1 95-1082R3
2
3 PROJECT: ATM Forum Technical Committee
4 LAN Emulation Sub Working Group
5
6 **
7
8 SOURCE: LAN Emulation SWG
9 Editor: John D. Keene
10 Interphase Corp.
11 13800 Senlac
12 Dallas, TX 75234
13
14 Phone: (214)654-5376
15 Fax: (214)654-5500
16 E-mail: jkeene@iphase.com
17
18 **
19
20 TITLE: LAN Emulation over ATM Version 2 - LNNI Specification - Draft 3
21
22 **
23
24 Date: April 15th, 1996
25
26 **
27
28 ABSTRACT: This document contains current draft of revision 2.0 of the LNNI specification.
29
30 **
31
32 Distribution List: LAN Emulation SWG
33
34 **
35 Notice: This contribution has been prepared to assist the ATM Forum. This proposal is made by
36 Interphase Corp. as a basis of discussion. This contribution should not be construed as a binding
37 proposal on Interphase Corp. Specifically, Interphase reserves the right to amend or modify the
38 statements made herein.
39 **

Change Log

Revision	Changes
Initial Draft	Created baseline document based on outline used by LANE 1.0 LUNI specification Incorporated text into baseline as per Motion 9 in 95-0811.
R1	Added text from 95-1177 to section 2.1 per Motion 8 (Oct '95). Added editor's notes for Motions 10 and 11 (Oct '95). Added text from 1176 as per Motion 12 (Oct '95). Added reference presented in 95-1159 to Section 3 from LUNI as per Motions 13 and 14 (Oct '95). Added text from 95-1174 as per Motion 15 (Oct '95).
R2	Added text from 95-1575 (section 2) to Section 3 - editorial comments from contribution were retained.
R3	Added text from 96-0233 Added text from 96-0235 Changed outline and moved text extensively within Section 2 and to Section 5.

The ATM Forum

Technical Committee

LAN Emulation over ATM Version 2 - LNNI Specification - Draft 3

ATM_Forum/95-1082R3

February 1996

(C) 1995 The ATM Forum. All Rights Reserved. No part of this publication may be reproduced in any form or by any means.

The information in this publication is believed to be accurate as of its publication date. Such information is subject to change without notice and the ATM Forum is not responsible for any errors. The ATM Forum does not assume any responsibility to update or correct any information in this publication. Notwithstanding anything to the contrary, neither The ATM Forum nor the publisher make any representation or warranty, expressed or implied, concerning the completeness, accuracy, or applicability of any information contained in this publication. No liability of any kind shall be assumed by The ATM Forum or the publisher as a result of reliance upon any information contained in this publication.

The receipt or any use of this document or its contents does not in any way create by implication or otherwise:

- Any express or implied license or right to or under any ATM Forum member company's patent, copyright, trademark or trade secret rights which are or may be associated with the ideas, techniques, concepts or expressions contained herein; nor
- Any warranty or representation that any ATM Forum member companies will announce any product(s) and/or service(s) related thereto, or if such announcements are made, that such announced product(s) and/or service(s) embody any or all of the ideas, technologies, or concepts contained herein; nor
- Any form of relationship between any ATM Forum member companies and the recipient or user of this document.

Implementation or use of specific ATM standards or recommendations and ATM Forum specifications will be voluntary, and no company shall agree or be obliged to implement them by virtue of participation in the ATM Forum.

The ATM Forum is a non-profit international organization accelerating industry cooperation on ATM technology. The ATM Forum does not, expressly or otherwise, endorse or promote any specific products or services.

NOTE: The user's attention is called to the possibility that implementation of the ATM interoperability specification contained herein may require use of an invention covered by patent rights held by ATM Forum Member companies or others. By publication of this ATM interoperability specification, no position is taken by The ATM Forum with respect to validity of any patent claims or of any patent rights related thereto or the ability to obtain the license to use such rights. ATM Forum Member companies agree to grant licenses under the relevant patents they own on reasonable and nondiscriminatory terms and conditions to applicants desiring to obtain such a license. For additional information contact:

The ATM Forum
Worldwide Headquarters
303 Vintage Park Drive
Foster City, CA 94404-1138
Tel: +1-415-578-6860
Fax: +1-415-525-0182

Contents

40 1. Introduction 7
41 1.1 Purpose of Document 7
42 1.2 Terminology 7
43 1.3 References 8
44 1.4 ATM Network Service Assumptions 9
45 2. Architectural Overview 10
46 2.1 Basic Concepts 10
47 2.1.1 Reference Model 11
48 2.1.2 Information Flows 14
49 2.1.3 A Note on Reliable Propagation 14
50 2.2 Topology 14
51 2.2.1 Spanning Tree Assumptions 14
52 2.2.2 Spanning Tree Usage 15
53 2.2.3 Server Discovery Mechanisms 16
54 2.2.4 Manual Configuration Considerations 16
55 2.2.5 Configuration Operations 16
56 2.2.6 Failures 16
57 2.3 Support for LUNI Messages 17
58 2.3.1 LE_TOPOLOGY_CHANGE Messages 17
59 2.3.2 Response Packet Forwarding 17
60 2.4 LNNI Protocol 17
61 2.4.1 Address Registration Model 17
62 2.4.2 Server Join Protocol 17
63 2.4.3 Server Topology Changes 18
64 2.4.4 LEC ID Allocation 18
65 2.4.5 The Intelligent BUS Optimization 19
66 2.4.6 LE_ARP Caching 19
67 2.4.7 Pre-Standard Distributed Implementation Considerations 19
68 3. Relationships to Other Services 19
69 3.1 LNNI to AAL Services 20

70 3.2 Connection Management Services 20
71 3.3 LNNI To Layer Management 20
72 3.4 LNNI Spanning Tree Versus 802.1D 20
73 4. LNNI Frame Formats 20
74 4.1 LNNI Control Frame 20
75 5. LNNI Protocols and Procedures 21
76 5.1 Overview 21
77 5.2 LECS to LECS 21
78 5.2.1 Initialization and Configuration 21
79 5.2.2 Run Time Operation 21
80 5.3 Server Initialization and Configuration 21
81 5.3.1 Configure Direct Connection 21
82 5.3.2 Configuration Frames 21
83 5.3.3 Add/Delete LESs 23
84 5.3.4 LES/BUS Initial Topology 23
85 5.3.5 Restart of LES/BUS 24
86 5.4 Server Topology 24
87 5.4.1 Spanning Tree 24
88 5.4.2 LNNI_LES_JOIN 24
89 5.4.3 Server Failure 24
90 5.5 Client Joining 24
91 5.5.1 LNNI_LEC_JOIN 24
92 5.5.2 LEC-ID Allocation 25
93 5.6 Client Address Registration 25
94 5.6.1 LNNI_REGISTER_REQUEST 25
95 5.6.2 LNNI_LEC_UNREGISTER 25
96 5.7 Client Address Resolution 26
97 5.7.1 Caching 26
98 5.7.2 LNNI_ARP 26
99 5.7.3 Address Resolution 26
100 5.8 Client Data Transfer 27
101 5.8.1 BUS Data Movement 27

102 5.9 Client Flush 27

103 5.10 Client Topology Change Notification 27

104 5.11 Client Terminate 27

105 6. Appendix 28

List of Figures

106 FIGURE 1 PEER-TREE MODEL 10
107 FIGURE 2 LNNI REFERENCE MODEL 11

List of Tables

108 TABLE 1. NORMATIVE STATEMENTS 8

109 # 1. Introduction

110 [Editor's Note: Contributions needed for this section.]

111 ## 1.1 Purpose of Document

112 This document specifies an implementation agreement for the LAN Emulation Service. This set of protocols are referred
113 to as the LAN Emulation Network-Network Interface (LNNI) protocols. The document includes the following:

114 • Architectural Framework and Service Interfaces
115 • Service Components and Functions
116 • Frame Formats
117 • Server/Server Protocols and Procedures

118 ## 1.2 Terminology

119 The following acronyms and terminology are used throughout this document:

120

121 AAL ATM Adaptation Layer
122 ARE All Routes Explorer
123 ATM Asynchronous Transfer Mode
124 B-LLI Broadband Low Layer Information
125 BN Bridge Number
126 BPP Bridge Port Pair (Source Routing Descriptor)
127 BPDU Bridge Protocol Data Unit
128 BUS Broadcast and Unknown Server
129 CPCS Common Part Convergence Sublayer
130 CPN Customer Premises Network
131 DA Destination MAC address
132 ELAN Emulated Local Area Network
133 IE Information Element
134 IEEE Institute of Electrical and Electronics Engineers
135 IETF Internet Engineering Task Force
136 IP Internet Protocol
137 LAN Local Area Network
138 LD LAN Destination
139 LE LAN Emulation
140 LE_ARP LAN Emulation Address Resolution Protocol
141 LEC LAN Emulation Client
142 LECID LAN Emulation Client Identifier
143 LECS LAN Emulation Configuration Server
144 LES LAN Emulation Server
145 LNNI LAN Emulation Network-Network Interface
146 LSB Least Significant Bit
147 LTH Length Field
148 LUNI LAN Emulation User-Network Interface
149 MAC Medium Access Control
150 MIB Management Information Base
151 MSB Most Significant Bit
152 MTU Message Transfer Unit
153 NDIS Network Driver Interface Specification
154 NSR Non-Source Routed
155 ODI Open Data-Link Interface

156 OSI Open Systems Interconnection
157 OUI Organizational Unit Identifier
158 PDU Protocol Data Unit
159 QOS/QoS Quality of Service
160 RC Routing Control
161 RD Route Descriptor
162 RFC Request For Comment (Document Series)
163 RI Routing Information
164 RII Routing Information Indicator
165 RT Routing Type
166 SA Source MAC address
167 SAP Service Access Point
168 SAAL Signaling AAL
169 SDU Service Data Unit
170 SR Source Routing (Bridging)
171 SRF Specifically Routed Frame
172 SRT Source Routing Transparent
173 SSCS Service Specific Convergence Sublayer
174 STE Spanning Tree Explorer
175 TB Transparent Bridging
176 TCP Transmission Control Protocol
177 TLV Type / Length / Value
178 UNI User-Network Interface
179 VCC Virtual Channel Connection
180 VPC Virtual Path Connection
181 VCI Virtual Channel Identifier
182 VPI Virtual Path Identifier
183

184 This document uses normative statements throughout as follows:

185 **Table 1. Normative Statements**

Statement	Verbal Form[1]
Requirement	MUST/MUST NOT
Recommendation	SHOULD/SHOULD NOT
Permission	MAY

186

187

188 ## 1.3 References

189 [1] The ATM Forum, *ATM User-Network Interface Specification, Version 3.0,* September 10, 1993.

190 [2] The ATM Forum, *ATM User-Network Interface Version 3.1 (UNI 3.1) Specification,* July 21, 1994.

191 [3] The ATM Forum, *LAN Emulation Over ATM Version 1.0 Specification (af-0021-000),* January, 1995.

[1]Verbal forms are based on ISO except for "Requirements," where ISO uses the terms "SHALL/SHALL NOT" instead of "MUST/MUST NOT" in this document.

192
1.4 ATM Network Service Assumptions

193 This LAN Emulation Over ATM specification is based on the ATM Forum User-Network Interface Specification,
194 Version 3.0 [1] or later. The specification provides example Information Element codings for UNI 3.0 and 3.1[2].

2. Architectural Overview

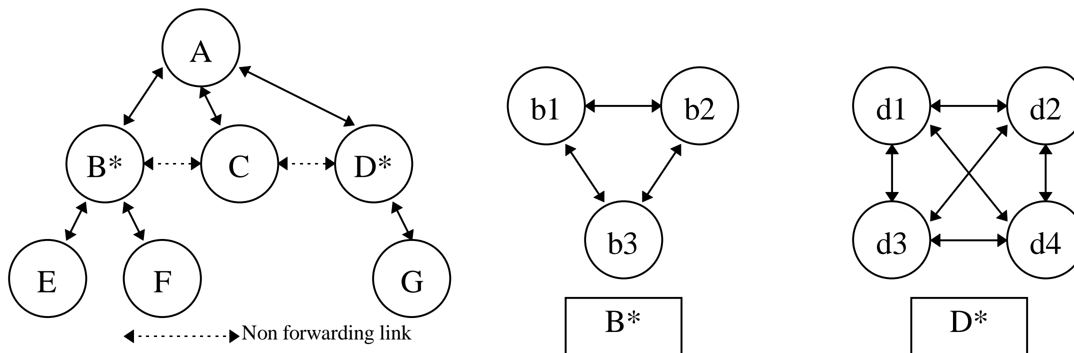
2.1 Basic Concepts

LAN Emulation as specified in [3] defines the interface between a LEC and the LAN Emulation Service entities. Interaction between the various LAN Emulation Service entities was not defined. This document defines the protocols and procedures used to implement interaction between these entities.

The model upon which the interactions is defined is called the “peer tree” model. Each node in the tree may be a single server entity (i.e., LES or BUS instance) or a "complex node" containing a collection of peer server-entities with fully-meshed interconnections. In the model, logical LES and BUS instances always occur in pairs, and the server topology may be controlled by the LECS. Any LES-BUS pair may be assigned the role of a pure tree node, a pure peer node, or a hybrid node with connections to both peer and tree node neighbors. Pure trees contain no complex nodes and pure meshes consist of a single complex tree node. Any LES-BUS pair may also serve zero or more local LE clients; thus, the model is classless in the sense that the pairs may be arbitrarily interchanged in the topology.

In addition to the tree connections between the peer nodes, the peer-tree model allows for inclusion of non-forwarding connections to provide redundancy to keep ELAN topology intact under LE Service entity failures. The non-forwarding connections can be located between any pair of simple peer nodes or between a simple peer node and a mesh node. When a LE Service entity fails along the tree or on the boundary between the tree and a mesh, the non-forwarding link connected to the nodes adjacent to the failed node will become active, thereby preventing loss of connectivity of the ELAN. The activation of non-forwarding links can be performed by the adjacent LE Service entities without requiring any LECS intervention.

Figure 1 Peer-Tree Model



[Editors Notes:

1. The LNNI protocol discussions will use a flow abstraction to describe the server forwarding rules (see section below).

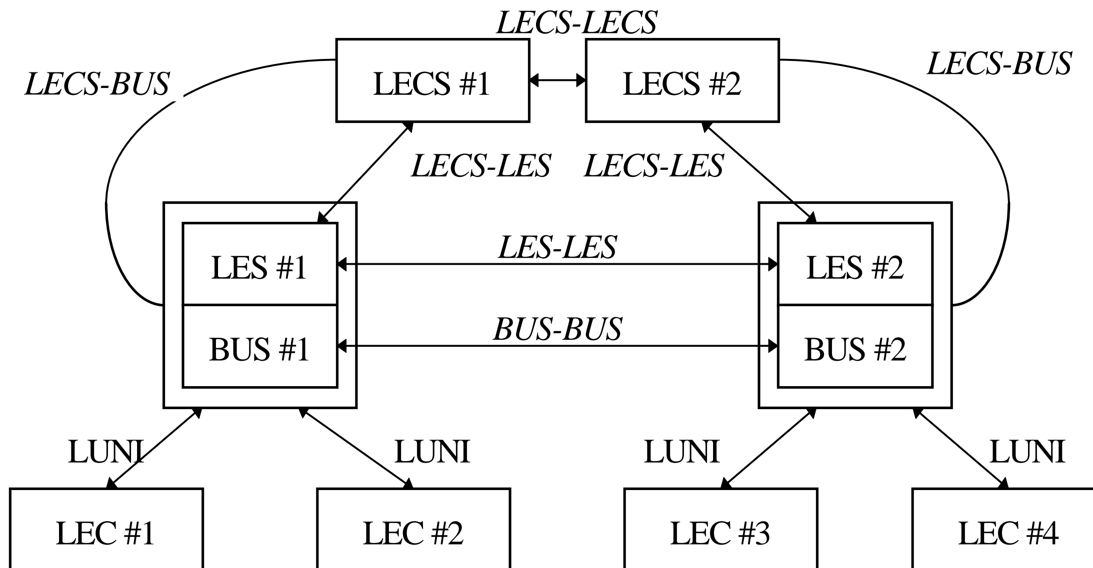
2. The LNNI protocol will define behavior for both static and/or dynamic failover and loop resolution for a given VCC topology.]

226 **2.1.1 Reference Model**

227 Figure 2 depicts the various components and their protocol interactions. The interfaces are LES-to-LES, BUS-to-
228 BUS, LECS-to-LES, and LECS-to-BUS. This section contains a high-level outline of information flows across each
229 of the interfaces. The discussion currently assumes that server-server topologies are managed by the LECS, and that
230 LES-to-LES/BUS-to-BUS communications are intact as long as the associated VCC(s) are maintained (i.e., "hello"
231 type status exchange messages are not part of the LES-to-LES and BUS-to-BUS flows).

232

233 **Figure 2 LNNI Reference Model**



234

235

236 2.1.1.1 LES-to-LES Information Flows

237 The LE servers maintain two levels of communication between each server and its immediate neighbors. At the
238 lowest level, a spanning tree algorithm is run [cite SPANOVER.TXT], determining which links in the tree are
239 active, and which peer nodes within each complex node have external connections. This information can also be
240 obtained via manual configuration, as per Civanlar and Gray's contribution [3], instead of from a spanning tree
241 algorithm.

242 At the higher level, LE servers must colaborate in order to handle address registration requests and queries. Each
243 tree node stores the address registration information registered by all of its hosts, and by all of its child servers in the
244 tree, and uses this information to answer ARP requests and forward packets efficiently. Address registration
245 information propagates up the tree to the root node of the tree, and ARP queries also propagate up the tree to the
246 root until they are answered. If an ARP request can not be handled at the top of the tree, the address may be an
247 address learnable only via the proxy ARP mechanism, and thus, if the root server can not resolve an ARP query, it
248 sets a special PROXYFLOOD bit in the LE_ARP_REQUEST packet and floods the updated message throughout
249 the entire tree. This special LE_ARP_REQUEST message is flooded to all servers, and forces each server to send
250 the appropriate ARP request message to all of its directly connected clients. Client registration requests are handled
251 by servers sending an LNNI_JOIN_REQUEST message up to the root of the tree. As the message passes through
252 all of the intermediate servers to the root server, they add the client's registration information to their local
253 databases. When a client or server crashes, a LNNI_UNREGISTER_REQUEST message is propagated throughout
254 the entire tree, directing each server to remove the appropriate client information from their caches.

Most messages, thus, are sent to the root node, and no further. A few messages, such as proxy ARP requests, topology change messages, and address deregistration messages, are flooded to all nodes in the tree. None of the messages that are sent more than one hop require any acknowledgement beyond a hop-by-hop acknowledgement.

2.1.1.1.1 LNNI_JOIN

When a LEC sends a join/registration request to its LES, the LES receiving the request creates a LNNI_JOIN_REQUEST message to verify the uniqueness of the binding. Such registration requests are not forwarded by the receiving LES to its LECs.

Any LES receiving such a request from the originator checks to determine if it has a duplicate binding already registered in its registration database. If there is a duplication, it sends back a negative LNNI_JOIN_RESPONSE to the requesting LES with a status code indicating the type of duplication. Any LES along the path to the originator, receiving the negative LNNI_JOIN_RESPONSE, must forward it back towards the originating LES. If it was the originating LES, it refuses to register the binding and returns a negative response to the LEC that requested it.

If there is not any duplication and if the LES has no other connections on which to forward the request, the LES sends back a LNNI_JOIN_RESPONSE with "No Duplicate Destination" status code. Any intermediate LES, upon determining that there is no duplication, must generate a positive response back to the requester. If there is not any duplication, the originating LES will accept the registration, create a new binding in its database and send a successful registration response to the LEC.

The LNNI_JOIN protocol must be reliable, possibly, by using a hop by hop acknowledgement scheme proposed in 96-0235.

2.1.1.1.2 LNNI_ARP

A LES responds to an address resolution request for a destination registered in that LES without passing it onto another LES. A LES may also respond to an address resolution request for a valid cache entry. The entries that are cached in a LES are those entries that are registered within other LESs, and learned from an address resolution message that originates from a LES which has the address registered in its database.

Since cached entries are learned from the address resolution messages, a flag is needed in the address resolution response to indicate whether that address is registered in the originating LES database, or not. The remote address flag may not be sufficient for this purpose as the learning-LES must differentiate between a response from a LES which cached the address as opposed to a LES which has the entry registered in its database. [Alternatively, a LES may decide to cache entries learned from another caching LES. This is for further study.]

2.1.1.1.3 LNNI_UNREGISTER

[Editors Note: This protocol allows each LES to maintain an up to date registration database and arp cache when LECs unregister.] When a LEC unregisters either explicitly or by dropping its connections from its LES, the LES generates an LNNI_UNREGISTER message and reliably sends to all its neighbors. Each LES receiving the LNNI_UNREGISTER_REQUEST will purge the corresponding registered and cached entry, and respond back with an LNNI_UNREGISTER_RESPONSE. This protocol allows each LES to maintain an up to date registration database and arp cache. Again, a reliable protocol is used.

2.1.1.1.4 LNNI_TOPOLOGY_CHANGE

[Editors Note: the LNNI_TOPOLOGY_CHANGE message in this section is one means of notifying LESs of a change. Other methods may be found which are more appropriate. The purpose of this section is to capture the discussion of the "marking flags" as a technique for reducing the amount of re-registration required after the healing of a network partition.] A new LNNI topology change protocol between LESs can be used as a means to identify failure/recovery of a LES. A LES issues a LNNI_TOPOLOGY_CHANGE when it identifies failure/recovery of a neighbor LES. The notice of a topology change is propagated through the mesh and tree of LESs by other LESs which receive it. This notification must be reliably propagated and thus can use the same hop by hop acknowledgement system used for the LNNI_JOIN. Multiple LNNI_TOPOLOGY_CHANGE messages might be initiated by different neighbors that identify the same failure. If such multiples of the same message are received, the receiving LES must make sure to propagate only one copy. The LNNI_TOPOLOGY_CHANGE message contains the id of the failed/recovered LES.

When an LNNI topology change message indicating a LES failure is received, the LESs must purge all registered or cached bindings learned from that failed LES, preferably, after a timer expires. In the Master LES approach,

305 through this timer, the failed LES's registered entries can be retained for a while just in case the failed LES recovers
306 shortly after a failure.

307 When a LES receives its first topology change message indicating failure of a server, it will set the "marking flag"
308 to indicate that a marking period starts. Until that flag is cleared, each LES will mark every newly registered
309 address. Similarly, each LES will mark unregistered addresses keeping them in a database with a "marked" status.
310 The marking is needed to ensure that when the network is healed, the registration/unregistration of "marked" entries,
311 those that are processed during a likely network partitioning, are re-synchronized within the merged network.

312 If there are other LES failures prior to the recovery of the first failed LES, the marking flag will be incremented
313 while keeping a list of the failed LESs. The flag will be decremented as each failed LES recovers, but it will not be
314 cleared until all failures are healed, or a timer expires.

315 If a new LES joins the topology when the network is in a marking state, the LES adjacent to the newly joined LES
316 will send its marking flag value so that the new LES knows whether it should perform marking or not.

317 Once the marking flag is cleared, meaning that the network has completely healed, all LESs will re-validate the
318 uniqueness of the marked registrations using LNNI_JOIN protocol. Similarly, using the LNNI_UNREGISTER
319 protocol, LESs will inform each other of the marked unregistrations so that each LES can purge the marked
320 unregistrations from their registration databases and ARP caches.

321 2.1.1.2 BUS-to-BUS Information Flows

322 In the LNNI baseline, LES and BUS servers are always colocated. Thus, the server registration procedures used by
323 the LESs can be shared by the colocated BUS processes. Nevertheless, there are some data flows specific to the
324 BUS processes.

325 In general, true broadcast traffic, as opposed to unknown traffic, must be flooded to all BUSs in the network.
326 Unknown packets, on the other hand, in an intelligent BUS environment, need only go up the tree as far as the
327 common parent of the sender and the receiver. If the unknown packet reaches the root of the tree without
328 encountering a server having the target MAC address registered, it needs to be flooded to all proxy hosts, since it is
329 an unknown packet whose destination is not registered by any servers in the tree, and only proxy hosts may
330 represent hosts having these unregistered MAC addresses.

331 2.1.1.3 LECS-to-LES Information Flows

332 The LECS-to-LES information flows are primarily for managing the distributed LES topology and tracking the
333 status of LES instances. These flows are used to implement the server discovery mechanisms and
334 redundancy/failover modes. From a topology perspective, the flows will support LES registration with the LECS
335 and LECS assignment of LES neighbors. During registration, a LES may provide its ATM address(es) to the LECS.
336 The LECS may provide each LES with a list of neighboring LESs. The ATM address and type (peer or parent) of
337 each neighbor will be specified. The LECS may provide dynamic updates that add/delete neighbors to/from the list.
338 The LECS will provide each LES with a LESID for use in server-server protocols, and may also manage the pool of
339 LECIDs. From a status perspective, the flows will allow the LECS to query the status of each LES instance, and
340 may allow the LECS to obtain resource utilization information for load balancing purposes. When a LES failure is
341 detected, the LECS may inform other LES instances to initiate prompt deletion of cached registration information
342 associated with clients of the failed LES.

343 2.1.1.4 LECS-to-BUS Information Flows

344 The LECS-to-BUS information flows may be viewed as being analogous to the LECS-to-LES flows. Alternatively,
345 the LECS-to-BUS interface may be perceived as being superfluous; in this line of thinking, management of the BUS
346 through the LES in LNNI is viewed as being orthogonal with LUNI v1.0, where the LES is the "logical manager" of
347 BUS ATM addresses. Furthermore, concerns have been raised that introduction of a non-orthogonal approach (i.e.,
348 the LECS-to-BUS interface) will require definition of a LES-to-BUS interface to resolve inconsistencies that may
349 occur. Additional work is needed to sort this out. Related issues include: (1) independence of LES and BUS
350 locations, (2) LES-to-BUS interface requirements in LNNI-aware LES-BUS pairs (i.e., how much coordination of
351 the pair is performed through the LECS?), and (3) ease of integrating existing distributed implementations into the
352 LNNI environment.

2.1.2 Information Flows

The LNNI Protocol uses the term "Information Flow" to describe communication paths between protocol entities. There are several reasons for this convention:

a) future versions of LAN Emulation may support packet encapsulation techniques other than the VC-muxing used by LANE 1.0. If a LLC/SNAP encapsulation were to be supported by LNNI, there is the scope for sharing VCCs between multiple flows and a protocol definition that is independent of this issue would be useful.

b) this clarifies the fact that there are no geographical constraints imposed on LANE protocol entities: communication between entities can occur over either dedicated or shared VCCs, or by some internal or proprietary ommunication path, without affecting the protocol definition itself.

c) the LNNI protocols are defined to assume a communications path is available for each Flow. The management of this path (e.g., SVC establishment, PVC configuration, VCC sharing) is separated from the main protocol: specification of the redundancy and failover aspects of the "tree" model in particular is made easier by this separation (see Section 5).

When the protocol requires an entity to establish a new "Flow," this may result in a new VCC being set up or it may just indicate that another communication path be multiplexed onto an existing VCC. It may also be implemented as a proprietary communication or a local signal between colocated entities.

2.1.3 A Note on Reliable Propagation

[Editor's Note: This section needs to be abstracted more.]

Some LNNI procedures must be performed reliably, meaning that they must be performed even when packets are dropped. For messages traversing a single link, this may require that we be able to detect a duplicate message if it has side effects that should not be repeated on a retransmission.

Other procedures require that a message propagate all the way to the root node before an acknowledgement is generated. For such messages, each server must generate a new transaction ID for each link of the path traversed by the message and require an acknowledgement. All of these messages are propagated hop-by-hop, and so there need not be any state maintained at a sending server after it has received the acknowledgment that its request packet has been received by the next server on the other side of the link. If a failure occurs up the line, the link will be reset. *[Editor's Note: We may be able to simplify this mechanism to avoid hop-by-hop acknowledgements.]*

[Editor's Note: This section may need to be simplified and the detail moved elsewhere.] A few other messages must be flooded reliably throughout the entire graph, such as the procedures that propagate server failure information. This type of operation can be handled similarly to the preceding protocol, only each server needs to generate a new transaction ID for each link upon which the message is flooded. As with the messages that simply go up to the root, these messages require an acknowledgement for every link on which they are sent, but after they are transmitted successfully over a link, no further state is required to track, for example, their response packets. Again, if a request packet can not be transmitted over a particular link, the link is simply reset, and the servers on the leafward side of the link will have to rejoin the network before they can function again.

2.2 Topology

2.2.1 Spanning Tree Assumptions

If spanning tree is used for configuration, we elect a root server for the entire tree. This node plays a very important role in the protocols described below.

Over time, all nodes perform a join-like protocol with the nodes already joined to the tree's root node. As new nodes join the set of servers joined with the root node, the servers adjacent to the newly joined nodes then join with them, leading to an ever-expanding set of nodes joined with the root. These servers all synchronize their address registration databases as part of joining.

398 A simple link state protocol, described below, runs when a node wants to join the coordinated set of servers. At that
399 time, the server will generate a unique (for all time) server instance ID (SII), which will be registered with all other
400 server nodes on the path to the root node. This will enable a server still joined to the root to know exactly which
401 servers have been lost, should a link fail. All address information registered by a server and stored at other servers
402 is tagged with this SII, and thus can be removed easily should an SII becomes invalid. This algorithm is described in
403 the next subsection.

404 **2.2.2 Spanning Tree Usage**

405 2.2.2.1 Mapping Spanning Tree to LNNI Peer-Tree Graphs

406 In order to use the spanning tree within the Emulated LAN's server graph, we need to explicitly map the server
407 graph into a bridged LAN in which the spanning tree could be run without significant change.

408 **2.2.2.1.1 Tree Portion**

409 For the tree portion of the peer-tree (with non-forwarding links), we propose to adopt the following mapping:

410 - each LNNI server (LES or BUS) maps into a "bridge".

411 - each server-to-server connection between a pair of servers maps into a "LAN".

412 - each server-to-server connection termination on a server maps into a "bridge port".

413 Here, we assume that the server-to-server links along the tree portion are point-to-point.

414 With the above mapping, each link between two simple nodes in the graph is equivalent to a LAN with two bridges,
415 each of which has a single bridge port on that LAN.

416 **2.2.2.1.2 Mesh Portion**

417 We can model a mesh as a single LAN, with each peer node having one port onto that LAN, and the port
418 representing *all* of its peer links. Any external (to the mesh) links for a peer node are represented as above: a
419 separate port to a LAN having two bridges. Note that when spanning tree turns off a mesh port, it effectively turns
420 off *all* of the server's connections to the other peer nodes in that mesh.

421 By using this model, we can map the entire peer tree graph into a single collection of LANs and bridges whose
422 spanning tree solution yields a loop free forwarding tree for LNNI control and data packets.

423 The spanning tree will make sure that the mesh node with the most direct connection towards the root of the
424 spanning tree (or the root itself, if the root is on the mesh) will be the "designated bridge" for the mesh. Links
425 "downward" to other parts of the tree from the mesh may come from any node in the mesh, although links from the
426 designated bridge will normally be preferred, because that node has the shortest path to the root.

427 In short, by using this "looks like a multi-drop LAN to the spanning tree" trick for meshes, the single spanning tree
428 algorithm allows any combination of connections between mesh nodes and non-mesh nodes, and builds the
429 appropriate peer tree automatically. The only requirement is that all of the nodes in a particular mesh have the same
430 idea about who is (and is not) in the mesh, but that requirement is not particular to spanning tree.

431 2.2.2.2 LNNI Spanning Tree Protocol

432 As the mapping from the LNNI graph to a bridged LAN is clear, it is possible to run the IEEE 802.1D spanning tree
433 code within each server and to provide the proper mappings (port priority, path costs etc.) in a separate module.
434 This allows to simply encapsulate 802.1D spanning tree packets in some LNNI header, and thus run 802.1D
435 directly. With the proper abstractions, it should even possible for a bridge to be able to use the same code for
436 running its real spanning tree code and for running the spanning tree within the LNNI algorithm.

437 2.2.2.3 Forwarding Rules in the Spanning Tree

438 The forwarding rules for this spanning tree are very simple. Whenever a packet is received from a port that has
439 been turned off by spanning tree, the packet is discarded. Whenever a packet is received from an active port, it is
440 forwarded out on all other active ports. Note that if a node is part of a mesh, one of its "ports" will represent all of
441 its connections to the other peer nodes in the mesh, and forwarding a packet to that port results in either a packet
442 transmission on a PMP connection, or N individual transmissions on a P2P connection. Similarly, if a mesh port is
443 turned off, all packets received from any of the connections from other peers are discarded.

2.2.3 Server Discovery Mechanisms

The following server discovery mechanism relies on the LECS to provide server entities with the addresses of neighboring servers. Servers will locate the LECS using the techniques defined for LE Clients in the [3]. Servers will then check in with the LECS and provide server ATM addresses, ELAN Names, and Server Names (the names allow the LECS to identify servers when ATM addresses are not known a priori). The LECS will assign Server IDs for use in server-server protocols, and provide localized topology information. For tree connections, each server will be given the ATM address of its parent and may also be provided the ATM addresses of its children for verification purposes. For mesh connections, each server will be provided with the ATM addresses of all peers. A new server configuration frame format will be defined to accommodate the required one-to-many mappings.

2.2.4 Manual Configuration Considerations

LUNI v1.0 allows ELANs to be manually configured. Some members of the LAN Emulation SWG have expressed a desire for the LNNI to also be able to operate in a manual configuration mode. Additional work is needed to appreciate the protocol implications of achieving this goal.

2.2.5 Configuration Operations

2.2.5.1 Configure Direct Connection

The LECS is the provider of the configuration information to both LECs and LESs/BUSs in an ELAN. A LES first determines its LECS address using the methods described in LUNI 1.0 and establishes a Configure Direct connection. A LECS which supports the LNNI protocol allows LES/BUS entities to establish Config Flows to it and handles queries. In this document, use of the term LECS implies one that can handle LES/BUS requests, as well as those from LECs.

The protocol allows sharing of a single VCC by multiple Configure Direct flows to the LECS. Note that the protocol maintains the principle from LUNI 1.0 that the presence or absence of a Configuration VCC has no relationship to the operational state of the entities at either end i.e. the connection may be brought up and down at any time.

2.2.5.2 LES/BUS Initial Topology

After successfully completing the configuration with the LECS, the LES/BUS establish control plane Mesh <m>, Forwarding <f> and redundant Non-forwarding <nf> VCCs to its neighbors. The type of the VCC indicates the forwarding rules that LES/BUS should implement *[Editor's Note: see 95-1177. This needs to be clarified in the context of connections and flows]*.

2.2.5.3 Add/Delete LESs/BUSs

A LECS is allowed to add/delete LESs to an existing ELAN. The changed configuration information due to added/deleted LES is sent to each LES by an unsolicited notification which may include updated configuration information, e.g., new LES/BUS addresses, connection types, etc. These require acknowledgement by LES.

LES is responsible for updating its associated BUS with any changes to the configuration: the protocol mechanisms for doing this are outside the scope of this document.

2.2.6 Failures

This section looks at server crashes and network partitions, both their occurrances and the recovery operations required after the crash or partition is repaired.

2.2.6.1 Server Failures

We look node and link failure, both of which are detected either by the loss of a control VC, or by a periodic LNNI_GETSYNC_REQUEST reporting that a link is in the unsynchronized state.

When a failure occurs, we must remove any address registration information for clients registered at LE servers no longer connected to the root node. Clients that were connected to these servers may well try to connect to the surviving servers; we must thus purge this information without preventing these same clients from re-registering. And we must also do this promptly, since otherwise this stale information may prevent these reconnecting clients from successfully joining the ELAN.

489 When a child link fails, the parent server notes the set of SIIs associated with that link, and knows that potentially all
490 of those servers are inaccessible. It thus removes any stored registration information tagged by any of those SIIs
491 from its own address database, and floods an LNNI_UNREGISTERSERVER_REQUEST throughout the remaining
492 set of joined servers, listing each affected SII. These requests tell the receiving server to flush all registration
493 information tagged with the server's SII. These requests only travel to the servers still joined to the root; no attempt
494 is made to send these messages to servers on the far side of a failed link. Instead, those servers will be forced to join
495 the LNNI network again.

496 If a LNNI_UNREGISTERSERVER_REQUEST message can not be sent along a link, the link must be put into the
497 unsynchronized state by the server failing to send the message, so as to force the leafward side of the link to rejoin
498 the LNNI network.

499 2.2.6.2 Server Recovery

500 When a network partition is repaired, two independently running ELANs must be merged. Each ELAN has a
501 spanning tree root, and during the merge operation, one of those roots will cease being the spanning tree root.

502 Given the synchronization protocol used here, a network partition's repair will manifest itself by various servers
503 changing their idea of which server is the spanning tree root. When a server sees a spanning tree root change, it puts
504 all of its links into the unsynchronized state, and then attempts to perform the server join protocol with the next
505 server along the path to the spanning tree root.

506 As each server joins with the next rootward server, the set of servers joined to the new root will grow, finally
507 expanding to include all servers in the two previously disconnected ELANs. Note that only the servers in one of
508 these two ELAN fragments will actually change their spanning tree root. The servers in the other fragment keep
509 operating normally throughout the partition recovery.

510 If the network is recovering from a simple server crash, the system goes through a similar recovery procedure. In
511 the affected part of the tree, the spanning tree links may change, and some set of servers will find that their link to
512 the root is no longer in the synchronized state. These servers will have to go through the basic server join protocol
513 again, as if their link had failed and been restarted. This may at times require those servesr to re-register their
514 clients with the root server.

515 2.3 Support for LUNI Messages

516 2.3.1 LE_TOPOLOGY_CHANGE Messages

517 These messages are simply flooded to all servers, who in turn flood them to all connected hosts.

518 2.3.2 Response Packet Forwarding

519 Certain response packets, including the very common LE_FLUSH_RESPONSE and LE_ARP_RESPONSE packets,
520 must be forwarded based upon the LEC ID of the REQUEST packet's originator. These packets can be efficiently
521 forwarded because the address registration information present at each server node is precisely the information
522 required to forward these response packets up to the common parent of the sender and receiver, and then back down
523 to the receiver.

524 2.4 LNNI Protocol

525 2.4.1 Address Registration Model

526 *[Editor's Note: Address Registration database model needs to be summarized here.]*

527 2.4.2 Server Join Protocol

528 *[Editor's Note: This section needs abstraction.]* The server join protocol deals with the tree of server nodes
529 consisting of simple and complex nodes. Each node has a single parent link and one or more child links.

530 Each server-to-server link has an associated link state, maintained at the parent side of the link. This state may be
531 any one of "unsynchronized," "connecting," or "synchronized." The link state may be changed by either side of the
532 link, but transitions in the direction from unsynchronized to synchronized may only be done at the request of the
533 child.

534 What do the link states mean? Loosely speaking, when a link is in the unsynchronized state, neither link control nor
535 registration information is passed along the link. When a link is in the connecting state, the child is establishing the
536 basic information needed to participate in the ELAN; in this state, link control information is passed up the link, but
537 no registration information is passed on the link. A link in the synchronized state is fully operational, allowing
538 registration information to pass in both directions. At this point, the server has joined the set of servers joined to the
539 root.

540 Servers have unique server instance IDs (SIIs), generated by the child server, and included on all control messages
541 sent by that server. This SII is generated by the child at the time that the link leaves the unsynchronized state.

542 As part of the spanning tree algorithm, each LE server maintains its own idea of the root server to which it is
543 connected. When the link to the root server changes, or the server otherwise discovers that its link to its parent is in
544 the unsynchronized state, the LE server must rejoin the LNNI network. It begins by sending an
545 LNNI_GETSYNC_REQUEST, and if the response indicates that the link has been put into the unsynchronized state
546 by its parent, the child sends its parent an LNNI_SETSYNC_REQUEST, requesting that the link be put into the
547 connecting state.

548 Once its link is in the connecting state, the child server sends a LNNI_SERVERJOIN_REQUEST containing its
549 ATM address and newly generated SII to its parent; it retransmits this request until it receives an
550 LNNI_SERVERJOIN_RESPONSE. When a node receives this request from a child on a link in the connecting
551 state, it records the SII in memory, associating this SII with the VC, and forwards the
552 LNNI_SERVERJOIN_REQUEST up the tree towards the root. The LNNI_SERVERJOIN_REQUEST propagates
553 up the tree until the request reaches the root, which generates an LNNI_SERVERJOINED_REQUEST with an error
554 code of 0 (success). The LNNI_SERVERJOINED_REQUEST packet is forwarded back down the tree until it gets
555 back to the originating LE server; these packets are acknowledged with LNNI_SERVERJOINED_RESPONSEs.

556 If anything goes wrong with this process, the link is put back into the unsynchronized state, and an attempt to
557 resynchronize the link is tried again a short time later.

558 Once a successful LNNI_SERVERJOINED_RESPONSE is received, the child sends an
559 LNNI_SETSYNC_REQUEST, placing the link in the synchronized state.

560 At this point, the newly joining LE server has two choices. If there are clients still registered with it, it can release
561 their control direct VCs and force them to re-register, which has the advantage of simplicity. It also could re-
562 register them, only disconnecting them if the attempt to re-register them fails (as suggested in 95-1393). Similarly,
563 if there are child servers still connected, it can either put their links into the unsynchronized state, forcing them to go
564 through the same server join protocol, or it can simply register all of their clients for them invisibly.

565 A commonly asked question has been: Is there any difference between performing this server join protocol and
566 simply killing off address registration info when a link to a child is lost.

567 There are indeed several differences. First, since the server join protocol unregisters clients in blocks labelled by
568 the servers' SIIs, this protocol is considerably more efficient at removing obsolete client registration information.
569 Moreover, the server join protocol ensures that when a network partition is repaired, all of the servers in one side of
570 the partition are forced to resend their address registration information to their new root server, thus providing a
571 mechanism for detecting conflicting address registrations that occur during a partition.

572 2.4.3 Server Topology Changes

573 *[Editor's Note: This section needs text describing what happens when servers lose connectivity (partitioning) with*
574 *each other.]*

575 2.4.4 LEC ID Allocation

576 This model can, with essentially no extra effort, allocate LEC IDs at the root node of the graph, since all client
577 registration requests must go through this node anyway. Static LEC ID allocation also works, although requires

578 additional network management overhead. The LECS assigns a block of LEC-ID numbers during the initial
579 configuration of a LES. *[Editor's Note: We need clarification on LEC-ID allocation schemes.*]

580 2.4.5 The Intelligent BUS Optimization

581 Each LES stores the address registration information for all of its descendants, including the link from which the
582 host's registration information request arrived. Thus, any LES, given the MAC address or ATM address of a host,
583 can easily determine the link over which the host's LES can be reached (or, more accurately, the link to use for the
584 first hop to the server directly connected to the target host).

585 An intelligent BUS can thus be implemented that, upon the receipt of an unknown packet, propagates the packet to
586 the target host's BUS, following a path up to the closest common ancestor of the sending host and the target host,
587 and then travelling back down the tree to the target host's BUS. If the unknown packet propagates all the way to the
588 root without finding the target host's address registration information, the packet may be addressed to proxy
589 connected host. In that case, the packet is flooded down the tree from the root, with a special encapsulation
590 indicating that the packet should be flooded to all BUSs, to all proxy hosts, but not to any non-proxy hosts.

591 2.4.6 LE_ARP Caching

592 *[Editor's Note: This section needs to be abstracted.]* LE ARP caching is an important LNNI performance
593 optimization; it provides the easiest way of sharing the load for processing LE_ARP_REQUESTs among all the
594 LESs in the network. Note that LE ARP caching denotes storing address registration information for hosts at LE
595 servers other than the one to which the host is connected; it does not denote any caching of proxy ARP information,
596 which is forbidden by the LUNI specification.

597 ARP information can be learned by a server's watching LE_ARP_RESPONSE packets go by. In order for these
598 caches to contain only up-to-date information, we must ensure that stale information is purged promptly. This
599 happens automatically by virtue of the protocols already used for propagating server failures and host
600 deregistrations. In both of these, cases, a request is propagated throughout the entire (remaining) LNNI tree,
601 indicating the set of hosts no longer connected to the tree, and thus which should be flushed from any caches. In the
602 case of a server crash, the protocol is a LNNI_UNREGISTERSERVER_REQUEST message, and in the case of a
603 simple host de-registration, the protocol is the LNNI_TERMINATE_REQUEST message.

604 2.4.7 Pre-Standard Distributed Implementation Considerations

605 It may be desirable for existing distributed implementations of the LES and/or BUS be able to function as a logical
606 LES-BUS pair in a LNNI environment. Additional work is needed to delineate the protocol support necessary to
607 achieve this goal. Factors to be considered include:

608 (1) a LES or BUS instance may have multiple ATM addresses on multiple UNIs,

609 (2) the LES is currently the "logical manager" of BUS ATM addresses,

610 (3) failure of a LES or BUS component of non-standard distributed implementation does not imply that the logical
611 LES or logical BUS has failed,

612 (4) failure of a LNNI-connecting component of a non-standard distributed implementation does not imply that the
613 LES-BUS pair should be removed from the LNNI topology (perhaps, only the LNNI access address/interface needs
614 to be changed), and

615 (5) clients may not contact the LECS when BUS connections are released if conditions are such that the clients can
616 reconnect to the logical BUS via the paired LES while maintaining compliance with LUNI v1.0.

617 3. Relationships to Other Services

618 This section specifies the service interfaces between the various LAN Emulation Service entities and the AAL,
619 Connection Management and Layer Management entities. The services are described in an abstract way and do not
620 imply any particular implementation, or any exposed interface.

621

```
/* Editor's Note: include by reference to save paper */
#include <Section 3 from LUNI 2.0>
```

[Editor's Note: After including, add at the end of section 3.2.1, include "Support of PVC ony is TBD.]

3.1 LNNI to AAL Services

3.2 Connection Management Services

3.3 LNNI To Layer Management

3.4 LNNI Spanning Tree Versus 802.1D

4. LNNI Frame Formats

[Editor's Note: This section could include all of the control frame text from the LUNI document assuming that we intend to use the same frame format.]

4.1 LNNI Control Frame

[Editor's Note: Contributions needed for this section.]

5. LNNI Protocols and Procedures

[Editor's Note: Contributions needed for this section.]

5.1 Overview

[Editor's Note: Need to describe functional blocks and how they will interact.]

5.2 LECS to LECS

5.2.1 Initialization and Configuration

5.2.1.1 LECS-LECS Information Flows

5.2.2 Run Time Operation

5.3 Server Initialization and Configuration

5.3.1 Configure Direct Connection

The LECS is the provider of the configuration information to both LECs and LESs/BUSs in an ELAN. A LES first determines its LECS address using the methods described in LUNI 1.0 and establishes a Configure Direct connection. A LECS which supports the LNNI protocol must support LES and BUS entities establishing Config Flows to it and queries in the format described below. In this document, use of the term LECS implies one that can handle LES and BUS requests, as well as those from LECs.

The LES should not release its configure direct connection with the LECS after completing its configuration. The LES is allowed to send additional LNNI_CONFIGURE_REQUESTs after the initialization. Similarly, the LECS is allowed to send an unsolicited LNNI_CONFIGURE_RESPONSE to inform a LES about a change in the server topology information.

The protocol does not preclude sharing of a single VCC by multiple configure direct connections to the LECS over the same VCC. Note that the protocol maintains the principle from LUNI 1.0 that the presence or absence of a Configuration VCC has no relationship to the operational state of the entities at either end i.e. the connection may be brought up and down at any time.

5.3.2 Configuration Frames

A LES requests configuration information from LECS by sending a LNNI_CONFIGURE_REQUEST. In response, the LES obtains the ATM address of all other LESs and may obtain additional configuration parameters specified in LUNI 1.0. The basic fields in the REQUEST frames are as follows:

```
LNNI_CONFIG_REQUEST(
        Source_ATM_Address,
        Transaction_ID
        )
```

In response to a LNNI_CONFIG_REQUEST, the LECS sends the requested information in an LNNI_CONFIGURE_RESPONSE which includes:

```
LNNI_CONFIG_RESPONSE(
    Source_ATM_Address,
    Transaction_ID,
    LES_ID,
    ranges of LEC_IDs,                                 optional
    BUS_ATM_Address,                                   optional
    sequence of { LES_ID, LES_ATM_Address, ... },      optional
    LES_Keepalive_Timeout,                             optional
    BUS_Maximum_Frame_Age,                             optional
    BUS_ID,                                            optional
    sequence of { BUS_ID, BUS_ATM_Address, ... },optional
    other reconfiguration-protocol config              optional
    )
```

[*It is T.B.D. whether a new Opcode value actually needs to be used or whether the above information can be coded as extensions to the LUNIv1.0 LE_CONFIGURE frame formats. Extra TLV values will need to be defined in any case. Details to be provided later.*]

The CONFIG_RESPONSE fields which are additional to the standard LUNI v1.0 fields are:

1. Transaction_ID: may be necessary for identifying lost messages or changed config information

2. LES-ID: An identifier for the LES, unique among all LESs on this ELAN.

3. Block of LEC-IDs: LEC-IDs for the LES to give out to LE Clients at Join time. [*N.B. this block could be implied by the LES-ID since LEC-IDs can be obtained by prepending the LES-ID*].

4. LES Neighbor List: a list of {LES_ID, LES_ATM_Address, Conn_Type , Conn_Priority} of all other neighboring LESs in the ELAN. Conn_Type is a flag indicating the desired type of connectivity to this neighbor: values are “Forwarding”, “Non-forwarding” or “Mesh” (<f>, <nf>, <m> as defined in [95-1393]. There may be a Conn_Priority for selecting which links to use and also to aid in determining the root of any dynamic tree (T.B.D.).

5. BUS ATM Address: optionally, may specify the BUS associated with this LES.

6. BUS-ID: An identifier for the BUS, unique among all BUSs on this ELAN.

7. BUS Neighbor List: a list of {BUS_ID, BUS_ATM_Address, Conn_Type , Conn_Priority} of all other neighboring BUSs in the ELAN. Conn_Type is a flag indicating the desired type of connectivity to this neighbor: values are “Forwarding”, “Non-forwarding” or “Mesh”. There may be a Conn_Priority for selecting which links to use and also to aid in determining the root of any dynamic tree (T.B.D.).

8. Other per-node parameters for configuring any dynamic reconfiguration protocol may also be needed here e.g. root priority for a spanning-tree protocol.

At any time, LECS may issue a LNNI_UPDATECONFIG_REQUEST which may contain some or all of the information listed above for a LNNI_CONFIG_RESPONSE. This is used primarily to update server components with altered configuration data:

```
LNNI_UPDATECONFIG_REQUEST(
    [same fields as CONFIG_RESPONSE above]
)
```

On receipt of an UPDATECONFIG_REQUEST, an LESs must generate a LNNI_UPDATECONFIG_RESPONSE frame:

```
LNNI_UPDATECONFIG_RESPONSE(
        Source_ATM_Address,
        Transaction_ID
        )
```

[*As discussed in [95-1393], it is expected that the LECS's configuration data will be relatively static i.e. it will be changing on a "human" sort of time scale: this makes the task of implementing a distributed LECS simpler and removes any real-time requirement on the synchronisation of this database. Note, however, that protocols for distribution of LECS functionality are outside the scope of this contribution.*

The above assumes that BUS topology is not identical to LES topology: some configuration parameters are then redundant if the topologies are equivalent.]

5.3.3 Add/Delete LESs

A LECS is allowed to add/delete LESs to an existing ELAN. The changed configuration information due to added/deleted LES is sent to each LES by an unsolicited LNNI_UPDATECONFIG_REQUEST which will include updated configuration information (new LES/BUS addresses, connection types, etc.). Such REQUESTs require acknowledgement by LES with a LNNI_UPDATECONFIG_RESPONSE message back to the LECS.

LES is responsible for updating its associated BUS with any changes to the configuration: the protocol mechanisms for doing this are outside the scope of this document.

5.3.4 LES/BUS Initial Topology

After successfully completing the configuration with the LECS, the LES and BUS establish control plane Mesh <m>, Forwarding <f> and redundant Non-forwarding <nf> connections to its neighbors. The type of the connection indicates the forwarding rules that LES/BUS should implement [see 95-1177].

Assuming that LES and BUS are paired, the configuration information provided to LESs is sufficient to establish both the initial LES-LES and BUS-BUS topologies. Each LES already needs to know the ATM address(es) of it's paired BUS(s) as specified in LUNI 1.0. This may be locally configured or optionally provided by LECS at configuration time.

[*If the LES and BUS topologies are assumed to be the same then some of the configuration information discussed above is redundant. There is also some scope for simplification of the configuration information based e.g. on a single "mesh/tree" flag or from extensions to LE_ARP protocol for discovering BUSs.*]

5.3.5 Restart of LES/BUS

[Editor's Note: Need clarification on whether warm start is different than cold start for LES/BUS.] After recovering from a failure, a LES establishes a configure direct connection to the LECS and reconfigures itself sending an LNNI_CONFIGURE_REQUEST. The LECS may decide to continue using the LES-ID and LEC-ID block assigned to that LES prior to the failure.

If that LES-ID and LEC-ID block are re-assigned to another newly initiated LES, then LECS must assign a new LES-ID and LEC-ID block to the recovered LES. The are assignment of LES-IDs and LEC-IDs from a dead LES to another LES may be performed upon a human intervention or upon expiration of a timer.

If LEC-ID blocks have been reassigned to another LES and then the previous owner becomes connected to the LES topology again, it will be necessary to specify mechanisms for merging their LE Clients to avoid duplicate LEC-IDs: this is for further study.

5.4 Server Topology

5.4.1 Spanning Tree

5.4.2 LNNI_LES_JOIN

5.4.3 Server Failure

5.5 Client Joining

5.5.1 LNNI_LEC_JOIN

[Editor's Note: This section needs to be pruned and consolidated.]

When a LEC sends a join/registration request to its LES, the LES receiving the request creates a LNNI_JOIN_REQUEST message to verify the uniqueness of the binding. Such registration requests are not forwarded by the receiving LES to its LECs.

Any LES receiving such a request from the originator checks to determine if it has a duplicate binding already registered in its registration database. If there is a duplication, it sends back a negative LNNI_JOIN_RESPONSE to the requesting LES with a status code indicating the type of duplication. Any LES along the path to the originator, receiving the negative LNNI_JOIN_RESPONSE, must forward it back towards the originating LES. If it was the originating LES, it refuses to register the binding and returns a negative response to the LEC that requested it.

If there is not any duplication and if the LES has no other connections on which to forward the request, the LES sends back a LNNI_JOIN_RESPONSE with "No Duplicate Destination" status code. Any intermediate LES, upon determining that there is no duplication, must generate a positive response back to the requester. If there is not any duplication, the originating LES will accept the registration, create a new binding in its database and send a successful registration response to the LEC.

The LNNI_JOIN protocol must be reliable, possibly, by using a hop by hop acknowledgement scheme proposed in 96-0235.

Each node in the tree is a master LES in terms of its descendants in the tree. That is, each LES stores the complete address registration database for all of its children. When an LE_JOIN_REQUEST is received by an LES, it must be propagated up to the root server as a LNNI_JOIN_REQUEST. As servers relay this message up to the root, they also update their local address registration databases to take note of the new client. This request is hop-by-hop reliable, with each hop sending an LNNI_JOIN_RESPONSE packet as the packet advances towards the root server.

Once the root has received the request, it generates an LNNI_JOINED_REQUEST message (acknowledged, hop-by-hop via a LNNI_JOINED_RESPONSE packet). A JOINED request corresponding to the JOIN request must eventually be received by the LES adding the registration information, or the registration request can not be validated. The root node generates the LNNI_JOINED_REQUEST as follows: the root verifies that there are no address duplications in the registration information, and also may allocate an LEC ID for the client. Once the root has registered the client, an LNNI_JOINED_REQUEST is generated, and is directed back towards the originating server.

One the host addresses have been registered, LE_ARP_REQUEST messages may be sent by host machines their connected LE server. These LE_ARP_REQUEST messages are sent up the tree to the root, with the first server having the ARP information for the target returning the response. If the request makes it all the way to the root without a response, the request is flooded back down with its PROXYFLOOD flag bit set, and each server in turn floods the corresponding LE_ARP_REQUEST message to all of its proxies, as well as propagating the modified ARP request throughout the tree.

When a host unregisters, either by dropping its control VCCs, or by sending an LE_TERMINATE_REQUEST protocol, the host's directly connected LES floods throughout the entire tree an LNNI_TERMINATE_REQUEST protocol, whose goal is ensuring both that the registration information stored along the path to the root has been removed, and that all LE ARP information cached by servers anywhere in the tree has been removed. This packet is acknowledged on a hop-by-hop basis by a LNNI_TERMINATE_RESPONSE packet.

5.5.2 LEC-ID Allocation

The LECS assigns a block of LEC-ID numbers during the initial configuration of a LES. *[Behavior of a LES when this block is exhausted is not addressed. Note that it is proposed here that LECS be responsible for unique allocation of LEC-IDs to LESs. An alternative approach, outlined in [95-1177], proposes a master LES be responsible for this. We believe that the allocation of blocks of LEC-IDs by LECS ahead of time offers a simpler solution and that this does not impose any significant real-time constraints on LECS and its failure modes. More dynamic methods of LEC_ID assignment are for further study*].

5.6 Client Address Registration

5.6.1 LNNI_REGISTER_REQUEST

[Editor's Note: This text needs more detail.] Client registration requests are handled by servers sending an LNNI_JOIN_REQUEST message up to the root of the tree. As the message passes through all of the intermediate servers to the root server, they add the client's registration information to their local databases. When a client or server crashes, a LNNI_UNREGISTER_REQUEST message is propagated throughout the entire tree, directing each server to remove the appropriate client information from their caches.

Most messages, thus, are sent to the root node, and no further. A few messages, such as proxy ARP requests, topology change messages, and address deregistration messages, are flooded to all nodes in the tree. None of the messages that are sent more than one hop require any acknowledgement beyond a hop-by-hop acknowledgement.

5.6.2 LNNI_LEC_UNREGISTER

When a LEC unregisters either explicitly or by dropping its connections from its LES, the LES generates an LNNI_UNREGISTER message and reliably sends to all its neighbors. Each LES receiving the LNNI_UNREGISTER_REQUEST will purge the corresponding registered and cached entry, and respond back with an LNNI_UNREGISTER_RESPONSE. This protocol allows each LES to maintain an up to date registration database and arp cache. Again, a reliable protocol is used.

5.7 Client Address Resolution

5.7.1 Caching

LE ARP caching is an important LNNI performance optimization; it provides the easiest way of sharing the load for processing LE_ARP_REQUESTs among all the LESs in the network. Note that LE ARP caching denotes storing address registration information for hosts at LE servers other than the one to which the host is connected; it does not denote any caching of proxy ARP information, which is forbidden by the LUNI specification.

ARP information can be learned by a server's watching LE_ARP_RESPONSE packets go by. In order for these caches to contain only up-to-date information, we must ensure that stale information is purged promptly. This happens automatically by virtue of the protocols already used for propagating server failures and host deregistrations. In both of these, cases, a request is propagated throughout the entire (remaining) LNNI tree, indicating the set of hosts no longer connected to the tree, and thus which should be flushed from any caches. In the case of a server crash, the protocol is a LNNI_UNREGISTERSERVER_REQUEST message, and in the case of a simple host de-registration, the protocol is the LNNI_TERMINATE_REQUEST message.

5.7.2 LNNI_ARP

A LES responds to an address resolution request for a destination registered in that LES without passing it onto another LES. A LES may also respond to an address resolution request for a valid cache entry. The entries that are cached in a LES are those entries that are registered within other LESs, and learned from an address resolution message that originates from a LES which has the address registered in its database.

Since cached entries are learned from the address resolution messages, a flag is needed in the address resolution response to indicate whether that address is registered in the originating LES database, or not. The remote address flag may not be sufficient for this purpose as the learning-LES must differentiate between a response from a LES which cached the address as opposed to a LES which has the entry registered in its database. *[Alternatively, a LES may decide to cache entries learned from another caching LES. This is for further study.]*

5.7.3 Address Resolution

LE servers must colaborate in order to handle address registration requests and queries. Each tree node stores the address registration information registered by all of its hosts, and by all of its child servers in the tree, and uses this information to answer ARP requests and forward packets efficiently. Address registration information propagates up the tree to the root node of the tree, and ARP queries also propagate up the tree to the root until they are answered. If an ARP request can not be handled at the top of the tree, the address may be an address learnable only via the proxy ARP mechanism, and thus, if the root server can not resolve an ARP query, it sets a special PROXYFLOOD bit in the LE_ARP_REQUEST packet and floods the updated message throughout the entire tree. This special LE_ARP_REQUEST message is flooded to all servers, and forces each server to send the appropriate ARP request message to all of its directly connected clients. LNNI_TOPOLOGY_CHANGE

A new LNNI topology change protocol between LESs can be used as a means to identify failure/recovery of a LES. A LES issues a LNNI_TOPOLOGY_CHANGE when it identifies failure/recovery of a neighbor LES. The notice of a topology change is propagated through the mesh and tree of LESs by other LESs which receive it. This notification must be reliably propagated and thus can use the same hop by hop acknowledgement system used for the LNNI_JOIN. Multiple LNNI_TOPOLOGY_CHANGE messages might be initiated by different neighbors that identify the same failure. If such multiples of the same message are received, the receiving LES must make sure to propagate only one copy. The LNNI_TOPOLOGY_CHANGE message contains the id of the failed/recovered LES.

When an LNNI topology change message indicating a LES failure is received, the LESs must purge all registered or cached bindings learned from that failed LES, preferably, after a timer expires. In the Master LES approach, through this timer, the failed LES's registered entries can be retained for a while just in case the failed LES recovers shortly after a failure.

When a LES receives its first topology change message indicating failure of a server, it will set the "marking flag" to indicate that a marking period starts. Until that flag is cleared, each LES will mark every newly registered address. Similarly, each LES will mark unregistered addresses keeping them in a database with a "marked" status. The marking is needed to ensure that when the network is healed, the registration/unregistration of "marked" entries, those that are processed during a likely network partitioning, are re-synchronized within the merged network.

If there are other LES failures prior to the recovery of the first failed LES, the marking flag will be incremented while keeping a list of the failed LESs. The flag will be decremented as each failed LES recovers, but it will not be cleared until all failures are healed, or a timer expires.

If a new LES joins the topology when the network is in a marking state, the LES adjacent to the newly joined LES will send its marking flag value so that the new LES knows whether it should perform marking or not.

Once the marking flag is cleared, meaning that the network has completely healed, all LESs will re-validate the uniqueness of the marked registrations using LNNI_JOIN protocol. Similarly, using the LNNI_UNREGISTER protocol, LESs will inform each other of the marked unregistrations so that each LES can purge the marked unregistrations from their registration databases and ARP caches.

5.8 Client Data Transfer

5.8.1 BUS Data Movement

5.9 Client Flush

5.10 Client Topology Change Notification

5.11 Client Terminate

6. Appendix

TBD

[End of Document]