Network Working Group
Request for Comments: 1329

P. Kuehn

May 1992

Thoughts on Address Resolution for Dual MAC FDDI Networks

Status of this Memo

This memo provides information for the Internet community. It does not specify an Internet standard. Distribution of this memo is unlimited.

1. Abstract

In this document an idea is submitted how IP and ARP can be used on inhomogeneous FDDI networks (FDDI networks with single MAC and dual MAC stations) by introducing a new protocol layer in the protocol suite of the dual MAC stations. Thus two dual MAC stations are able to do a load splitting across the two rings and use the double bandwidth of 200 Mbits/s as single MAC stations. The new layer is an extension of layer 3. For the user, the higher layer protocols, IP and ARP the property "dual MAC" is transparent. No modification is required in the protocol suite of single MAC stations and transparent bridges.

2. Acknowledgements

This paper is a result of a diploma thesis prepared at the Technical University of Munich, Lehrstuhl fuer Kommunikationsnetze, in co-operation with the Siemens Nixdorf AG. The author would like to thank Jrg Eberspher and Bernhard Edmaier from the university, Andreas Thimmel and Jens Horstmeier from the SNI AG at Augsburg for the helpful comments and discussions.

3. Conventions

Primary MAC, P-MAC	MAC, placed on the primary ring
Secondary MAC, S-MAC	MAC, placed on the secondary ring
Inhomogeneous ring	configuration of a dual FDDI ring with single MAC and dual MAC stations
DMARP	Dual MAC Address Resolution Protocol

4. Assumptions

When a dual FDDI ring wraps, both MACs in a dual MAC station are assumed to remain connected to the ring. ANSI is just investigating whether the Configuration Management in the Station Management of a

FDDI station can be modified to allow this. According to the FDDI SMT standard [1], different addresses are required for all MACs on the primary and the secondary ring.

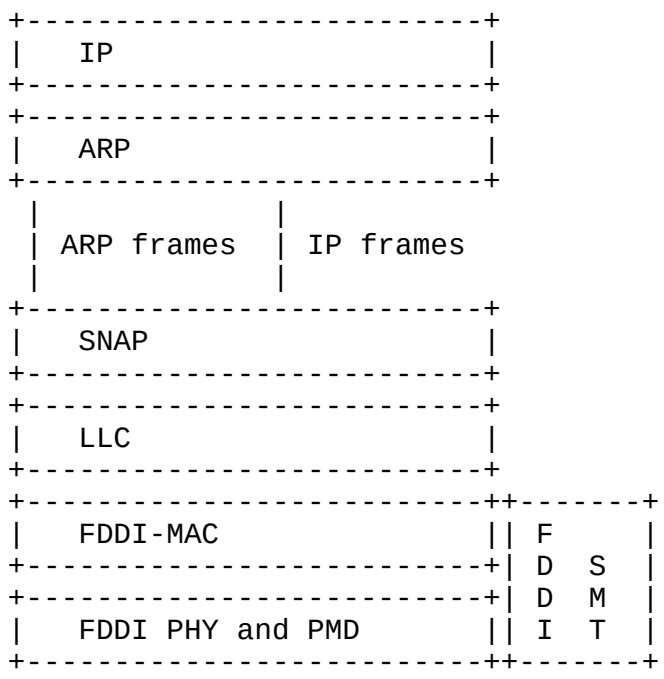
In this paper, the MAC in a single MAC station is assumed to reside on the primary ring. The application of single MAC stations which have their MAC attached to the secondary ring is not precluded, but therefor additional connectivity between the two rings is required. These configurations are beyond the scope of this document.

5. The Application of Transparent Bridges

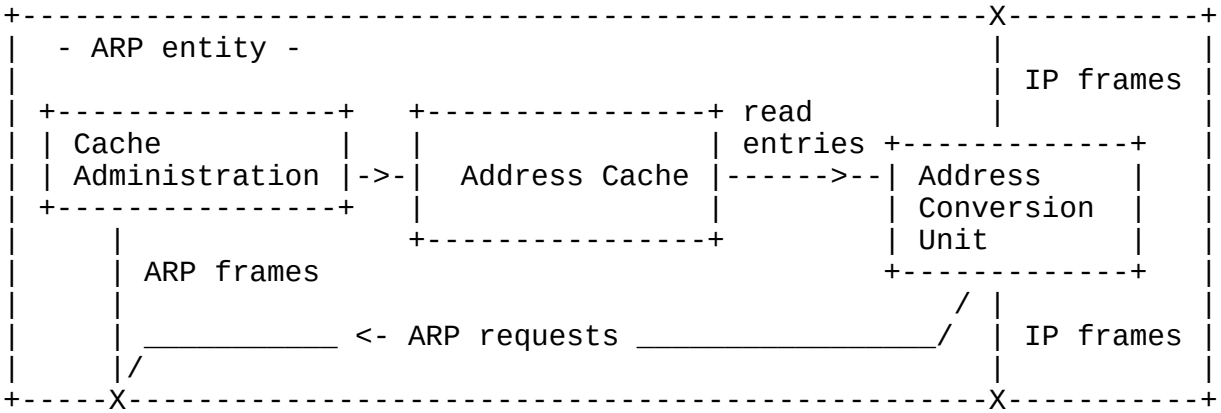
Transparent bridges can provide links to other 802 LANs or further inhomogeneous FDDI rings. The connection between two inhomogeneous FDDI rings can be realized by one or two transparent bridges. When two transparent bridges are used, one transparent bridge links the primary rings, the other the secondary rings. If two secondary rings are connected by a transparent bridge, a path of transparent bridges must exist between the two primary rings. No transparent bridges are allowed between the primary and the secondary ring.

6. Protocol Layers in Single MAC Stations

The new protocol layer, named load sharing layer, is drafted to be introduced only in dual MAC stations. In single MAC stations, IP and ARP are working on top of the Subnetwork Access Protocol (SNAP) 04] and the Logical Link Control protocol (802.2 LLC) [3]. LLC type 1 is used because connectionless services are investigated only.



For the ARP layer, the following model is assumed:

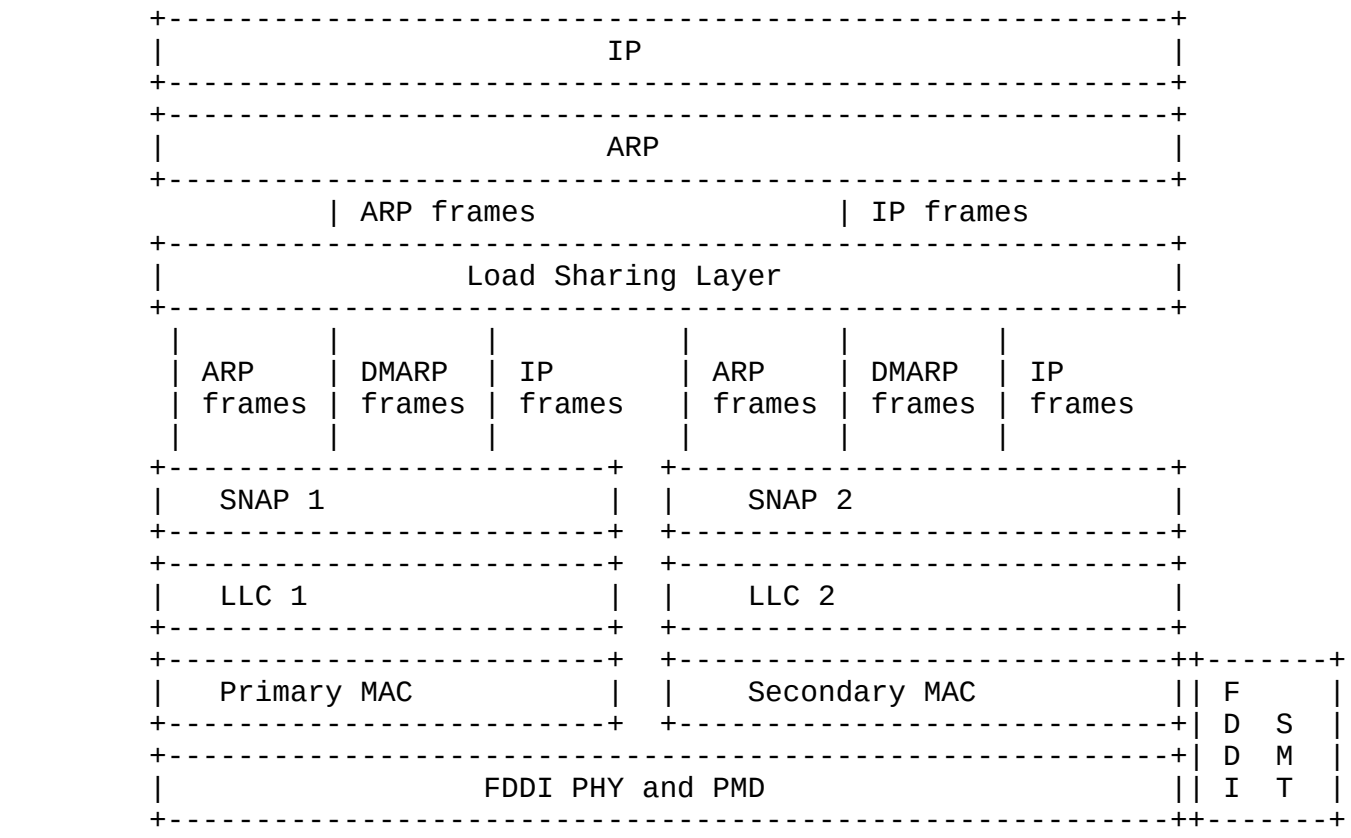


The Address Conversion Unit handles the actual conversion of IP addresses to hardware addresses. For this purpose, it uses the information in the ARP cache. The cache administration communicates with other ARP entities by ARP and creates, deletes and renews the entries in the cache.

7. Protocol Layers in Dual MAC Stations

The load sharing layer provides the same interface to ARP as SNAP does. To exchange information about addresses and reachability, the load sharing entities in dual MAC stations communicate with the Dual

MAC Address Resolution Protocol (DMARP). For the transmission of DMARP frames the SNAP SAP of LLC is used, as for IP and ARP, too. The Organizationally Unique Identifier (OUI) in the SNAP header is set to zero (24 bit), the EtherType field (16 bit) contains a new number indicating DMARP, which is not defined yet.



8. Running Inhomogeneous FDDI Rings

8.1. Exchange of Primary MAC Addresses between Stations

IP and higher layer protocols only use the network independent IP addresses. The ARP entity takes upon the conversion of an IP address to the appropriate hardware address. To make the property dual MAC" transparent, ARP may only know the addresses of MACs on the primary ring. Therefore, the load sharing entity always delivers ARP frames to SNAP 1 for transmission. By this way, communication with ARP is done over the primary ring in normal state. A secondary MAC can receive an ARP frame when the dual ring is wrapped and the destination hardware address is a multicast or broadcast address. These frames will be discarded because they were received twice.

By this way, the associations of IP addresses to primary MAC addresses for the single MAC and dual MAC stations are stored in the ARP cache. The ARP cache contains no secondary MAC addresses.

8.2. Exchange of Secondary MAC Addresses between Dual MAC Stations

The load sharing layer needs to know the secondary MAC addresses of the other dual MAC stations. The DMARP is used to get these addresses. Whenever the load sharing entity delivers an ARP frame to SNAP 1, a DMARP reply frame will be sent on the secondary ring, containing the stations primary and secondary MAC address. The destination hardware address in this DMARP frame is the broadcast MAC address, the EtherType field in the SNAP header identifies DMARP. The IP destination address is copied from the ARP frame. If the ARP frame that was transmitted parallel to the DMARP reply was a request, an ARP reply frame will be sent back to the sending station by the ARP entity in the receiving station. When the load sharing layer in the receiving station delivers this ARP reply frame to SNAP 1, it sends a DMARP reply frame on the secondary ring.

By this way, DMARP exchanges the additionally required secondary MAC addresses between the dual MAC stations. This is done parallel to the exchange of the ARP frames.

8.3. Communication of Dual MAC Stations on Different Dual FDDI Rings

If two inhomogeneous dual FDDI rings are connected by one transparent bridge, dual MAC stations placed on different dual FDDI rings cannot perform a load sharing. If both dual FDDI rings remain in normal state, no DMARP reply frames get from one secondary ring to the other secondary ring. A dual MAC station realizes another dual MAC station placed on the other dual ring as a single MAC station, because it only receives ARP frames from it. If one of the dual rings is wrapped, a DMARP reply frame can get on the primary ring of the other dual ring. A target station on the unwrapped ring receives this DMARP frame by the primary MAC and the load sharing entity stores the contained addresses in an entry in the address cache. This entry is marked with a control bit, named the OR-bit Other ring bit"). No load sharing will be done with a station related to an entry with the OR-bit set.

If both dual FDDI rings are wrapped, the MACs of all stations reside on one ring. Now, dual MAC stations placed on different dual rings can communicate with DMARP. If a DMARP reply frame is received by the primary MAC and no entry exists for the sending station, a new entry with OR-Bit set will be created. Otherwise, the OR-bit will be set in the existing entry. If a DMARP reply frame is received by the secondary MAC and an entry with OR-bit set already exists for the

sending station, the bit will not be reset.

This mechanism provides that no load sharing will be done between Dual MAC stations on different dual rings if the dual rings are linked with one transparent bridge. An additional DMARP error frame is used to provide against errors when a DMARP reply frame gets lost on the ring.

8.4. Timeout of Entries Marked with OR-Bit Set

If a FDDI ring is wrapped, the DMARP reply frames are received by the primary and secondary MACs of the target dual MAC stations. In that case, the entries for dual MAC stations on the same dual ring are also marked with the OR-bit, although the load sharing is possible between these stations.

When an OR-bit in an entry is set for the first time, a timer entity is started. If the timer entity runs out, a DMARP request frame is sent over SNAP 2 to the secondary MAC of the associated target) station. Then the entry will be discarded.

If the request cannot be received by the target station because the network configuration has changed, there is no entry in the address cache for this station any more and no load sharing is computed. If the target station receives the DMARP request frame, it sends back a DMARP reply frame.

8.5. Problems with the Application of Large FDDI Networks

With an increasing number of dual FDDI rings, each one linked together by two transparent bridges, the probability increases, that one of these inhomogeneous dual FDDI rings is wrapped in the moment when two dual MAC stations exchange ARP frames and DMARP replies.

If two dual MAC stations are communicating for the first time, the probability decreases that a load sharing is really computed after the exchange of DMARP replies, although this would be possible according to the network configuration. It relies upon the fact, that DMARP replies get to the primary ring over the wrapped dual ring and only entries marked with the OR-bit set are created. To solve this problem further expedients are invented:

At first, entries in the address cache can be marked read-only by the setting of the R-bit. In dual MAC stations, entries can be written manually for other dual MAC stations that are frequently talked to or that have a special importance. The control bits of these entries cannot be changed by DMARP.

Next, additional control bits are introduced. One of these bits is the Hold-bit (H-bit). When two dual MAC stations exchange ARP frames and DMARP replies to create entries in their address caches, one station starts sending a DMARP reply, first. According to the network state, it sends an additional DMARP error frame, a moment later. Within a maximum period of time (see "Configuring the Timer Parameters"), all frames arrive at the neighbour station and are received by the primary and/or secondary MAC. If the OR-bit was not set for an entry within this period of time, it is clear, that no further DMARP frames will be received, which result in setting the OR-bit. For such an entry the H-bit is set. As the reception of reply and error frames is not sufficient for setting the OR-bit when the H-bit is set, the load sharing is assumed to be sure. The correctness of the H-bit will be verified in relatively long time periods by queries (query and hold frames) at the station associated.

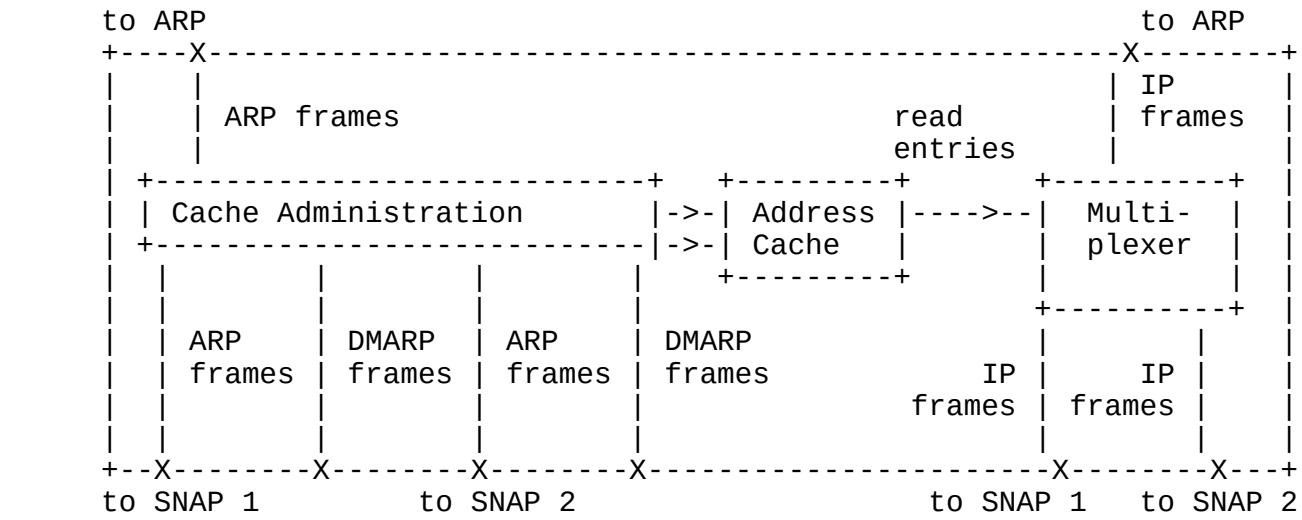
For two communicating stations there exists a possibility to get information from a third station. Always, when the OR-bit is set for an entry in a dual MAC station, a search frame is transmitted by the secondary MAC, containing the own primary MAC address and the primary MAC address of the counter station. If a third station can compute a sure load sharing with both stations (the H-bit is set for the associated entries), the stations can perform a load sharing between them, too. The third station informs these stations by sending found frames to them.

8.6. Multicast and Broadcast Addresses in IP Frames

If the destination hardware address of an IP frame is a multicast or broadcast hardware address, the frame is always delivered to SNAP 1 and sent on the primary ring, because one of the addressed stations could be a single MAC station. IP frames which are delivered to the load sharing entity by SNAP 2 are discarded by the load sharing entity. Thus, the duplication of these frames can be prevented.

9. Internal Structure

One load Sharing entity exists in the load sharing layer. This load sharing entity consists of the address cache, the cache administration and the multiplexer.



9.1. The Address Cache

In the address cache, the associations of primary MAC addresses to secondary MAC addresses are stored for other dual MAC stations on the network. There are no entries for single MAC stations.

Because the OR- and the LS-bit (see table) always have inverted values, one of the bits is redundant. Afterwards the examination of an entry state gets easier by the introduction of both bits, they are defined together. The ARP is able to support other protocol address formats than the IP format. To support this ARP property by DMARP, the protocol type number as used in the ARP frames is stored in every entry of the address cache. So, a dual MAC station is able to communicate with another station with DMARP, even if the other station does not use IP. The numbers used in DMARP frames and the address cache for the protocol type and the address length are taken over from ARP.

name	length	comment
P-MAC address	48 bit	Address of the primary MAC in an other dual MAC station
S-MAC address	48 bit	Address of the secondary MAC in that station
LS-bit	1 bit	A load sharing can be performed with that station ("Load sharing bit")

OR-bit	1 bit	No load sharing may be done with that station ("Other ring bit")
H-bit	1 bit	The load sharing with that station is trusty. ("Hold bit")
Q-bit	1 bit	A query frame was sent to that station, no hold frame was received yet ("Query bit")
R-bit	1 bit	This entry cannot be changed by DMARP ("Read-only bit")
V-bit	1 bit	The entry is valid ("Valid bit")
subscript	32 bit	Unique number, identifying this entry
protocol type	16 bit	Number of the protocol type that was last used in that station

9.2. The Multiplexer

The multiplexer deals with multiplexing the IP frames upon the two FDDI rings. Broadcast and multicast frames are always sent on the primary ring. Otherwise, the contents of the address cache and a load sharing criteria are used to decide on which of the rings an IP frame has to be transmitted. If there is no entry for the primary MAC address of the destination station in the cache, the IP frame is transmitted on the primary ring. If there is an entry for the destination station and the LS-bit is set, a load sharing can be done with this station. Later on a load sharing criteria, which is beyond the scope of this document, decides, which one of the rings is used for transmission. An example for a load sharing criteria is the length of the transmit queues in the MACs. The multiplexer requires an abstract function only, which returns the appropriate ring for the transmission of an actual IP frame.

Additionally, the multiplexer filters the received IP frames: multicast or broadcast frames received from the secondary MAC are discarded.

9.3. The Cache Administration

The cache administration creates and deletes the entries in the address cache. For this purpose, it communicates with other load sharing entities in other dual MAC stations with the DMARP. The cache administration handles the delivery of ARP frames to the ARP and the SNAP entity in the station, respectively.

The cache administration needs three timers for the communication with the DMARP, which have to be supported by the system environment. Each of these timers must support a timer entity for each entry in the address cache, whereby a single one is running at a time.

Supported timer services:

```
TIMER_request(time, name, subscript)
TIMER_response(name, subscript)
TIMER_cancel(name, subscript):
```

A timer entity is started by the service TIMER_request and cancelled by the TIMER_cancel service request. The TIMER_response service indicates that a timer entity has run out. The parameter name is the name of a timer: OR-Entry-Timer, Hold-Timer, or Query-Timer. Each entry in the address cache is uniquely identified by a number subscript). This number is also the number of an associated timer entity. How to dispose these numbers is a question of implementation. The parameter time determines the time period when the timer runs out. This parameter has the value OR-set-timeout for the OR-Entry-Timer, Hold-time for the Hold-Timer and Query-time for the Query-Timer.

9.4. Configuring the Timer Parameters

The OR-set-timeout parameter for the OR-Entry-Timer

The period of time, determined by this parameter, should be essentially longer than the maximum time for a frame to travel around the entire network. The expression entire network means the network which is constituted by the subnetworks linked together with transparent bridges. When entries with OR-bit set are created continuously for a dual MAC station by the timeout mechanism, this parameter determines the periods of time between the consecutive requests that are sent to this station. If the state of the dual FDDI ring changes and an entry with LS-bit set could be created, this parameter additionally determines the maximum time until the new entry is created. (If an entry could not be created by transmission of search frames.) Therefore, the OR-set-timeout parameter should be set to some 10 seconds.

The Hold-time parameter for the Hold-Timer

The period of time, determined by this parameter, should as well be essentially longer than the maximum time for a frame to travel around the entire network. When two stations communicate for the first time, they exchange ARP frames and DMARP replies. The Hold-time parameter determines the period of time until the load sharing is assumed to be accomplished after the setting of the LS-bit. In this period of time, the frames mentioned above must have reached its destination. If an entry would be marked with the H-bit incorrectly, the time until it gets corrected will be relatively long (Query time). Proposed dimension: several minutes.

The Query-time parameter for the Query-Timer

When an entry is marked with LS- and H-bit it is assumed, that load sharing can be performed with the associated station. To allow the correction of a wrong value of the H-bit, the correctness of the H-bit is tested in periods of time, determined by the parameter Query-time. It is tested whether a frame is received, which was sent by the secondary MAC to the secondary MAC address of the target station. (The target station acknowledges the reception of the query frame by a hold frame.) To limit the traffic caused by the query and hold frames, the parameter Query-time should be set to several minutes.

9.5. Format of DMARP Frames

fieldname	length	comment
hardware type	16 bit	1 = "ethernet"
protocol type	16 bit	2048D = "Internet Protocol"
length of hardware addresses	8 bit	Value in octets, 6 for 48 bit MAC addresses
length of protocol addresses	8 bit	Value in octets, 4 for Internet addresses
operation	16 bit	1: "reply" 2: "request" 3: "error" 4: "search" 5: "found"

6: "query"
7: "hold"

1. hardware address ... octets

2. hardware address ... octets

protocol address ... octets
sender

protocol address ... octets
receiver

The value for the field "protocol type" is the same as in ARP frames.

9.6. Contents of DMARP Frames

In the following tables of DMARP frames, the fields containing the length and type of protocol and hardware addresses are omitted.

Format:

Operation	1. hardware address	2. hardware address	protocol address sender	protocol address receiver

Operation = 1 (reply), 2 (request), 3 (error):

Operation	P-MAC address sender	S-MAC address sender	protocol address sender	protocol address receiver

Operation=4 (search)	P-MAC address sender	P-MAC address counter-station	protocol address sender	broadcast protocol address

Operation=5 (found)	P-MAC address sender	S-MAC address counter-station	protocol address sender	broadcast protocol address

Operation=6 (query)	S-MAC address sender	P-MAC address counter-station	protocol address sender	broadcast protocol address

Operation=7 (hold)	P-MAC address sender	S-MAC address sender	protocol address sender	protocol address receiver

Apart from the error frames all frames are sent on the secondary ring. The reply, error and search frames are addressed to the broadcast hardware address. The request, found, query and hold frames are addressed to an individual secondary MAC address.

10. Formal Description

The following description is written in ESTELLE.

10.1. Global Constants, Variables and Types

```
default individual queue;

timescale ...;

type

 PDU_type        = ... ; (* format of a Protocol Data Unit:
                            String of variable length              *)
 HW_addr_type    = ... ; (* format of a 48 bit MAC address         *)
 PR_addr_type    = ... ; (* General: format of a protocol address
                            in an ARP or DMARP frame               *)
 IP_addr_type    = ... ; (* General: format of an IP address       *)
 QoS_type        = ... ; (* General: format of a Quality-of-
                             -Service statement                    *)
 timer_name_type = ... ; (* Type for the name of a system timer    *)

 flag = (reset,set);

var

(*
  The values of these variables are set in the initialization part or
  by external management functions.
*)
```

```
My_P_MAC_addr      : HW_addr_type; (* Address of the MAC, placed on
                                      the primary ring              *)
My_S_MAC_addr      : HW_addr_type; (* Address of the MAC, placed on
                                      the secondary ring            *)
My_IP_address      : IP_addr_type; (* IP address of this station    *)
Broadcast_HW_addr  : HW_addr_type; (* Broadcast MAC address (48 bit) *)
Broadcast_IP_addr  : IP_addr_type; (* Broadcast IP address          *)
dmarp_QoS          : QoS_type;     (* Quality_of_Service-statement
                                      for DMARP frames              *)

ethernet         : integer; (* Type statement in DMARP frames       *)
ip               : integer; (* Number for IP as protocol type       *)
fddi_addr_length : integer; (* Length of a MAC address in octetts   *)
ip_addr_length   : integer; (* Length of a IP address in octetts    *)

OR_set_timeout   : integer; (* Parameter for the OR-Entry-Timer     *)
Query_time       : integer; (* Parameter for the Hold-Timer         *)
Hold_time        : integer; (* Parameter for the Query-Timer        *)
```

10.2. Channels

```
channel SAPchn(User,Provider);
by User :
 UNITDATA_request
 (
   Source_addr  : HW_addr_type;
   Dest_addr    : HW_addr_type;
   QoS          : QoS_type;
   PDU          : PDU_type;
 )
by Provider :
 UNITDATA_indication
 (
   Source_addr  : HW_addr_type;
   Dest_addr    : HW_addr_type;
   QoS          : QoS_type;
   PDU          : PDU_type;
 )

channel System_Access_Point_chn(User,Provider);
by User:
 TIMER_request(Time       : integer;
               Timer_id   : timer_name_type;
               subscript  : integer);

 TIMER_cancel(Timer_id    : timer_name_type;
              subscript   : integer);
```

```
   by Provider:
    TIMER_response(Timer_id  : timer_name_type;
                   subscript : integer);
```

10.3. The Module Header and Interaction Points

```
   module LS_module systemprocess;
    ip LS_ARPSAP     : SAPchn(Provider);
       LS_IPSAP      : SAPchn(Provider);
       SNAP1_ARPSAP  : SAPchn(User);
       SNAP1_LSSAP   : SAPchn(User);
       SNAP1_IPSAP   : SAPchn(User);
       SNAP2_ARPSAP  : SAPchn(User);
       SNAP2_LSSAP   : SAPchn(User);
       SNAP2_IPSAP   : SAPchn(User);
       LS_System_Access_Point : System_Access_Point_chn(User);
   end;
```

10.4. The Modulebody of the Load Sharing Entity

```
   body LS_body for LS_module;

   module multiplexer_module process;
    ip LS_IPSAP    : SAPchn(Provider);
       SNAP1_IPSAP : SAPchn(User);
       SNAP2_IPSAP : SAPchn(User);
   end;

   module cache_administration_module process;
    ip LS_ARPSAP    : SAPchn(Provider);
       SNAP1_ARPSAP : SAPchn(User);
       SNAP1_LSSAP  : SAPchn(User);
       SNAP2_ARPSAP : SAPchn(User);
       SNAP2_LSSAP  : SAPchn(User);
       LS_System_Access_Point : System_Access_Point_chn(User);
   end;

   body cache_administration_body for cache_administration_module;
     (* defined later *)
   end;

   body multiplexer_body for multiplexer_module;
     (* defined later *)
   end;

   modvar
    cache_administration : cache_administration_module;
```

```
    multiplexer          : multiplexer_module;

   initialize
   begin
    ethernet         := 1;
    ip               := 2048;
    fddi_addr_length := 6;
    ip_addr_length   := 4;
    init cache_administration      with cache_administration_body;
    init multiplexer               with multiplexer_body;
    attach LS_IPSAP                to multiplexer.LS_IPSAP;
    attach SNAP1_IPSAP             to multiplexer.SNAP1_IPSAP;
    attach SNAP2_IPSAP             to multiplexer.SNAP2_IPSAP;
    attach LS_ARPSAP               to cache_administration.LS_ARPSAP;
    attach SNAP1_ARPSAP            to cache_administration.SNAP1_ARPSAP;
    attach SNAP1_LSSAP             to cache_administration.SNAP1_LSSAP;
    attach SNAP2_ARPSAP            to cache_administration.SNAP2_ARPSAP;
    attach SNAP2_LSSAP             to cache_administration.SNAP2_LSSAP;
    attach LS_System_Access_Point  to cache_administration.
                                       LS_System_Access_Point;
   end; end;
```

10.5. The Modulebody for the Multiplexer

```
body multiplexer_body for multiplexer_module;

type
  Type_of_addr_type = (individual, multi, broad);
  ring_type         = (primary, secondary);

var
  act_S_MAC_addr : HW_addr_type;

function determ_addrtype(HW_addr: HW_addr_type): Type_of_addr_type;
primitive;
(*
  Returns the type of a hardware address.
  (Individual, multicast or broadcast address)
*)

function get_cacheentry(prtype: integer; P_MAC_addr: HW_addr_type;
 var S_MAC_addr : HW_addr_type): boolean;
primitive;
(*
 Returns the associated secondary MAC address for a given primary MAC
 address and protocol type. If an entry exists, the value TRUE is
 returned.
*)
```

```
function ls_criteria : ring_type;
(*
 Returns the ring on which the actual frame should be transmitted.
*)
primitive;

trans

when LS_IPSAP.UNITDATA_request(Source_addr,Dest_addr,QoS,PDU) begin
 if determ_addrtype(Dest_addr) <> individual then
 output SNAP1_IPSAP.UNITDATA_request(Source_addr,Dest_addr,QoS,PDU);
 else begin
  if get_cacheentry(ip,Dest_addr,act_S_MAC_addr) and
   (ls_criteria=secondary) then
  output SNAP2_IPSAP.UNITDATA_request(My_S_MAC_addr,
   act_S_MAC_addr,QoS,PDU);
  else
  output SNAP1_IPSAP.UNITDATA_request(Source_addr,Dest_addr,QoS,PDU);
 end;
end;

when SNAP1_IPSAP.UNITDATA_indication(Source_addr,Dest_addr,QoS,PDU)
begin
 output LS_IPSAP.UNITDATA_indication(Source_addr,Dest_addr,QoS,PDU);
end;

when SNAP2_IPSAP.UNITDATA_indication(Source_addr,Dest_addr,QoS,PDU)
begin
 if determ_addrtype(Dest_addr) = individual then begin
  Dest_addr := My_P_MAC_addr;
  output LS_IPSAP.UNITDATA_indication(Source_addr,Dest_addr,QoS,PDU);
 end;
end;
```

10.6. The Modulebody for the Cache Administration

```
body cache_administration_body for cache_administration_module;

type
 arp_pdu_type = record
  hwtype        : integer;
  prtype        : integer;
  HW_length     : integer;
  PR_length     : integer;
  operation     : (request,reply);
  HW_sender     : HW_addr_type;
  PR_sender     : PR_addr_type;
  HW_receiver   : HW_addr_type;
```

```
  PR_receiver   : PR_addr_type;
 end;

 dmarp_operation_type = (request,reply,error,search,found,query,hold);

 dmarp_pdu_type = record
  hwtype        : integer;
  prtype        : integer;
  HW_length     : integer;
  PR_length     : integer;
  operation     : dmarpoperation_type;
  HW_1          : HW_addr_type;
  HW_2          : HW_addr_type;
  PR_sender     : PR_addr_type;
  PR_receiver   : PR_addr_type;
 end;

var
 arp_pdu        : arp_pdu_type;
 dmarp_pdu      : dmarp_pdu_type;
 send_pdu       : dmarp_pdu_type;
 act_P_MAC_addr : HW_addr_type;

function my_pr_address(prtype : integer ; praddr : PR_addr_type):
boolean;
(*
 Returns TRUE, if praddr is my station address, the protocol type is
 prtype. (2048d for the Internet protocol)
*)
primitive;

function get_my_pr_addr(prtype : integer) : PR_addr_type;
(*
 Returns my station address, the protocol has the number prtype.
*)

function extract_arp_pdu(PDU : PDU_type) : arp_pdu_type;
(*
 Returns the data contained in an ARP PDU as a record.
*)
primitive;

function extract_dmarp_pdu(PDU : PDU_type) : dmarp_pdu_type;
(*
 Returns the data contained in an DMARP PDU as a record.
*)
primitive;
```

```
function assemble_dmarp_pdu(dmarp_pdu : dmarp_pdu_type): PDU;
(*
 Returns a DMARP PDU from the data in the record.
*)
primitive;

procedure create_entry(prtype: integer; P_MAC_addr: HW_addr_type;
 S_MAC_addr: HW_addr_type; LS_Bit: flag; OR_Bit: flag;
 H_Bit: flag; Q_Bit: flag; R_Bit: flag; V_Bit: flag);
(*
 Creates a new entry in the address cache, if no entry with the given
 primary MAC address or R-bit set to one exists. The protocol type has
 the number prtype. The control bits are set as given in the parameters,
 the LS-bit is set last.
*)
primitive;

function search_entry(prtype : integer; P_MAC_addr : HW_addr_type):
boolean;
(*
 Returns TRUE if an entry with the primary MAC address P_MAC_addr and
 the given protocol type was found in the address cache.
*)
primitive;

procedure update_entry(prtype: integer; P_MAC_addr: HW_addr_type;
 S_MAC_addr: HW_addr_type);
(*
 Searches an entry with the given primary MAC address P_MAC_address and
 updates the secondary MAC address in the entry if the R-bit is set to
 zero.
*)
primitive;

procedure reset_LS_bit(prtype: integer; P_MAC_addr : HW_addr_type);
(*
 Searches an entry with the given primary MAC address P_MAC_address and
 resets the LS-bit if the R-bit is reset.
*)
primitive;

procedure set_Q_bit(prtype: integer; P_MAC_addr : HW_addr_type);
(*
 Searches an entry with the given primary MAC address P_MAC_address and
 sets the Q-bit if the R-bit is reset.
*)
primitive;
```

```
function H_bit_set(prtype: integer; P_MAC_addr : HW_addr_type):
boolean;
(*
 Returns TRUE if an entry exists with H-bit set to one and the given
 P-MAC address.
*)
primitive;

function OR_bit_set(prtype: integer; P_MAC_addr : HW_addr_type):
boolean;
(*
 Returns TRUE if an entry exists with OR-bit set to one and the given
 P-MAC address.
*)
primitive;

function LS_bit_set(prtype: integer; P_MAC_addr : HW_addr_type):
boolean;
(*
 Returns TRUE if an entry exists with LS-bit set to one and the given
 P-MAC address.
*)
primitive;

function Q_bit_set(prtype: integer; P_MAC_addr : HW_addr_type):
boolean;
(*
 Returns TRUE if an entry exists with Q-bit set to one and the given
 P-MAC address.
*)
primitive;

function get_subscript(prtype: integer; P_MAC_addr : HW_addr_type):
integer;
(*
 Returns the subscipt number of an entry with the given primary MAC
 address.
*)
primitive;

function get_broadcast_addr(prtype : integer): PR_addr_type;
(*
 Returns the broadcast protocol address for the given protocol type.
*)

function get_P_MAC_addr(subscript : integer) : HW_addr_type;
(*
 Returns the primary MAC address of the entry with the given subscript
```

```
 number.
*)
primitive;

function get_S_MAC_addr(prtype: integer; P_MAC_addr: HW_addr_type):
 HW_addr_type;
(*
 Returns the secondary MAC address of the station with the given primary
 MAC address.
*)
primitive;

procedure delete_entry(subscript : integer);
(*
 Deletes the entry with the given subscript number if the R-bit is
 reset.
*)
primitive;

function get_pr_type(subscript : integer) : integer;
(*
 Returns the protocol type for the entry with the given subscript
 number.
*)
primitive;

function get_pr_length(prtype : integer) : integer;
(*
 Returns the length of a protocol address.
*)
primitive;


trans

when LS_ARPSAP.UNITDATA_request(Source_addr,Dest_addr,QoS,PDU)
begin
 arp_pdu := extract_arp_pdu(PDU);
 output SNAP1_ARPSAP.UNITDATA_request(Source_addr,Dest_addr,QoS,PDU);
 dmarp_pdu.hwtype               := ethernet;
 dmarp_pdu.prtype               := arp_pdu.prtype;
 dmarp_pdu.HW_length            := fddi_addr_length;
 dmarp_pdu.PR_length            := arp_pdu.PR_length;
 dmarp_pdu.operation            := reply;
 dmarp_pdu.HW_1                 := My_P_MAC_addr;
 dmarp_pdu.HW_2                 := My_S_MAC_addr;
 dmarp_pdu.PR_sender            := arp_pdu.PR_sender;
 dmarp_pdu.PR_receiver          := arp_pdu.PR_receiver;
```

```
 PDU := assemble_dmarp_pdu(dmarp_pdu);
 output SNAP2_LSSAP.UNITDATA_request(My_S_MAC_addr,Broadcast_HW_addr,
  dmarp_QoS,PDU);
end;


when SNAP1_ARPSAP.UNITDATA_indication(Source_addr,Dest_addr,QoS,PDU)
begin
 output LS_ARPSAP.UNITDATA_indication(Source_addr,Dest_addr,QoS,PDU);
end;

when SNAP2_ARPSAP.UNITDATA_indication(Source_addr,Dest_addr,QoS,PDU)
begin end;

when SNAP1_LSSAP.UNITDATA_indication(Source_addr,Dest_addr,QoS,PDU)
begin
 dmarp_pdu := extract_dmarp_pdu(PDU);
 if ((dmarp_pdu.operation = error) or (dmarp_pdu.operation = reply))
 then begin
  if my_pr_address(dmarp_pdu.prtype,dmarp_pdu.PR_receiver) then begin
   if not H_bit_set(dmarp_pdu.prtype,dmarp_pdu.HW_1) then begin
    if not OR_bit_set(dmarp_pdu.prtype,dmarp_pdu.HW_1) then begin
     if LS_bit_set(dmarp_pdu.prtype,dmarp_pdu.HW_1) then begin
      output LS_System_Access_point.TIMER_cancel(
       "Hold_Timer",get_subscript(dmarp_pdu.prtype,dmarp_pdu.HW_1));
      create_entry(dmarp_pdu.prtype,dmarp_pdu.HW_1,dmarp_pdu.HW_2,
       reset,set,reset,reset,reset,set);
     end;
     output LS_System_Access_point.TIMER_request(
      OR_set_timeout,"OR_Entry_Timer",
      get_subscript(dmarp_pdu.prtype,dmarp_pdu.HW_1));
     send_pdu.hwtype    := ethernet;
     send_pdu.prtype    := dmarp_pdu.prtype;
     send_pdu.HW_length := fddi_addr_length;
     send_pdu.PR_length := dmarp_pdu.PR_length;
     send_pdu.operation := search;
     send_pdu.HW_1      := My_P_MAC_addr;
     send_pdu.HW_2      := dmarp_pdu.HW_1;
     send_pdu.PR_sender := get_my_pr_addr(dmarp_pdu.prtype);
     send_pdu.PR_receiver := get_broadcast_addr(dmarp_pdu.prtype);
     PDU := assemble_dmarp_pdu(dmarp_pdu);
     output SNAP2_LSSAP.UNITDATA_request(
      My_S_MAC_addr,Broadcast_HW_addr,dmarp_QoS,PDU);
    end else begin
     if dmarp_pdu.operation=error then
     update_entry(dmarp_pdu.prtype,dmarp_pdu.HW_1,dmarp_pdu.HW_2);
    end;
   end else begin
```

```
    if dmarp_pdu.operation = error then
    update_entry(dmarp_pdu.prtype,dmarp_pdu.HW_1,dmarp_pdu.HW_2);
   end;
  end else begin
   if my_pr_address(dmarp_pdu.prtype,dmarp_pdu.PR_sender) and
    (dmarp_pdu.operation = reply) then begin
    dmarp_pdu.operation := error;
    PDU := assemble_dmarp_pdu(dmarp_pdu);
    output SNAP1_LSSAP.UNITDATA_request(
     My_P_MAC_addr,Broadcast_HW_addr,dmarp_QoS,PDU);
   end else begin
    if dmarp_pdu.operation=error and
     search_entry(dmarp_pdu.prtype,dmarp_pdu.HW_1) then
    update_entry(dmarp_pdu.prtype,dmarp_pdu.HW_1,dmarp_pdu.HW_2);
end; end; end; end;


when SNAP2_LSSAP.UNITDATA_indication(Source_addr,Dest_addr,QoS,PDU)
begin
 dmarp_pdu := extract_dmarp_pdu(PDU);
 if (dmarp_pdu.operation = found) and
  my_pr_address(dmarp_pdu.prtype,dmarp_pdu.PR_receiver) then begin
  if not H_bit_set(dmarp_pdu.prtype,dmarp_pdu.HW_1) then begin
   if OR_bit_set(dmarp_pdu.prtype,dmarp_pdu.HW_1) then begin
    output LS_System_Access_Point.
     TIMER_cancel("OR_Entry_Timer",
     get_subscript(dmarp_pdu.prtype,dmarp_pdu.HW_1));
   end;
   if LS_bit_set(dmarp_pdu.prtype,dmarp_pdu.HW_1) then begin
    output LS_System_Access_Point.
     TIMER_cancel("Hold_Timer",
     get_subscript(dmarp_pdu.prtype,dmarp_pdu.HW_1));
   end;
   create_entry(dmarp_pdu.prtype,dmarp_pdu.HW_1,dmarp_pdu.HW_2,
    set,reset,set,reset,reset,set);
   output LS_System_Access_Point.TIMER_request(Query_time,"Query_Timer",
    get_subscript(dmarp_pdu.prtype,dmarp_pdu.HW_1));
  end;
 end else begin
  if (dmarp_pdu.operation = reply) or
   (dmarp_pdu.operation = request) then begin
   if search_entry(dmarp_pdu.prtype,dmarp_pdu.HW_1) then
    update_entry(dmarp_pdu.prtype,dmarp_pdu.HW_1,dmarp_pdu.HW_2);
  end;
  if (dmarp_pdu.operation=request) and
   my_pr_address(dmarp_pdu.prtype,dmarp_pdu.PR_receiver) then begin
   send_pdu.hwtype      := dmarp_pdu.hwtype;
   send_pdu.prtype      := dmarp_pdu.prtype;
```

```
   send_pdu.HW_length   := fddi_addr_length;
   send_pdu.PR_length   := dmarp_pdu.PR_length;
   send_pdu.operation   := reply;
   send_pdu.HW_1        := My_P_MAC_addr;
   send_pdu.HW_2        := My_S_MAC_addr;
   send_pdu.PR_sender   := get_my_pr_addr(dmarp_pdu.prtype);
   send_pdu.PR_receiver := dmarp_pdu.PR_sender;
   PDU := assemble_dmarp_pdu(dmarp_pdu);
   output SNAP2_LSSAP.UNITDATA_request(
    My_S_MAC_addr,Broadcast_HW_addr,dmarp_QoS,PDU);
  end else begin
   if my_pr_address(dmarp_pdu.prtype,dmarp_pdu.pr_receiver) then begin
    case dmarp_pdu.operation of
     reply: begin
      if not ( OR_bit_set(dmarp_pdu.prtype,dmarp_pdu.HW_1) or
       LS_bit_set(dmarp_pdu.prtype,dmarp_pdu.HW_1) )then begin
       create_entry(dmarp_pdu.prtype,dmarp_pdu.HW_1,dmarp_pdu.HW_2,
        set,reset,reset,reset,reset,set);
       output LS_System_Access_Point.TIMER_request(Hold_time,
        "Hold_Timer",get_subscript(dmarp_pdu.prtype,dmarp_pdu.HW_1));
      end;
     end;

     error: begin
      if not ( OR_bit_set(dmarp_pdu.prtype,dmarp_pdu.HW_1) or
       H_bit_set(dmarp_pdu.prtype,dmarp_pdu.HW_1) ) then begin
       if LS_bit_set(dmarp_pdu.prtype,dmarp_pdu.HW_1) then
       output LS_System_access_point.TIMER_cancel(
        "Hold_Timer",get_subscript(dmarp_pdu.prtype,dmarp_pdu.HW_1));
       create_entry(dmarp_pdu.prtype,dmarp_pdu.HW_1,dmarp_pdu.HW_2,
        reset,set,reset,reset,reset,set);
       output LS_System_access_point.TIMER_request(
        OR_set_timeout,"OR_Entry_Timer",
        get_subscript(dmarp_pdu.prtype,dmarp_pdu.HW_1));
       send_pdu.hwtype          := ethernet;
       send_pdu.prtype          := dmarp_pdu.prtype;
       send_pdu.HW_length       := fddi_addr_length;
       send_pdu.PR_length       := dmarp_pdu.PR_length;
       send_pdu.operation       := search;
       send_pdu.HW_1            := My_P_MAC_addr;
       send_pdu.HW_2            := dmarp_pdu.HW_1;
       send_pdu.PR_sender       := get_my_pr_addr(dmarp_pdu.prtype);
       send_pdu.PR_receiver     := get_broadcast_addr(dmarp_pdu.prtype);
       PDU := assemble_dmarp_pdu(dmarp_pdu);
       output SNAP2_LSSAP.UNITDATA_request(
        My_S_MAC_addr,Broadcast_HW_addr,dmarp_QoS,PDU);
      end;
     end;
```

```
search: begin
 if not (dmarp_pdu.HW_1=My_P_MAC_addr or
  dmarp_pdu.HW_2=My_P_MAC_addr) then begin
  if H_bit_set(dmarp_pdu.prtype,dmarp_pdu.HW_1) and
   H_bit_set(dmarp_pdu.prtype,dmarp_pdu.HW_2) then begin
   send_pdu.hwtype      := ethernet;
   send_pdu.prtype      := dmarp_pdu.prtype;
   send_pdu.HW_length   := fddi_addr_length;
   send_pdu.PR_length   := dmarp_pdu.PR_length;
   send_pdu.operation   := found;
   send_pdu.HW_1        := dmarp_pdu.HW_2;
   send_pdu.HW_2        := get_S_MAC_addr(dmarp_pdu.prtype,
                           dmarp_pdu.HW_2);
   send_pdu.PR_sender   := get_my_pr_addr(dmarp_pdu.prtype);
   send_pdu.PR_receiver := get_broadcast_addr(dmarp_pdu.prtype);
   PDU := assemble_dmarp_pdu(send_pdu);
   output SNAP2_LSSAP.UNITDATA_request(My_S_MAC_addr,
    get_S_MAC_addr(dmarp_pdu.prtype,dmarp_pdu.HW_1),dmarp_QoS,PDU);
   send_pdu.HW_1 := dmarp_pdu.HW_1;
   send_pdu.HW_2 := get_S_MAC_addr(dmarp_pdu.prtype,
    dmarp_pdu.HW_1);
   PDU := assemble_dmarp_pdu(send_pdu);
   output SNAP2_LSSAP.UNITDATA_request(My_S_MAC_addr,
    get_S_MAC_addr(dmarp_pdu.prtype,dmarp_pdu.HW_2),dmarp_QoS,PDU);
  end;
 end;
end;


Query: begin
 if dmarp_pdu.HW_2 = My_P_MAC_addr then begin
  send_pdu.hwtype          := ethernet;
  send_pdu.prtype          := dmarp_pdu.prtype;
  send_pdu.HW_length       := dmarp_pdu.HW_length;
  send_pdu.PR_length       := dmarp_pdu.PR_length;
  send_pdu.operation       := hold;
  send_pdu.HW_1            := My_P_MAC_addr;
  send_pdu.HW_2            := My_S_MAC_addr;
  send_pdu.PR_sender       := get_my_pr_addr(dmarp_pdu.prtype);
  send_pdu.PR_receiver     := dmarp_pdu.PR_sender;
  PDU := assemble_dmarp_pdu(send_pdu);
  output SNAP2_LSSAP.UNITDATA_request(
   My_S_MAC_addr,dmarp_pdu.HW_1,dmarp_QoS,PDU);
 end;
end;
Hold: begin
 if H_bit_set(dmarp_pdu.prtype,dmarp_pdu.HW_1) then
 reset_Q_bit(dmarp_pdu.prtype,dmarp_pdu.HW_1);
```

```
     end;
    end;
   end;
  end;
 end;
end;


when LS_System_Access_Point.TIMER_response(Timer_name,subscript) begin
case Timer_name of
 "OR_Entry_Timer": begin
  act_P_MAC_addr := get_P_MAC_addr(subscript);
  if OR_bit_set(get_pr_type(subscript),act_P_MAC_addr) then begin
   send_pdu.hwtype      := ethernet;
   send_pdu.prtype      := get_pr_type(subscript);
   send_pdu.HW_length   := fddi_addr_length;
   send_pdu.PR_length   := get_pr_length(send_pdu.prtype);
   send_pdu.operation   := request;
   send_pdu.HW_1        := My_P_MAC_addr;
   send_pdu.HW_2        := My_S_MAC_addr;
   send_pdu.PR_sender   := get_my_pr_addr(send_pdu.prtype);
   send_pdu.PR_receiver := get_broadcast_addr(send_pdu.prtype);
   PDU := assemble_dmarp_pdu(send_pdu);
   output SNAP2_LSSAP.UNITDATA_request(
    My_S_MAC_addr,get_S_MAC_addr(send_pdu.prtype,act_P_MAC_addr),
    dmarp_QoS,PDU);
   delete_entry(subscript);
  end;
 end;
 "Hold_Timer": begin
  act_P_MAC_addr := get_P_MAC_addr(subscript);
  if (not H_bit_set(get_pr_type(subscript),act_P_MAC_addr)) and
   LS_bit_set(get_pr_type(subscript),act_P_MAC_addr) then begin
   set_H_bit(get_pr_type(subscript),act_P_MAC_addr);
   output LS_System_Access_point.TIMER_request(
    Query_time,"Query_Timer",subscript);
  end;
 end;
 "Query_Timer": begin
  act_P_MAC_addr       := get_P_MAC_addr(subscript);
  send_pdu.hwtype      := ethernet;
  send_pdu.prtype      := get_pr_type(subscript);
  send_pdu.HW_length   := fddi_addr_length;
  send_pdu.PR_length   := get_pr_length(send_pdu.prtype);
  send_pdu.PR_sender   := get_my_pr_addr(send_pdu.prtype);
  send_pdu.PR_receiver := get_broadcast_addr(send_pdu.prtype);
  if Q_bit_set(get_pr_type(subscript),act_P_MAC_addr) then begin
   send_pdu.HW_1      := My_P_MAC_addr;
```

```
   send_pdu.HW_2      := My_S_MAC_addr;
   send_pdu.operation := request;
   PDU := assemble_dmarp_pdu(send_pdu);
   output SNAP2_LSSAP.UNITDATA_request(
    My_S_MAC_addr,get_S_MAC_addr(send_pdu.prtype,act_P_MAC_addr),
    dmarp_QoS,PDU);
   delete_entry(subscript);
  end else begin
   send_pdu.HW_1      := My_S_MAC_addr;
   send_pdu.HW_2      := get_P_MAC_addr(subscript);
   send_pdu.operation := query;
   PDU := assemble_dmarp_pdu(send_pdu);
   output SNAP2_LSSAP.UNITDATA_request(
    My_S_MAC_addr,get_S_MAC_addr(send_pdu.prtype,send_pdu.HW_2),
    dmarp_QoS,PDU);
   set_Q_bit(send_pdu.prtype,send_pdu.HW_2);
end; end; end; end; end; (* body *)
```

11. Summary

The introduction of the load sharing layer in the protocol layering of the dual MAC stations allows the application of IP and ARP on inhomogeneous FDDI rings. The protocol suite of single MAC stations needs no modification.

By the load sharing layer, the property "dual MAC" is transparent for ARP, IP and the higher layer protocols.

In dual MAC stations, any load sharing criteria may be implemented in the multiplexer of the load sharing entity. The conversion of addresses, the exchange of address and reachability information between dual MAC stations and the proper transmission of multicast and broadcast frames is taken upon by the load sharing entity.

12. References

[1] ANSI, "FDDI Station Management (SMT)", ANSI X3T9/90-X3T9.5/84-49 Rev 6.2, May 1990.

[2] ANSI, "FDDI Media Access Control (MAC-2)", X3T9/90-X3T9.5/88-139 Rev 3.2, June 1990.

[3] ISO, "Information processing systems- Local area networks- Part 2: Logical link control", ISO 8802-2:1989, August 1989.

[4] IEEE, "Draft Standard P802.1A Overview and Architecture", P802.1A/D9-89/74, September 1989.

[5] Plummer, C., "An Ethernet Address Resolution Protocol --or-- Converting Network Protocol Addresses to 48.bit Ethernet Address for Transmission on Ethernet Hardware", RFC 826, MIT, November 1982.

[6] Reynolds, J., and Postel, J., "Assigned Numbers", RFC 1060, USC/Information Sciences Institute, March 1990.

[7] Postel, J., "Internet Protocol", RFC 791, USC/Information Sciences Institute, September 1981.

[8] Katz, D., "A Proposed Standard for the Transmission of IP Datagrams over FDDI Networks", RFC 1188, Merit/NSFNET, October 1990.

[9] Internet Engineering Task Force, Braden, R., Editor, "Requirements for Internet Hosts -- Communication Layers", RFC 1122, IETF, October 1989.

[10] Katz, D., "The Use of Connectionless Network Layer Protocols over FDDI Networks", Merit/NSFNET, 1990.

13. Security Considerations

Security issues are not discussed in this memo.

14. Author's Address

Peter Kuehn
Raiffeisenstrasse 9b
8933 Untermeitingen
Germany

Phone: .. 82 32 / 7 46 02
EMail: thimmela@sniabg.wa.sni.de