PDU	DATA
AWAIT_ CACC_PDU	T1_TIMER_EXPIRED	DISCONNECT_ INDICATION REPORT_FAULT	IDLE
AWAIT_ CACC_PDU	RECEIVE_CREJ_PDU	DISCONNECT_ INDICATION REPORT_FAULT	IDLE
AWAIT_ CACC_PDU	DISCONNECT_ REQUEST	SEND_CREJ_PDU	IDLE
AWAIT_ CACC_PDU	All Other Events	REPORT_FAULT	IDLE
AWAIT_ ACTIVATION	RECEIVE_CREQ_PDU	SEND_CACC_PDU	AWAIT_ ACTIVATION
AWAIT_ ACTIVATION	RECEIVE_CACT_PDU	CONNECT_INDICATION	DATA
AWAIT_ ACTIVATION	RECEIVE_DATA_PDU	CONNECT_INDICATION DATA_INDICATION	DATA
AWAIT_ ACTIVATION	All Other Events	SEND_CREJ_PDU REPORT_FAULT	IDLE
DATA	RECEIVE_DATA_PDU	DATA_INDICATION	DATA
DATA	DATA_REQUEST	SEND_DATA_PDU	DATA
DATA	DISCONNECT_ REQUEST	SEND_CCLO_PDU	IDLE
DATA	RECEIVE_CCLO_PDU	DISCONNECT_ INDICATION	IDLE
DATA	All Other Events	REPORT_FAULT	DATA

Table 3.1: State Transition Table - Terminal Node

STATE TRANSITION TABLE - TRANSIT NODE			
Current State	Event	Action	Next State
IDLE	RECEIVE_CREQ_PDU (VC available)	FLOOD START_T5_TIMER	FLOODING
IDLE	RECEIVE_CREQ_PDU (No VC available)	SEND_CREJ_PDU REPORT_FAULT	IDLE
DATA or IDLE	RECEIVE_CCLO_PDU	CLEAR_VC SEND_CCLO_PDU	IDLE
DATA or IDLE	RECEIVE_CREJ_PDU	CLEAR_VC SEND_CREJ_PDU	IDLE
DATA	RECEIVE_CACT_PDU	SEND_CACT_PDU	DATA
DATA	RECEIVE_DATA_PDU	SEND_DATA_PDU	DATA
FLOODING	RECEIVE_CREQ_PDU	FLOOD_UPDATE	FLOODING
FLOODING	RECEIVE_CACC_PDU (Lower difficulty, Flood incomplete)	FLOOD_UPDATE MAKE_BEST SEND_CREJ_PDU (previous best)	FLOODING
FLOODING	RECEIVE_CACC_PDU (Greater difficulty, Flood incomplete)	FLOOD_UPDATE SEND_CREJ_PDU	FLOODING
FLOODING	RECEIVE_CACC_PDU (Lower difficulty, Flood complete)	MAKE_BEST SEND_CREJ_PDU (previous best) SEND_CACC_PDU STOP_T5_TIMER	IDLE
FLOODING	RECEIVE_CACC_PDU (Greater difficulty, Flood complete)	SEND_CREJ_PDU SEND_CACC_PDU STOP_T5_TIMER	IDLE
FLOODING	RECEIVE_CREJ_PDU (Flood incomplete)	FLOOD_UPDATE	FLOODING
FLOODING	RECEIVE_CREJ_PDU (Flood complete, Dest. found)	SEND_CACC_PDU STOP_T5_TIMER	IDLE
FLOODING	RECEIVE_CREJ_PDU (Flood complete, Dest. not found)	SEND_CREJ_PDU STOP_T5_TIMER	IDLE
FLOODING	T5_TIMER_EXPIRED	SEND_CREJ_PDU (Current best &) Requesting Nodes)	IDLE

Table 3.2: State Transition Table - Transit Node

STATE TRANSITION TABLE - BOTH NODE TYPES			
Current State	Event	Action	Next State
All States	T3_TIMER_EXPIRED	SEND_LSTA_PDU RESET_T3_TIMER	No change
All States	RECEIVE_LSTA_PDU	CHECK_VC RESET_T4_TIMER CLEAR_RETRY_COUNTER	No change
All States	T4_TIMER_EXPIRED and RETRY_COUNT <= N4	RESET_T4_TIMER RETRY_COUNTER++	No Change
All States	T4_TIMER_EXPIRED and RETRY_COUNT > N4	REPORT_FAULT LINK_SHUT_DOWN	IDLE

Table 3.3: State Transition Table - Both Node Types

Chapter 4

Acknowledgements

Many people have been working on the Caroline Project, and have contributed to the development of the concepts included in this document. Their work is most gratefully acknowledged.

In particular, the author would like to mention the members of the student project team which first attempted to state the flooding algorithm: Peter Litwiniuk, Minh Phan, Paul Richardson and Paul Stuart; the project leader Tony McGregor, who proposed the "reverse flood metric" now included in the protocol; and Rik Harris who, along with Tony, provided invaluable feedback on the paper as he implemented the flooding algorithm in the prototype transit node.