Network Working Group
Internet-Draft
Expires: April 27, 2005

J. Maloy
Ericsson
S. Blake
Modularnet
M. Koning
WindRiver
J. Hadi Salim
Znyx
H. Khosravi
Intel

October 27, 2004

TIPC: Transparent Inter Process Communication Protocol, a Layer 2 TML for the ForCES protocol
draft-maloy-tipc-01.txt

Status of this Memo

Copyright Notice

Conventions used in this document

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in [RFC2119].

Abstract

This document describes TIPC, a protocol intended to be used as a TML (Transport Mapping Layer) for the ForCES protocol[ForCES] when that protocol is transported over L2 carriers such as Ethernet, RapidIO or PCI-Express. TIPC is specially designed for efficient communication within clusters of loosely coupled nodes, and may as such even be used outside the context of being a ForCES protocol carrier. It would even be an excellent candidate as a ForCES pre-association phase setup protocol.

TIPC is a reliable transport protocol typically operating on top of L2 packet networks, but it should also work well on higher-level protocols such as DCCP, TCP, or SCTP.

TIPC offers the following services to its users:

- A functional addressing scheme providing full addressing transparency over the whole cluster.
- A topology information and subscription service, providing up-to-date information about functional and physical topology.
- Lightweight, highly reactive connections reporting errors or destination unreachability within a fraction of a second.
- A reliable multicast service, based on functional addressing, but using the underlying network multicast service when possible.
- Acknowledged, loss-free, error-free, non-duplicated transfer of user data, both in connectionless and connection-oriented mode.
- Configurable congestion control both at bearer, link, and connection level.
- Data fragmentation conforming to discovered carrier MTU size.
- Bundling of multiple user messages into a single TIPC packet in situations where messages cannot be sent immediately.

- Transparent, link-level load sharing and redundancy, through support of heterogeneous multi-homing.
- A slim, non-layered protocol header allowing efficient protocol implementations.

Apart from common process-to-process communication, the design of TIPC even includes the possibibily to commmunicate process-to-kernel and kernel-to-kernel, still with full addressing and interface transparency.

Table of Contents

1. Introduction

This section explains the rationale behind the development of the Telecom Inter Process Communication (TIPC) protocol. It also gives a brief introduction to the services provided by this protocol, as well as the basic concepts needed to understand the further description of the protocol in this document.

1.1 Motivation

There are no standard protocols available today that fully satisfy the special needs of application programs working within highly available, dynamic cluster environments. Clusters may grow or shrink by orders of magnitude, having member nodes crashing and restarting, having routers failing and replaced, having functionality moved around due to load balancing considerations, etc. All this must be possible to handle without significant disturbances of the service offered by the cluster. In order to minimize the effort by the application programmers to deal with such situations, and to maximize the chance that they are handled in a correct and optimal way, the cluster internal communication service should provide special support helping the applications to adapt to changes in the cluster. It should also, when possible, leverage the special conditions present within cluster environments to present a more efficient and more fault-tolerant communication service than more general protocols are capable of. This is the purpose of TIPC.

Version 1 of TIPC has been widely deployed in customer networks. This document describes version 2 of TIPC. An open source implementation of version 2 is available at [TIPC]

1.1.1 Existing Protocols

TCP [RFC793] has the advantage of being ubiquitous, stable, and wellknown by most programmers. Its most significant shortcomings in a real-time cluster environment are the following:

- It lacks any notion of functional addressing and addressing transparency. Mechanisms exist (DNS, CORBA Naming Service) for transparent and dynamic lookup of the correct IP-adress of a destination, but those are in general too static and expensive to use.
- TCP has non-optimal performance, especially for intra-node communication and for short messages in general. For intra-node communication there are other and more efficient mechanisms available, at least on Unix, but then the location of the destination process has to be assumed, and can not be changed. It is desirable to have a protocol working efficiently for both intra-node and inter-node messaging, without forcing the user to

distinguish between these cases in his code.
- The rather heavy connection setup/shutdown scheme of TCP is a disadvantage in a dynamic environment. The minimum number of packets exchanged for even the shortest TCP transaction is nine (SYN, SYNACK etc.), while with TIPC this can be reduced to two, or even to one if connectionless mode is used.
- The connection-oriented nature of TCP makes it impossible to support true multicast.

SCTP [RFC2960] is message oriented, it provides some level of user connection supervision, message bundling, loss-free changeover, and a few more features that may make it more suitable than TCP as an intra-cluster protocol. Otherwise, it has all the drawbacks of TCP already listed above.

Apart from these weaknesses, neither TCP nor SCTP provide any topology information/subscription service, something that has proven very useful both for applications and for management functionality operating within cluster environments.

Both TCP and SCTP are general purpose protocols, in the sense that they can be used safely over the Internet as well as within a closed cluster. This virtual advantage is also their major weakness: they require funtionality and header space to deal with situations that will never happen, or only infrequently, within clusters.

1.1.2 Assumptions

TIPC [TIPC] has been designed based on the following assumptions, empirically known to be valid within most clusters.
- Most messages cross only one direct hop.
- Transfer time for most messages is short.
- Most messages are passed over intra-cluster connections.
- Packet loss rate is normally low; retransmission is infrequent.
- Available bandwidth and memory volume is normally high.
- For all relevant bearers packets are check-summed by hardware.
- The number of inter-communicating nodes is relatively static and limited at any moment in time.
- Security is a less crucial issue in closed clusters than on the Internet.

Because of the above one can use a simple, traffic-driven, fixed-size sliding window protocol located at the signalling link level, rather than a timer-driven transport level protocol. This in turn gives a lot of other advantages, such as earlier release of transmission buffers, earlier packet loss detection and retransmission, earlier detection of node unavailability, only to mention some. Of course, situations with long transfer delays, high loss rates, long messages,

security issues, etc. must be dealt with as well, but rather from the viewpoint of being exceptions than as the general rule.

1.2 Architectural View

TIPC should be seen as a layer between the TIPC user, the ForCES protocol, and a packet transport service such as Ethernet, ATM, DCCP, TCP, or SCTP. The latter are denoted by the generic term "bearer service" or just "bearer" throughout this document.

TIPC provides reliable transfer of user messages between TIPC users, or more specifically between two TIPC ports, which are the endpoints of all TIPC communication. A TIPC user normally means a user process, but may also be a kernel-level function or a driver, for which a specific interface has been defined.

Described by standard terminology TIPC spans the level of transport, network, and signalling link layers, although this does not inhibit it from using another transport level protocol as bearer, so that e.g. an SCTP association may serve as bearer for a TIPC signalling link.

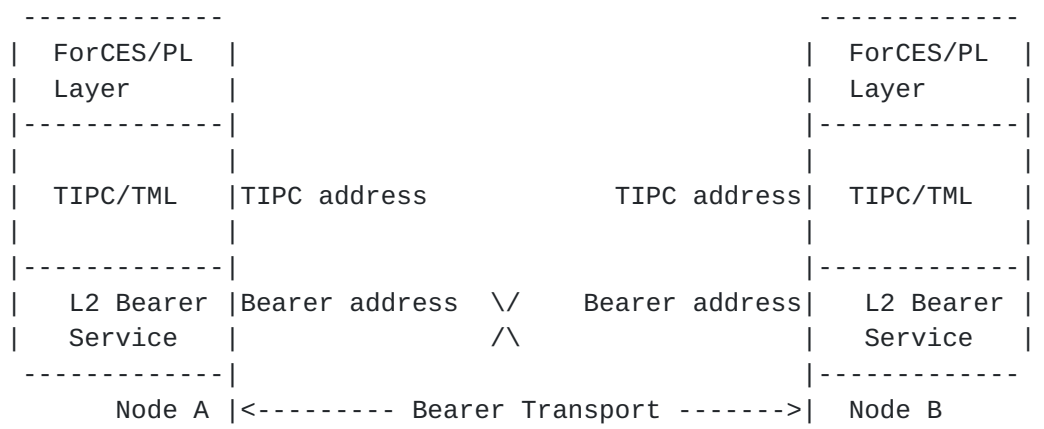

Figure 1: Architectural view of TIPC

1.3 Functional View

Functionally TIPC can be described as consisting of several layers performing different tasks, as shown in Figure 2. It must be emphasized that this layering reflects a functional model, not the way TIPC should be (or actually is) implemented.

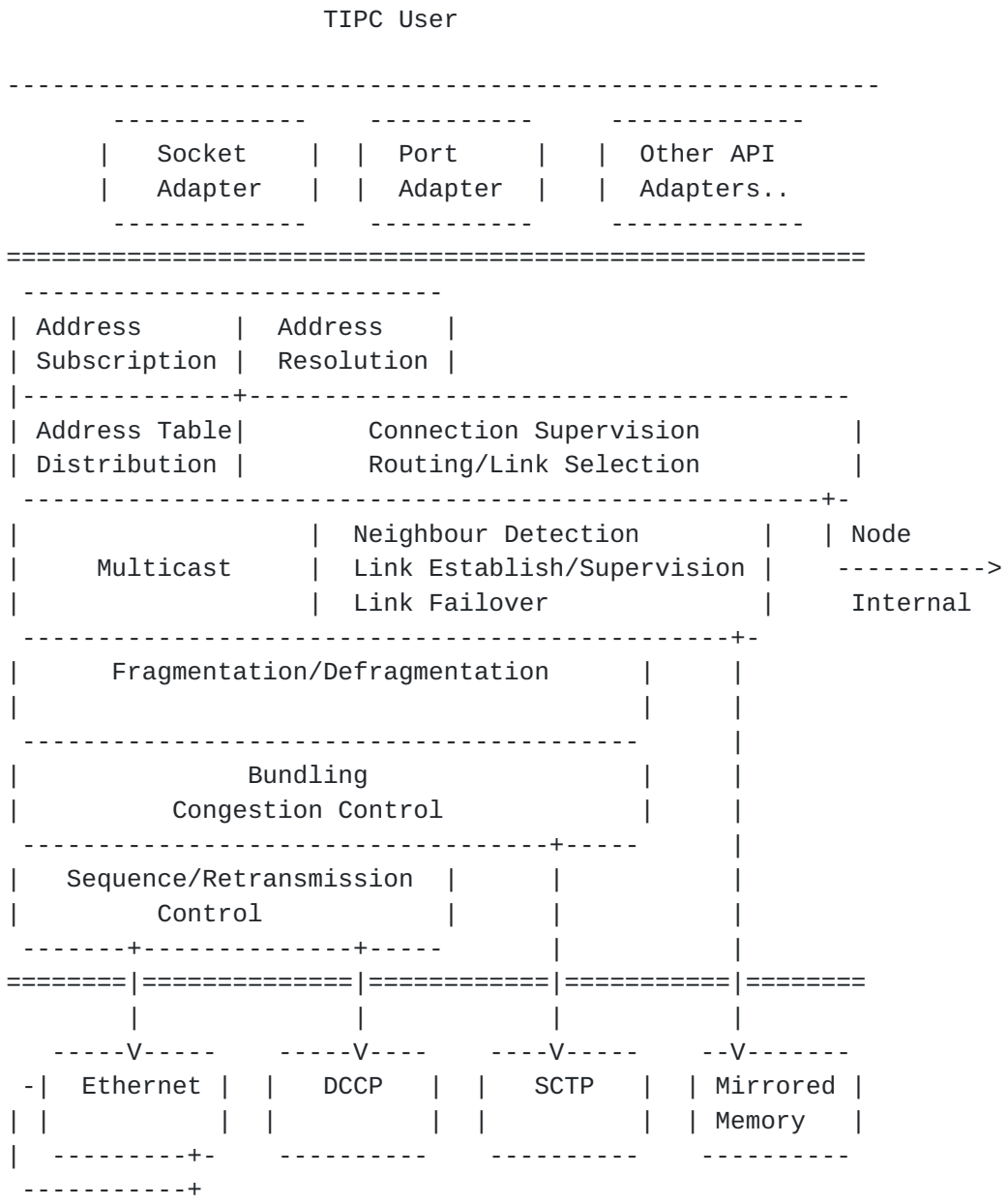

Figure 2: Functional view of TIPC

1.3.1 API Adapters

TIPC makes no assumptions about which APIs should be used, except that they must allow access to the TIPC services. It is possible to provide all functionality via a standard socket interface, an asynchronous port API, and any other form of dedicated interface that can be motivated. In these layers there is also support for transport-level congestion and overload protection control.

1.3.2 Address Subscription

The service "Topology Information and Subscription" provides the ability to interrogate and if necessary subscribe for the availability of a functional or physical address. This helps the application to synchronize its startup, and may even serve as a simple, distributed event channel if used with care.

1.3.3 Address Distribution

Functional addresses must be equally available within the whole cluster node. In order for a message to reach its destination they must also at some stage be translated into a physical address. For performance and fault tolerance reasons it is not acceptable to keep the necessary translation tables in one node, but rather TIPC must ensure that they are distributed to all nodes in the cluster, and that they are kept consistent at any time. This is the task of the Address Distribution Service, also called Name Distribution Service.

1.3.4 Address Translation

The translation from a functional to a physical address is performed on-the-fly during message sending by this functional layer. It goes without saying that this step must use an efficient algorithm, but in many cases it can even be omitted altogether. When it makes sense, the sender may choose to use a physical address instead, e.g. a server responding to a connection setup request, or when communication is connection-oriented.

1.3.5 Multicast

This layer, supported by the underlying three layers, provides a reliable intra-cluster broadcast service, typically defined as a semi-static multicast group over the underlying bearer. It also provides the same features as an ordinary unicast link, such as message fragmentation, message bundling, and congestion control.

1.3.6 Connection Supervision

There are several mechanisms to ensure immediate detection and report of connection failure.

1.3.7 Routing and Link Selection

This is the step of finding the correct destination node, and, if applicable, the correct next-hop router node, plus selecting the right link to use for reaching that node. If the destination node turns out to be the own node, the rest of the stack is omitted, and the message is sent directly to the receiving port.

1.3.8 Neighbour Detection

When a node is started it must make the rest of the cluster aware of its existence, and itself learn the topology of the cluster. By default this is done by use of broadcast, but there are other methods available.

1.3.9 Link Establishment/Supervision

Once a neighbouring node has been detected on a bearer, a signalling link is established towards it. The functional state of that link has to be supervised continuously, and proper action taken if it fails.

1.3.10 Link Failover

TIPC on a node will establish one link per-destination node and functional bearer instance, typically one per-configured ethernet interface. Normally these will run in parallel and share load equally, but special care has to be taken during the transition period when a link comes up or goes down, to ensure the guaranteed cardinality and sequentiality of the message delivery. This is done by this layer.

1.3.11 Fragmentation/Defragmentation

When necessary TIPC fragments and reassembles messages that can not be contained within one MTU-size packet.

1.3.12 Bundling

Whenever there is some kind of congestion situation, i.e. when a bearer or a link can not immediately send a packet as requested, TIPC starts to bundle messages into packets already waiting to be sent. When the congestion abates the waiting packets are sent immediately,

and unbundled at the receiving node.

1.3.13 Congestion Control

When a bearer instance becomes congested, e.g. it is unable to accept more outgoing packets, all links on that bearer are marked as congested, and no more messages are attempted to be sent over those links until the bearer opens up again for traffic. During this transition time messages are queued or bundled on the links, and then sent whenever the congestion has abated. A similar mechanism is used when the send window of a link becomes full, but affects only that particular link.

1.3.14 Sequence and Retransmission Control

This layer ensures the cardinality and sequentiality of packets over a link.

1.3.15 Bearer Layer

This layer adapts to some connectionless or connection-oriented transport service, providing the necessary information and services to enable the upper layers to perform their tasks.

1.4 Terminology

1.4.1 ForCES Terminolgy

- ForCES Protocol: While there may be multiple protocols used within the overall ForCES architecture, the term "ForCES protocol" refers only to the protocol used at the Fp reference point in the ForCES Framework in RFC3746 [RFC3746]. This protocol does not apply to CE-to-CE communication, FE-to-FE communication, or to communication between FE and CE managers. Basically, the ForCES protocol works in a master-slave mode in which FEs are slaves and CEs are masters.
- ForCES Protocol Layer (ForCES PL): A layer in ForCES protocol architecture that defines the ForCES protocol messages, the protocol state transfer scheme, as well as the ForCES protocol architecture itself (including requirements of ForCES TML (see below)). Specifications of ForCES PL are defined by this document.
- ForCES Protocol Transport Mapping Layer (ForCES TML): A layer in ForCES protocol architecture that specifically addresses the protocol message transportation issues, such as how the protocol messages are mapped to different transport media (like TCP, IP, ATM, Ethernet, etc), and how to achieve and implement reliability, multicast, ordering, etc. This document defines an L2/Ethernet

based ForCES TML.

1.4.2 TIPC Specific Terminolgy

- Port: The endpoint of all user communication. On Unix it typically takes the form of a socket.
- Zone: A "super-cluster" of clusters interconnected via TIPC.
- Cluster: A part of a zone where all nodes are directly interconnected (fully meshed).
- Node: A physical computer within a cluster, identified by a TIPC address.
- System Node: A node having direct links to all other system nodes in the cluster, and a TIPC address defined within a certain range. When using the term 'node' in the remainder of this document we normally mean 'system node',unless the context makes a different interpretation obvious.
- Secondary Node: A node identified by a TIPC address within a certain range, and potentially having limited physical connectivity to the rest of the cluster. Secondary nodes can communicate with all system nodes in the cluster, and vice versa, but the messages may have to pass via a system node acting as router. Secondary nodes can not communicate with each other.
- Link: A signalling link connecting two nodes, performing tasks such as message transfer, sequence ordering, retransmission etc. A node pair may be interconnected by 1 or 2 parallel links, in load sharing or active/standby configuration.
- Bearer: A generic term for an instance of a physical or logical transport media, such as Ethernet, ATM/AAL or DCCP.
- Network Address: A TIPC internal node identifier. It is in reality a 32 bit integer, subdivided into three fields (8/12/12), representing zone, cluster and node number respectively. Normally depicted as <Z.C.N>.
- Network Identity: A TIPC internal identifier, used to keep different TIPC networks separated from each other, e.g. on a LAN in a lab environment.
- Location transparency, sometimes called addressing tranparency, is the ability to let processes communicate within a cluster without either of them knowing the physical location of their peer.
- Port Name: (or just Name) A persistent functional address identifying a port within a zone. A port may move between nodes while retaining its name. For load sharing and redundancy purposes several ports may bind to the same name.
- Port Identity: A volatile address identifying a unique physical port within a zone. Once a physical port is deleted its identity will not be reused for a very long time.
- Connection: A logical channel for passing messages between two ports. Once a connection is established no address need be indicated when sending a message from any of the endpoints. A

connection also implies automatic supervision of the endpoints' existence and state.

- Message Bundling: The act of bundling several messages into one bearer level packet, typically an Ethernet frame. TIPC bundles messages e.g. during media congestion.
- Message Fragmentation: Dividing a long message into several bearer-level packets, and reassembling the fragments at the receiving end.
- Message Forwarding: Ability to pass a received message on to a new destination port while pretending that the original sender port is the original sender.
- Link Failover: Moving all traffic from a failing link to the remaining link, while retaining original sequence order and cardinality.
- Naming Table: A TIPC internal table which keeps track of the mapping between port names and corresponding port identities. It performs an on-the-fly translation from the one to the other during the message transfer phase.
- Message: The unit of data delivered from one user to another, i.e. between ports.
- Packet: The unit of data sent over a bearer. It may contain one or more complete TIPC messages, as well as fragments of a message.
- Broadcast: The notion of sending a copy of the same message to all other nodes in the cluster. Note that what is considered a broadcast from the TIPC viewpoint typically is mapped onto a multicast at the bearer (Ethernet or DCCP) level.
- Multicast: Sending a copy of the same message to muliple receivers by one user call. In TIPC multicasts may be transferred both by broadcast and unicast between nodes, dependent of the number of identified receivers and the capabilities of the bearer layer.
- Unicast: Sending a message to one particular destination, i.e. over a TIPC link.
- Domain: A TIPC network addess designating a part of a TIPC network. E.g., <Z.C.N> means the specific node with that address, <Z.C.0> any node within the specified cluster, and <Z.0.0> any node within the specified zone. <0.0.0> means any node, anywhere within the network, except when it is used as Lookup Domain.
- Scope: A domain around a given node, as seen from that node. E.g. <own_zone.own_cluster.own_node> or <own_zone.0.0>.

1.5 Abbreviations

- MAC - Message Authentication Code [RFC2104]
- MTU - Maximum Transmission Unit
- API - Application Programming Interface
- RTT - Round Trip Time, the elapsed time from the moment a message is sent to a destination to the moment it arrives back to the sender, provided the message is immediately bounced back from

the sender. Typically on the order of 100 usecs, process-to-process, between 2 Ghz CPUs via a 100 Mbps Ethernet switch.

2. Mapping ForCES/PL to TIPC/TML

2.1 Fulfilment of TML Requirements

- Reliability: TIPC is a protocol guaranteeing sequential, loss-free, non-duplicated delivery of checksummed messages, as described in 3.7.3. Whenever needed, each individual socket can be set to be "unreliable", meaning that all messages sent from that socket has the "drop"-bit (see 4.1.2) set.
- Security: For now, TIPC can only guarantee message and endpoint authenticity for closed networks, e.g. a trusted LAN or bus. Since no router can yet forward TIPC/Ethernet packets it is impossible to inject spoofed packets into such a network. How this should be handled when the nodes connected to the LAN or bus can not be trusted remains TBD. The same is valid for message encryption.
- Congestion Control: TIPC provides three levels of congestion control, as described in in sections 3.5.5,3.7.6,3.7.7 and 3.9.6. The ForCES PL may receive indication of destination socket or node congestion when setting up a connection. For established connections, socket congestion is handled transparently by the TIPC connection flow control scheme, while node congestion will result in connection abortion. TIPC will also inform the PL layer about the reason for any connection abortion, such as node overload, node crash, or process crash. When a connection, is aborted, the indication will be given to the PL immediately. As connections require very few system resources, in particular regarding supervision timers, CE-FE connections can normally be established directly process to process, with no restraints on number of parallel connections. Connections dedicated to traffic data transfer should be set to "non-reliable" in the FE-CE direction, to make it possible to fence off DoS attacks, while the CE-FE direction, as well as both directions of control data connections, should be established as "reliable".
- Uni/multi/broadcast: TIPC provides functional multicast, and broadcast as a special case of that, to the PL layer. This function takes advantage of any broadcast transport facility in the bearer, such as Ethernet, and will use replicated unicast if this feature is missing, as with TCP. Furtheremore, it is configurable when L2 broadcast should be used, so that multicasts identified to have only a few destination nodes, as well as ditto retransmissions, in reality may be sent as replicated unicast.
- Timeliness: Messages are delivered without any delay whatsoever over L2 networks. With Ethernet this will in practice mean a delivery time, process-to-process, in the order of 100 microseconds of a typical one-packet message. TIPC does not allow obsoleting messages.

- HA considerations: L2 link failure detection and failover is handled transparently by TIPC, and does not affect the PL layer. If there is a complete communication failure between two nodes, the PL layer will be informed. Any non-delivered messages will be returned to the sending PL, along with the failure reason, and it is up to the PL to handle such a situation as intelligently as possible.
- Encapsulations: The TIPC message formats are defined in section 4 of this document. There is no particular encapsulation distinguishing the PL layer from other users.
- TIPC provides four message importance priorities, instead of eight, as required in [ForCES]. However, the rationale for requiring as much as eigth levels seems weak; extensive experience from use of TIPC indicates that four levels is perfectly adequate. If it is decided that the ForCES PL must have eight levels, those will have to be mapped down 2-to-1 to the TIPC priorities.

2.2 Address Mapping

[ForCES] decribes two address levels, the node level (CE/FE identity and multicast addresses), and the LFB level (type/instance tuple). These can easily be mapped down to the TIPC address concept of Port Name and Port Name Sequence. The following example illustrates such a mapping:

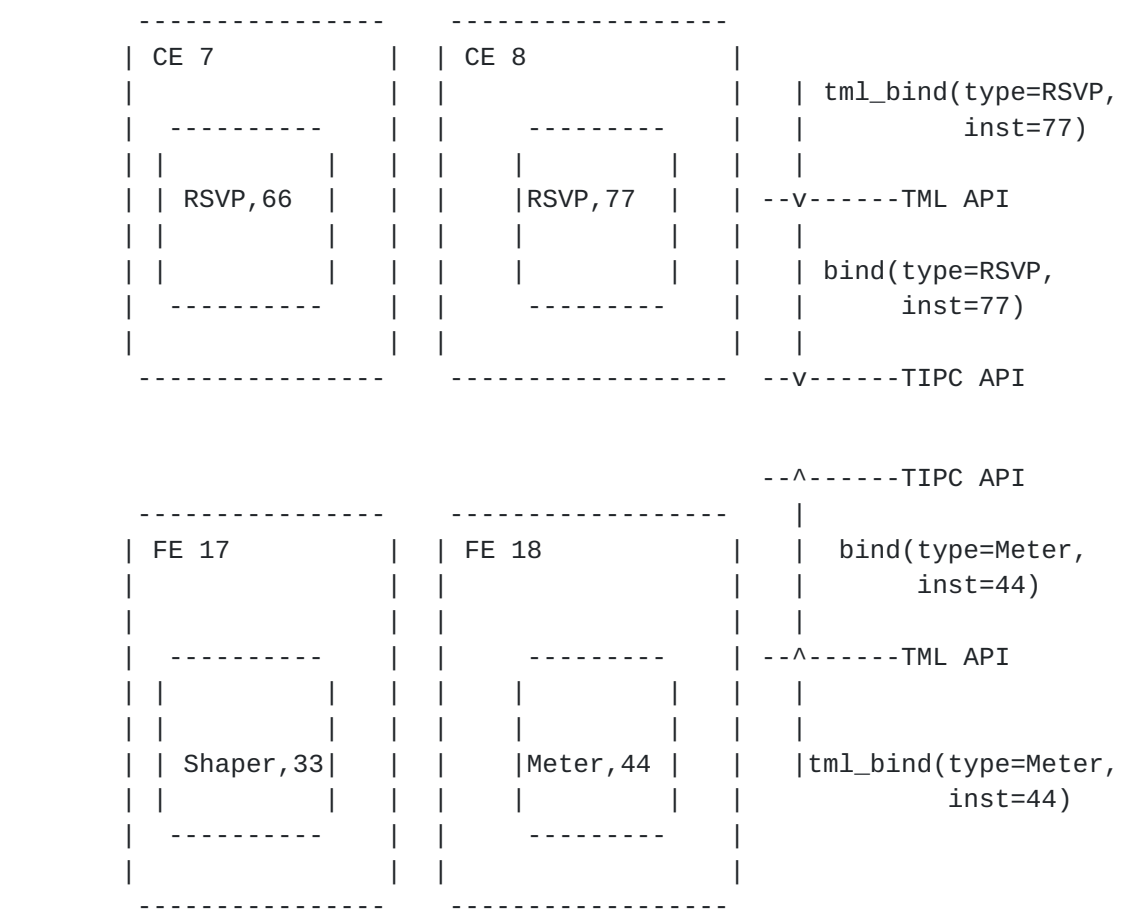

Figure 3: ForCES/PL to TIPC/TML address mapping

An LFB wanting to send a message to a block in a CE would use the following call:

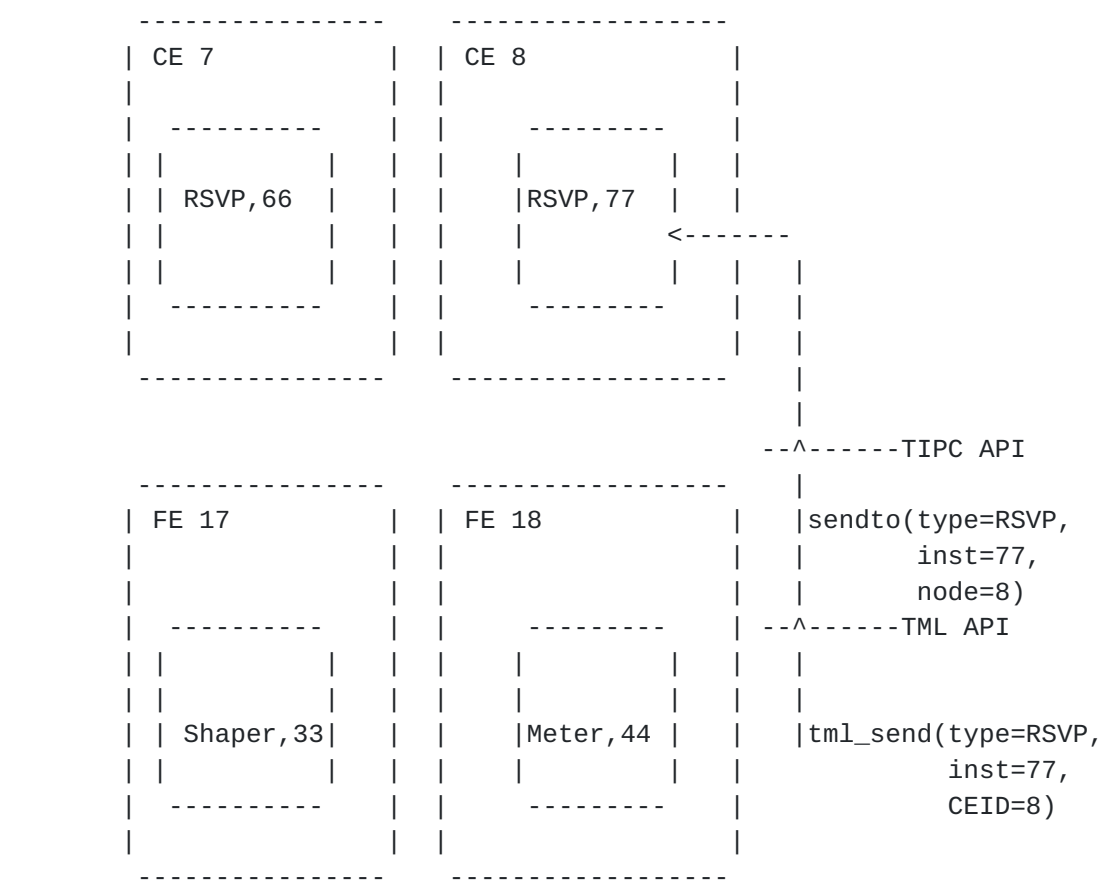

Figure 4: FE-CE messaging

Note that unless we coordinate the instance numbers for a block across the whole cluster, the constructed addresses are not guaranteed unique. Such a coordination can be achieved either statically through configuration data, or dynamically through use of the TIPC topology subscription mechanism. But this is not a prerequisite for this model to work, since the the current version of ForCES/PL layer anyway assumes that the sender knows the identity of the destination node. However, the model becomes immensely more flexible if we can remove this assumption, and ensure that block addresses are cluster unique. The message sender in the example above would never need to indicate the CEID, and would never need to be updated in case the configuration changes, e.g. that RSVP number 77 moves over to CE number 7.

FE and CE Protocol Objects can be considered as just other LFB's and can connect to each other as in the next example.

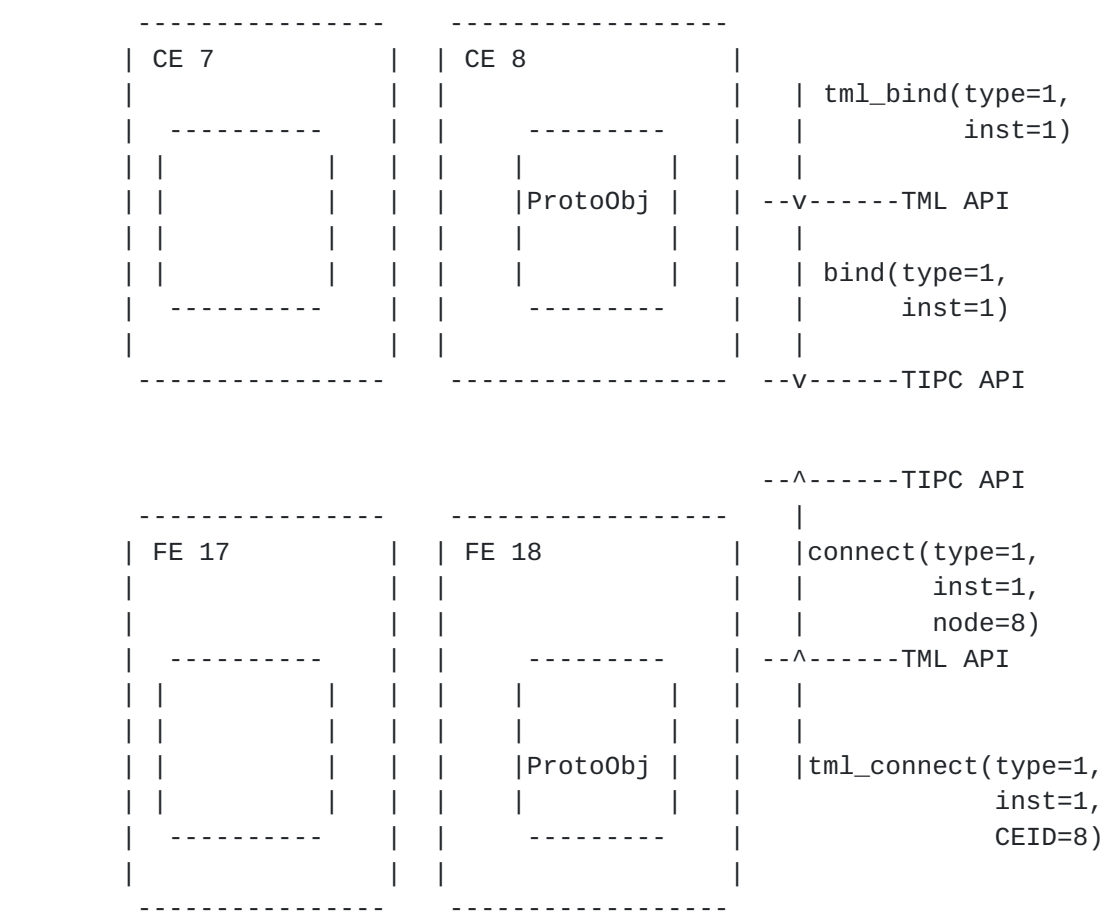

Figure 5: FE-CE Protocol Object connect

Note that there is nothing stopping any LFB to establish its own connection to any block in a CE (or another FE). Given that TIPC connections don't need individual heartbeating this should not be a problem, but whether this is desirable or anticipated by the authors of [ForCES] remains unclear. Also note that FE/CE protocol objects don't need to start any heartbeating at all. If they want a node failure detection time lower than the TIPC default value, they only need to configure (dynamically) the corresponding TIPC links accordingly. Since TIPC does its heartbeating in driver mode,and also subscribes for low-level carrier failure detection, there is unlikely ForCES/PL can do this better.

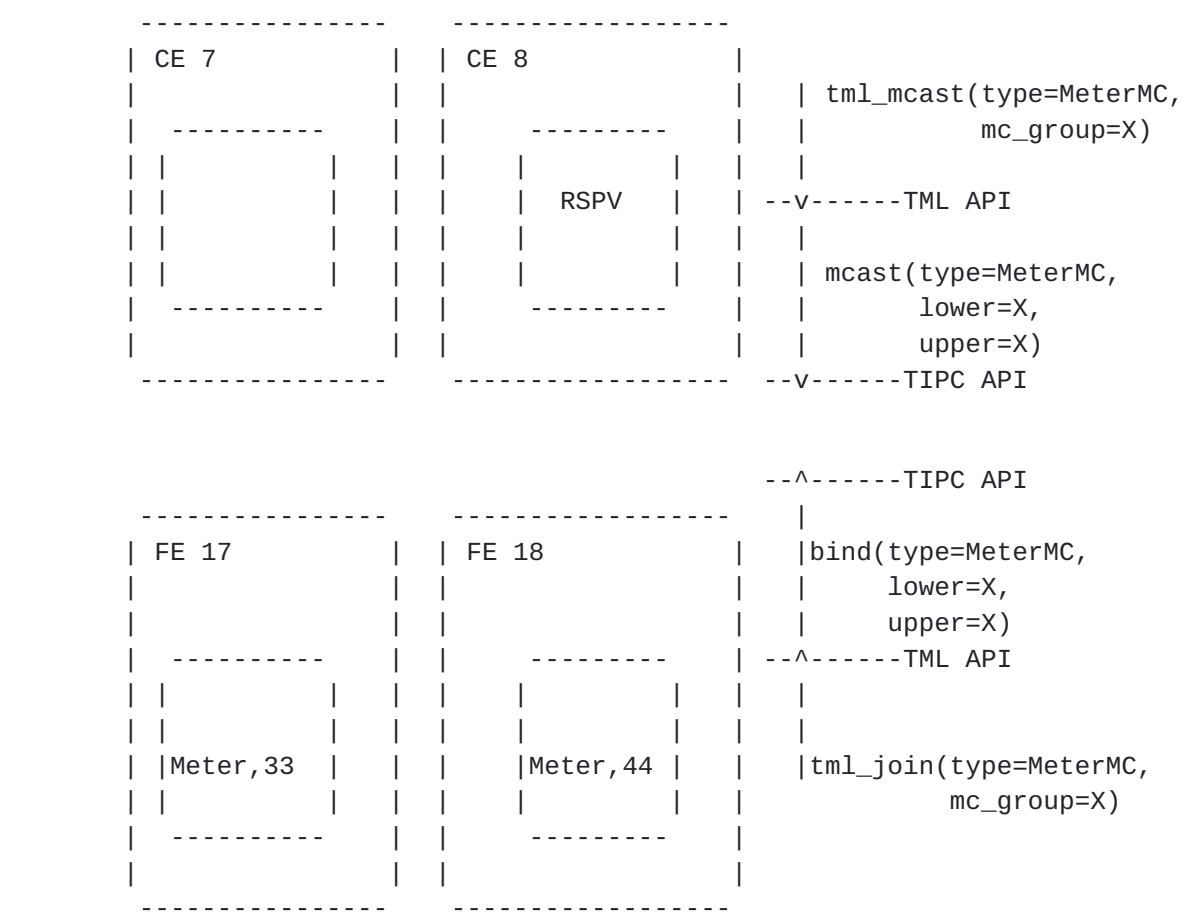

Figure 6: CE-FE Multicast

Multicast addresses are mapped to TIPC port name sequences according to the figure above. We have to assign a new type identity,"MeterMC" instead of "Meter" to avoid introducing a limitation: otherwise "ordinary" Meter instance numbers might collide with the value range of multicast addresses (see [ForCES]),and cause utter confusion. Of course, the binding of the multicast address can still be done to the same socket as the unicast address.

3. TIPC Features

3.1 Network Topology

From a TIPC viewpoint the network is organized in a five-layer structure:

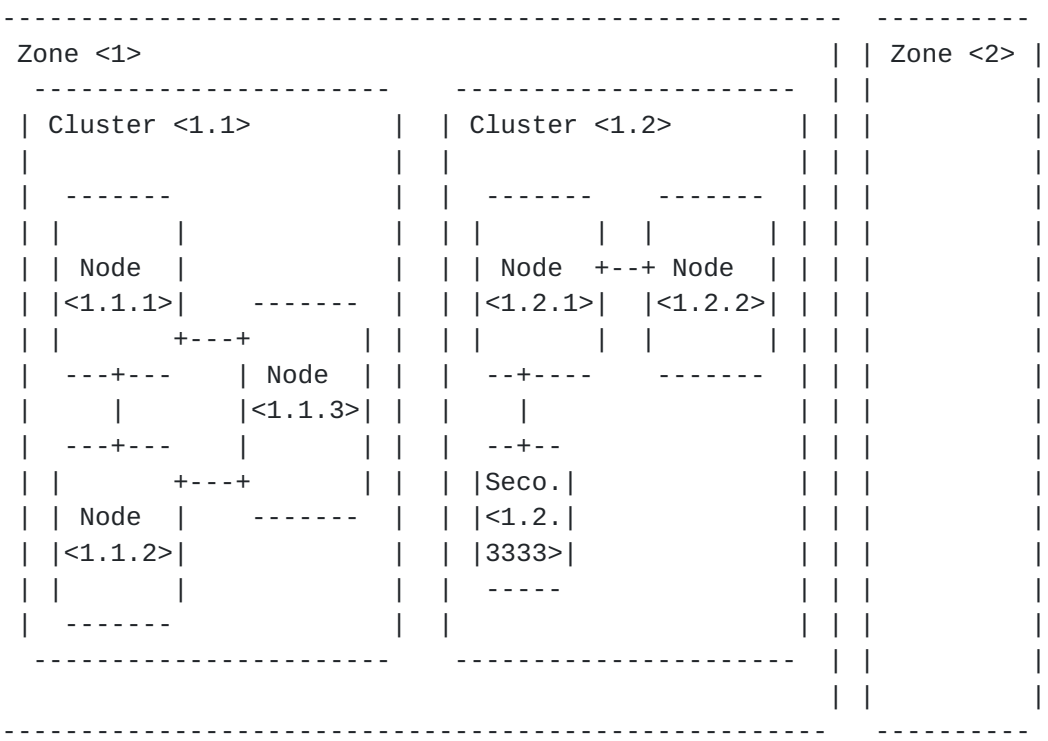

Figure 7: TIPC network topology

3.1.1 Network

The top level is the TIPC network as such. This is the ensemble of all zones interconnected via TIPC, i.e. the domain where any node can reach any other node by using a TIPC network address. The zones within such a network must be directly interconnected all-to-all via TIPC links, since there is no zone-level routing, i.e. a message can not pass from one zone to another via an intermediate zone. Any number of links between two zones is permitted, and normally there will be more than one for redundancy reasons.

It is possible to create distinct, isolated networks, even on the same LAN, reusing the same network addresses, by assigning each

network a Network Identity. This identity is not an address, and only serves the purpose of isolating networks from each other. Networks with different identities can not communicate with each other via TIPC.

3.1.2 Zone

The next level in the hierarchy is the zone. This is the largest scope of location transparency within a network, i.e. the domain where a programmer does not need to worry about network addresses. The maximum number of zones in a network is 255, but this may be implementation dependent, and should be configurable.

3.1.3 Cluster

The third level is the cluster. Cluster nodes within a zone must be interconnected in a full mesh, but just as with zones, there is no need for fully meshed node links between clusters. The maximum number of clusters within a zone is 4095, but this should be made configurable in the actual implementation.

3.1.4 Node

The fourth level is the individual system node, or just node. Nodes within a cluster must be interconnected all-to-all. There may be up to 2047 system nodes in a cluster.

3.1.5 Secondary Node

The fifth level is the secondary node. Secondary nodes belong to a cluster, just like system nodes, and provide the same properties regarding location transparency and availability as system nodes. The difference is that secondary nodes don't need full physical connectivity to all other nodes in the cluster, -one link to one system node is sufficient, although there may be more for redundancy reasons.

There may be up to 2047 secondary nodes in a cluster, the node part of their identities being within the range 2048-4095. In fact, from a TIPC viewpoint this special address is the only thing distinguishing a secondary node from a system node.

TIPC does not allow secondary nodes to establish links directly to each other, since they are supposed to play a subordinate role in the system.

3.2 Addressing

3.2.1 Location Transparency

TIPC provides two functional address types, Port Name and Port Name Sequence, to support location transparency, and two physical address types, Network Address and Port Identity, to be used when physical location knowledge is necessary for the user.

3.2.2 Network Address

A physical entity within a network is identified internally by a TIPC Network Address. This address is a 32-bit integer, structured into three fields, zone (8 MSB), cluster, (12 bits), and node (12 LSB). The address is only filled in with as much information as is relevant for the entity concerned, e.g. a zone may be identified as 0x03000000 (<3.0.0>), a cluster as 0x03001000 (<3.1.0>), and a node as 0x03001005 (<3.1.5>). Any of these formats is sufficient for the TIPC routing function to find a valid destination for a message.

3.2.3 Port Identity

This address is produced internally by TIPC when a port is created, and is only valid as long as that physical instance of the port exists. It consists of two 32-bit integers. The first one is a random number with a period of 2^31-1, the second one is a fully qualified network address with the internal format as described earlier. A port identity may be used the same way as a port name, for connectionless communication or connection setup, as long as the user is aware of its limitations. The main advantage with using this address type over a port name is that it avoids the potentially expensive binding operation in the destination port, something which may be desirable for performance reasons.

3.2.4 Port Name

A port name is a persistent address typically used for connectionless communication and for setting up connections. Binding a port name to a port roughly corresponds to binding a socket to a port number in TCP, except that the port name is unique and has validity for the whole publishing scope indicated in the bind operation, not only for a specific node. This means that no network address has to be given by the caller when setting up a connection, unless he explicitly wants to reach a certain node, cluster or zone.

A port name consists of two 32-bits integers. The first integer is called the Name Type, and typically identifies a certain service type or functionality. The second integer is called the Name Instance,

and is used as a key for accessing a certain instance of the requested service.

The type/instance structure of a port name helps giving support for both service partitioning and service load sharing.

When a port name is used as destination address for a message, it must be translated by TIPC to a port identity before it can reach it destination. This translation is performed on a node within the lookup scope indicated along with the port name.

3.2.5 Port Name Sequence

To give further support for service partitioning TIPC even provides an address type called Port Name Sequence, or just Name Sequence. This is a three-integer structure defining a range of port names, i.e. a name type plus the lower limit of and the upper boundary of the range. By allowing a port to bind to a sequence, instead of just an individual port name, it is possible to partition the service's range of responsibility into sub-ranges, without having to create a vast number of ports to do so.

There are very few limitations on how name sequences may be bound to ports. One may bind many different sequences, or many instances of the same sequence, to the same port, to different ports on the same node, or to different ports anywhere in the cluster or zone. The only restriction, in reality imposed by the implementation complexity it would involve, is that no partially overlapping sequences of the same name type may exist within the same publishing scope.

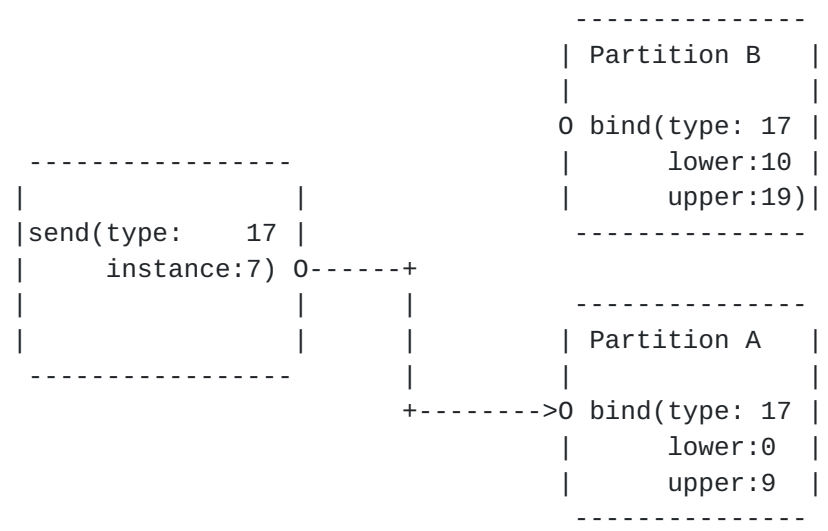

Figure 8: Functional addressing, using port name and port name sequence

When a port name is used as a destination address it is never used alone, contrary to what is indicated in Figure 8. It has to be accompanied by a network address stating the scope and policy for the lookup of the port name. This will be described later.

3.2.6 Multicast Addressing

The concept of functional addressing is also used to provide multicast functionality. If the sender of a message indicates a port name sequence instead of a port name, a replica of the message will be sent to all ports bound to a name sequence fully or partially overlapping with the sequence indicated.

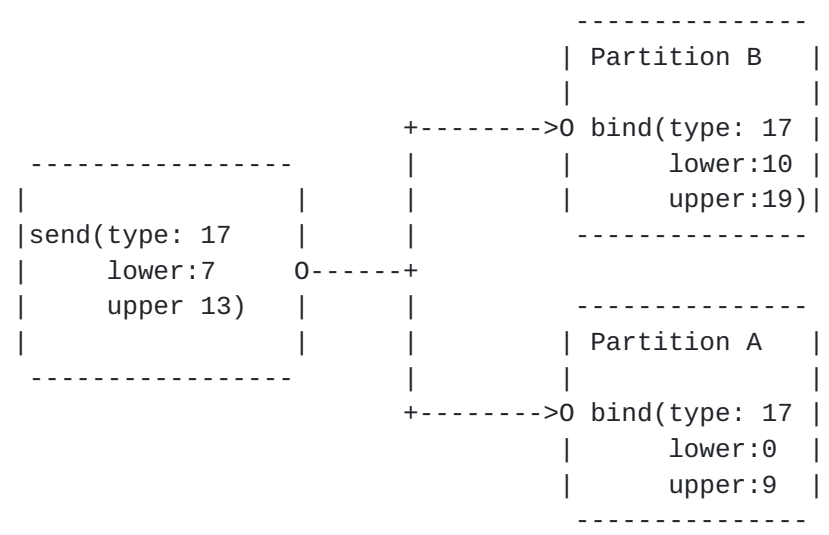

Figure 9: Functional multicast, using port name sequence

Only one replica of the message will be sent to each identified target port, even if it is bound to more than one overlapping name sequence.

This function will whenever possible and considered advantageous make use of the reliable cluster broadcast service also supported by TIPC.

3.2.7 Publishing Scope

The default visibility scope of a published (bound) port name is the local cluster. If the publication issuer wants to give it some other visibility he must indicate this explicitly when binding the port. The scopes available are:

Value	Meaning
1	Visibility within whole own zone
2	Visibility within whole own cluster
3	Visibility limited to own node

3.2.8 Lookup Policies

When a port name is looked up in the TIPC internal naming table for translation to a port identity the following rules apply:

If indicated lookup domain is <Z.C.N>, the lookup algorithm must choose a matching publication from that particular node. If nothing

is found on the given node, it must give up and reject the request, even if other matching publications exist within the zone.

If the lookup domain is <Z.C.0>, the algorithm must select round-robin among all matching publications within that cluster, treating node local publications no different than the others. If nothing is found within the given cluster, it must give up and reject the request, even if other matching publications exist within the zone.

If the lookup domain is <Z.0.0>, the algorithm must select round-robin among all concerned publications within that zone, treating node or cluster local publications no different than the others. If nothing is found, it must give up and reject the request.

A lookup domain of <0.0.0> means that the nearest found publication must be selected. First a lookup with scope <own zone.own cluster.own node> is attempted. If that fails, a lookup with the scope <own zone.own cluster.0> is tried, and finally, if that fails, a lookup with the scope <own zone.0.0>. If that fails the request must be rejected.

Round-robin based lookup means that the algorithm must select equally among all the matching publications within the given scope. In practice this means stepping the root pointer to a circular list referring to those publications between each lookup.

3.2.9 Name Translation

Recommended Algorithm.

3.2.10 Distributed Naming Table

The TIPC internal naming table is used for translation from a port name to a corresponding port identity, or from a port name sequence to a corresponding set of port identities. In order to achieve acceptable translation times and fault tolerance, an instance of this table must exist on each node. Each instance of the table must be kept consistent with all other instances within the same zone, and there must be no unnecessary delays in the update the neighbouring table instances when a port name sequence is published or withdrawn. Inconsistencies are only tolerated for the short timespan it takes for update messages to reach the neigbouring nodes, or for the time it takes for a node to detect that a neighbouring node has disappeared.

When a node establishes contact with a new node in the cluster or the zone, it must immediately send out the necessary number of

NAME_DISTRIBUTOR/ PUBLICATION messages to that node, in order to let it update its local naming table instance.

When a node looses contact with another node, it must immediately clean its naming table from all entries pertaining to that node.

When a port name sequence is published on a node, TIPC must immediately send out a NAME_DISTRIBUTOR/PUBLICATION message to all nodes within the publishing scope, in order to have them update their tables.

When a port name sequence is withdrawn on a node, TIPC must immediately send out a NAME_DISTRIBUTOR/WITHDRAWAL message to all nodes within the publishing scope, in order to have them remove the corresponding entry from their tables.

Temporary table inconsistencies may occur, despite the above, and are handled as follows: If a successful lookup on one node leads to a non-existing port on another node, the lookup is repeated on that node. If this lookup succeeds, but again leads to a non-existing port, another lookup is done. This procedure can be repeated up to six times before giving up and rejecting the message.

3.3 Topology Services

TIPC provides a mechanism for inquiring about or subscribing for the availability of port names or ranges of port names. The service is built on and uses the contents of the node local instance of the naming table.

3.3.1 Inquiry

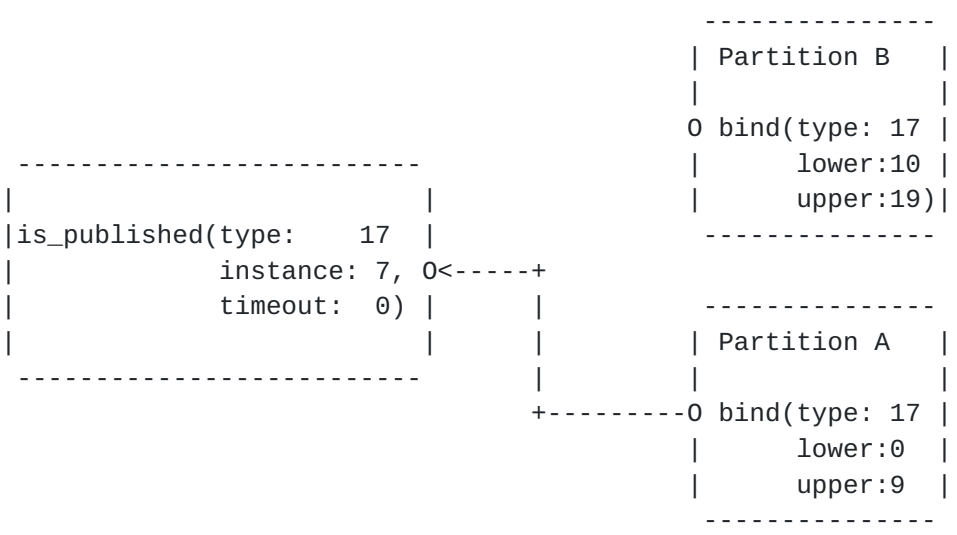

Figure 11: Inquiry about existence of a port name

Inquiries are synchronous requests to TIPC about a port name. A timer value in msecs may be given along with the request, indicating that the call should not return until the port name has been published, or until the timer expires, whichever comes first, indicated in the return value of the call. A timeout of zero instructs the call to return immediately, a timeout of 0xffffffff indicates that the call should not return until the port name requested has been published.

3.3.2 Subscriptions

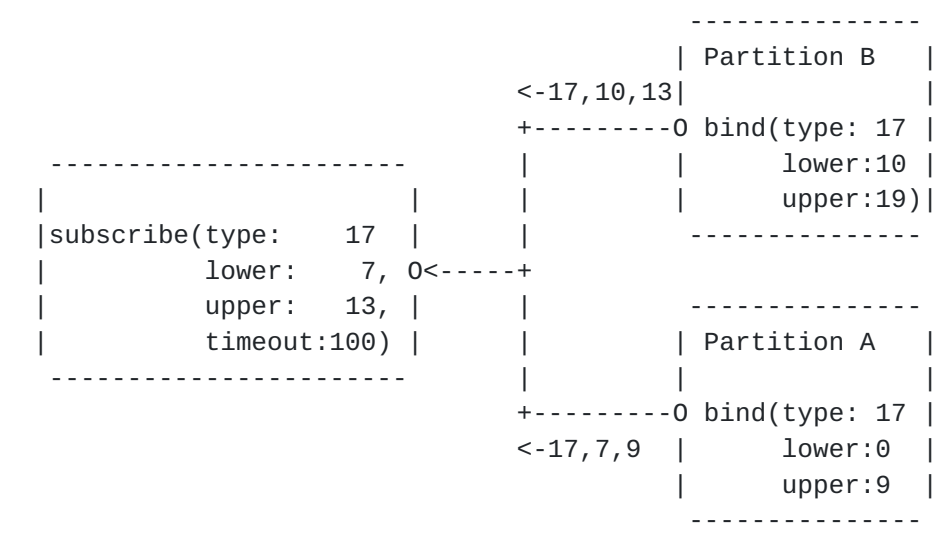

Figure 12: Subscription about existence of sequences within a range

A subscription is a non-blocking request to TIPC, telling it to indicate when a name sequence within the requested range is published or withdrawn. Such events will be issued repeatedly for any changes pertaining to the range until the given timer expires. The timer values are interpreted the same way as for inquiries. Subscription for a particular port name is equivalent to indicating the same value in "lower" and "upper".

Each event will indicate the overlapping part between the requested range and the actual published range, as it is also shown in the figure above.

3.3.3 Functional Topology

The functional topology of the cluster can be continuously kept track of by subscribing for the relevant port names or sequences.

3.3.4 Physical Topology

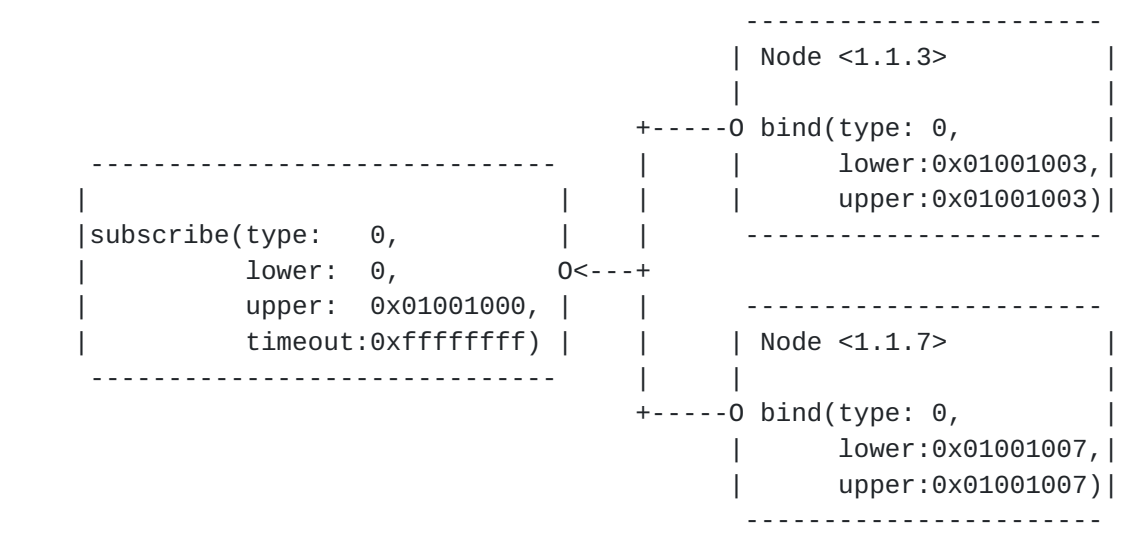

Figure 13: Subscription for physical topology of cluster <1.1>

The physical cluster topology can be considered a special case of the functional topology, and can be kept track of in the same way. Hence, to subscribe for the availability/disappearance of a specific node, a group of nodes, or a remote cluster, the user specifies a dedicated port name sequence, representing this "function". In this particular case, TIPC will itself publish the corresponding port name as soon as it discovers or looses contact with a node. The special name type 0 (zero) is used for this purpose.

3.4 Ports

3.4.1 Port State Machine

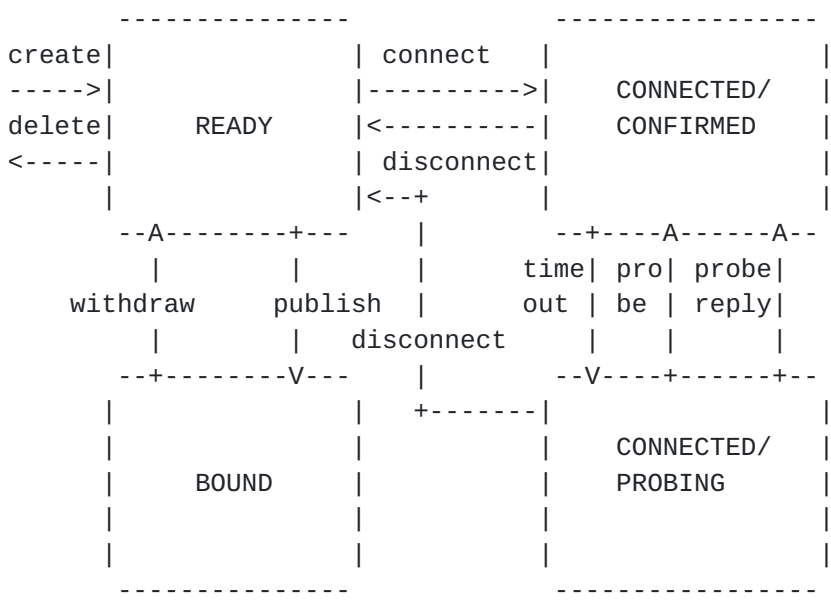

Figure 14: Port FSM for non-error events

The port state machine is relatively simple for normal, non-error events, as illustrated in Figure 14.

A port has three main STATES, as described below:

READY: The port is in its basic state, and is ready to receive any normal state event.
BOUND: The port has been bound to (published with) one or more port name sequences.
CONNECTED: The port has been connected to some other port in the network, i.e. it has stored the identity of that port, and a flag "connected" is set in the port.

The CONNECTED state has two sub-states, reflecting its supervision of the connected peer:

CONNECTED/CONFIRMED: The port has had confirmed that the other port exists, through reception a payload message or CONN_MANAGER message from the peer within the last timer interval.
CONNECTED/PROBING: During the last timer expiration, it sent out a CONN_PROBE message to the peer, and now awaits the unconditional CONN_PROBE_REPLY message from the other end, or any data or CONN_PROBE message from the peer that can confirm the correct state of that port. See the detailed description of how this is handled later in this section.

The following EVENTS may occur to a port:

```
      CREATE:        Trivial
      PUBLISH:       Bind a port name sequence to a port.
      WITHDRAW:      Unbind the relation between a port name sequence
                     and a port.
      CONNECT:       Connect the port to another port.
      DISCONNECT:    Disconnect the port from the port it is connected
                     to.
      TIMEOUT:       Check if a sent CONN_PROBE was reponded to. Order
                     new timer.
      PROBE:         Receive a CONN_PROBE from peer.
      PROBE_REPLY:   Receive a CONN_PROBE_REPLY from peer.
      SEND_CONN:     Send a data message of type CONN_MSG.
      SEND_CONNLESS: Send a data message of type NAMED_MSG or
                     DIRECT_MSG.
      REC_CONN:      Receive data message of type CONN_MSG.
      REC_DIRECT:    Receive a DIRECT_MSG data message.
      REC_NAMED:     Receive a NAMED_MSG data message.
      REC_CONN_ERR:  Receive CONN_MSG data message with error code.
      REC_CLESS_ERR: Receive DIRECT_MSG or NAMED_MSG with error code.
      LOST_NODE:     Receive indication that contact with peer node
                     lost.
      DELETE:        Not so trivial.
```

A port may also take the following ACTIONS, depending on event:

```
      SEND_PRB:      Send a CONN_PROBE to peer.
      SEND_REPLY:    Send a CONN_PROBE_REPLY to peer.
      ABORT_REM:     Send one DATA_NON_REJECTABLE/CONN_MSG/
                     NO_REMOTE_PORT to peer.
      ABORT_SELF:    Send one DATA_NON_REJECTABLE/CONN_MSG to self,
                     with the appropriate error code, NO_REMOTE_NODE
                 or NO_REMOTE_PORT.
      DISCONNECT:    Disconnect.
      WITHDRAW:      Withdraw all publications pertaining to this port.
      REJ_CALL:      Reject user call with interface specific error
                     code.
      REJ_MSG:       Reject message with error code NO_REMOTE_PORT.
      DROP:          Drop message silently.
```

The state machine in Figure 14 only covers the normal events and state transitions in a port. The following table gives a more comprehensive picture. If there is no arrow "->" in a field it means that the port remains it its current state.

Event:	READY	BOUND	CONNECTED	
			CONFIRMED	PROBING
CREATE:	->!	-	-	-
PUBLISH:	->BOUND		REJ_CALL	
WITHDRAW:	REJ_CALL	->READY	REJ_CALL	
CONNECT:	->CONN/CONF	REJ_CALL	REJ_CALL	
DISCONNECT:	REJ_CALL	REJ_CALL	-> READY	
TIMEOUT:	-	-	SEND_PRB -> PROBING	ABORT_SELF -> READY
PROBE:	SEND_REPLY	ABORT	SEND_REPLY -> CONFIRMED	
PROBE_REPLY:	ABORT_REM	ABORT		->CONFIRMED
SEND_CONN:	REJ_CALL	REJ_CALL		
SEND_CONNLESS:		->READY	REJ_CALL	
REC_CONN:	->CONN/CONF	ABORT_REM		->CONFIRMED
REC_DIRECT:			REJ_MSG	
REC_NAMED:			REJ_MSG	
REC_CONN_ERR:	DROP	DROP	DISCONNECT -> READY	
LOST_NODE:	-	-	ABORT_SELF -> READY	
REC_CLESS_ERR:			DROP	
DELETE:	->0	WITHDRAW	ABORT_REM	

Figure 17: Complete port FSM

The reason for having a background probing of connections is explained in Section 3.5. The recommended timer interval for this probing is 3600 s, making it probable that the timer will never have to expire.

3.5 Connections

User Connections must be kept as lightweight as possible because of their potential huge number, and because it must be possible to establish and shut down thousands of connections per second on a node.

3.5.1 Connection Setup

How a connection is established and terminated is not defined by the protocol, only how they are supervised, and if necessary, aborted. Instead, this is left to the end user to define, or to the actual implementation of the user API-adapter. The following figures show two examples of this.

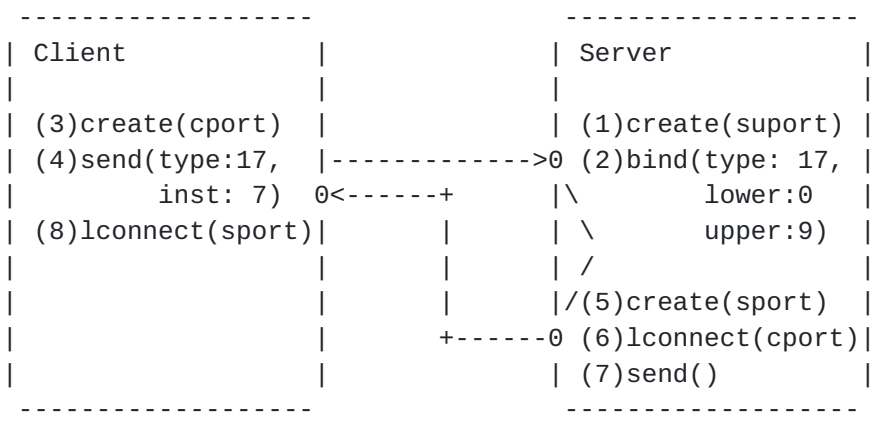

Figure 18: Example of user defined establishment of a connection

Figure 18 shows an example where the user himself defines how to set up the connection. In this case, the client starts with sending one payload- carrying NAMED_MSG message to the setup port (suport)(4). The setup server receives the message, and reads its contents and the client port (cport) identity. He then creates a new port (sport)(5), and connects it to the client port's identity(6). The lconnect() call is a purely node local operation in this case, and the connection is not fully established until the server has fulfilled the request and sent a response payload-carrying CONN_MSG message back to the client port(7). Upon reception of the response message the client reads the server port's identity and performs an lconnect() on it(8). This way, a connection has been established without sending a single protocol message.

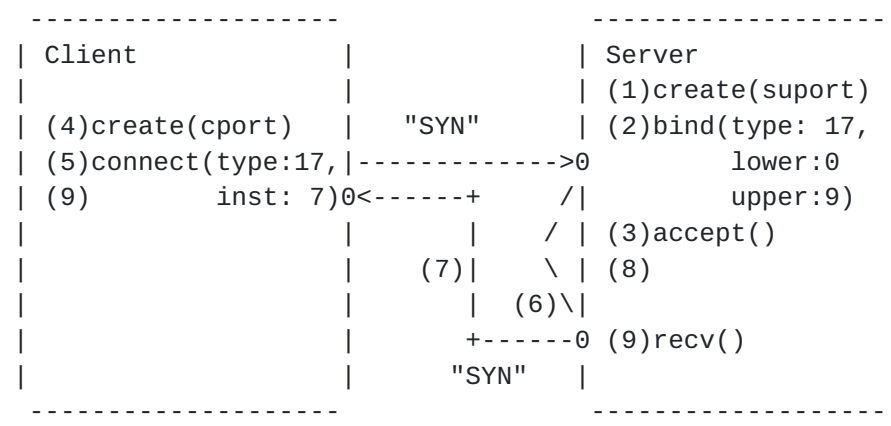

Figure 19: TCP-style connection setup

Figure 19 shows an example where the user API-adapter supports a TCP-style connection setup, using hidden protocol messages to fulfil the connection. The client starts with calling connect()(5), causing the API to send an empty NAMED_MSG message ("SYN" in TCP terminology) to the setup port. Upon reception, the API-adapter at the server side creates the server port, peforms a local lconnect()(6) on it towards the client port, and sends an empty CONN_MSG ("SYN") back to the client port (7). The accept() call in the server then returns, and the server can start waiting for messages (8). When the second SYN message arrives in the client, the API-adapter performs a node local lconnect() and lets the original connect() call return (9).

Note the difference between this protocol and the real TCP connection setup protocol. In our case there is no need for SYN_ACK messages, because the transport media between the client and the server (the node-to-node link) is reliable.

Also note from these examples that it is possible to retain full compatibility between these two very different ways of establishing a connection. Before the connection is established, a TCP-style client or server should interpret a payload message from a user-controlled counterpart as an implicit SYN, and perform an lconnect() before queueing the message for reading by the user. The other way around, a user-controlled client or server must perform an lconnect() when receiving the empty message from its TCP-style counterpart.

3.5.2 Connection Shutdown

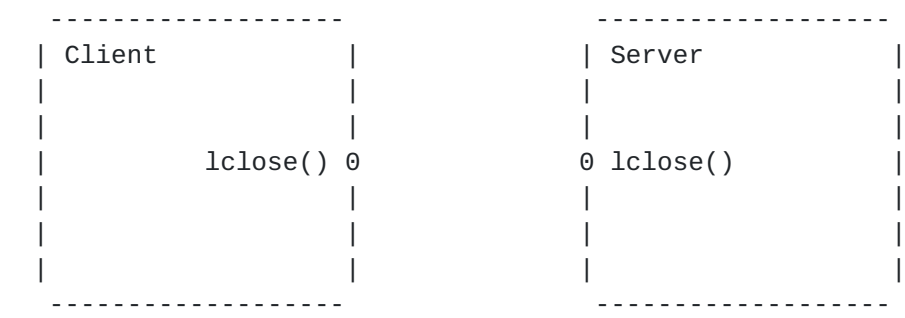

Figure 20: Example of user defined shutdown of a connection

Figure 20 shows the simplest possible user defined connection shutdown scheme. If it inherent in the user protocol when the connection should be closed, both parties will know the right moment to perform a "node local close" (lclose()) and no protocol messages need to be involved.

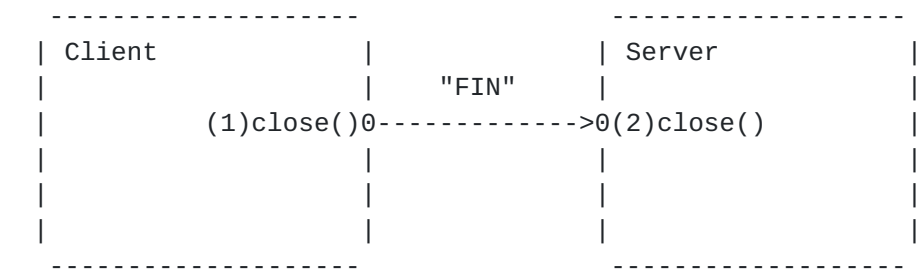

Figure 21: TCP-style connection shutdown

In Figure 21 a TCP-style connection close() is illustrated. This is simpler than the connection setup case, because the built-in connection abortion mechanism of TIPC can be used. When the client calls close() (1) TIPC must delete the client port. As will be described later, deleting a connected port has the effect that a DATA_NON_REJECTABLE/CONN_MSG ("FIN" in TCP terminology) with error code NO_REMOTE_PORT is sent to the other end. Reception of such a message means that TIPC at the receiving side must shut down the connection, and this must be done already before the server is notified. The server must then call close() (2), not to close the connection, but to delete the port. TIPC does not send any "FIN" this time, the server port is already disconnected, and the client port is anyway gone. If both endpoints call close() simultaneously, two "FIN" messages will cross each other, but at the reception they will have no effect, since there is no destination port, and they

must be discarded by TIPC.

Note even here the automatic compatibility with a user-defined peer and a TCP-style ditto: Any user, no matter the user API, must at any moment be ready to receive a "connection aborted" indication, and this is what in reality happens here.

3.5.3 Connection Abortion

When a connected port receives an indication from the TIPC link layer that it has lost contact with its peer node, it must immediately disconnect itself and send an empty CONN_MSG/NO_REMOTE_NODE to its owner process.

When a connected port is deleted without a preceding disconnect() call from the user it must immediately disconnect itself and send an empty CONN_MSG/NO_REMOTE_PORT to its peer port. This may happen when the owner process crashes, and the OS is reclaiming its resources.

When a connected port receives a timeout call, and is still in CONNECTED/PROBING state since the previous timer expiration,it must immediately disconnect itself and send an empty CONN_MSG/NO_REMOTE_PORT to its owner process.

When a connected port receives a rejected previously sent message, (a CONN_MSG with error code), it must immediately disconnect itself and deliver the message, with data contents, to its owner process.

When a port participating in a multi-hop connection receives a CONN_MSG where the connection level sequence number does not fit, it must immediately disconnect itself, send an empty CONN_MSG/COMM_ERROR to its owner process, and another empty CONN_MSG/COMM_ERROR to its peer port.

When a connected port receives a CONN_MSG from somebody else than its peer port, it must immediately send an empty CONN_MSG/NO_CONNECTION to the originating port.

When TIPC in a node receives a CONN_MSG/TIPC_OK for which it finds no destination port, it must immediately send an empty CONN_MSG/NO_REMOTE_PORT back to the originating port.

When a bound port receives a CONN_MSG from anybody,it must immediately send an empty CONN_MSG/NO_CONNECTION to the originating port.

3.5.4 Connection Supervision

In almost all practical cases the mechanisms for resource cleanup after process failure, rejection of messages when no destination port is found, and supervision of peer nodes at the link level, is sufficient for immediate failure detection and abortion of connections.

However, because of the non-specified connection setup procedure of TIPC, there exists cases when a connection may remain incomplete. This may happen if the client in a user-defined setup/shutdown scheme forgets to call lconnect() (see Figure 20), and then deletes itself, or if one of the parties fails to call lclose() (see Figure 21). These cases are considered very rare, and should normally have no serious consequenses for the availability of the system, so a slow background timer is judged sufficient to discover such situations.

When a connection is established each port starts a timer, whose purpose is to check the status of the connection. It does this by regularly (typical configured interval is once an hour) sending a CONN_PROBE message to the peer port of the connection. The probe has two tasks; first, to inform that the sender is still alive and connected; second, to inquire about the state of the recipient.

A CONN_PROBE or a CONN_PROBE_REPLY reply MUST be immediately responded to according to the following scheme:

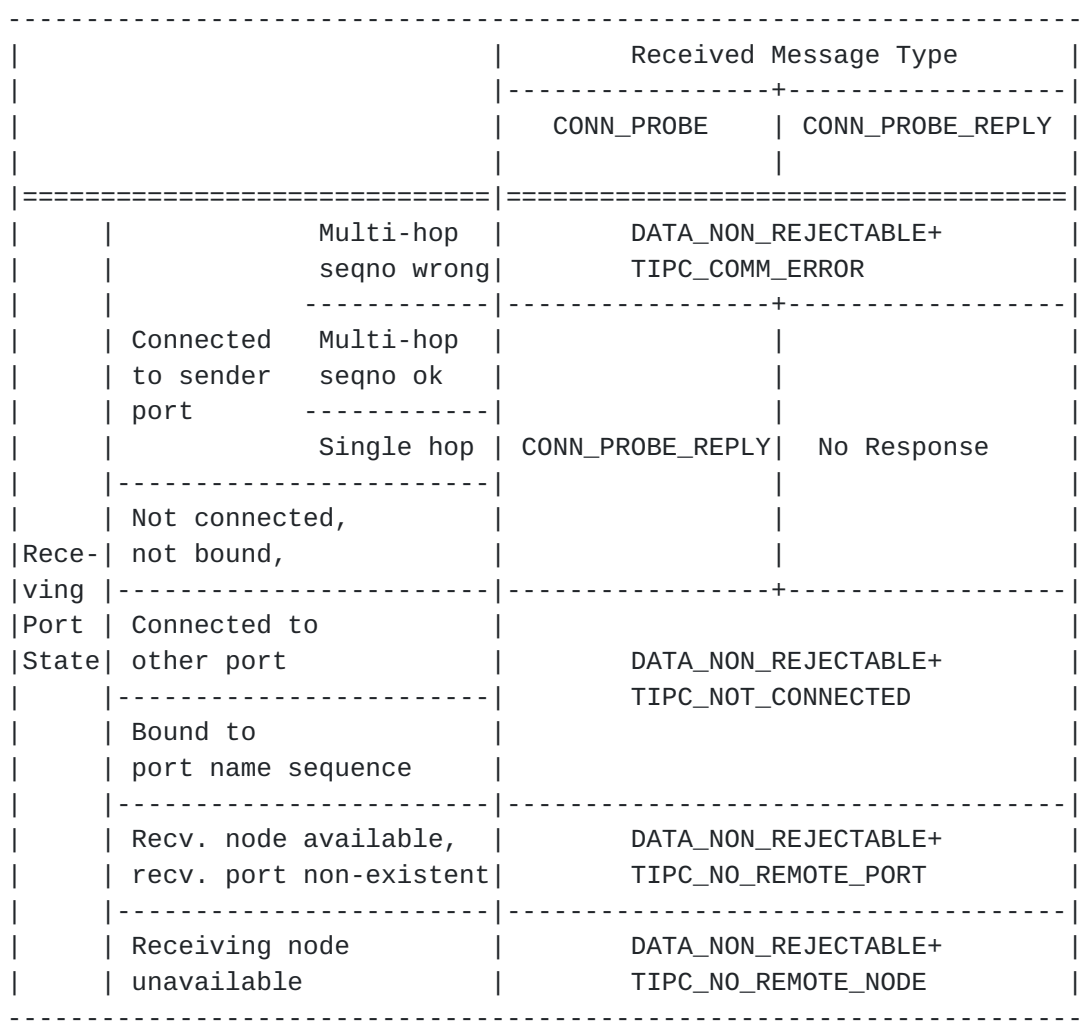

Figure 22: Response to probe/probe replies vs port state.

If everything is well then the receiving port will answer with a probe reply message, and the probing port will go to rest for another interval. It is inherent in the protocol that one of the ports - the one connected last - normally will remain passive in this relationship. Each time its timer expires it will find that it has just received and replied to a probe, so it will never have any reason to explicitly send a probe itself.

When an error is encountered, one or two empty CONN_MSG data are generated, one to indicate a connection abortion in the receiving port, if it exists, and one to do the same thing in the sending port.

The state machine for a port during this message exchange is described in Section 3.5.

3.5.5 Flow Control

The mechanism for end-to-end flow control at the connection level has as its primary purpose to stop a sending process from overrunning a slower receiving process. Other tasks, such as bearer, link, network, and node congestion control, are handled by other mechanisms in TIPC. Because of this, the algorithm can be kept very simple. It works as follows:

1. The message sender (the API-adapter) keeps one counter, SENT_CNT, to count messsages he has sent, but which has not yet been acnkowledged. The counter is incremented for each sent message.
2. The receiver counts the number of messages he delivers to the user, ignoring any messages pending in the process in-queue. For each N message, he sends back a CONN_MANAGER/ACK_MSG containing this number in its data part.
3. When the sender receives the acknowledge message, he subtracts N from SENT_CNT, and stores the new value.
4. When the sender wants to send a new message he must first check the value of SENT_CNT, and if this exceeds a certain limit, he must abstain from sending the message. A typical measure to take when this happens is to block the sending process until SENT_CNT is under the limit again, but this will be API-dependent.

The recommended value for the send window N is at least 200 messages, and the limit for SENT should be at least 2*N.

3.5.6 Sequentiality Check

Inter-cluster connection-based messages, and intra-cluster messages between cluster nodes and secondary nodes, may need to be routed via intermediate nodes if there is no direct link between the two. This implies a small, but not negligeable risk that messages may be lost or re-ordered. E.g. an intermediate node may crash, or it may have changed its routing table in the interval between the messages. A connection level sequence number is used to detect such problems, and this must be checked for each message received on the connection. If the sequence number does not fit in sequence, no attempts of re-sequencing should be done. The port discovering the sequence error must immediately abort the connection by sending one empty CONN_MSG/COMM_ERROR message to itself, and one to the peer port.

The sequence number must not be checked on single-hop connections, where the link protocol guarantees that no such errors can occur.

The first message sent on a connection has the sequence number 42.

3.6 Neighbour Detection

3.6.1 Link Requests

At startup, or when otherwise told to, TIPC will send out Link Request messages to its neighbouring nodes, informing about its existence, and requesting to have a set of links set up from the destination domain towards itself. The structure of Link Configuration messages is described in Section 4.3.9.

A node receiving a Link Request, first checks whether it belongs to the destination domain stated in the message, and if the Network Identity of the message is equal to its own. If that is not the case, it ignores the message.

Otherwise, depending on the destination domain and the sender node address, message, the node goes through one of the following steps:

- If the destination domain is exactly the own node, and if a link does not already exist, it creates a link to the sender node. A response is sent back to the sender node if the Response Expected field is set.
- Otherwise, if the sender node belongs to the own cluster, and if a link does not already exist, it creates a link to the sender node. A response is sent back to the sender node if the Response Expected field is set.
- Otherwise, if the destination domain comprises, but is larger than the own node, and the sender node belongs to a remote cluster, the node initiates the Inter Cluster Link Setup algorithm.

3.6.2 Inter-Cluster Link Setup

3.6.2.1 Link Entropy

The inter-cluster link setup algorithm has the goal of setting up a specified number of links from each node in one cluster, to a correponding number of different nodes in another cluster. Dual links between a node pair are not permitted. The algorithm takes all measures necessary to ensure that exactly the requested number of links are created and maintained at any time; nothing less, nothing more. We call this the Link Entropy Check Algorithm.

The algorithm also ensures that all created links are distributed smoothly over the two clusters. The following applies:

- If two clusters are of equal size, each of the nodes in the two clusters will have exactly the number of links specified.
- If two clusters are of different size, all nodes in the bigger cluster will have exactly the specified number of links. Some nodes in the smaller cluster will have more links than specified,

to fulfil the requirement for the bigger one. These extra links will be smoothly distributed, so that no node in the smaller cluster will have more than one link more than any of the others.
- If a node is added to the smaller cluster, some of the existing extra links will be moved to the new node, to keep the optimal distribution.
- If a node is added to a bigger or equal-sized cluster, its new links will be established to nodes in the other clusters having the fewer links.
- If a node disappears or is removed from a cluster, the connected nodes in the other re-establish links to some of the remaining nodes, in such a way that smooth distribution is maintained, and the nodes regains the specified number of links.
- Whenever a new link is established beyond the specified link number for a node, it checks link entropy by sending a CHECK_LINK_COUNT message to all its peer nodes. If any of the receiving nodes finds it has more inter-cluster links than its specified number, it knows that the link to the message sender is redundant, and terminates it.

The recommended specified number of inter-cluster links per-node is two.

3.6.2.2 Initiation of Setup

The first inter cluster contact may be established in two ways:
- A node is ordered through the management interface to send a Link Request to a specific node in the other cluster, hence creating a Pilot Link. This link request must have the Response Expected field set to non-zero.
- Both clusters are within a bearer multicast/broadcast domain, e.g. the same LAN, and the neigbour detection broadcasts are configured to be accepted by foreign clusters, i.e. destination domain is <Z.0.0> or <0.0.0>. These link requests must have the Response Expected field set to non-zero. Possible redundant links created this way will be removed later through the link entropy check.

Thereafter the following sequence of events follows:
- When a first inter-cluster link comes up, all the other nodes in both clusters will immediately become aware of it. This is because of the distribution of ROUTE_ADDITION (see Section 3.8) messages from the establishing nodes.
- Any node in a cluster becoming aware of the existence of a new cluster, immediately sends a GET_NODE_INFO message to the router node, in order to obtain the bearer level address (IP- or Ethernet) needed to reach the remote node.
- When the bearer address is obtained, the nodes send a Link Request message to the identified remote node. This Link Request must be

"open", i.e. both the node part of the destination domain field and the contents of the Response Expected field must be zero. When the romote node receive the request, it does not immediately establish a link, but follows a procedure described in the following section.

- Until the first own inter-cluster link is established, each node repeatedly, but using the same exponential backoff algorithm as for broadcasted Link Requests (see description later), send out new Link Requests to the other cluster. Duplicates of such requests are detected and dropped, as will be described later.

As can be seen from this description, and from the following sections, the scenario for setting up inter-cluster links is extremely chaotic in the beginning, with all nodes in both clusters simultaneously trying to set up links to the opposite cluster. The way Link Requests and link establishments are handled, still ensures that this phase will eventually settle down to a link pattern with the optimal number and distribution of links, and remain that way as long as the two clusters are in contact.

3.6.2.3 Handling of Inter Cluster Link Requests

When a node receives a Link Request from a remote cluster, and the request contains a domain larger than the node itself, it initiates the following sequence of events.

- The node creates a Link Probe message, whose task it is to roam around in the cluster to find the most suitable node to establish a link back to the original remote node. The structure of Link Probe Messages is described in Section 4.
- If there is no previous contact with the remote cluster, i.e. the node's routing table shows that there are no other nodes having links to the sender's cluster, and the Response Expected field of the request is non-zero,the node creates a link, called the Pilot Link, and sends a response back to the originating node. While the link activation is ongoing, i.e. until the pilot link is up in WORKING_WORKING state, or is found to have failed, the link probe is parked at the receiving node, and all subsequent link requests from the originating cluster are ignored.
- If there is previous contact, or when the pilot link comes up, the link probe is sent on a tour in the cluster, using the Sequence Tag to determine each next hop. At each node on the tour it counts the number of links back to the originating cluster, and stores that value if it is lower than the Lowest Link Count This Tour field in the probe. If at any node it discovers a working link back to the requesting node, it decrements the Requested Links field with one and continues to the next node in the sequence. If the Requested Links field reaches zero, or a parked probe from the same remote node is found, the link probe is

dropped as a duplicate, and no more action is taken on behalf of the original link request.

- o After a tour, i.e. when the probe is back at the node receiving the original link request, the value of Lowest Link Count This Tour field is stored in the Lowest Link Count field, and the probe is sent out on a new tour, comprising the first node.
- o When the probe encounters a node having the same number of links to the remote cluster as Lowest Link Count, and where there is no previous link towards the requesting node, the probe is parked on that node, a link endpoint is created, and a "reverse" Link Request message is sent directly back to the requesting node. In this request, the destination domain field must be fully specified, and the Response Expected field set to zero. No link endpoint is created yet.
- o When the requesting node in the other cluster receives the reverse link request, there are four possible responses:
  1. It finds it already has enough (i.e. the requested number of) links to the responding cluster. It sends back a DROP_LINK_REQUEST message to the responding node, instructing it to delete its link endpoint and the parked link probe.
  2. It may find it already has a link, working or in progress, to the responding node. If this race condition is true, it returns a LINK_REQUEST_REJECTED message to that node. This forces the responding node to to pass the parked link probe to the next node in the cluster sequence.
  3. It may find that itself has a parked link probe, trying to establish a link to the responder. If this race condition is true, it must decide which of the mutual setup attempts should be aborted. Hence, if the numerical value of the own node address is lower than that of the responding node, it returns a LINK_REQUEST_REJECTED message to that node. This forces the reponding node to pass the parked link probe to the next node in the cluster sequence.
  4. None of the previous conditions are true. It returns a LINK_REQUEST_ACCEPTED message to the reponding node. That node creates its link endpoint, decrements the Requested Links counter of the parked link probe, and if it is still non-zero, it passes it on to the next node in the cluster.

Note that the three new message types introduced here are sent as ordinary TIPC messages, using an ordinary TIPC port. This can be done because there is already at least a pilot link between the two clusters. The address used is the port name <1,msg_type>, using the responder node as lookup domain.

When the probe has established all the requested links, or after a maximum of 10 complete cluster tours, it is dropped.

3.6.3 Multicast Link Setup

When the bearer media makes it possible, TIPC uses a special auto-configuration protocol for neighbour detection and link creation. This is the case e.g. with Ethernet and DCCP, where multicast transmission of packets is possible. The protocol for this is fairly simple: Immediately after start, or when it detects that a new interface has become active, a node starts to repeatedly broadcast Link Request messages to other presumed members of the network.

The Destination Domain field in these messages should be configurable when TIPC is started, but is typically set to <own_zone.own_cluster.0>.

The broadcast interval has a start value of 125 msec, and is multiplied by a factor 4 at each transmission, until it reaches a configurable upper limit, default set to 32000 msec, at which point it stops increasing. The broadcasts continue at this rate as long as the node is up. A node receiving a Link Request, checks whether it belongs to the destination domain stated in the message, and if the Network Identity of the message is equal to its own. If that is the case, if a link does not already exist, and the Response Expected field is non-zero, it creates its end of the link. Thereafter, it answers with a unicast Link Request back to the requesting node. This will in its turn create the other end of the link, if there is not one already, and the next phase, the link activation phase, begins.

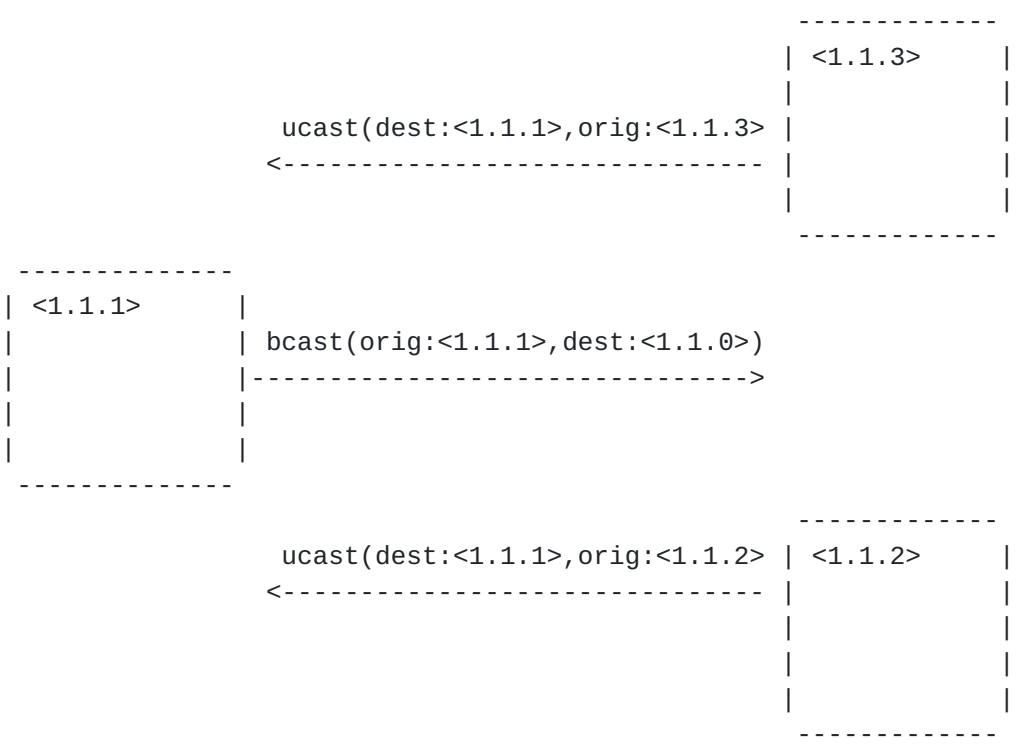

Figure 23: Neighbour Detection

There are two reasons for the continuous broadcasting decribed above. First, it should always be possible for two nodes to discover each other, even if the communication media between them is non-functional at the start moment. E.g. in a dual-switch system, one of the node cables may have been faulty or disconnected from the beginning, while the cluster is still fully connected and functional via the other switch. In such cases one should not be forced to restart one of the nodes to set up the links, and any more manual intervention than plugging in a working cable should be unnecessary. Second, it should be possible to replace (hot-swap) an interface card with one having a different MAC address, still without having to restart the node. When a node receives a Link Request its originating MAC address is always checked against the one previously stored for that destination, and if they differ the old one is replaced. This way, a replaced interface board will be detected and taken into traffic within 32 seconds of the replacement.

The structure and semantics of Link Request messages is described in Section 4.3.9.

3.7 Links

3.7.1 Link Activation

Link activation and supervision is completely handled by the generic part of the protocol, in contrast to the partially media-dependent neighbour detection protocol.

The following FSM describes how a link is activated and supervised.

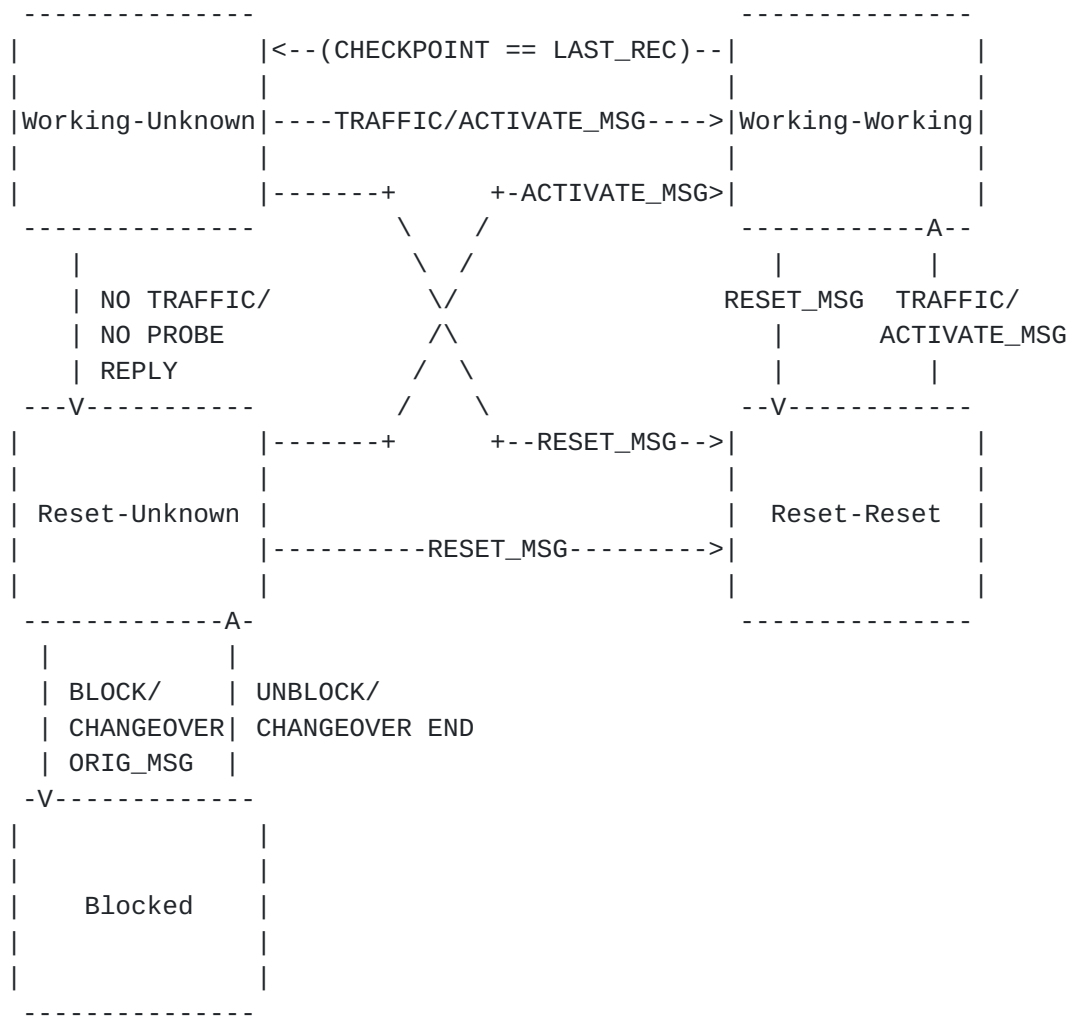

Figure 24: Link finite state machine

A link enpoint's state is defined by the own endpoint's state, combined with what is known about the other endpoint's state. The following states exist:

Reset-Unknown

Own link endpoint reset, i.e. queues are emptied and sequence numbers are set back to their initial values. The state of the peer endpoint is unknown. LINK_PROTOCOL/RESET_MSG messages are sent periodically at CONTINUITY_INTERVAL to inform peer about the own endpoint's state, and to force it to reset its own enpoint,if this has not already been done. If the peer endpoint is rebooting, or has reset for some other reason, it will sooner or later also reach the state Reset-Unknown, and start sending its own RESET_MSG messages periodically. At least one of the endpoints, and often both, will eventually receive a RESET_MSG and transfer to state Reset-Reset. If the peer is still active, i.e. in one of the states Working-Working or Working-Unknown, and has not yet detected the disturbance causing this endpoint to reset, it will sooner or later receive a RESET_MSG, and transfer directly to state Reset-Reset. If a LINK_PROTOCOL/ ACTIVATE_MSG message is received in this state, the link endpoint knows that the peer is already in state Reset-Reset, and can itself move directly on to state Working-Working. Any other messages are ignored in this state. CONTINUITY_INTERVAL is calculated as the smallest value of LINK_TOLERANCE/4 and 0.5 sec.

Reset-Reset

Own link endpoint reset, peer endpoint known to be reset, since the only way to reach this state is through receiving a RESET_MSG from peer. The link endpoint is periodically at CONTINUITY_INTERVAL sending ACTIVATE_MSG messages. This will will eventually cause peer to transfer to state Working-Working. The own endpoint will also transfer to state Working-Working as soon as any message which is not a RESET_MSG is received.

Working-Working

Own link endpoint working. Peer link endpoint known to be working, i.e. both can send and receive traffic messages. A periodic timer with the interval CONTINUITY_INTERVAL checks if anything has been received from the peer during the last interval. If not,state transfers to state Working-Unknown.

Working-Unknown

Own link endpoint working. Peer link endpoint in unknown state. LINK_PROTOCOL/STATE_MSG messages with the PROBE bit set are sent at an interval of CONTINUITY_INTERVAL/4 to force a response from peer. If a calculated number of probes (LINK_TOLERANCE/(CONTINUITY_INTERVAL/4) remain unresponded, state transfers to Reset-Unknown. Own link endpoint is reset, and the link is considered lost. If, instead, any kind of message, except LINK_PROTOCOL/RESET_MSG and LINK_PROTOCOL/ACTIVATE_MSG is

received, state transfers back to Working-Working. Reception of a RESET_MSG in this situation brings the link to state Reset-Reset. ACTIVATE_MSG will never received in this state.

Blocked

The link endpoint is blocked from accepting any packets in either direction, except incoming, tunneled CHANGEOVER_PROTOCOL/ORIG_MSG. This state is entered upon the arrival of the first such message, and left when the last has been counted in and delivered. See description about the changeover procedure later in this section. The Blocked state may also be entered and left through the management commands BLOCK and UNBLOCK. This is also described later.

A newly created link endpoint starts from the state Reset-Unknown. The recommended default value for LINK_TOLERANCE is 0.8 sec.

3.7.2 Link Continuity Check

During normal traffic both link enpoints are in state Working-Working. At each expiration point, the background timer checkpoints the value of the Last Received Sequence Number. Before doing this, it compares the check- point from the previous expiration with the current value of Last Received Sequence Number, and if they differ, it takes the new checkpoint and goes back to sleep. If the two values don't differ, it means that nothing was received during the last interval, and the link endpoint must start probing, i.e. move to state Working-Unknown.

Note here that even LINK_PROTOCOL messages are counted as received traffic, altough they don't contain valid sequence numbers. When a LINK_PROTOCOL message is received, the checkpoint value is moved,instead of Last Received Sequence Number, and hence the next comparison gives the desired result.

3.7.3 Sequence Control and Retransmission

Each packet eligible to be sent on a link is assigned a Link Level Sequence Number, and appended to a send queue associated with the link endpoint. At the moment the packet is sent, its field Link Level Acknowledge Number is set to the value of Last Received Sequence Number.

When a packet is received in a link endpoint, its send queue is scanned, and all packets with a sequence number lower than the arriving packet's acknowledge number (modulo 2^16-1) are released.

If the packet's sequence number is equal to Last Received Sequence

Number + 1 (mod 2^16-1), the counter is updated, and the packet is delivered upwards in the stack. A counter, Non Acknowledged Packets, is incremented for each message received, and if it reaches the value 10, a LINK_PROTOCOL/STATE_MSG is sent back to the sender. For any message sent, except BCAST_PROTOCOL messages, the Non Acknowledged Packets counter is set to zero.

Otherwise, if the sequence number is lower, the packet is considered a duplicate, and is silently discarded.

Otherwise,if a gap is found in the sequence, the packet is sorted into the Deferred Incoming Packets Queue associated to the link endpoint, to be re-sequenced and delivered upwards when the missing packets arrive. If that queue is empty,the gap is calculated and immediately transferred in a LINK_PROTOCOL/STATE_MSG back to the sending node. That node must immediately retransmit the missing packets. Also, for each 8 subsequent received out-of-sequence packets, such a message must be sent.

3.7.4 Message Bundling

Sometimes a packet can not be sent immediately over a bearer, due to network or recipient congestion (link level send window overflow), or due to bearer congestion. In such situations it is important to utilize the network and bearer as efficiently as possible, and not stop important users from sending messages before this is absolutely unavoidable. To achieve this, messages which can not be transmitted immediately are bundled into already waiting, packets whenever possible, i.e. when there are unsent packets in the send queue of a link. When the packet finally arrives at the receiving node it is split up to its individual messages again. Since the bundling layer is located below the fragmentation layer in the functional model of the stack, even message fragments may be bundled with other messages this way, but this can only happen to the last fragment of a message, the only one normally not filling an entire packet by itself.

It must be emphasized that message transmissions never are delayed in order to obtain this effect. In contrast to TCP's Nagle Algorithm, the only goal of the TIPC bundling mechanism is to overcome congestion situations as quickly and efficiently as possible.

3.7.5 Message Fragmentation

When a message is longer than the identified MTU of the link it will use, it is split up in fragments, each being sent in separate packets to the destination node. Each fragment is wrapped into a packet headed by an TIPC internal header (see Section 4.2). The User field of the header is set to MSG_FRAGMENTER, and each fragment is assigned

a Fragment Number relative to the first fragment of the message. Each fragmented message is also assigned a Fragmented Message Number, to be present in all fragments. Fragmented Message Number must be a sequence number with the period of 2^16-1. At reception the fragments are reassembled so that the original message is recreated, and then delivered upwards to the destination port.

3.7.6 Link Congestion Control

TIPC uses a common sliding window protocol to handle traffic flow at the signalling link level. When the send queue associated to each link endpoint reaches a configurable limit, the Send Window Limit, TIPC stop sending messages over that link. Packets may still be appended to or bundled into waiting packets in the queue, but only after having been subject to a filtering function, selecting or rejecting user calls according to the sent message's importance priority. DATA_LOW messages are not accepted at all in this situation. DATA_NORMAL messages are still accepted, up to a configurable limit set for that user. All other users also have their individually configurable limits, recommended to be assigned values in the following ascending order: DATA_LOW, DATA_NORMAL, DATA_HIGH, DATA_NONREJECTABLE, CONNECTION_MANAGER,BCAST_PROTOCOL, ROUTE_DISTRIBUTOR, NAME_DISTRIBUTOR, MSG_FRAGMENTER. MSG_BUNDLER messages are not filtered this way, since those are packets created at a later stage. Whether to accept a message due for fragmentation or not is decided on its original importance, set before the fragmentation is done. Once such a message has been accepted, its individal fragments must be handled as being more important than the original message.

When the part of the queue containing sent packets again is under the Send Window Limit, the waiting packets must immediately be sent, but only until the Send Window Limit is reached again.

3.7.7 Bearer Congestion Control

When the local bearer media becomes overloaded, e.g. when an Ethernet circuit runs out of send buffers, the Bearer Congestion Control function may be activated. This function keeps track of the current state of the bearer, and stops accepting any packet send calls until the bearer is ready for it again. During this interval TIPC users may still perform send calls, and packets will be accumulated in the affected links send queues according to the same rules as for Link Congestion Control, but all actual transmission is stopped.

When the congestion has abated, the bearer opens up for traffic again, and the links having packets waiting to be sent are scheduled

round-robin for sending their unsent packets. This level of congestion control is optional, and its activation should be configurable.

3.7.8 Link Load Sharing vs Active/Standby

When a link is created it is assigned a Link Priority, used to determine its relation to a possible parallel link to the same node. There are two possible relations between parallel working links.

Load Sharing

Load Sharing is used when the links have the same priority value.Payload traffic is shared equally over the two links, in order to take full advantage of available bandwidth. The selection of which link to use must be done in a deterministic way, so that message sequentiality can be preserved for each individual sender port. To obtain this a Link Selector is used. This must be value correlated to the sender in such a way that all messages from that sender choose the same link, while guaranteeing a statistically equal possibility for both links to be selected for the overall traffic between the nodes. A simple example of a link selector with the right properties is the last two bits of the random number part of the originating port's identity, another is the same bits in Fragmented Message Number in message fragments.

Active/Standby

When the priority of one link has a higher numeral value than that of the other, all traffic will go through that link, denoted the Active Link. The other link is kept up and working with the help of the continuity timer and probe messages, and is called the Standby Link. The task of this link is to take over traffic in case the active link fails.

Link Priority has a value within the range [1,31]. When a link is created it inherits a default priority from its corresponding bearer, and this should normally not need to be changed thereafter. However, Link Priority must be reconfigurable in run-time.

3.7.9 Link Changeover

When the link configuration between two nodes changes, the moving of traffic from one link to another must be performed in such a way that message sequentiality and cardinality per sender is preserved. The following situations may occur:

Active Link Failure

Before opening the remaining link for messages with the failing link's selector, all packets in the failing link's send queue must wrapped into messages (tunneling messages) to be sent over the remaining link, irrespective of whether this is a load sharing active link or a standby link. These messages are headed by a TIPC Internal Header, the User field set to CHANGEOVER_PROTOCOL, Message Type set to ORIG_MSG. On the tunneling link the messages are subject to congestion control, fragmentation and bundling, like any other messages. Upon arrival in the arriving node, the tunneled packets are unwrapped, and moved over to the failing links receiving endpoint. This link endpoint must now be reset, if it has not already been done, and itself initiate tunneling of its own queued packets in the opposite direction. The unwrapped packets' original sequence numbers are compared to Last Received Sequence Number of the failed links receiving endpoint, and are delivered upwards or dropped according to their relation to this value. There is no need for the failing link to consider packet sequentiality or possible losses in this case, - the tunneling link must be considered a reliable media guaranteeing all the necessary properties. The header of the first ORIG_MSG sent in each direction must contain a valid number in the Message Count field, in order to let the receiver know how many packets to expect. During the whole changeover procedure both link endpoints must be blocked for any normal message reception, to avoid that the link is inadvertently activated again before the changeover is finished. When the expected number of packets has been received, the link endpoint is deblocked, and can go back to the normal activation procedure.

Standby Link Failure

This case is trivial, as there is no traffic to redirect.

Second Link With Same Priority Comes Up

When a link is active, and a second link with the same priority comes up, half of the traffic from the first link must be taken over by the new link. Before opening the new link for new user messages, the packets in the existing link's send queue with a link selector corresponding to the new link must be transmitted over that link. This is done by wrapping copies of these packets into messages (tunnel messages) to be sent over the new link. The tunnel messages are headed by a TIPC Internal Header, the User field set to CHANGEOVER_PROTOCOL, Message Type set to DUPLICATE_MSG. On the tunneling link the messages are subject to congestion control, fragmentation and bundling, just like any other messages. Upon arrival in the arriving node, the tunneled packets are unwrapped, and delivered to the original links

receiving endpoint, just like any other packet arriving over that link's own bearer. If the original packet has already arrived over that bearer, the tunneled packet is dropped as a duplicate, otherwise the tunneled packet will be accepted, and the original packet dropped as a duplicate when it arrives.

Second Link With Higher Priority Comes Up

When a link is active, and a second link with a higher numerical priority comes up, all traffic from the first link must be taken over by the new link. The handling of this case is identical to the case when a link with same priority comes up. After the traffic takeover has finished, no more senders will select the old link, but this does not affect the takeover procedure.

3.8 Routing

TIPC support routing of packets when necessary, both between clusters, between zones, and between system nodes and secondary nodes.

3.8.1 Routing Algorithm

Available routes to a remote zone, cluster or secondary node are explored in the following order:

First, look for any direct links to the destination node, and if there are any, select one using the normal link selection algorithm.

Second, if no direct link is found, and if the message is an inter-cluster message, look for a direct link to any node within the remote zone/cluster, and send the message there. This selection must be performed in a deterministic way, to minimize the risk for disordered message arrivals. If the destination or sender of the message is a secondary node, this step of the lookup is omitted.

Third, if there are no such direct links,the algorithm must look for a node within the own cluster, known to have a direct link to the destination node. Selection of this intermediate node must also be done in a deterministic way, e.g. using the Link Selector. If the destination or origin of the message is a secondary node, and if no router node is found, the message must be rejected with error code TIPC_NO_REMOTE_NODE.

As last resort, if all previous attempts have failed, the algorithm will look for any cluster local node known to have a link to any node within the remote zone/cluster. If no such router node is found, the message must be rejected with error code TIPC_NO_REMOTE_NODE.

3.8.2 Routing Table

Each node in a cluster must be kept up-to-date about all available routes to secondary nodes, external clusters, and external zones. This is done by letting each node broadcast any change in its direct connectivity to all the other nodes in the cluster, the same way the nameing table is kept updated. Because of the multiplicative effect, the number of potential routes between two big zones may be very large, and the routing table structure must reflect this. As an extreme scenario, take a zone with 1000 nodes, divided into five clusters with 200 nodes each, each node having two links to two different nodes in the foreign clusters, and dual links to all nodes within the local cluster. Each node would have to store knowledge about 999 external nodes, 800 of those representing nodes in the remote clusters. For each of these 800 nodes, information about 2 routes must be stored, apart from the four direct inter-cluster links.

To continue this example, we see that each node would have 199 x 2 + 5 x 2 = 308 links to maintain. With a continuity timer for each link expiring e.g. every 400 msec, corresponding to a link tolerance of 1.6 sec, there would be 600 expiring timers per second on each node. Again, assuming a a worst-case scenario with idle links, 1200 probe messages would have to be sent and received per second per node. To handle a kernel timer and send a probe message takes less than 5 usecs of CPU-time on a single 2 Ghz CPU, and to receive and handle a probe at kernel level takes about the same time. Hence, the background load for maintaining all necessary links even within such a huge zone would not exceed 2.5%.

3.8.3 Routing Table Updates

There are five different cases when a node's routing table must be updated:

A link towards a zone/cluster external node comes up:

- Broadcast a ROUTE_ADDITION message to all system nodes within the own cluster, informing them that the new destination can be reached via this node.

A link towards a secondary node comes up:

- Broadcast a ROUTE_ADDITION message to all system nodes within the own cluster, informing that the new destination can be reached via this node.
- Send a LOCAL_ROUTING_TABLE message to the secondary node,informing about existence of all system nodes within cluster.

A new cluster local system node becomes available:

- Send a SEC_ROUTING_TABLE message to the new node, containing information about all cluster secondary nodes which can be reached via this node.
- Send an EXT_ROUTING_TABLE message to the new node, containing information about all cluster external nodes which can be reached via this node.
- Send a ROUTE_ADDITION message to all directly connected secondary nodes, informing about the existence of the new node.

The direct contact with a zone/cluster external node or secondary node is lost:

A cluster local system node becomes unavailable:
- Remove all references to this node from the local routing tables. This is a completely node local operation.
- Send ROUTE_REMOVAL messages to all directly connected secondary nodes, informing them about the loss of the node.

The specific message structure used in these situations is described in Section 4.

3.9 Multicast Transport

To effectively support the functional multicast service described in a previous section, a reliable cluster broadcast service is provided by TIPC.

Although seen as a broadcast service from a TIPC viewpoint, at the bearer level this service is typically implemented as a multicast group comprising all nodes in the cluster.

At the multicast/broadcast sending node a sequence of actions is followed:
- When a functional multicast is requested, TIPC first looks up all matching destinations in its name translation table.
- The destination addresses are filtered down to a list containing the network addresses of and the exact number of destination nodes in the indicated lookup domain.
- If the own node is on the list, a replica is sent to the functional multicast receive function in the own node.
- If the lookup domain indicated goes beyond the own cluster, a replica of the message is sent to the identified external zones or clusters. There the message is subject to a new multicast lookup and cluster broadcast, if necessary.

3.9.1 Conditional Cluster Broadcast

If the lookup domain comprises the node's own cluster, and if the

number of identified target nodes in the cluster is less than the configurable parameter Broadcast Limit(recommended default value: 8), or if no cluster broadcast is supported, a replica of the multicast message is sent via a link to each of these nodes. Since a link can be considered a reliable media, performing fragmentation and retransmission if necessary, we can trust that the replicas will reach their destinations, and no more action need to be taken.

Otherwise, if the number of destination nodes exceeds Broadcast Limit, and a cluster broadcast service is available, the message is sent as one or more broadcast packets to all nodes in the cluster. If necessary the message is fragmented into packets to fit into the MTU of the media. Each packet is assigned a broadcast sequence number and added to a broadcast transmission/retransmission queue before being sent. If there is no congestion in the bearer, or the transmission queue has not exceeded the Broadcast Window Limit, the packet or packets are sent immediately. If there is any congestion, the packets are queued according to their importance, bundled into waiting buffers if possible. Broadcast Window Limit should be configurable, with a recommended default value of 48. All this is analogous to how the node-to-node link protocol works.

If not already running, a background timer is started, to expire at a Broadcast Continuity Interval. Broadcast Continuity Interval is recommended to be 16 * RTT, or, since most OS:es don't support timers of that resolution, as fast as the OS can support. As long as there are non-acknowledged packets in the broadcast out-queue, the timer continues to run, but no more timers are started.

At next expiration the timer checks the acknowledge status for each destination node, and releases all buffers which have been acknowledged by all nodes in the cluster. In order to keep the send-queue as short as possible at any time, this step is also performed at each attempted sent broadcast, at each received multicast, and at each received BCAST_PROTOCOL/STATE_MSG.

3.9.2 Conditional Tunneled Retransmission

For the still unacknowledged packets sent before the previous timer expiration, and for packets older than 16 positions from the last sent packet in the send queue, the timer function calculates how to do the retransmission. If the number of missing nodes in the acknowledge list for a message is less than Broadcast Limit, it sends a replica of that packet via a link to each of the missing nodes. Because the packet must be recognized as a missing broadcast at the receiving node, it is tunneled over the link, i.e. wrapped into a packet of type BCAST_PROTOCOL/BCAST_MSG. Since a link can be trusted as a reliable media, the original packet is now discarded. This step

is also performed at each received BCAST_PROTOCOL/STATE_MSG containing a non-zero sequence gap field.

Otherwise, If the number of missing acknowledging nodes is larger than Broadcast Limit, the unacknowledged packets are re-broadcast again. Note that packets sent after the previous timout expiration are not retransmitted, because those may potentially have been sent immediately before the current timer expired.

In order to keep all receiving nodes updated about sent broadcast packets, even during low traffic, each sent LINK_PROTOCOL/STATE_MSG contains the sequence number of the "next sent broadcast packet from this node". This gives the receiving nodes an opportunity of early detection of lost packets,and to send a BCAST_PROTOCOL/STATE_MSG demanding retransmission.

When receiving a cluster broadcast, or a tunneled retransmission,the following is is done at the receiving node:

- The Broadcast Sequence Number is checked against the Next Expected Broadcast Packet counter for the corresponding sender node, and if it fits, this counter is updated.
- Otherwise, if the packet turns out to be a duplicate, it is silently discarded.
- Otherwise, if a gap is found in the sequence, the packet is queued in a Deferred Incoming Multicast Packets Queue, to be resequenced and delivered upwards later. If the last 4 bits of the sequence number are equal to the last 4 bits of the node's Sequence Tag, AND if this queue was empty,the gap is calculated and immediately transferred in a BCAST_PROTOCOL/STATE_MSG back to the sending node This must immediately retransmit the missing packets. Also, for each 8 received out-of-sequence packet fulfilling the sequence criteria described above, such a message must be sent. All this is analogous to how the node-to-node link protocol works.

3.9.3 Piggybacked Acknowledge

All packets, without exception, passed from one node to another, contain a valid value in the field Acknowledged Bcast Number. Since there is always some traffic going on between all nodes in the cluster (in the worst case only link supervision messages), the receiving node can trust that the Last Acknowledged Bcast counter it has for each node is kept well up-to-date. This value will under no circumstances be older than one CONTINUITY_INTERVAL, so it will inhibit a lot of unnecessary retransmissions of packets which in reality have already be received at the other end.

3.9.4 Coordinated Acknowledge Interval

If the received packet fits in sequence as described above, AND if the last four bits of the sequence number of the packet received are equal to the last four bits of the own node's Sequence Tag, AND if the difference to the last sent piggybacked Acknowledged Bcast Number to that node is more than 8, a BCAST_PROTOCOL/STATE_MSG is generated and sent back to the receiving node, acknowledging the packet, and implicitly all previously received packets. This means that e.g. node <Z.C.1> will only explicitly acknowledge packet number 1, 17, 33, and so on, node number <Z.C.2> will acknowledge packet number 2, 18, 34, etc. This condition significantly reduces the number of explicit acknowledges needing to be sent, taking advantage of the normally ongoing traffic over each link.

A node's Sequence Tag is the node's sequence number in relation to the network address of the other nodes in the cluster. E.g, if a cluster consists of the nodes <1.1.17>, <1.1.123> and <1.1.555>, those will assign themselves the sequence tags 1, 2, and 3, respectively. This way, we make the protocol behaviour independent of how node addresses are assigned in the cluster. The sequence tags must be recalculated locally on each node when the cluster topology changes.

3.9.5 Replicated Delivery

When an in-sequence functional multicast is finally is delivered upwards in the stack, be it via cluster broadcast,replicated multicast, or tunneled broadcast retransmission, TIPC looks up in the naming table and finds all node local destination ports. The destination list created this way is stripped of all duplicates, so that only one message replica is sent to each identified destination port.

3.9.6 Congestion Control

Messages sent over the "broadcast link" are subject to the same congestion control mechanisms as point-to-point links, with prioritized transmission queue appending, message bundling, and as last resort a return value to the sender indicating the congestion. Typically this return value is taken care of by the socket layer code, blocking the sending process until the congestion abates. Hence, the sending application should never notice the congestion at all.

3.10 Fault Handling

Most functions for improving system fault tolerance are described

elswhere, under the repective functions, but some aspects deserve being mentioned separately.

3.10.1 Fault Avoidance

Strict source address check

After the neighbour detection phase, a message arriving to a node must have a not only a valid Pevious Node address, but this must belong to one of the nodes known having a direct link to the destination. The node may in practice be aware of at most a few hundred such nodes, while a network address is 32 bits long. The risk of accepting a garbled message having a valid address within that range, a sequence number that fits into the reception window, and otherwise valid header fields, is extremely small, no doubt less than one to several billions.

Sparse port address space

As an extra measure, TIPC uses a 32-bit pseudo-random number as the first part of a port identity. This gives an extra protection against corrupted messages, or against obsolete messages arriving at a node after long delays. Such messages will not find any destination port, and be attempted returned to the sender port. If there is no valid sender port, the message should be quietly discarded.

Name Table Keys

When a naming table is updated with a new publication, each of those are qualified with a Key field, that is only known by the publishing port. This key must be presented and verified when the publication is withdrawn, in all instances of the naming table. If the key does not fit, the withdrawal is refused.

Link Selectors

Whenever a message/packet is sent or routed, the link used for the next-hop transport is always selected in a deterministic way, based on the sender port's random number. The risk of having packets arriving in disorder is hence non-existent for single-hop messages, and extremely low for multi-hop messages.

Repeated Name Lookups

If a lookup in the naming table has returned a port identity that later turns out to be false, TIPC performs up to 6 new lookups before giving up and rejecting the message.

Routing Counter

To eliminate the risk of having messages roaming around in the network a routing counter is present in the TIPC header. This counter is updated for each inter-node hop and for each naming table lookup the message is subject to. If this counter reaches the upper limit, seven, the message is rejected back to the sender port.

3.10.2 Fault Detection

The mechanisms for fault detection have been described in previous sections, but some of them will be briefly repeated here:

- Transport Level Sequence Number, to detect disordered multi-hop packets.
- Connection Supervision and Abortion mechanism.
- Link Supervision and Continuation control.

3.10.3 Fault Recovery

When a failure has been detected, several mechanisms are used to eliminate the impact from the problem, or when that is impossible, to help the application to recover from it:

Link Changeover

When a link fails, its traffic is directed over to the redundant link, if any, in such a way that message sequentiality and cardinality is preserved. This feature is described in Section 3.7.9.

Returning Messages to Sender

When no destination is found for a message, the 1024 first bytes of it is returned to the sner port, along with an approriate error code. This helps the application to identify the exact instant of failure, and if possible, to find a new destination for the failed call. The complete list of error codes and their significance is described in Section 4.

3.10.4 Overload Protection

To overcome situations where the congestion/flow control mechanisms described earlier in this section are inadequate or insufficient, TIPC must provide two additional overload protection services:

Node Overload Protection

TIPC must maintain a global counter on each node, keeping track of the total number of pending, unhandled payload messages on the node. When this counter reaches a critical value, which should be

configurable, TIPC must selectively reject new incoming messages. Which messages to reject must be based on the following criteria:

* Message importance. DATA_LOW messages should be rejected at a lower threshold than DATA_NORMAL messages, which should be rejected before DATA_HIGH messages. DATA_NONREJECTABLE should not be rejected at all.
* Message type. Connectionless messages should be rejected earlier than connection oriented messages. Rejecting such messages normally means rejecting a service request form the beginning, causing less disturbances than interrupting a transaction already in progress, e.g. an ongoing phone call.

Process Overload Protection

TIPC must maintain a counter for each process, or if this is impossible, for each port, keeping track of the total number of pending, unhandled payload messages on that process or port. When this counter reaches a critical value, which should be configurable, TIPC must selectively reject new incoming messages. Which messages to reject should be based on the same criteria as for the node overload protection mechanism, but all thresholds must be set significantly lower. Empirically a ratio 2:1 between the node global thresholds and the port local thresholds has been working well.

4. TIPC Packet Format

A TIPC packet is composed of a header and a user data part. Two different header formats are used, one for payload (user-to-user) messages, and one for TIPC internal protocol messages.

4.1 TIPC Payload Message Header

4.1.1 Payload Message Header Format

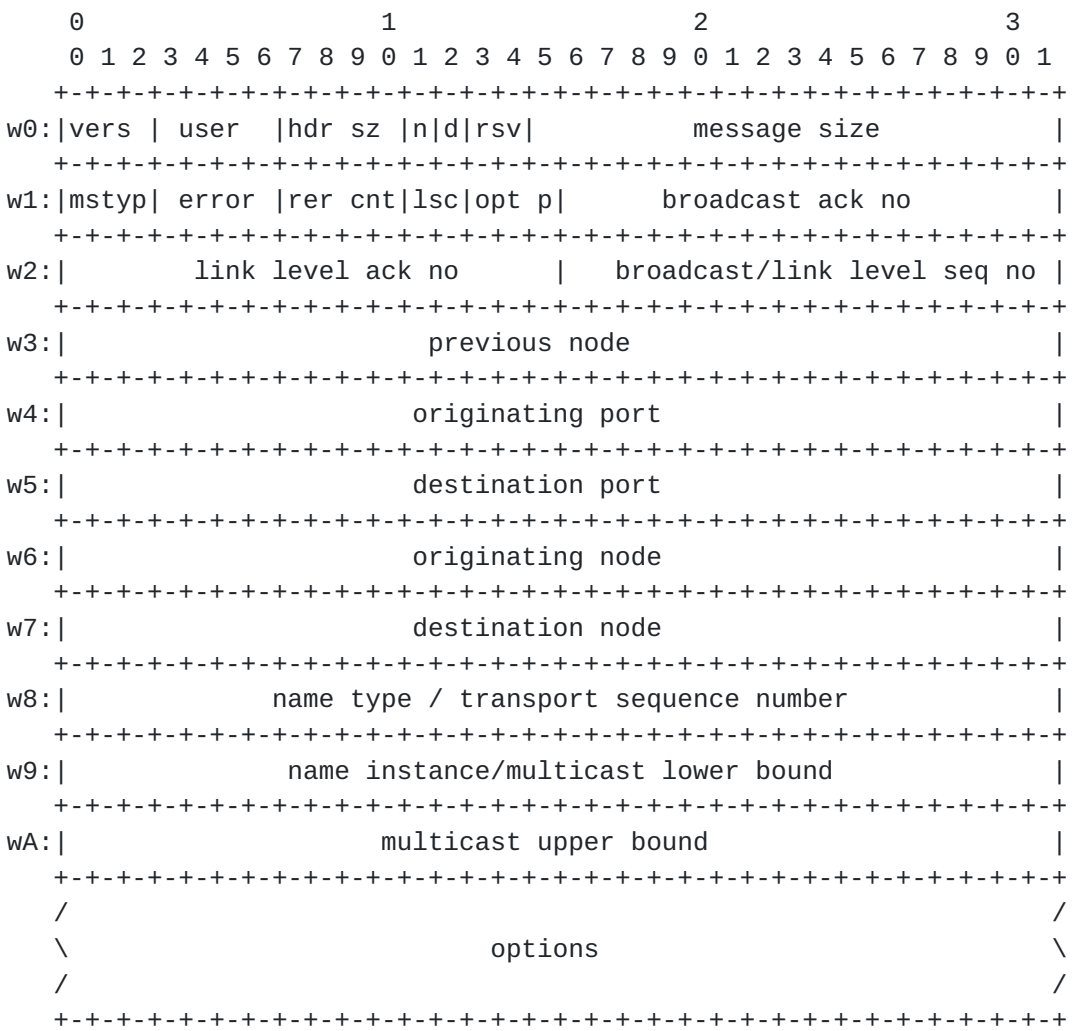

Figure 25: TIPC payload message header format

Each 32-bit word in the header is transmitted as an integer coded in network byte order.

The first four words of the header have an identical format in all

messages, independently of whether they are internal or payload messages.

4.1.2 Payload Message Header Field Descriptions

Version: 3 bits

To enable future upgrades the protocol version must be present in the header.The current version of the protocol is 2.

User: 4 bits

This field not only indicates whether the message is a protocol message or a data (payload) message, but in the latter case even the importance priority of the user. The following values are used:

ID Value	User	
0	Low Priority Payload Data	(DATA_LOW)
1	Normal Priority Payload Data	(DATA_NORMAL)
2	High Priority Payload Data	(DATA_HIGH)
3	Non-Rejectable Payload Data	(DATA_NON_REJECTABLE)

Header Size: 4 bits

Since header size both is variable and has been given a semantic meaning it must be present in the header. This feature also provides forward compatibility in case we need to extend the header format in the future. If options are present the header size will comprise the options size. The unit used is 32 bit words, implying that the maximum allowed header size is 60 bytes.

Non-sequenced: 1 bit

Indicates whether a packet belongs to the sequence of a link or not. Broadcast packets and neighbour detetction packet have this bit set. If the packet is a broadcast it must be subject to special treatment at the receiving node, with some of the fields having a different meaning than in the unicast case. This is described in section 3.

Drop: 1 bit

If this bit is set,the message should be dropped in case of congestion or overload,instead of being returned to the sender. A connection may have this bit set to 1 for messages going in one direction, and to 0 for those going in the opposite direction.

Default value is 0.

Reserved: 2 bits

Unused in this protocol version, but must be set to zero in order to facilitate compatibility if used in the future.

Message Size: 17 bits

This field indicates the size of the complete message, end-to-end, inclusive header size. The maximum size of a data message is internally set to 66060 bytes, i.e. 66000 bytes of user data plus plus a maximal header, inluding options. The limit of 66000 was chosen to make it possible to tunnel maximum-size IP-messages through TIPC, but technically this can easily be extended, since there is an adjacent unused field of three bits.

Message Type: 3 bits

ID Value	User	
0	Message sent on connection	(CONN_MSG)
1	Logical multicast message	(MCAST_MSG)
2	Message with port name destination address	(NAMED_MSG)
3	Message with port identity destination address	(DIRECT_MSG)

Error Code: 4 bits

ID Value	Meaning	
0	No error	(TIPC_OK)
1	Destination port name unknown	(TIPC_ERR_NO_NAME)
2	Destination port does not exist	(TIPC_ERR_NO_PORT)
3	Destination node unavailable	(TIPC_ERR_NO_NODE)
4	Destination node overloaded	(TIPC_ERR_OVERLOAD)
5	Connection Shutdown (No error)	(TIPC_CONN_SHUTDOWN)

Reroute Counter: 4 bits

This counter has the purpose of stopping messages from roaming around in the system. This may, at least theoretically, happen in case of temporary naming table or routing table inconsistency. The counter is incremented each time a lookup is done in the naming table, and each time the message makes an inter-node hop. When the counter reaches a limit, (seven in the current implementation) the counter is reset and the message is rejected

with the appropriate error code.

Lookup Scope: 2 bits

When a port name has been successfully translated to a port identity, the field "Destination Node" is filled with a complete node address. This also means that the scope of the original lookup domain is lost, since this is indicated by the value of this field before the lookup. Sometimes, e.g. because of temporary inonsistency ot the naming table during update, the destination port turns out to not exist, and one or more new lookups must be performed. In order to do this correctly, the original lookup scope must be preserved in the message, and that is done in this field. The following values apply:

ID Value	Meaning
1	Zone Scope
2	Cluster Scope
3	Node Scope

The lookup domain is recreated based on the complete destination node address and the lookup scope.

Options Position: 3 bits

The position of the first word of the options field, if any. If zero there are no options. Otherwise it will have values between 1 (word 8/byte 32) and 7 (word 14/byte 56).

Broadcast Acknowledge Number: 16 bits

All messages, irrespective of user and type, carry a valid value in this field. It informs the recipient about the last in-sequence broadcast packet received from the recipient node in the sender node. This gives the recipient node a chance to release sent broadcast buffers, or to retransmit broadcast packets in case it discovers a lag-behind for this node.

Link Level Acknowledge Number: 16 bits

All messages, except broadcast messages and LINK_PROTOCOL messages of type RESET_MSG and ACTIVATE_MSG, carry a valid value in this field. It informs the recipient about the last in-sequence packet received on this link in the sender node. This gives the recipient node a chance to release sent buffers.

Link Level Sequence Number: 16 bits

All non-broadcast packets, except LINK_PROTOCOL, contain a sequence number valid for their particular link, in order to keep track of message flow, detect packet losses, duplicates or out-of-sequence packets. The first packet sent after a link reset has the sequence number 0.

Broadcast Sequence Number: 16 bits

All broadcast packets contain a sequence number valid for the sending node, in order to keep track of message flow, detect packet losses, duplicates or out-of-sequence packets. Since such packets are unrelated to any links, and are easily identified by the 'broadcast' bit, we can reuse the physical area of the 'Link Level Sequence Number' for this field.

Previous Node: 32 bits

The network address of the last node visited by the message. In the case of intra-cluster messages this is most often, but not always, identical to the node from which the message originates.

Originating Port: 32 bits

This field is specific for data messages, and contains the random number identifying the originating port locally on the originating node.

Destination Port: 32 bits

The random number identifying the destination port on the destination node. For NAMED_MSG and MCAST_MSG messages this field is set to zero until a lookup in the naming table has found a destination. As long as the value remains zero new lookups will be performed until a destination is found, or until 'Reroute Counter' reaches the upper limit.

Originating Node: 32 bits

The node from which the message originally was sent.

Destination Node: 32 bits

The final destination node for a message, when known. For port name addressed messages this field has a slightly different meaning before and after the final destination is determined. See Section 3.2.8.

Transport Level Sequence Number: 32 bits

For port named messages a connection sequence number has no meaning, just as connection based messages never contain a port name. Because of this mutual exclusion we can use the same physical space for both these fields. The connection sequence number is only defined and checked for potentially routed messages, i.e. for messages passed between different clusters, or between secondary nodes and system nodes. The first message sent in each direction on such a connection has the sequence number 42.

Port Name Type: 32 bits

The type part of a destination port name or port name sequence.

Port Name Instance: 32 bits

The instance part of a destination port name.

Port Name Sequence Lower: 32 bits

The 'lower' boundary of a destination port name sequence. Uses the same physical field as 'Port Name Instance' described above.

Port Name Sequence Upper: 32 bits

The 'upper' boundary of a destination port name sequence.

4.1.3 Payload Message Header Size

The header is organized so that it should be possible to omit certain parts of it, whenever any information is dispensable. The following header sizes are used:

Cluster Internal Connection Based Non-Routed Messages:

Such messages per definition do only one hop over an inherently reliable link, so all fields from word 6 and onwards are redundant or irrelevant. The message header can be limited to 24 bytes. By ensuring that no other messages have this particular header size, this can indeed be used as a test that we are dealing with that kind of message, and some code optimization can be done based on this knowledge.

Direct Addressed Messages:

These are connection-less messages containing a port identity as destination address, i.e. the fields 'destination port' and 'destination node' are filled in and non-zero. All fields from word 7 and onwards are irrelevant, and the message size can be set

to 32.

Connection Based Potentially Routed Messages:

Inter cluster connection based messages, and intra-cluster messages between cluster nodes and secondary nodes, may need to be routed via intermediate nodes if there is no direct link between the two. 'Originating node' may hence differ from 'previous node', so this field must be present. Since there is now a small, but not negligeable risk that messages may be lost or arrive in disorder (the intermediate node may crash), a transport level connection sequence number is needed for problem detection. A header size of 36 bytes is required.

Port Name Addressed Messages:

These are connection-less messages containing a port name as destination address, i.e. the fields 'name type' and 'name instance' have valid values, while 'destination port' is zero before the name table lookup, and non-zero after a sucessful lookup. 'Destination node' may be zero or have a valid value before lookup,but has a valid value after a sucessful lookup. The header size is set to 40.

Multicast Messages:

Multicast messages are similar to port name addressed messages, except that the destination address is a range (port name sequence) rather than a port name. An extra word, the 'upper' part of the port name sequence must be present, so the header size ends up at 44.

Messages with Options:

All message headers may exceptionally be appended with an 'options' field, e.g. containing tracing information or a time stamp. In this case the total header length may take any four-byte aligned value up to the maximum, 60. There is one anomaly here, however. The non-routed 24-byter header can not take any option without invalidating the semantic meaning of the header size. Hence, such headers must be expanded to the full 36 bytes before any options can be added.

4.2 TIPC Internal Header

4.2.1 Internal Message Header Format

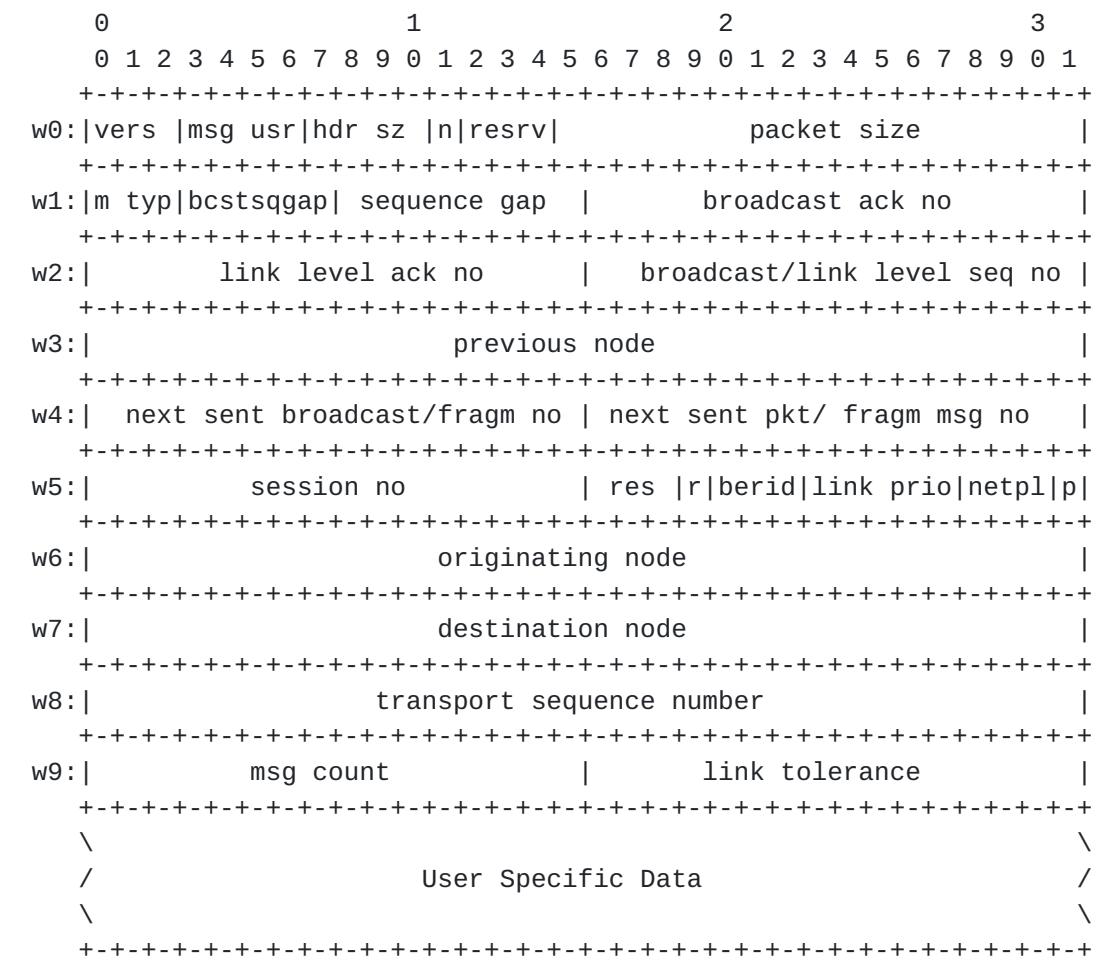

Figure 30: TIPC Internal Message Header Format

The internal header has one format and one size, 40 bytes. Some fields are only relevant to some users, but for simplicity in understanding and implementation a we present it as single header format.

4.2.2 Internal Message Header Fields Description

The first four words are almost identical to the corresponding part of the data message header. The differences are described next.

User: 4 bits

For TIPC internal messages this field has a different set of values than for data messages. The following values are used:

ID Value	User	
5	Broadcast Maintenance Protocol	(BCAST_PROTOCOL)
6	Message Bundler Protocol	(MSG_BUNDLER)
7	Link State Maintenance Protocol	(LINK_PROTOCOL)
8	Connection Manager	(CONN_MANAGER)
9	Routing Table Update Protocol	(ROUTE_DISTRIBUTOR)
10	Link Changeover Protocol	(CHANGEOVER_PROTOCOL)
11	Name Table Update Protocol	(NAME_DISTRIBUTOR)
12	Message Fragmentation Protocol	(MSG_FRAGMENTER)

Packet Size: 17 bits. Used by: All users

This is physically the same field as 'Message Size' in data messages, but now indicates the actual size of the packet it envelopes.

Message Type: 4 bits. Used by: All users

The values in this field is defined per user. See the section describing each separate user below.

Broadcast Sequence Gap: 5 bits. Used by: LINK_PROTOCOL

The field 'Error Code','Reroute Count', 'Lookup Scope' and 'Options Position' fields have no relevance for LINK_PROTOCOL/STATE_MSG messages, so these 13 bits can be recycled in such messages. 'Broadcast Sequence Gap' informs the recipient about the size of a gap detected in the sender's received broadcast packet sequence, from 'Broadcast Acknowledge Number' and onwards. The receiver of this information must immediately retransmit the missing packets.

Sequence Gap: 8 bits. Used by: LINK_PROTOCOL

The field 'Error Code','Reroute Count', 'Lookup Scope' and 'Options Position' fields have no relevance for LINK_PROTOCOL/STATE_MSG messages, so these 13 bits can be recycled in such messages. 'Sequence Gap' informs the recipient about the size of a gap detected in the sender's received packet sequence, from 'Link Level Acknowledge Number' and onwards. The receiver of this information must immediately retransmit the missing packets.

Next Sent Broadcast: 16 bits.Used by: LINK_PROTOCOL

In order to speed up detection of lost broadcasts packets all LINK_PROTOCOL/STATE_MSG messages contain this information from the sender node. If the receiver finds that this is not in

accordance with what he has received, he immediately sends a BCAST_PROTOCOL/STATE_MSG back to the sender, with Sequence Gap set appropriately.

Fragment Number: 16 bits.Used by: MSG_FRAGMENTER

Occupying the same space as 'Next Sent Broadcast' this value indicates the number of a message fragment within a fragmented message, starting from 1.

Next Sent Packet: 16 bits. Used by: LINK_PROTOCOL

Link protocol messages bypass all other packets in order to maintain link integrity, and hence can not have sequence numbers valid for the ordinary packet stream. But all receivers are dependent of this information to detect packet losses, and cannot completely rely on the assumption that a sequenced packet will arrive within acceptable time. To guarantee a worst case packet loss detection time, even on low-traffic links,the equivalent information to a valid sequence number has to be conveyed by the link continuity check (STATE_MSG) messages, and that is the purpose of this field.

Fragment Number: 16 bits.Used by: MSG_FRAGMENTER

Occupying the same space as 'Next Sent Packet', this value identifies a a fragmented message on the particular link where it is sent.

Session Number: 16 bits. Used by: LINK_PROTOCOL

The risk of packets being reordered by the router is particularly elevated at the moment of first contact between nodes, so a check of sequentiality is needed even for LINK_PROTOCOL/RESET_MSG messages. The session number starts from a random value, and is incremented each time a link comes up. This way, redundant RESET_MSG messages, delayed by the router and arriving after the link has been brought to a working state,can be identified and ignored.

Reserved: 3 bits

Must be set to zero.

Redundant Link: 1 bit

When set, this bit informs the other endpoint that the sender thinks it has a second working link towards the destination. This

information is needed by the recipient on order know whether he should initiate a changeover procedure or not in case of link failure. Under certain, extremely transient and rare, circumstances, it is not sufficient for an endpoint to know its own link view to perform a correct changeover.

Bearer Identity: 3 bits

When a bearer is registered with the link layer of TIPC in a node, it is assigned a unique ideitifying number in the range [0,7]. This number will not necessarily be the same in different nodes, so a link endpoint to needs to know the other endpoint's assigned identity for the same bearer. This is needed during the link changeover procedure, in order to identify the destination bearer instance of a tunneled packet.

Link Priority: 5 bits. Used by: LINK_PROTOCOL

When there are more than one link between two nodes, one may want to use them in load sharing or active/standby mode. Equal priority between links means load sharing, different priorities means that the link with the higher numerical value will take all traffic. By offering a value range of 32 one can build in a default relation between different bearer types,(e.g. DCCP is given lower priority than Ethernet), and no manual configuration of these values should normally be needed.

Network Plane: 3 bits. Used by: LINK_PROTOCOL

When multiple parallel routers and multiple network interfaces are used it is useful, although not strictly needed by the protocol, to have a network pervasive identifier telling which interfaces are connected to which routers. This relieves system managers from the burden of manually keeping track of the actual physical connectivity. Typically, the identifier 0 would be presented to the operator as 'Network A', identity 1 as 'Network B' etc. This identity must be agreed upon in the whole network, and therefore this field is present and valid in the header of all LINK_PROTOCOL messages. The 'negotiation' consists of letting the node with the lowest numeral value of its network address, typically node <1.1.1>, decide the identities. All others must strictly update their identities to the value received from any lower node.

Probe: 1 bit. Used by: LINK_PROTOCOL

This one-bit field is used only by messages of type LINK_PROTOCOL/ STATE_MSG. When set it instructs the receiving link endpoint to immediately respond with a STATE_MSG. The Probe bit MUST NOT be

set in the responding message.

Originating Node: 32 bits. Used by: NAME_DISTRIBUTOR

The node from which the message originally was sent.

Destination Node: 32 bits. Used by: NAME_DISTRIBUTOR

The final destination node for a message.

Transport Level Sequence Number: 32 bits. Used by: NAME_DISTRIBUTOR

The sequence number of the NAME_DISTRIBUTOR message. Only used and valid when the message needs routing.

Message Count: 16 bits. Used by: MSG_BUNDLER, CHANGEOVER_PROTOCOL

This field is used for two different purposes. First, the message bundling function uses it to indicate how many packets are bundled in a bundle packet. Second, when a link goes down, the endpoint detecting the failure must send an ORIG_MSG to the other endpoint (tunneled through the remaing link) informing it about how many tunneled packets to expect. This gives the other endpoint a chance to know when the changeover is finished, so it can return to the normal link setup procedure.

Link Tolerance: 16 bits. Used by: LINK_PROTOCOL

Each link endpoint must have a limit for how long it can wait for packets from the other end before it declares the link failed. Initially this time may differ between the two endpoints, and must be negotiated. At link setup all RESET_MSG messages in both directions carry the sender's configured value in this field, and the highest numerical value will be the one chosen by both endpoints. In STATE_MSG messages this field is normally zero, but if the value is explicitly changed at one endpoint, e.g. by a configuration command, it will be carried by the next STATE_MSG and force the other endpoint to also change its value. Subsequent STATE_MSG messages return to the zero value. The unit of the value is [ms].

4.3 Message Users

4.3.1 Broadcast Protocol

User: 5 (BCAST_PROTOCOL).

Message Types:

ID Value	Meaning
0	Tunneled, retransmitted broadcast packet (BCAST_MSG)

A BCAST_MSG message wraps a packet that was originally sent as broadcast, but which has retransmitted. Depending on the number of missing acknowledges (see Section 3.9.4), the retransmission may be performed as a new broadcast, or as a limited sequence of BCAST_MSG unicasts to the nodes missing in the acknowledge list.

4.3.2 Message Bundler Protocol

User: 6 (MSG_BUNDLER)

Message Types: None

A MSG_BUNDLER packet contains as many bundled packets as indicated in Message Count. All bundled messages start at a 4-byte aligned position in the packet. Each bundled packet is a complete packet, including header, but with the fields Broadcast Acknowledge Number, Link Level Sequence Number and Link Level Acknowledge Number left undefined. Any kind of packets, except LINK_PROTOCOL and MSG_BUNDLER packets, may be bundled.

4.3.3 Link State Maintenance Protocol

User: 7 (LINK_PROTOCOL)

ID Value	Meaning	
0	Detailed state of a working link endpoint	(STATE_MSG)
1	Reset receiving endpoint, sender is RESET_UNKNOWN	(RESET_MSG)
2	Sender in RESET_RESET,ready to receive traffic	(ACTIVATE_MSG)

RESET_MSG messages must have a data part that must be a zero-terminated string. This string is the name of the bearer instance used by the sender node for this link. Examples of such names is "eth0","vmnet1" or "udp". Those messages must also contain valid values in the fields Session Number, Link Priority and Link Tolerance.

ACTIVATE_MSG messages do not need to contain any valid fields except Message User and Message Type.

STATE_MSG messages may leave bearer name and Session Number

undefined, but Link Priority and Link Tolerance must be set to zero in the normal case. If any of these values are non-zero, it implies an order to the receiver to change its local value to the one in the message. This must be done when a management command has changed the corresponding value at one link endpoint, in order to enforce the same change at the other endpoint. Network Identity must be valid in all messages.

Link protocol messages must always be sent immediately, disregarding any traffic messages queued in the link. Hence, they can not follow the ordinary packet sequence, and their sequence number must be ignored at the receiving endpoint. To facilitate this, these messages should be given a sequence number guaranteed not to fit in sequence. The recommended way to do this is to give such messages the next unassigned Link Level Sequence Number + 362768. This way, at the reception the test for the user LINK_PROTOCOL needs to be performed only once, after the sequentiality check has failed, and we never need to reverse the Next Received Link Level Sequence Number.

4.3.4 Connection Manager

Although a TIPC internal user, Connection Manager is special, because it uses the 36-byte header format of CONN_MSG payload messages instead of the 40-byte internal format. This is because those messages must contain a destination port and a originating port.

The following message types are valid for Connection Manager:

User: 8 (CONN_MANAGER).

Message Types:

ID Value	Meaning	
0	Probe to test existence of peer	(CONN_PROBE)
1	Reply to probe, confirming existence	(CONN_PROBE_REPLY)
2	Acknowledge N Messages	(MSG_ACK)

MSG_ACK messages are used for transport-level congestion control, and carry one network byte order 32-byte integer as data. This indicates the number of messages acknowledged, i.e. actually read by the port sending the acknowledge. This information makes it possible for the other port to keep track of the number of sent, but not yet received and handled messages, and to take action if this value surpasses a certain threshold.

The details about why and when these messages are sent are described in Section 3.5.4.

4.3.5 Routing Table Update Protocol

User: 9 (ROUTE_DISTRIBUTOR).

Message Types:

ID Value	Meaning	
0	Sender's routes to cluster external nodes	(EXT_ROUTING_TABLE)
1	Sender's routes to cluster internal nodes	(LOCAL_ROUTING_TABLE)
2	Sender's routes to secondary nodes	(SEC_ROUTING_TABLE)
3	New route to a node	(ROUTE_ADDITION)
4	Lost contact with a node	(ROUTE_REMOVAL)

EXT_ROUTING_TABLE messages have the following structure:

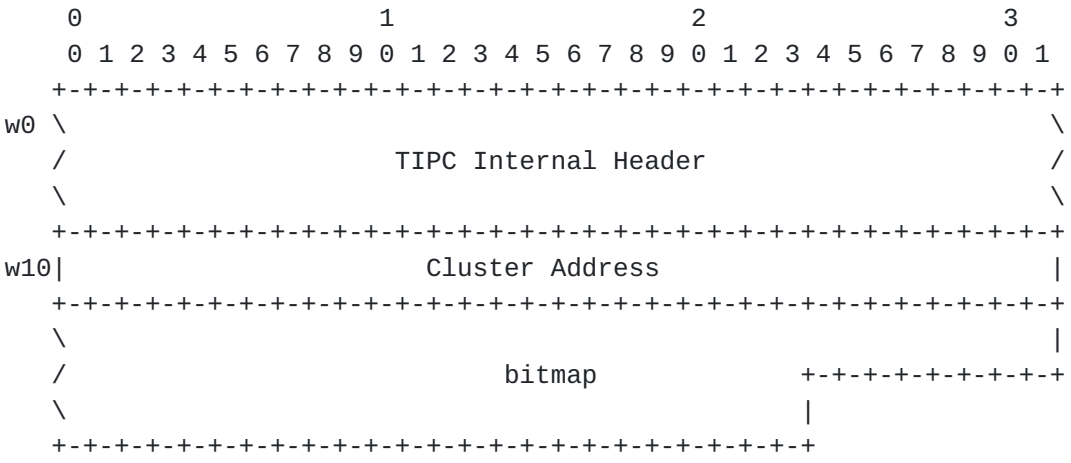

Figure 36: External Routing Table message format

The first four bytes of the message payload is a TIPC Network Address in network byte order, indicating the remote cluster concerned by the table. The rest of the message is a bitmap, indicating to which nodes within that cluster the sending node has direct links. E.g. if node <1.1.7> has a direct link to the nodes <2.3.4> and <2.3.5>, Cluster Address will contain <2.3.0>, and bit 4 and 5 of the remaining data is set to a non-zero value. The message need not be longer than to contain the last non-zero bit in the map. Along with the Originating Node field this message contains all information needed for any node in the cluster to know how to reach the two

nodes.

LOCAL_ROUTING_TABLE and SEC_ROTING_TABLE messages have the same structure as EXT_ROUTING_TABLE, but Cluster Address may be left undefined, since the receiving nodes already know to which cluster they belong.

ROUTE_ADDITION and ROUTE_REMOVAL messages have the following format:

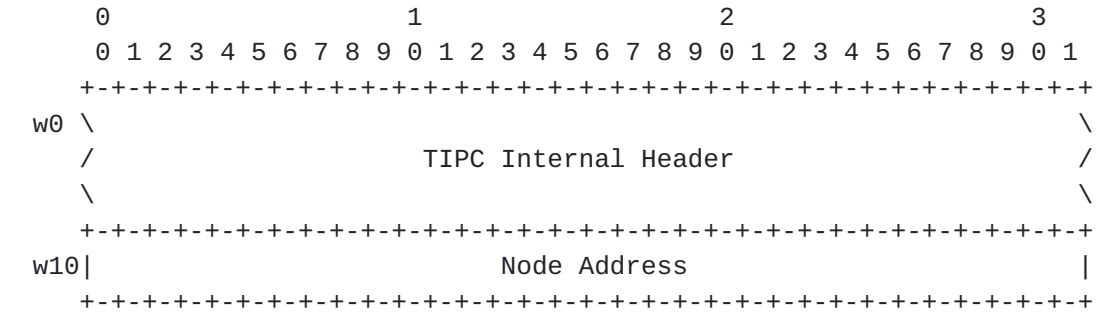

Figure 37: Route Addition/Removal message format

Here the only information needed is a Network Address indicating which node the route addition/loss concern, i.e. to which node the sender node has established/lost direct contact.

4.3.6 Link Changeover Protocol

User: 10 (CHANGEOVER_PROTOCOL)

ID Value	Meaning	
0	Tunneled duplicate of packet	(DUPLICATE_MSG)
1	Tunneled original of packet	(ORIGINAL_MSG)

DUPLICATE_MSG messages contain no extra information in the header apart from the first thee words. The first ORIGINAL_MSG message sent out MUST contain a valid value in the Message Count field, in order to inform the recipient about how many such messages, inclusive the first one, to expect. If this field is zero in the first message, it means that there are no packets wrapped in that message, and none to expect.

4.3.7 Name Table Update Protocol

User: 11 (NAME_DISTRIBUTOR)

ID Value	Meaning	
0	One or more port name publications	(PUBLICATION)
1	Withdraw port name publication	(WITHDRAWAL)

These messages have the following structure:

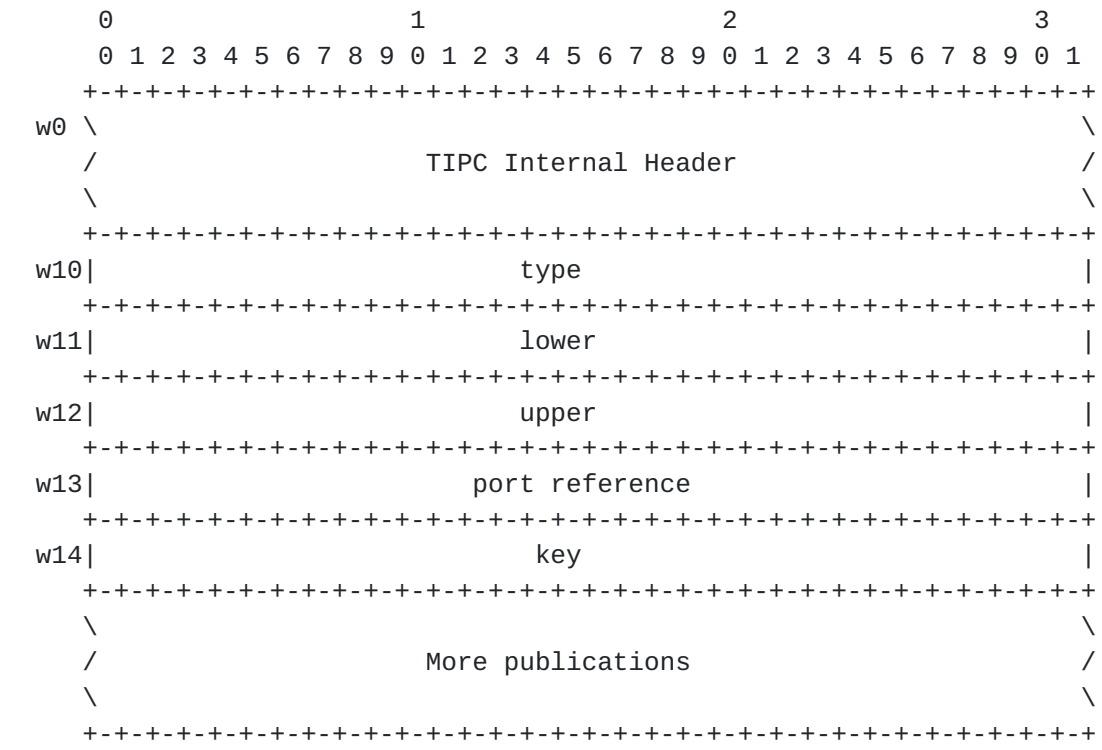

Figure 40: Name Table Update message format

PUBLICATION messages may contain one or more Publications. A Publication consists of the five-word field listed in the figure. Each field is stored in network byte order, and have the following meaning.

- Type: The 'type' part of a published/withdrawn port name sequence.
- Lower: The 'lower' part of a published/withdrawn port name sequence.
- Upper: The 'upper' part of a published/withdrawn port name sequence.
- Port Reference: The random number part of the publishing port's identity.
- Key: A key created by the publishing port. Must be presented to the receiver in the corresponding WITHDRAW message.

The number of publications in the message is calculated by the receiver based on the message length. The Node Identity part of each publication is the same as the messages Originating Node.

WITHDRAWAL messages may only contain one publication item.

4.3.8 Message Fragmentation Protocol

User: 12 (MSG_FRAGMENTER)

ID Value	Meaning
0	First fragment of message (FIRST_FRAGMENT)
1	Body fragment of message (FRAGMENT)
2	Last fragment of message (LAST_FRAGMENT)

All packets contain a dedicated identifier, Fragmented Message Number, to distinguish them from packets belonging to other messages from the same node. All packets also contain a sequence number within its respective message, the Fragment Number field, in order to, if necessary, reorder the packets when they arrive to the detination node. Both these sequence numbers must be incemented modulo 2^16-1.

4.3.9 Neighbour Detection Protocol

User: 13 (LINK_CONFIGURATION) The protocol for neighbour detection uses a special message format, with the following generic structure:

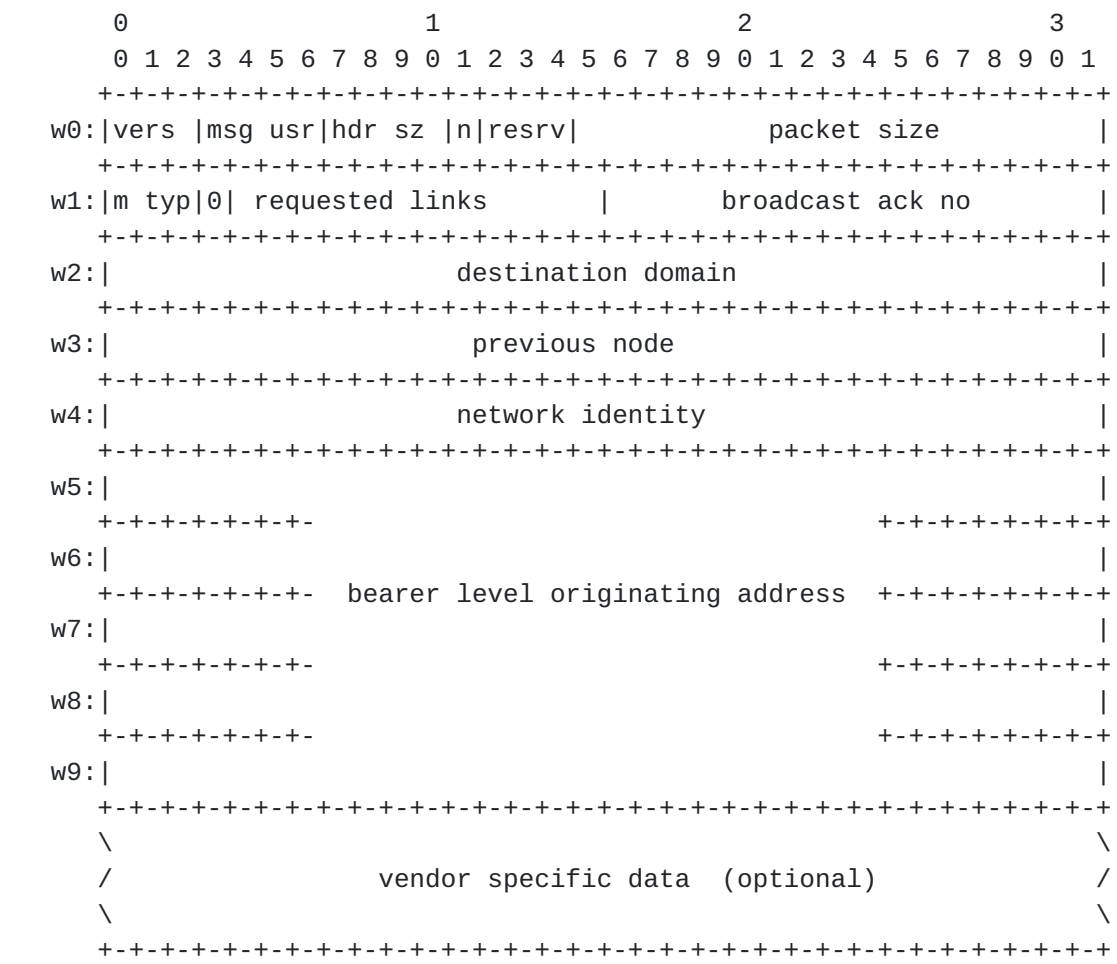

Figure 42: Link Configuration message format

- Header Size: 40 bytes.
- Non-sequenced: This bit is set to 1 for this user.
- Packet Size: Header Size plus size of Optional Data.
- Message Type: 0 if the message is a link request (e.g. a broadcast neighbour detection), 1 if it is a response to such a request.
- Requested Links (12 bits): The number of links the sender node wants to establish to the destination domain, over the specific bearer used. For a link request to a specific node this number must be 1, as only one link is permitted per bearer and node pair. Recommended default value for this field is 2.
- Originating Node: The network address of the originating node.
- Destination Domain: The domain to which the link request is directed. <Z.C.N> means that the sender requests a link to that specific node, and nothing else. <Z.C.0> or <Z.0.0> means that the sender wants the specified number of links to that cluster or zone, but not necessarily to the first node where the message

arrives. <0.0.0> means that the sender does not know anything about the receiver's identity, but wants links to anybody within the cluster receiving the message.

- Network Identity: The sender node's network identity. Receivers having a network identity different from the one in the message must ignore the message.
- Bearer Level Originating Address: A Bearer Address containing the Ethernet or IP(v4 or v6)-address+port number of the sender. Note that with IP-protocols this is only a hint, as the IP-address may have been replaced by NAT. In such cases, it is the reponsibility of the corresponding adaptation layer to extract the correct sender address from the incoming message.
- Vendor Specific Data: The contents of this optional field is vendor specific.

Link Probe Message

The messages used for finding the optimal location for responding to a Link Request are called Link Probes:

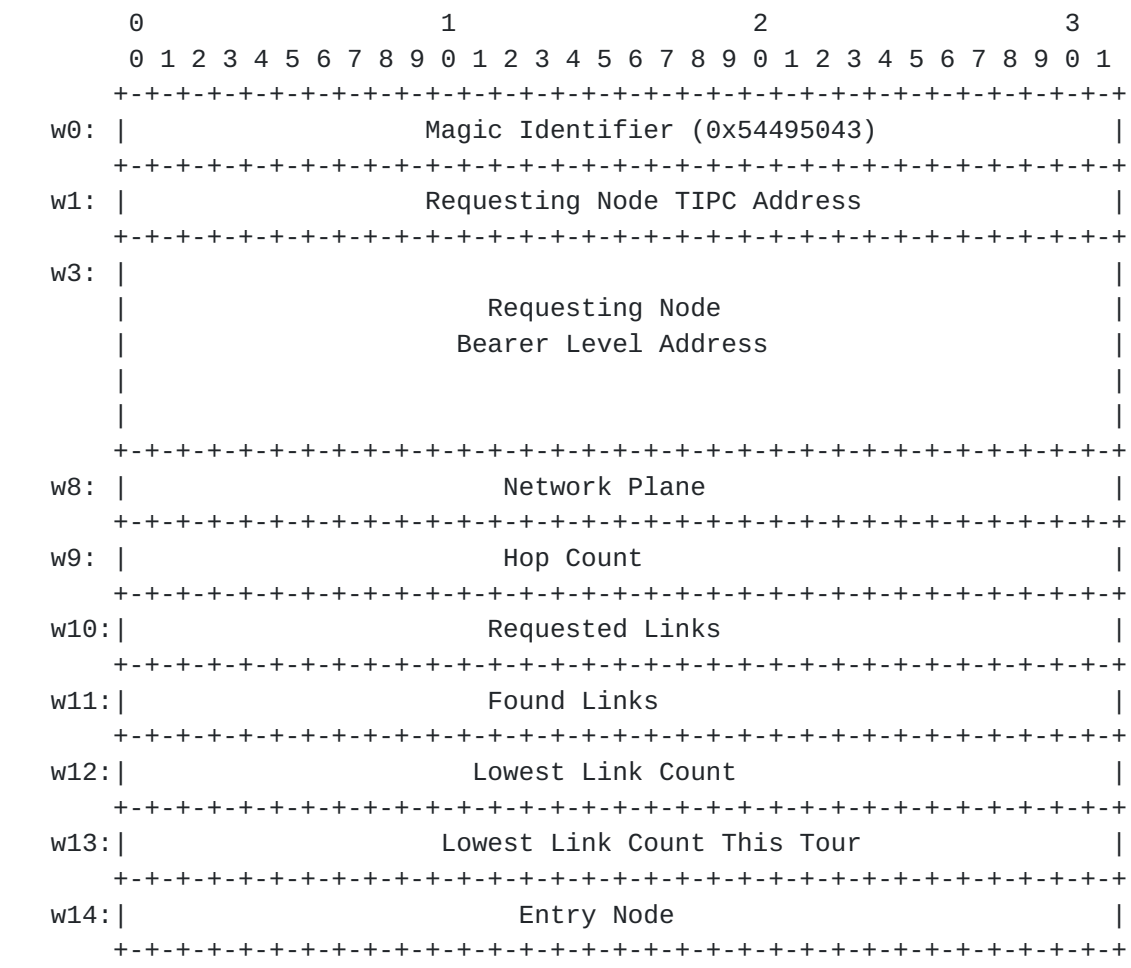

Figure 43: Link Probe message format

* Requesting Node: The node in the remote cluster which originally sent out the request.
* Requesting Node Bearer Level Address: The bearer address (e.g. IP or Ethernet address) of the node in the remote cluster which originally sent out the request.
* Network Plane: The identifier of the network where the Link Request originally was received. E.g. 0 for Network A, 1 for Network B.
* Hop Count: The number of node hops this probe has performed. If this number exceeds 10 * size of cluster, the probe must be discarded.
* Requested Links: The number of links the requesting node wants to this cluster.
* Found Links: The number of found, established links to the originating node so far. When this number equals 'Requested Links', the establishing procedure is finished,and the probe

can be discarded.

* Lowest Link Count: The lowest number of links to the requesting cluster found in the cluster during the previous tour. If, at the next tour, a node is found with this number of links, a link setup attempt is initiated. For each fulfilled tour, this field is updated with the contents of Lowest Link Count This Tour.
* Lowest Link Count This Tour: The lowest number of links to the requesting cluster found in the cluster during the current tour. This is useful if the probe fulfils a tour without finding the Lowest Link Count number of links, which may happen when the number of links is changing very rapidly.
* Entry Node: The node that received the original link request. This helps identifying when the probe has fulfilled a tour. Hop Count can not be trusted alone for this, since the number of nodes may change during the tour.

Link Probe Messages are sent to port name <1,302> using the identified next hop node as lookup domain.

Get Node Info Message

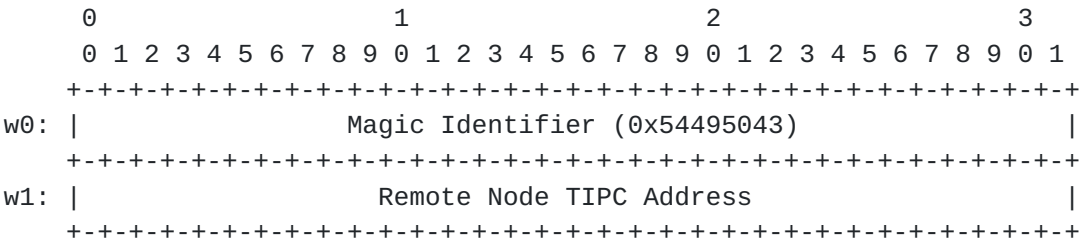

Figure 44: Get Node Info message format

Remote Node Address: The cluster remote node for which the information is requested.

Get Node Info Messages are sent to port name <1,300> using the identified router node as lookup domain.

Get Node Info Result Message

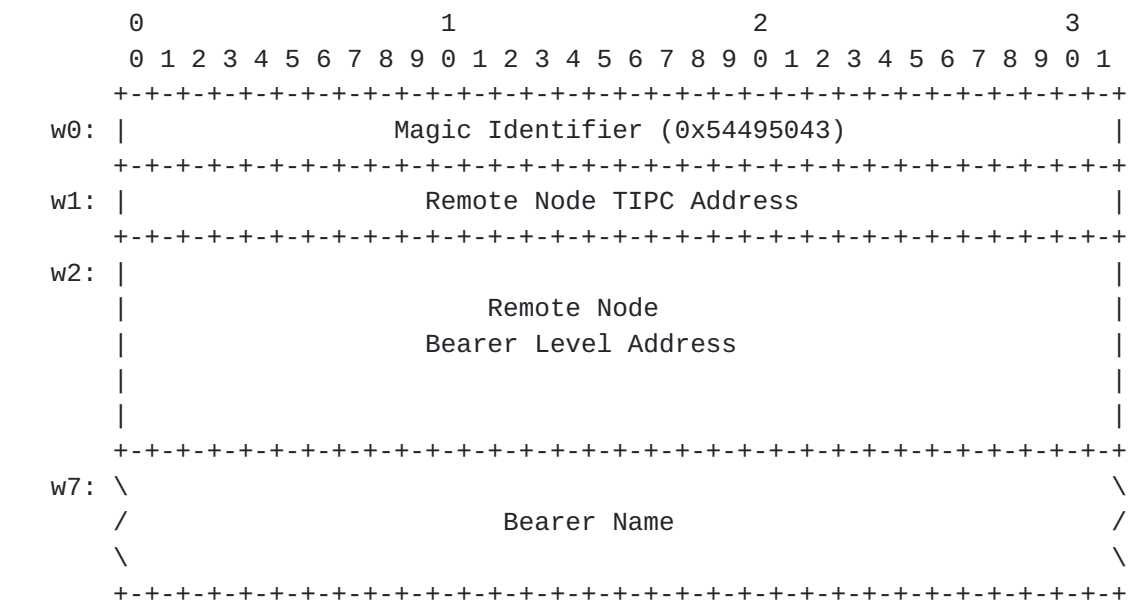

Figure 45: Get Node Info Result message format

- Remote Node TIPC Address: The cluster remote node for which the information is valid.
- Remote Node Bearer Level Address: The bearer address (e.g. IP or Ethernet address) of the node in the remote cluster.
- Bearer Name: The string identifying the bearer where the Bearer Level Address is valid, i.e. the bearer to be used for the Link Requests to be sent.

Get Node Info Result Messages are sent to port name <1,301> using the node that sent the corresponding Get Node Info message as lookup domain.

Link Request Accepted Message

This message only has the four-byte Magic Identifier as data. It is sent to port name <1,304>, using the node that sent the corresponding Link Request message as lookup domain.

Link Request Rejected Message

This message only has the four-byte Magic Identifier as data. It is sent to port name <1,303>, using the node that sent the corresponding Link Request message as lookup domain.

Drop Link Request Message

This message only has the four-byte Magic Identifier as data. It is sent to port name <1,305>, using the node that sent the

corresponding Link Request message as lookup domain.

Check Link Count Message

This message only has the four-byte Magic Identifier as data. It is sent to port name <1,306>, using the node node to check as lookup domain.

4.4 Media Adapter Protocols

The protocol for adapting to various underlying media is described in the following sections. For the time being only one such mapping is publicly available, the one for Ethernet

4.4.1 Ethernet Adaptation

Ethernet Number, formally assigned from IEEE: 0x88ca

Ethernet Adaptation Protocol Header

No header is added at this level.

5. Management

The management interface towards TIPC is a message interface. A TIPC node is managed by sending a correctly formatted message to that node. using a port name destination address with Type set to 0, and Instance set to the network address of the target node.

5.1 Command Types

There are three groups of management commands:

- Group 1: Read-only commands, or other non-intrusive commands. E.g. commands reading statistics, table contents etc. or commands resetting statitics. These commands may be called from anywhere, by sending connectionless messages.
- Group 2: Intrusive commands. Commands changing settings in TIPC must be handled very strictly. Before issuing such commands, a manager must establish an exclusive connection towards TIPC on the concerned node. Exclusive means than no more than one such link may exist at any time towards a node. Typically, a manager process will establish such a management link to all nodes in the cluster or zone at system start, and then hold on to that link as long as he is up.
- Group 3: Remote Subscriptions. Port name sequence subcriptions can be ordered not only from local users via the ordinary API, but also remotely, by issuing a correctly formatted management message towards a specific node. This way, it is possible to e.g. subscribe for a certain port name in a remote zone, even if the subscriber is located outside the publishing scope of the requested name sequence. Since such remote subscriptions are potentially intrusive, there can only exist a limited number, which should be configurable, at any time. Just like with write commands, they require that a connection is set up towards the target node. One such connection is established per-subscription.

5.2 Command Message Formats

All fields described as integers in the following sections are transferred in network byte order.

5.2.1 Command Messages

The first 16 bytes of all command messages have the same structure:

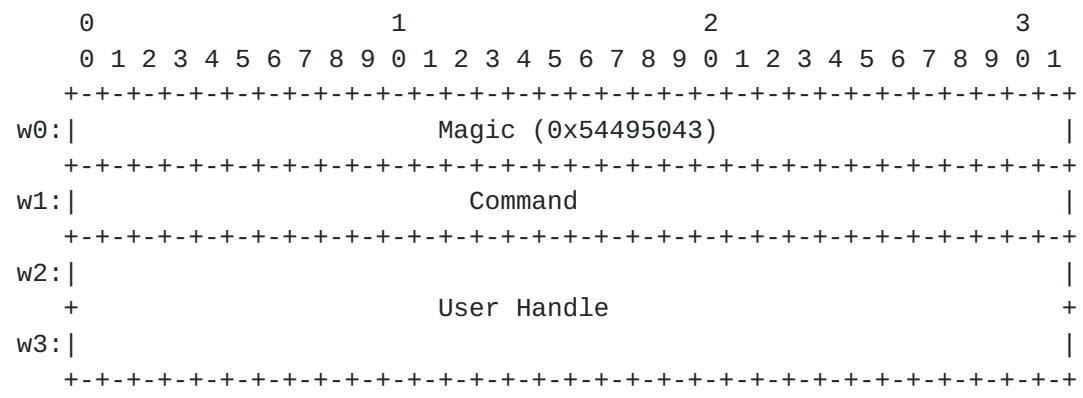

Figure 46: Command Message header format

Magic: 32 bit integer

This is an identifier with the value 0x54495043 ('T','I','P','C'), meant to protect against accidental access by corrupted or wrongly addressed command messages.

Command: 32 bit integer

This is the command issued by this message.

User Handle: 64 bits

A handle at the user's disposal, for storing e.g. a pointer.

5.2.2 Command Response Messages

The first 24 bytes of all command response messages have the same structure:

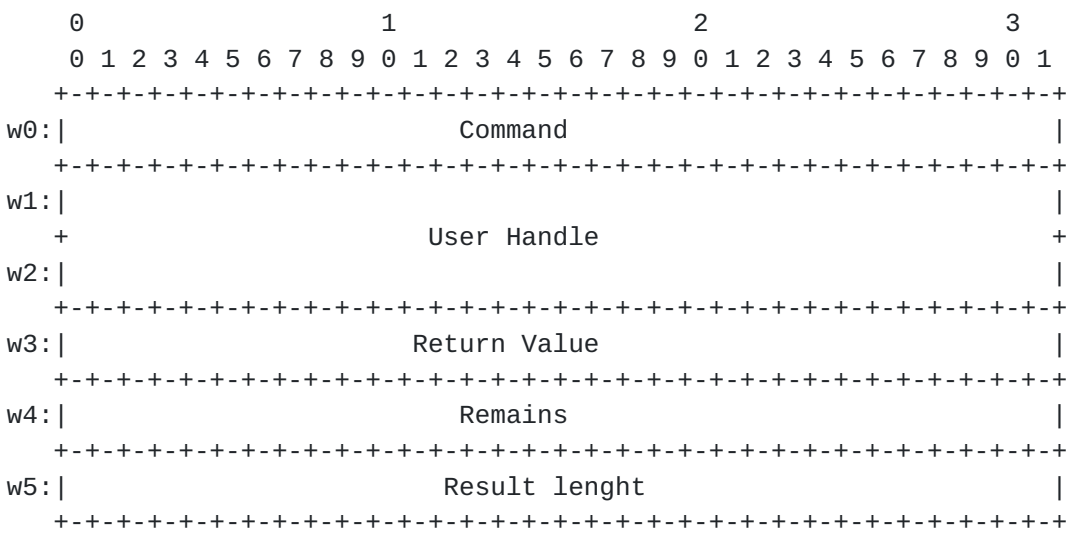

Figure 47: Command Response Message header format

Command: 32 bit integer

The original command issued in the corresponding command message.

User Handle: 64 bits

The original handle passed by the corresponding command message.

Return Value: 32 bit integer

If the command was successful, this field is 0, otherwise 0xffffffff.

Remains: 32 bit integer

If the command resulted in more return data than can be stored in one message, this field indicates the number of bytes to be expected in subsequent messages.

Result Length: 32 bit integer

The length of the remainder of the message, i.e. the command specific result.

5.3 Commands

5.3.1 Group 1: Query Commands

Get Port Statistics Group 1 command: 211

Get statistics for a certain port.
Command message:

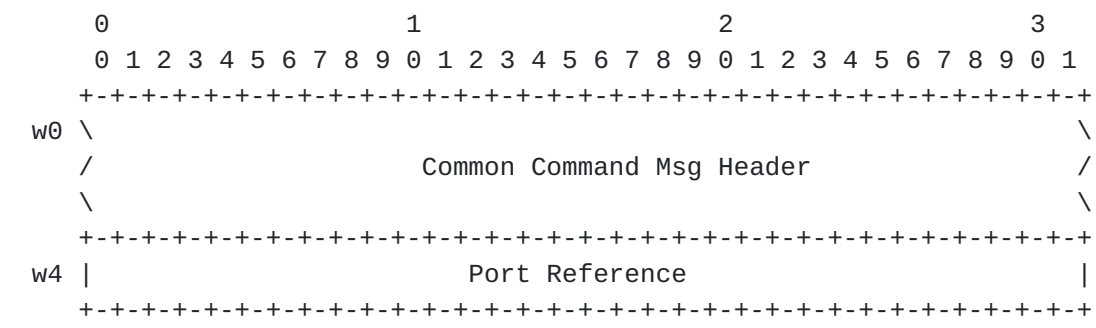

Figure 48: Get Port Statistics Query message format

Port Reference contains the random number part of the identity of the port from which statistics is wanted.
Result message:

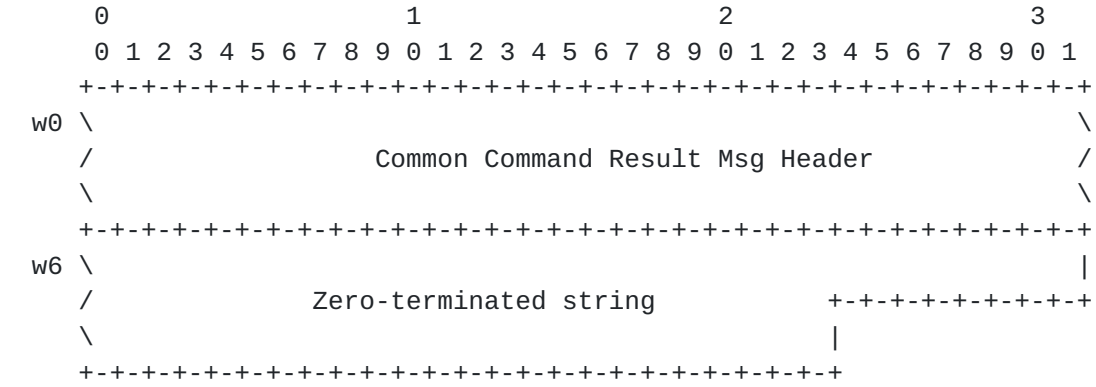

Figure 49: Get Port Statistics Query Result message format

Reset Port Statistics Group 1 command: 212

Command message:

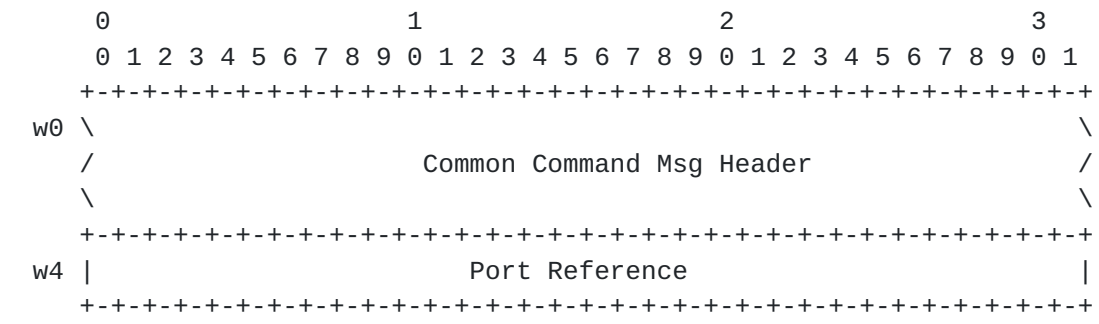

Figure 50: Reset Port Statistics Command message format

Port Reference contains the random number part of the identity of the port for which the statistics should be reset.
Result message:
The result of this command is a Common Command Result Header, indicating success or failure.

Get Nodes Group 1 command: 201

Get information about all nodes within a given domain known by the target node.
Command message:

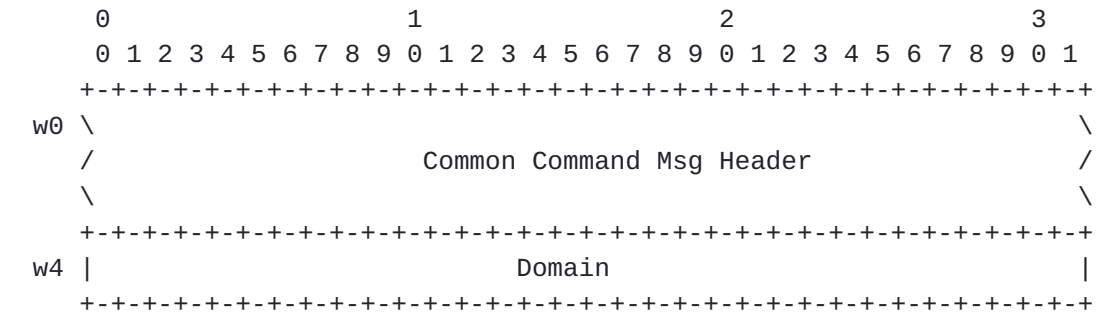

Figure 51: Get Nodes Query Command message format

Domain is a Network Address in the form <0.0.0>, <Z.0.0> etc.
Result message:

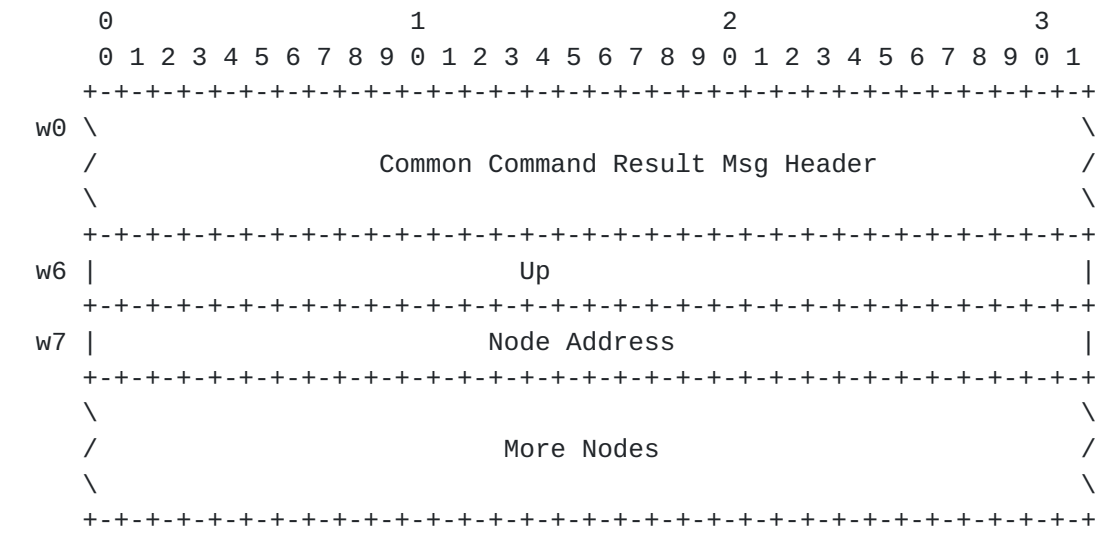

Figure 52: Get Nodes Query Result message format

Result data is a sequence of structures, each consisting of two integers.

* Up: Integer indicating whether the indicated node is reachable or not from the target node. Non-zero means it is reachable.
* Node Address: The address of the known node.

Get Links Group 1 command: 182

Get information about all links from the target node to other nodes within the given domain.
Command message:

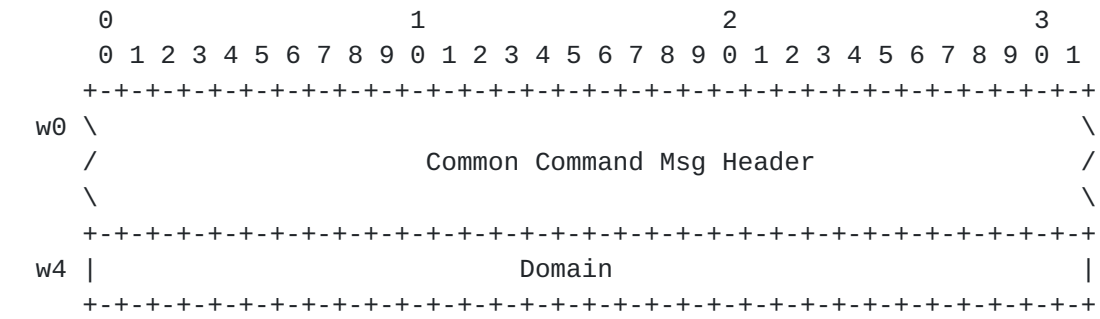

Figure 53: Get Links Query Command message format

Domain is a Network Address in the form <0.0.0>, <Z.0.0> etc.
Result message:

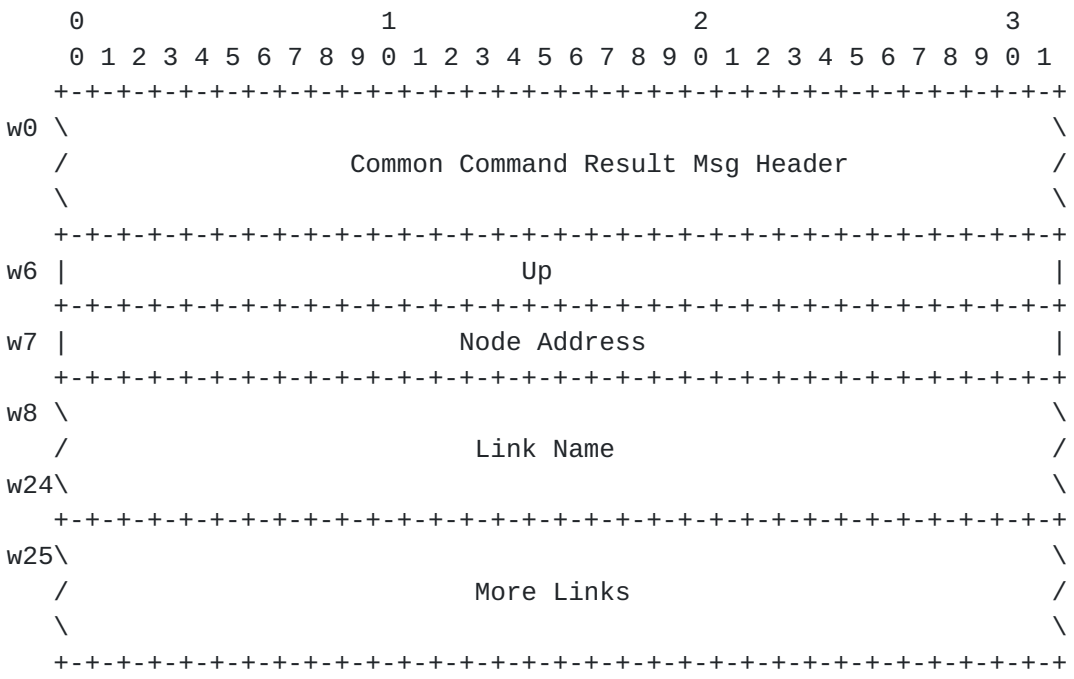

Figure 54: Get Links Query Result message format

Result data is a sequence of structures, each consisting of three elements.

* Up: Integer indicating whether the indicated link is working or not.
* Node Address: The node at the other end of the link.
* Link Name: A 72 byte field, contianing a zero-terminated string, uniquely identifying the link.

Get Routes Group 1 command: 230

Get all routes to a given domain known by the target node.
Command message:

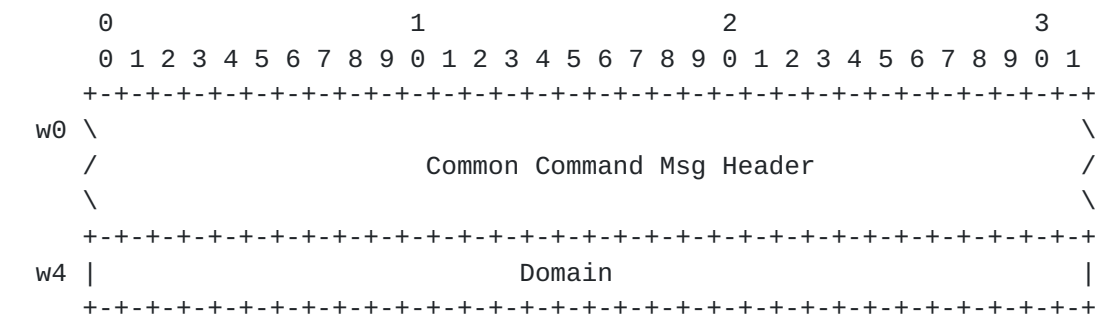

Figure 55: Get Routes Query Command message format

Domain is a Network Address in the form <0.0.0>, <Z.0.0> etc.
Result message:

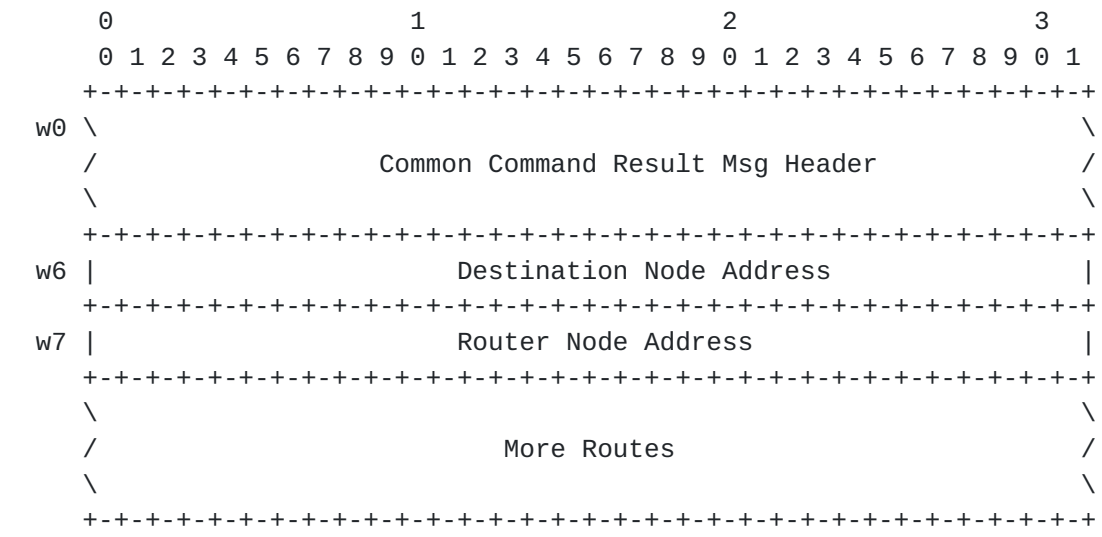

Figure 56: Get Routes Query Result message format

Result data is a sequence of structures, each consisting of two elements:

* Node Address: The network address of the node to which the route leads.
* Router Address: The network address of the cluster local node through which the indicated detination node can be reached.

Get Links Statistics Group 1 command: 183

Get statistics from the link indicated by Link Name.
Command message:

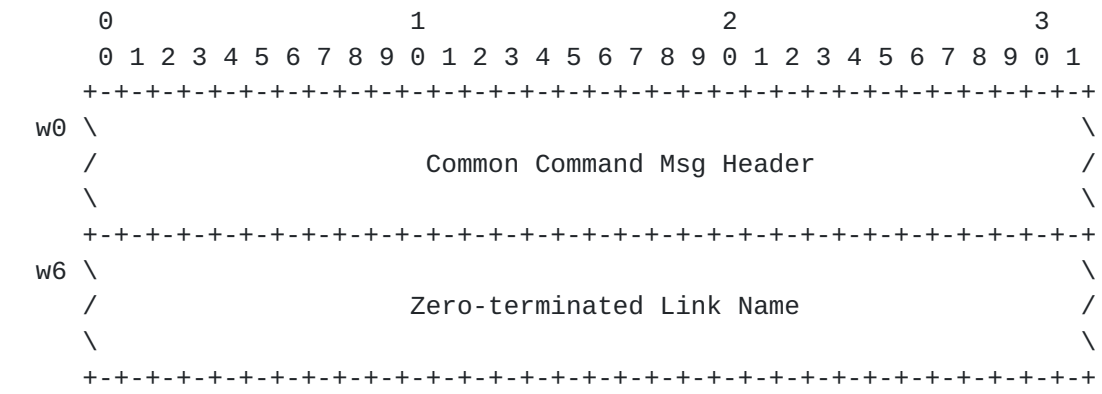

Figure 57: Get Links Statistics Query Command message format

Link Name is a 72-byte field, containing the name of the requested link.
Result message:

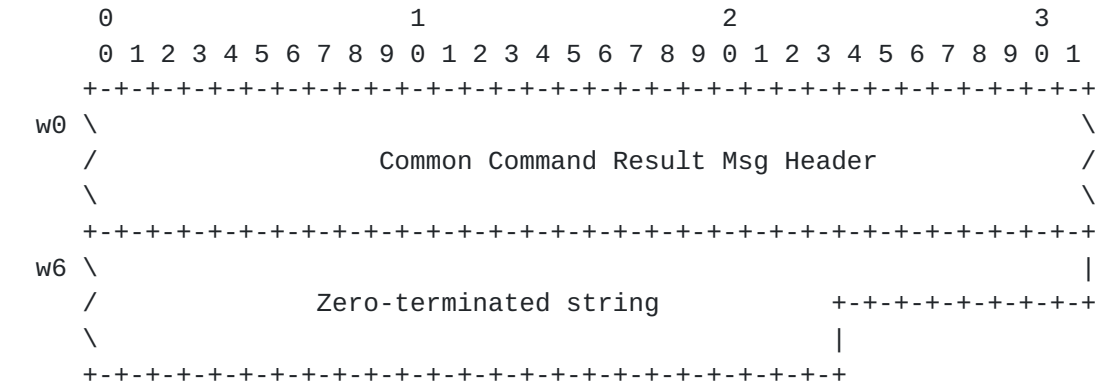

Figure 58: Get Links Statistics Query Result message format

Reset Links Statistics Group 1 command: 184

Reset statistics for the link indicated by Link Name.
Command message:
Same as for Get Link Statistics.
Result message:
The result of this command is a Common Command Result Header, indicating success or failure.

Get Peer Address Group 1 command: 193

Get the bearer level address. e.g. ethernet or IP-address:portno, for the other endpoint of the indicated link.
Command message:

Same as for Get Link Statistics.

Result message:

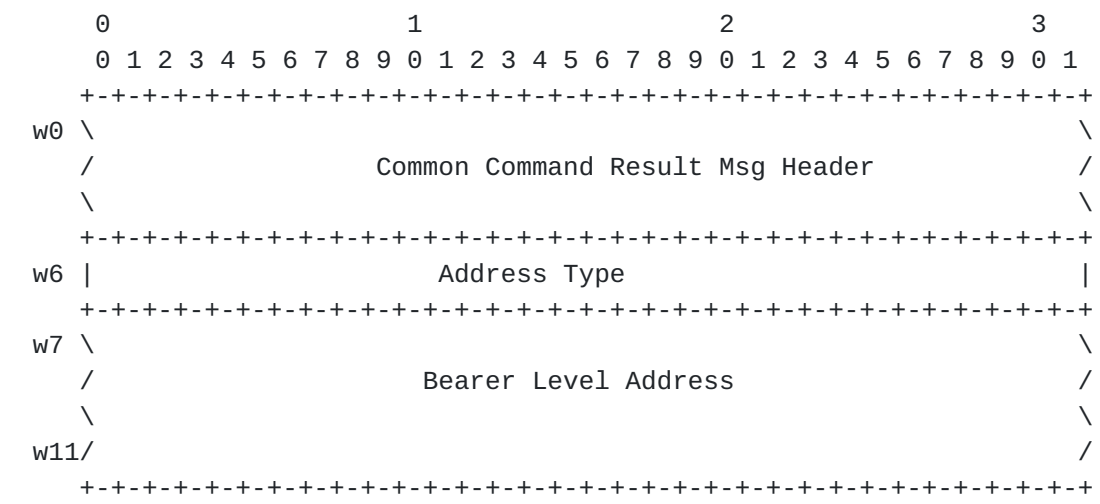

Figure 59: Get Peer Address Query Result message format

The integer Address Type has the following defined values for now:

Value	Type
0	6-byte ethernet address
1	Socket address IPv4.
2	Socket address IPv6.

An Ipv4 socket address has the following structure:

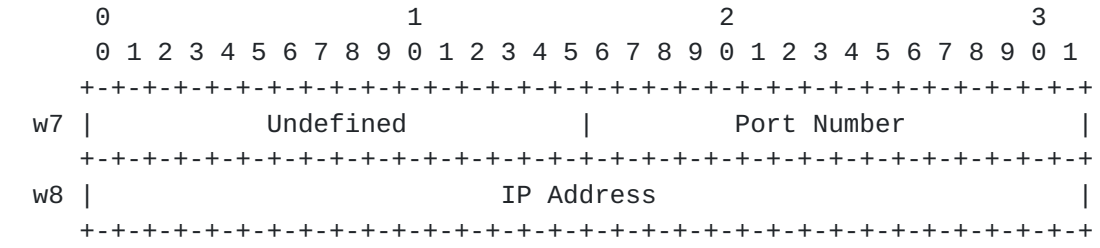

Figure 61: Socket Address format for peer address

Port Number and IP address are integers transferred in network byte order.

An Ipv6 socket address has the following structure:

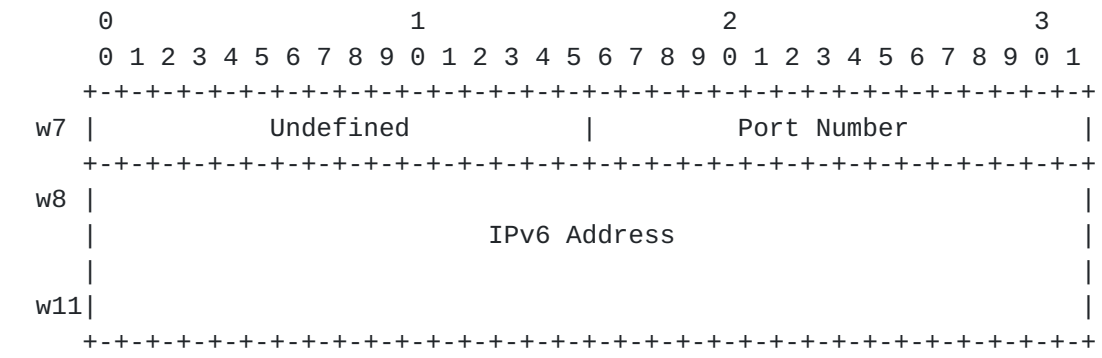

Figure 62: Socket Address format for peer address

Port Number and IP address are integers transferred in network byte order.

Get Name Table Group 1 command: 220

Get selected contents of the target node's naming table:
Command message:

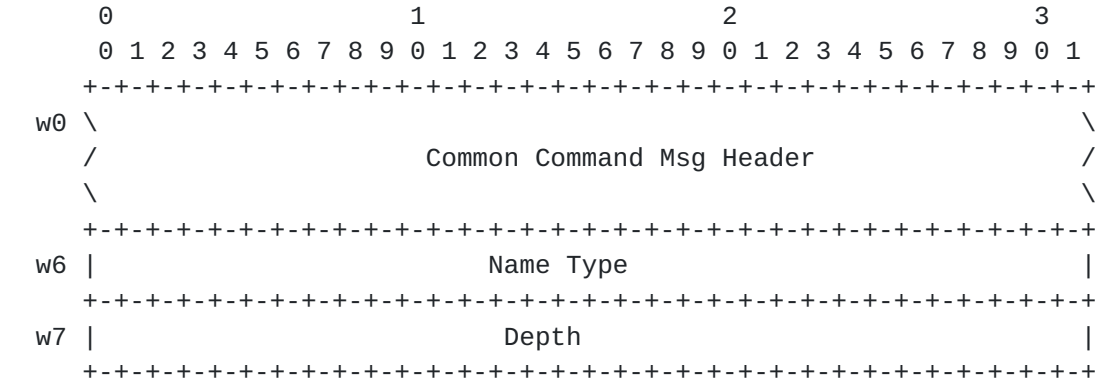

Figure 63: Get Name Table Query Command message format

Name Type is an integer indicating which name table entry is to be investigated. If this field is zero, all types in the table will be investigated and returned.
Depth is an integer indicating how much information is wanted about the requested entry or entries. It may have the following values:

Value	Description
0	All information about the entry, i.e. all known publications.
1	All known sequences for the requested name type.

2 Only the Name Type of the entry.

If both Name Type and Depth are 0 the whole table contents is returned. A depth value of 2 only makes sense with The Name Type value 0.
Result message:

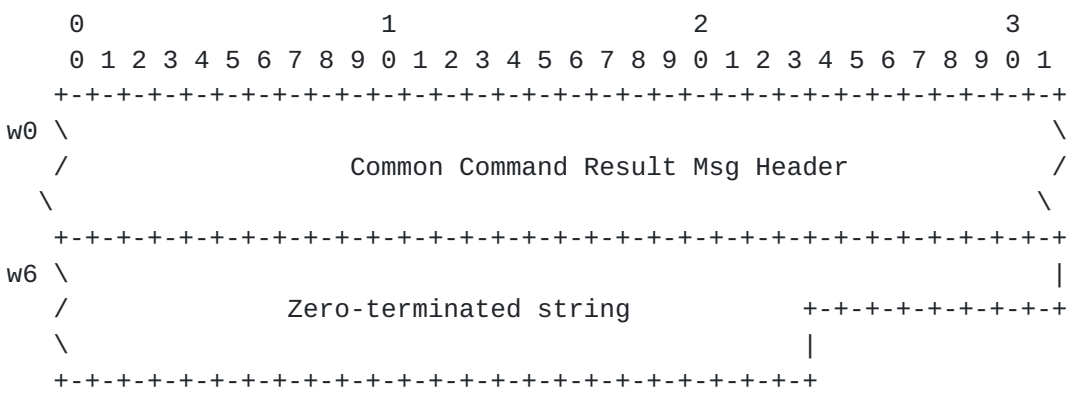

Figure 65: Get Name Table Query Result message format

Get Bearers Group 1 command: 190

Get information about all bearer instances found on the target node.
Command message:
Only a common command message header, with no arguments.
Result message:

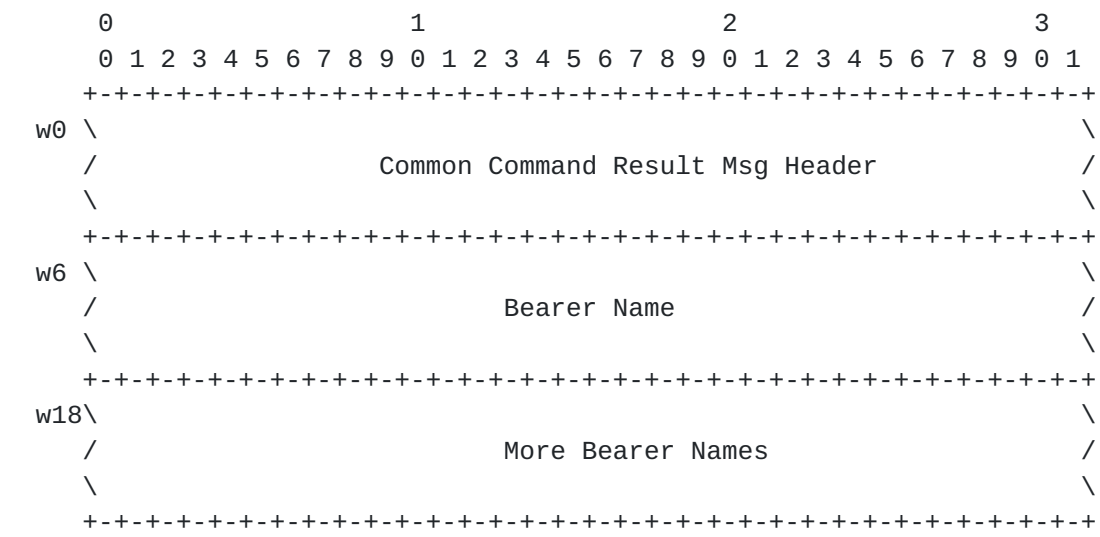

Figure 66: Get Bearers Query Result message format

Result data is a sequence of 48-byte sized structures, each consisting of a zero-terminated string, uniquely identifying each bearer.

Get Media Group 1 command: 194

Get information about all bearer types registered on the target node.
Command message:
Only a common command message header, with no arguments.
Result message:

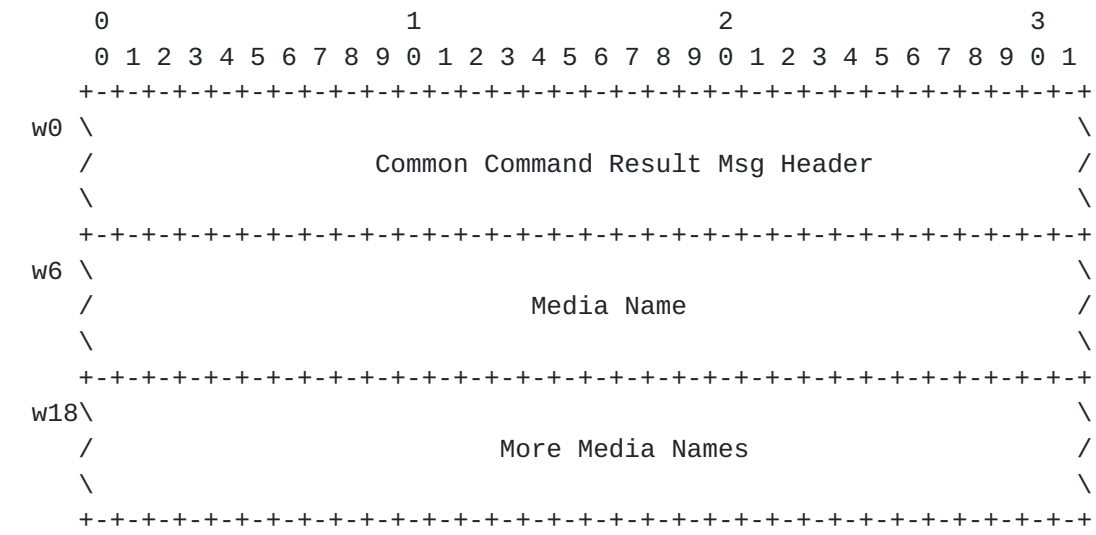

Figure 67: Get Media Query Result message format

Result data is a sequence of 48-byte sized structures, each consisting of a zero-terminated string, uniquely identifying each bearer.

Get Ports Group 1 command: 210

Get reference (random number part of identity) of all existing ports on the node.
Command message:
Only a common command message header, with no arguments.
Result message:

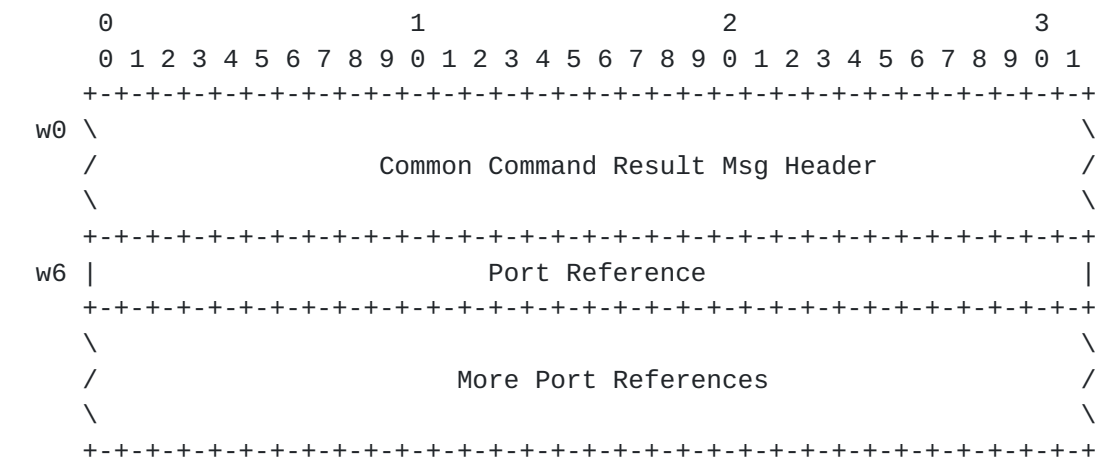

Figure 68: Get Ports Query Result message format

5.3.2 Group 2: Manipulating Commands

Establish Management Connection Group 1 command: 111

Command message:
Only a common command message header, with no arguments.
Result message:
A common command result message header, indicating success or failure.

Create Link Group 2 command: 180

Establish a link to the indicated domain.
Command message:

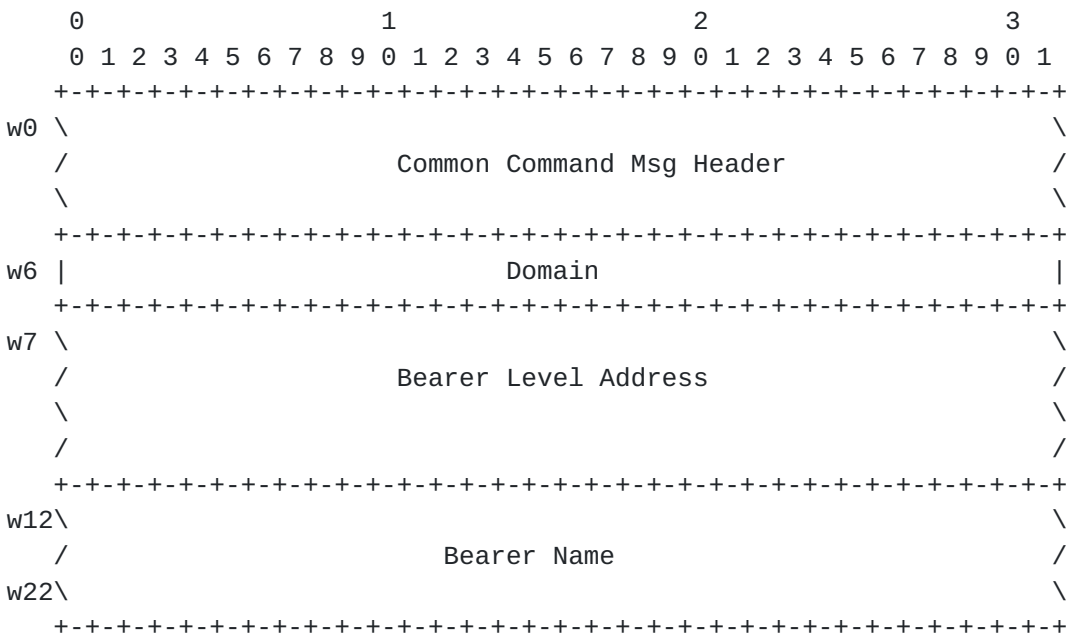

Figure 69: Create Link Command message format

Domain is a network address in one of the formats <0.0.0>, <Z.C.0> etc. Bearer Level Address is the address to be used for the establishment. Bearer Name indicates the bearer instance through which the establishment should be attempted.
Result message:
A common command result message header, indicating success or failure.

Remove Link Group 2 command: 181

Remove the link indicated by Link Name.
Command message:
Same as for Get Link Statistics.
Result message:
A Common Command Result Header, indicating success or failure.

Block Link Group 2 command: 185

Set the link indicated by Link Name in BLOCKED state.
Command message:
Same as for Get Link Statistics.
Result message:
A Common Command Result Header, indicating success or failure.

Unlock Link Group 2 command: 186

Set the blocked link indicated by Link Name in RESET_UNKNOWN state.
Command message:
Same as for Get Link Statistics.
Result message:
A Common Command Result Header, indicating success or failure.

Set Link Tolerance Group 2 command: 187

Set the link tolerance of the link indicated by Link Name.
Command message:

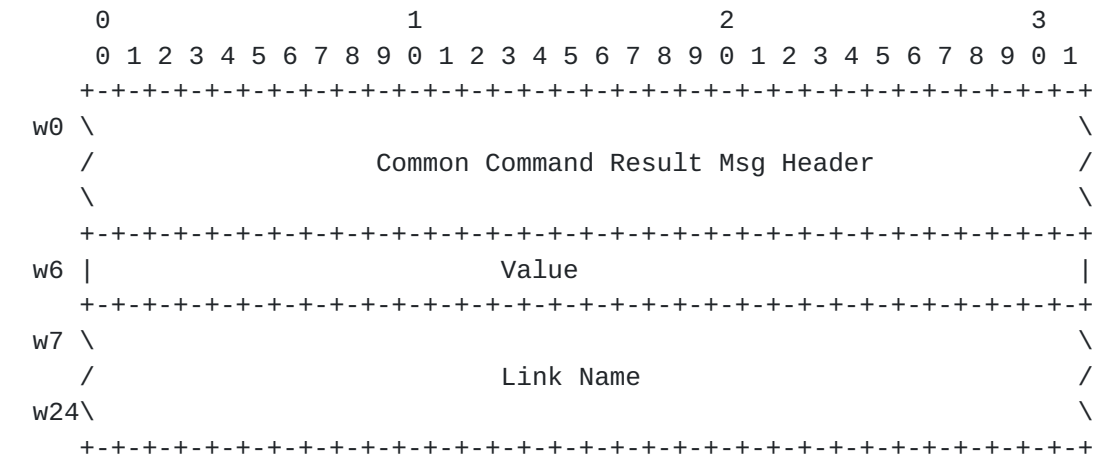

Figure 70: Set Link Tolerance Command message format

Result message:
A Common Command Result Header, indicating success or failure.

Set Link Priority Group 2 command: 188

Set the link priority of the link indicated by Link Name.
Command message:
Same as for Set Link Tolerance.
Result message:
A Common Command Result Header, indicating success or failure.

Set Link Window Group 2 command: 189

Set Send Window Limit for the link indicated by Link Name.
Command message:
Same as for Set Link Tolerance.
Result message:
A Common Command Result Header, indicating success or failure.

Enable Bearer Group 2 command: 191

Enable the bearer instance indicated by Bearer Name.
Command message:

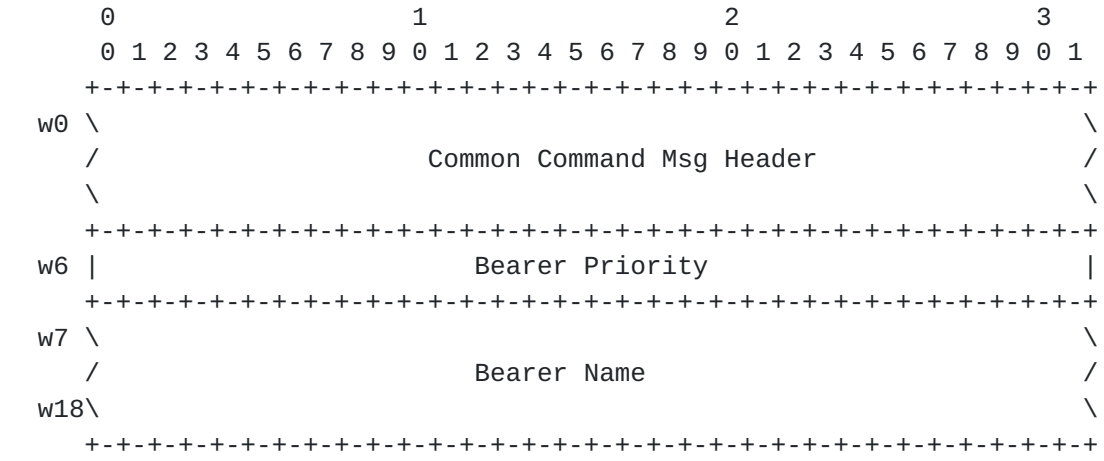

Figure 71: Enable Bearer Command message format

Bearer Priority is the default priority given to all links established over this bearer.

Result message:
A common command result message header, indicating success or failure.

Disable Bearer Group 2 command: 192

Disable the bearer instance indicated by Bearer Name.
Command message:

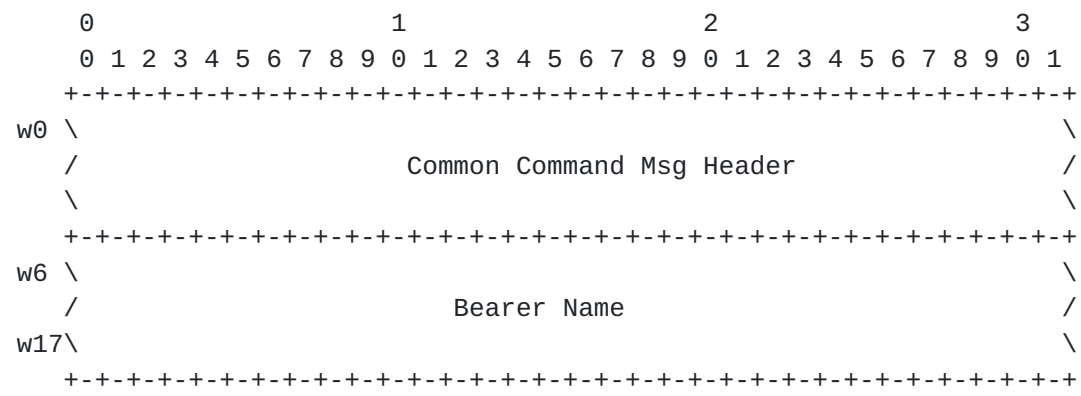

Figure 72: Disable Bearer Command message format

Result message:
A common command result message header, indicating success or failure.

5.3.3 Group 3: Subscriptions

Subscribe For Port Name Sequence Group 3 command: 174

Command message:

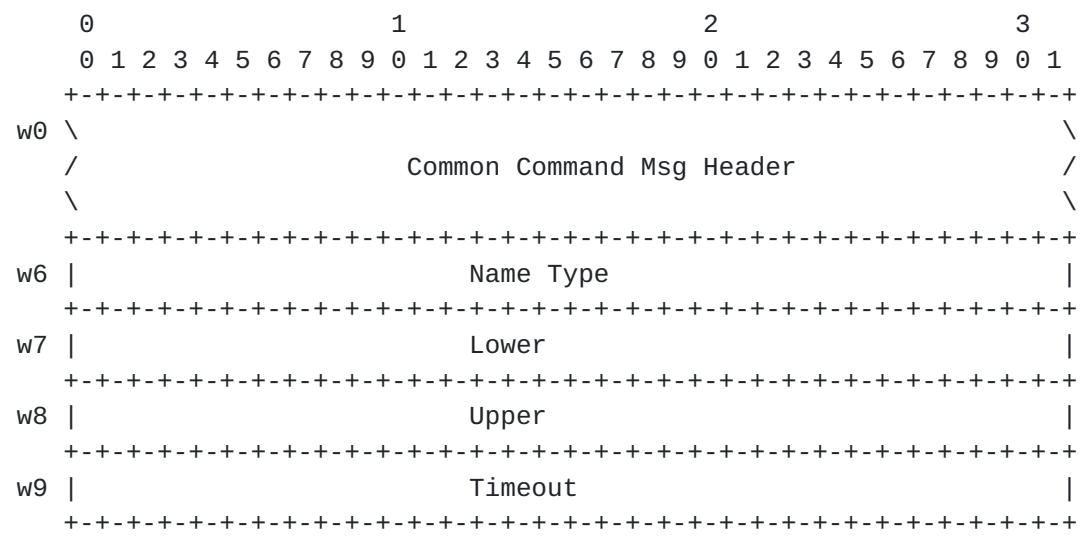

Figure 73: Subscribe for Port Name Sequence Command message format

All the four words in the command argument are integers. Their values and interpretation are the same as described in Section 3.2.5.

Result messages:

* A common command result message header, indicating success or failure. If successful, TIPC has established a connection towards the port which sent out the original command message.
* For each change in the naming table pertaing to the subcribed name sequence, a message with the following structure will be received:

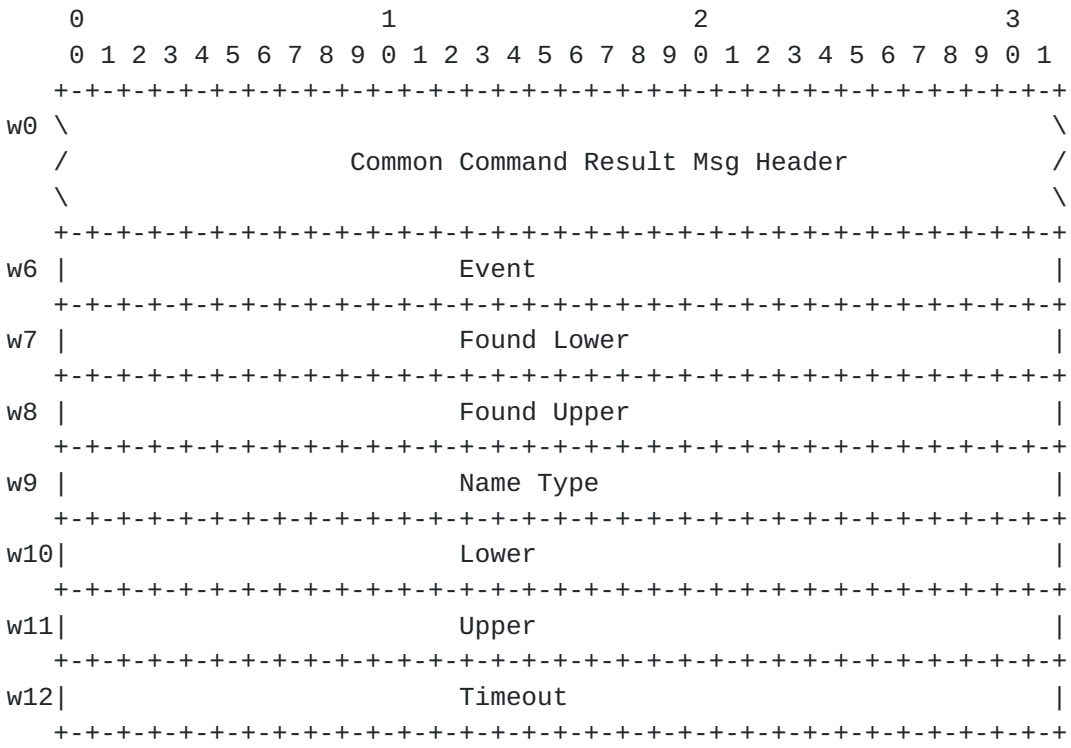

Figure 74: Subscribe for Port Name Sequence Result message format

Event may have the following values:

Hex Value	Description
0x800	A sequence overlapping with the requested range was published.
0x1000	No sequences overlapping with the requested range remain.
0x2000	The subscription exceeded the limit set in Timeout. The connection was shut down.

Found Lower and Found Upper indicate the overlapping part between the subscribed values and the actually published values. Name Type, Lower etc. are the same values as originally passed in the command message.

Subscribe For Link Group 3 command: 67

Subscribe for working state of the link indicated in Link Name
Command message:
Same as for Get Link Statistics
Result messages:

* A common command result message header, indicating success or failure. If successful, TIPC has established a connection towards the port which sent out the original command message.
* For each change in status of the concerned link, a message with the following structure will be received:

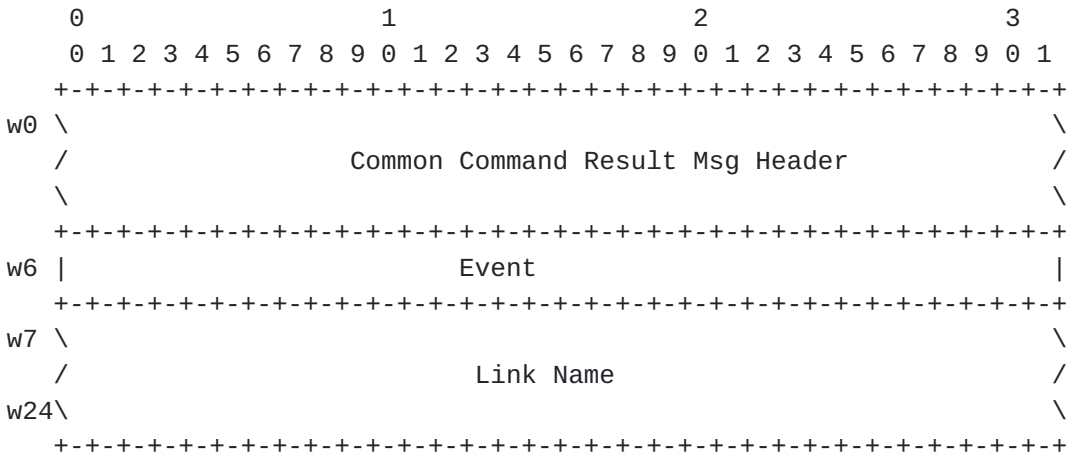

Figure 76: Subscribe for Link Result message format

Event may have the following values:

Hex Value	Description
0x800	The link went up to WORKING_WORKING state.
0x1000	The link went down to RESET_UNKNOWN state.
0x2000	The subscription exceeded the limit set in Timeout. The connection was shut down.

Link Name is the original link name, sent with the command message.

6. Security

TIPC is a special-purpose transport protocol designed for operation within a secure, closed network interconnecting nodes within a cluster. TIPC does not possess any native security features, and hence rely on the properites of the selected bearer protocol, e.g. IP-Sec, when such features are needed

7 References

[ForCES] Doria et al., A., "ForCES Protocol Specification", September 2004, <http://www.ietf.org/internet-drafts/draft-ietf-forces-protocol-00.txt>.

[RFC2026] Bradner, S., "The Internet Standards Process -- Revision 3", RFC 2026, BCP 9, October 1996, <http://www.rfc-editor.org/rfc/rfc2026.txt>.

[RFC2104] Krawczyk, H., Bellare, M. and R. Canetti, "HMAC: Keyed-Hashing for Message Authentication", RFC 2104, February 1997, <http://www.rfc-editor.org/rfc/rfc2104.txt>.

[RFC2119] Bradner, S., "Key words for use in RFCs to Indicate Requirement Levels", RFC 2119, March 1997.

[RFC2406] Kent, S. and R. Atkinson, "IP Encapsulating Security Payload (ESP)", RFC 2406, November 1998, <http://www.rfc-editor.org/rfc/rfc2406.txt>.

[RFC2408] Maughan, D., Schertler, M., Schneider, M. and J. Turner, "Internet Security Association and Key Management Protocol", RFC 2408, November 1998, <http://www.rfc-editor.org/rfc/rfc2408.txt>.

[RFC2434] Narten, T. and H. Alvestrand, "Guidelines for Writing an IANA Considerations Section in RFCs", RFC 2434, BCP 26, October 1998, <http://www.rfc-editor.org/rfc/rfc2434.txt>.

[RFC2460] Deering, S. and R. Hinden, "Internet Protocol, Version 6 (IPv6) Specification", RFC 2460, December 1998, <http://www.rfc-editor.org/rfc/rfc2460.txt>.

[RFC2581] Allman, M., Paxson, V. and W. Stevens, "TCP Congestion Control", RFC 2581, April 1999, <http://www.rfc-editor.org/rfc/rfc2581.txt>.

[RFC2960] Stewart, R., Xie, Q., Morneault, K., Sharp, C., Schwarzbauer, H., Taylor, T., Rytina, I., Kalla, M., Zhang, L. and V. Paxson, "Stream Control Transmission Protocol", RFC 2960, October 2000, <http://www.rfc-editor.org/rfc/rfc2960.txt>.

[RFC768] Postel, J., "User Datagram Protocol", RFC 768, STD 6, August 1980, <http://www.rfc-editor.org/rfc/rfc768.txt>.

[RFC793] Postel, J., "Transmission Control Protocol", RFC 793, STD 7, September 1981, <http://www.rfc-editor.org/rfc/rfc793.txt>.

[TIPC] Maloy, J., "Telecom Inter Process Communication", January 2003, <http://tipc.sourceforge.net>.

Authors' Addresses

Jon Paul Maloy
Ericsson
Research Canada
8400, boul. Decarie
Ville Mont-Royal, Quebec H4P 2N2
Canada

Phone: +1 514 576-2150
EMail: jon.maloy@ericsson.com

Steven Blake
Modularnet
Raleigh, NC 27606
USA

Phone:
EMail: steven.blake@modularnet.com

Maarten Koning
WindRiver
15983 Loyalist Campway
RR2
Bloomfield, ON KOK 1G0
Canada

Phone: +1 613 399-5669
EMail: maarten.koning@windriver.com

Jamal Hadi Salim
Znyx
195 Staford Road West,
Suite 104
Nepean, ON K2H 9C1
Canada

Phone: +1 613 596-1138
EMail: hadi@znyx.com

Hormuzd M. Khosravi
Intel
2111 NE 25th Avenue,
Hillsboro, OR 97124
USA

Phone: +1 503 264-0334
EMail: hormuzd.m.khosravi@intel.com

Intellectual Property Statement

Disclaimer of Validity

Copyright Statement

Acknowledgment

Funding for the RFC Editor function is currently provided by the Internet Society.