ToLHnet
Tree or Linear Hopping network

DII – Università Politecnica delle Marche

September 9, 2014

Contents

Chapter 1

Introduction

ToLHnet (which stands for "tree or linear hopping network") is a powerful yet simple networking protocol developed in order to support the creation of mixed networks, employing wired and wireless connections over different media among thousands of nodes. It is based on tree routing, with special care to support the degenerate case of linear routing, to keep implementation on nodes simple and protocol overhead low.

The protocol has been developed with the main goal of keeping both the complexity of the firmware on the nodes, and the overhead introduced by the network layer on the transmission, as low as possible. Because of this, the protocol has been designed so as to allow a strongly asymmetrical implementation, moving most of the complexity out of the standard nodes and into a single special node that will be the master controller of the network.

This master controller shall have larger computing and memory resources than those required by the other nodes (it will typically be implemented on a single-board-computer equipped with a standard operating system) and will take care of computing routing tables, assigning addresses, and configuring the network. It can also act as a gateway towards other networks.

The most important features of ToLHnet are:

- seamlessly support of wired or wireless media
- relatively large address space: about 60000 nodes
- low network overhead (typically 4 bytes, no need for MAC-layer addressing, payloads up to 240 bytes)
- low complexity of node firmware ($<$12 kB on one of the tested platforms)

1.1 Network Topology

The ToLHnet network can span multiple physical transmission media, either by means of wired (point-to-point or multidrop) or wireless connections. The network layer, as described in this document, abstracts from the details of actual physical transmission,

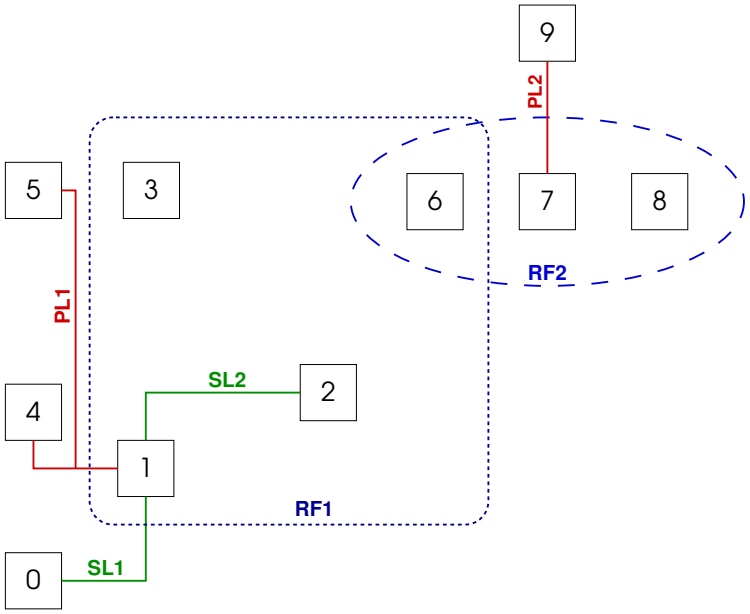

Figure 1.1: A simple example of node positions and connection highlighting the physical network topology.

relying only on basic services provided by the underlying layers, like the basic transmission or reception of data frames.

It's important to distinguish between the physical network topology, that can have arbitrary complexity, from the logical one used to route data packets, that is a tree topology with the master node at its root.

An example of physical topology is shown in Fig. 1.1, where several nodes are connected through different physical media: two point-to-point serial links ("SL1" and "SL2"), two separated power-line buses ("PL1" and "PL2") and two wireless connections ("RF1" and "RF2"), the latter being separated by operating on different frequencies or merely by physical factors.

We can define a "physical broadcast domain" (PBD) as a set of nodes that share the same transmission medium and that can all directly reach each other, e.g., a set of nodes connected to the same wired bus, or a set of wireless nodes close together enough so as to allow direct communication. Generally speaking, a node can belong to many PBDs, and different PBDs can overlap one another.

In the shown example, one can identify two wired PBDs ("PL1" and "PL2") and two wireless PBDs ("RF1" and "RF2"); these domains overlap in several ways, for example node 6 belongs to "RF1" and "RF2", while node 7 belongs to "RF2" and "PL2".

A node belonging to more than one PBD can act as a router and route packets to other connected nodes. As explained in this document, any node can be a router, without the need of a special configuration. An end-node is just a special case of a node that does not route data to any other node.

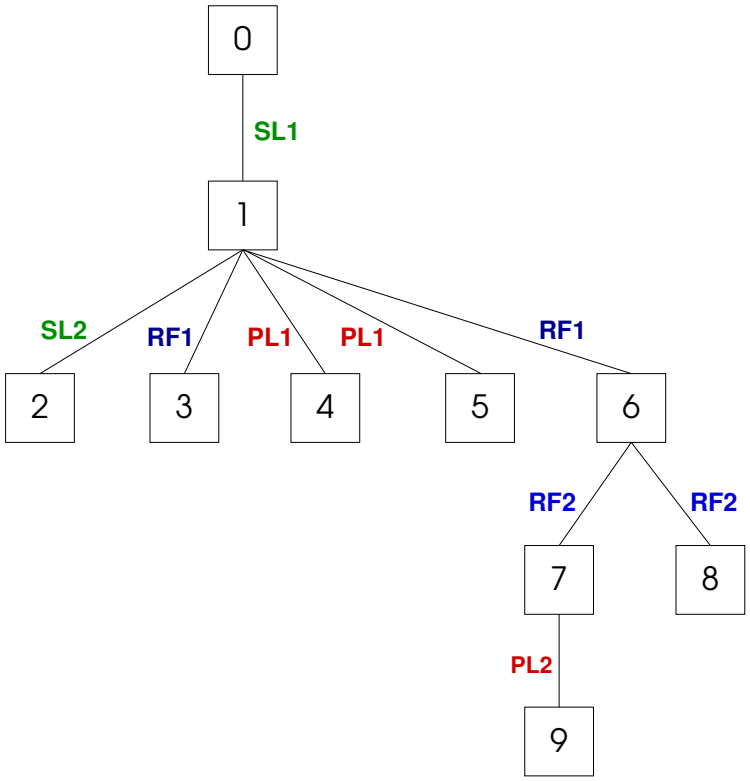

Figure 1.2: A simple example of a logical tree topology.

1.1.1 Logical Topology (Tree)

To simplify the protocol implementation on the nodes, ToLHnet uses a logical topology based on a tree, extracted from the physical one, where data packets travel across the tree branches only, intermediate nodes act as routers and only one packet is transmitted on the whole tree at a given time.

Building the logical tree starts from the physical topology; the latter can be formally described by an undirected graph, and weights can be associated to its edges to take into account the different capacity, delay, cost (energy consumption) and reliability (related to the distance) of each link. Using well-known algorithms (e.g. Dijkstra's), it is possible to extract a tree from this graph, optimizing for the relevant parameters, having its root at the node 0 (master node).

An example of such a tree, derived from the previous physical topology, is shown in Fig. 1.2.

As said, the master node (node 0) is typically a different device than the other nodes, with greater computational capabilities; in our example the master node is connected with a point-to-point connection to one of the other nodes (node 1), but this configuration is not mandatory.

The master node, besides processing data packets like the other nodes, must perform

additional tasks, like:

- compute the logical tree from the physical topology, or store a pre-computed tree structure
- build the routing tables of all the nodes, according to the tree structure
- configure the nodes during the network setup, assigning their addresses and routing tables
- manage the network errors: lost packets, timeout computation and management, connection loss between blocks of nodes, etc.
- interact with the user or other interface systems, and perform the more complex application's tasks

See Sec. 2.7 for details on some of the additional functions provided by the master node.

As explained later, every node receiving a packet, explicitly directed to it or to be routed to another node, decides, according to its routing rules, if it has to accept and process the packet or discard it. The latter case can be needed because, due to different broadcast domains (PBDs), whose reliability can change, a packet can be received through a link not belonging to the tree (side-edge reception), as a side-effect of a transmission directed to another node in the routing chain, even if the packet is actually destined to the node itself. In this case the packet must be ignored, to not generate collisions by trying to route it, or reply to it, while other nodes are transmitting as well. The node must instead wait until it receives the same packet at the right time and/or through the proper link.

This check is made possible by several protocol features:

- every node is assigned, during the initial configuration, a "depth" value, that is the number of needed hops to reach the node from the master node
- every data packet contains information about the number of hops performed up to a given moment, updated at every retransmission.

Each node can therefore compare the hop number of a packet, together with other information such as the packet direction, to its own configuration and decide if the packet is actually directed to it.

1.2 Network Layer Model

The ToLHnet protocol can be conceptually organized in the layers shown in Fig. 1.3, compared to the OSI protocol model.

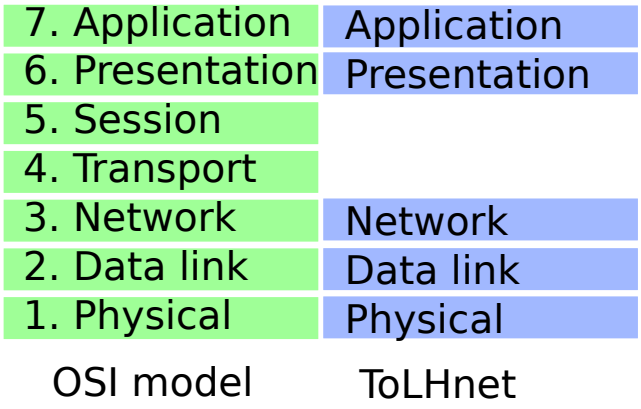

Figure 1.3: ToLHnet layers.

1.2.1 Physical and Data Link Layer

As said, the ToLHnet protocol is independent from the physical communication media, allowing the integration of different media in an uniform way. To this purpose, the functions that the data link layer must provide are minimal, including the transmission of data frames, notification of data reception and of possible errors.

Each specific implementation will therefore provide a custom version for this layer. Nonetheless, to ensure interoperability among nodes using the same physical transmission standard, this document provides a series of reference implementations for some wired or wireless communication channels. See appendix ... for details.

1.2.2 Network Layer

The network layer is the central part of the ToLHnet standard. As said, there are two versions of the network stack:

- standard version, available in all the nodes of a network, typically embedded systems with limited resources; this version performs the basic tasks of packet routing
- extended version, implemented in the master node, performing the tasks of the standard version together with the additional functions of the master (see Sec. 2.7)

See Sec. 2 for details.

1.2.3 Presentation Layer

This layer implements a higher-level interface with the network layer, interpreting a series of more user-friendly commands and translating them into one or more network communications.

This layer is implemented by the master node. See Sec. 3 for details.

1.2.4 Application Layer

The application layer provides the services typical of the specific application. As such, its implementation is independent from the ToLHnet standard.

Nonetheless, given that the information exchanged between the application layer and the lower ones is partly dependent on the network packet format, the ToLHnet standard defines a protocol for the application layer, logically organizing the application in a set of registers, readable and writable through dedicated commands, with side effects depending on the application. It also provides a mean to send messages, from a generic node, indicating an event not related to any request.

Following this implementation, an (even if limited) compatibility can be obtained among different applications using the ToLHnet standard. Future versions of this standard might also define a semantics for the registers of the application layers, and some means of "discovery" of the available services. That will allow the implementation of fully-compliant applications, ensuring a high level of interoperability with other systems, together with maximum flexibility for those systems.

See Sec. 4 for details.

Chapter 2

Network Layer

As said, the ToLHnet network is logically organized as a tree (see example Fig. 1.2). Data travel across the network in form of packets, described in this chapter, following the branches of the tree.

Typically (?) a communication only happens from the master node to a generic node, and from a node to the master node, the latter only as a reply to a request from the master.

Communication is unicast: a data packet is addressed to a single node identified by a unique address.

2.1 Addressing

There are two kind of addresses used in the ToLHnet protocol:

Hardware Address
: 48 bits long, uniquely and permanently identifies each node. Typically programmed during production on a non-volatile memory, or inferred from unique identifiers of the underlying hardware. In the text it's also referred to as "MAC address" in conformity with network terminology, even if in this case it has nothing to do with a MAC layer. This kind of address is used during the initial configuration only, in order to assign a shorter network address to the node.

Network Address
: 16 bits long, it's assigned by the master node to each other node during network configuration, and it's then used for all the following communications. The address 0 is reserved to the master node.

Each network packet (see Sec. 2.2) can include one or more of such addresses, among the source network address, the destination network address and the destination MAC address.

field	size (bits)	description
T	1	Target: 0 if the packet is to be processed by the network layer, 1 if it's a message for the application layer.
CODE	3	Type of command or message carried by the packet. Represents the semantics of the packet. See Sec. 2.6.
SEQ	4	Sequence number of the packet, used to discard duplicated packets. See Sec 2.5.
AM	2	Addressing mode. Specifies which addresses are present in the header. See Sec. 2.2.1 and Table 2.2.
D	1	Direction: 0 if the packet is directed towards the tree leaves, 1 if directed towards the tree root.
HOPS	5	Current routing depth of the packet, that is the distance, in number of hops, from the master node to the current node handling the packet[1]. This field is updated by every router when forwarding the packet (see Sec 2.5). This field can handle 31 depth levels; if it's saturated (11111), the optional field "HOPS_EXTENSION" is used instead, to extend the maximum depth[2] to about 65000.
HOPS_EXTENSION	16	Optional, present if the "HOPS" field is 11111.
SRC_ADDRESS	16	Optional (Sec. 2.2.1). Network address of the source node. If omitted it is 0 (packet coming from the master node).
DST_ADDRESS	16	Optional (Sec. 2.2.1). Network address of the destination node. If omitted it is 0 (packet directed to the master node).
MAC_ADDRESS	48	Optional (Sec. 2.2.1). Hardware address of the destination node, used during node configuration only.
DEPTH	16	Optional (Sec. 2.2.1). Routing depth of the destination node, used during node configuration only, together with "MAC_ADDRESS" field.
payload	var.	packet payload (0–240 bytes)

Table 2.1: Fields in network packet.

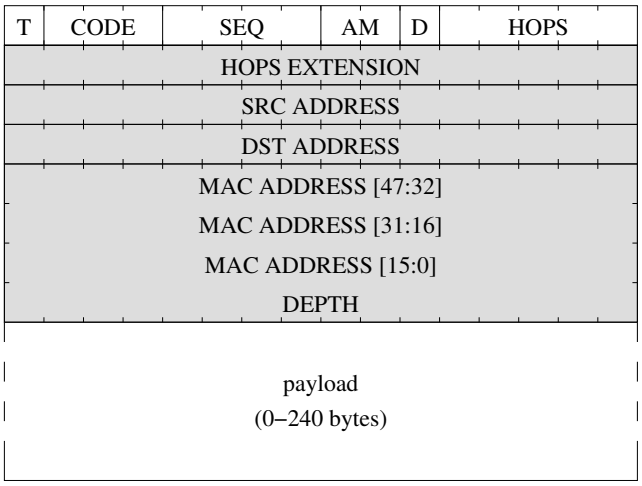

Figure 2.1: Network-level packet structure.

2.2 Network Packet

Fig. 2.1 shows the structure of the network packet. Gray fields are optional, their presence depends on the value of other fields.

Tab. 2.1 lists the meaning of each field in the network packet.

It can be seen that, in the common case in which one of the endpoints of the communication is the master node, only one network address need be specified, and if there are less than 32 hops in the tree, the header size is just 4 bytes.

2.2.1 Addressing modes

Tab. 2.2 shows the possible combinations of addresses included in a packet, according to the content of "AM" field:

SRC
: Source network address specified; destination = 0 (master node).

SRC + DST
: Source and destination network address specified; communication between two arbitrary nodes.

DST
: Destination network address specified; source = 0 (master node).

[1] In a network like the one depicted in the previous example (Fig. 1.2) where the master node is not part of the network itself, it is conventional to assign a depth of 0 to the first node attached to the master node, that is the node 1 in the example, and not to the master node itself. Similarly, the number of hops would be 0 (not 1) when reaching the node 1.

[2] This allows ToLHnet to scale well to the special case of a linear network, e.g. a long string of nodes where each node can transmit only to the previous and following one.

AM	fields present
00	SRC
01	SRC + DST
10	DST
11	DST + MAC/DEPTH

Table 2.2: Addressing mode field description. These two bits specifies which addresses are present in the header.

DST + MAC/DEPTH
Destination MAC address, network address and depth specified; used during initial configuration only.

2.3 Routing tables

All the nodes in the network possess a network address, a depth level in the tree, and a routing table, all of them assigned by the master node during the initial configuration. These parameters are used to decide how to process incoming packets.

When a node accepts a packet, it must find a matching rule in the routing table, as described below, stating what to do with the packet, whether accepting it as the packet's recipient or route it to another node, and in the latter case what's the interface to route the packet through.

The protocol is especially designed to keep the routing table as small as possible, often in the order of just one entry per physical interface.

There is no difference between an end-node (tree leave) and a router node, except that an end-node will have the smallest possible routing table, with one rule matching its own network address for packets to be accepted locally, and another catch-all rule for transmitting packets to its parent (see Sec. 2.4).

Each entry (rule) in the routing table contains a span of network addresses to match, a direction value (+1 for the routes towards leaves, -1 for the route towards the parent, 0 for the local node address) and an interface specification stating what transmission medium to use to reach the addresses specified by the span.

Routing rules are stored in each device in implementation-specific format. Hereafter they will be written in a symbolic format, closely related to their actual contents.

There are two kind of rules: mask-based rules and span-based rules. See the "CONFIG" command (Sec. 2.6.3) for their exact representation in the context of network configuration.

2.3.1 Mask-Based Rules

A mask-based routing rule is formatted as follows:

```
XXXX/L (MMMM) V II
```

The XXXX field identifies the network address span of destination, as explained below. This kind of rule uses a 16-bit mask (MMMM) having a certain number of most significant bits set to 1, and the remaining bits set to 0. The length field L (0–15) indicates the number of bits in the mask set to 1, so the mask is not explicitly assigned, but derived from L. For example, a length of 4 indicates a mask with value 0xF000.

In order to match such a rule, the actual destination address of a packet is first logically ANDed with this mask and then compared to the destination field XXXX.

This kind of rule allows to partition a network in "subnetworks", much like used, for example, in the IP protocol.

When searching for a matching rule, the table must be searched in descending order by the mask length, so that a rule with a longer mask will be matched preferentially.

Once a rule is matched, the II field indicates the transmission interface to which the packet must be routed. II must be lesser than 0x7F, so that it's usual to denote the interfaces with ASCII values of printable characters. Values can be assigned arbitrarily, but must consistently identify each interface through the whole network. The special value 0 is for the local interface, that is a packet to be accepted locally. The value 0x7F has special meaning (See the "CONFIG" command, Sec. 2.6.3). Other values are undefined.

Lastly, the V field is the hop number to be added to the packet's own hop number ("HOPS" field) before further routing: $+1$ for the routes towards leaves, -1 for the route towards the parent, 0 for local acceptation.

2.3.2 Span-Based Rules

A span-based routing rule is formatted as follows:

```
XXXX-YYYY V II
```

The XXXX and YYYY fields identify a network address interval to compare the packet destination address with (endpoints included). In case of a single address, the rule can be abbreviated as:

```
XXXX V II
```

The V and II fields have the same meaning as in the previous case.

In order to be sorted together with mask-based rules, to choose a preferential rule among several matching ones, provided that mask-based rules are searched in descending order by the mask length, for span-based rules a conventional value of 15 is assumed for the mask length when searching the table for a matching rule.

2.4 Network Configuration

During the initial network setup, the master node must configure all the network nodes by assigning them:

- their network address
- their depth in the tree
- their routing table, if any

The configuration is performed by sending "CONFIG" packets (see Sec. 2.6.3) to every node in the network.

Given that a node initially does not know such parameters, the master node must first address the nodes by mean of their MAC address, using network packets with addressing mode "DST + MAC/DEPTH". Other configuration commands can then be sent to the same node using the normal addressing (network address).

A node receiving a "CONFIG" packet with "DST + MAC/DEPTH" address, decides that it is the actual recipient of the command using the MAC address and the number of hops of the packet, both specified in the packet's header itself (see Sec. 2.5). Reception of such a packet means the start of a configuration process, and the node resets its network parameters as follows:

- stores the packet destination network address as its network address
- stores the packet "depth" value as its depth in the tree
- resets its routing table and populates it as directed by the records in the command's payload (if any).

"CONFIG" packets received subsequently with another addressing mode (using the network address and not the MAC address) indicate routing rules that the node must add, delete or modify, but without resetting the previous information. See the description of "CONFIG" command (Sec. 2.6.3) for details.

When a node resets its routing table upon receiving the first configuration command, in order to function properly the node must set two additional implicit rules to its table; such rules are not explicitly included in the configuration sent by the master node.

The first implicit rule is for packets to be accepted locally, and can be coded as follows:

```
XXXX 0 00
```

where `XXXX` is the network address of the node, as it's being assigned by the configuration command, and the value 00 for the interface indicates the local acceptation. This is the rule that will be matched by packets directed to the node itself, and will typically be the the only rule with null direction value.

The second implicit rule is for packets to be routed to the parent node, and can be coded as follows:

```
0000/0 (0000) -1 II
```

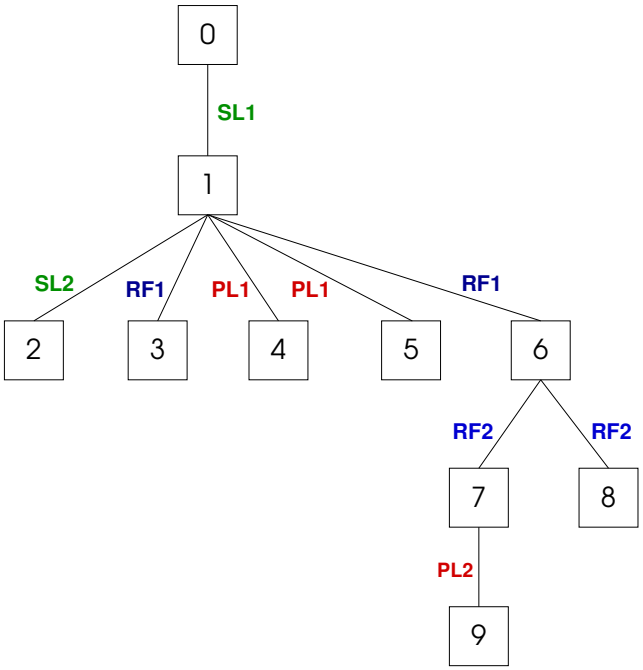

Figure 2.2: A simple example of a logical tree topology.

where `II` is the interface from which the configuration command has been received, identifying the connection with the parent node. This is a last-resort, catch-all rule (rules are searched in descending order of mask length), and it's matched for packets whose destination address is not known by the node as belonging to its sub-tree, and so the node does not know how to route them to destination. This can be a parent node or any node in another portion of the tree. The rule states that such a packet must be routed to the parent node, which in turn may have the destination node in its sub-tree or can decide to route it backwards again. Note that this rule also matches the case of a packet directed to the master node (address 0). Also note that typically this will be the only rule with negative direction.

It can be seen that the master node needs to configure the nodes with rules having "+1" direction only, given that the other two cases are covered by the implicit rules. Such positive-direction rules tell a node which are its child nodes, therefore stating the tree topology, and by which interfaces to reach them.

Given that a node can not route packets to its child nodes before being configured, the master node shall first send the "CONFIG" command to its direct child nodes, then to nodes at the subsequent depth layer, and so on.

The master node can dynamically compute the network configuration, and therefore the routing tables of the nodes, or use a pre-computed configuration (see Sec. 2.7).

As an example, the configuration that the master node would compute for the network in Fig. 1.2, reported here as Fig. 2.2, will be shown. The numbers associated to nodes indicate their network addresses. Let's assume for simplicity that the nodes'

MAC addresses only differ in their last hexadecimal digit, in its turn corresponding to the node's network address.

Conventionally, in a network like the one in the example, where the master node is point-to-point connected to another network node, the depth value 0 is assigned to the latter node (node 1 in the example), and not to the master. The hop number in packets will also reflect this, so that a packet will have a 0 hop number when reaching the node with 0 depth. This allows the saving of a depth level in the packet header, and the master can be reached anyway by mean of its special address 0. Care must be taken to not decrement the hop count of a packet if it's already 0.

Therefore, in the example network, node 1 has depth 0, nodes 2 through 6 have depth 1, nodes 7 and 8 have depth 2 and node 9 has depth 3.

A further prerequisite is the mapping of the several physical interfaces to a unique code ("`II`" field in routing rules). Let's choose the following:

'A' serial link SL1

'B' serial link SL2

'C' radio domain RF1

'D' power-line bus PL1

'E' radio domain RF2

'F' power-line bus PL2

With these assumptions, the configuration for the example network can be written as follows, where nodes are identified by their fictitious MAC address:

- node XXXXXXXXXXX1

  depth 0000
  network address 0001
  routing table

```
0002       +1 'B'
0003       +1 'C'
0004-0005  +1 'D'
0006       +1 'C'
```

- node XXXXXXXXXXX2

  depth 0001
  network address 0002
  routing table

—

- node XXXXXXXXXXX3

  depth 0001
  network address 0003
  routing table
  —

- node XXXXXXXXXXX4

  depth 0001
  network address 0004
  routing table
  —

- node XXXXXXXXXXX5

  depth 0001
  network address 0005
  routing table
  —

- node XXXXXXXXXXX6

  depth 0001
  network address 0006
  routing table
  0007-0008 +1 'E'

- node XXXXXXXXXXX7

  depth 0002
  network address 0007
  routing table
  0009 +1 'F'

- node XXXXXXXXXXX8

  depth 0002
  network address 0008
  routing table
  —

- node XXXXXXXXXXX9

  depth 0003
  network address 0009
  routing table
  —

2.5 Packet Processing and Routing

As said, a node receiving a packet on any of its configured interfaces decides, according to its own depth, to its routing table and to the packet header's fields, if it must accept the packet as its recipient, route it for another node, or discard it.

The algorithm used by the ToLHnet protocol and hereafter described, ensures that only one packet is transmitted across the network at a given time, and only through the actual tree branches, discarding side-edge-coming packets. It can be seen that this algorithm is intrinsically collision-free.

2.5.1 Sequence Numbers and Duplicated Packets

The sequence number of a packet is used to detect duplicated packet transmissions, like a node retransmitting the same command/message to the same node, possibly because the first reply to the source had been lost. The implementation must track the sequence number of the last seen incoming packet for each peer in the communication, according to the hardware resources available on each device. Every node can store the sequence number received within the last packet only, or by more than one source address if it has the resources to do so. If not, stored sequence numbers must time out after a while to free storage and allow communication with other peers to happen. In the typical scenario of a node that receives packets from the master node only, or that communicates with a limited number or peers, this task can be implemented with very little usage of hardware resources.

A node transmitting a packet that is not a reply to another packet must generate a suitable sequence number, i.e., a number different from the last one used in the communication between the same two nodes, with a more or less sophisticated algorithm

(the simplest one being using consecutive numbers, but care must be taken to prevent the possibility of a sequence number collision due to field overflow). Communications are typically originated by the master node, whose network layer can implement more optimized algorithms for this task (see Sec. 2.7).

In contrast, the sequence number of a packet sent as a reply to another packet must be the same as the one in the received packet.

The distinction between a newly generated packet and a reply to a previous command is trivial in the case of commands processed by the network layer (T=0). For transmissions originated upon request by the application layer, the network layer identifies the following two cases:

- if the code of the packet to be transmitted is "ACK" or "NACK", the packet is meant to be a reply to a command received by the application layer, and the network layer uses the same sequence number as the last received packet for the outgoing packet
- otherwise the packet is meant to be a new request, and the network layer generates a new sequence number for the outgoing packet

In addition, a node must store the last packet it sent (or more than one). In the case of packets destined to the application layer (T=1), when detecting a retransmission of the same command from the same source node:

- if a packet was previously sent to the same node with the same sequence number, thus as a reply from the application layer to the command being re-received, the node must reply with the same packet sent when the command was first received
- otherwise the packet did not generate a reply, so the mode must ignore it again

Both the two cases must happen without notifying the upper layer of the duplicates reception and/or reply. This is to avoid the undesirable multiple execution of a command at the application layer, with side effects not predictable by the network layer.

In contrast, commands for the network layer (T=0) must not be checked for duplicates, nor a reply must be re-sent automatically without executing the command. Commands directed to the network layer must be implemented so that the execution of the same command more than once does not have a different effect than a single execution. The rationale is to avoid potential network deadlocks due to nodes that do not execute configuration commands issued by the master because of a mismatch in the sequence number management between the two nodes.

2.5.2 Packet Processing Algorithm

The detailed algorithm for the processing of an incoming packet is conceptually described below. All the cases in which a packet is discarded are part of the strategy to ignore the packets received through side-edge channels, as already explained.

1. if the received packet contains the destination MAC address and the "depth" field, and the MAC address matches the node's MAC address:
   (a) if the hop number of the packet equals the depth attribute of the packet itself[3]: accept the packet locally (go to (10))
   (b) else discard the packet and stop
2. if the hop number of the packet does not equal the node's depth: discard the packet and stop
3. check the packet input path by finding the matching routing rule for the source address; this rule can have negative or positive direction, if the packet comes from the parent node or a child node, respectively. If the rule does not exist or its specified interface is different than the one from which the packet has been received, discard the packet and stop
4. check the packet output path by finding the matching routing rule for the destination address; if the rule does not exist discard the packet and stop
5. if the two matching rules (input and output) have both negative direction, discard the packet and stop. This is the case of a packet coming from the parent node but not directed to the node's sub-tree
6. if the input routing rule's direction and the packet's direction ("D" packet's field) are the same, discard the packet and stop. This is the case of a packet that would be retransmitted towards the same direction it came from
7. at this point the packet is considered accepted for processing by the node, to be further routed or accepted locally
8. if the packet has the "TRACE" code, possibly update the packet's payload with the node's tracing information (see Sec. 2.6.5)
9. if the interface specified by the output routing rule is 0 (local destination), accept the packet locally (go to (10)); else:
   (a) set the packet's direction ("D" field) to the direction specified by the output routing rule
   (b) if the packet has negative direction and its hop number is greater than 0, decrement its hop number
   (c) if the packet has positive direction, increment its hop number
   (d) transmit the packet through the interface specified by the output routing rule

[3] This kind of packet is used during the initial node configuration (see Sec. 2.4 and Sec. 2.6.3), when the node has not yet been assigned its own depth. The "depth" field of the packet indicates the depth in the tree to which the packet is destined, that will also become the node's depth as a result of the configuration command.

(e) stop

10. at this point the packet is considered locally accepted (destined to the node itself)
11. if the packet is destined to the application layer (T=1), and the sequence number is the same as that of a previous packet from the same node:
    (a) if a reply was already sent to the command and it's stored in memory, send the same reply again
    (b) stop
12. update the information relative to the sequence number of the received packet
13. if the packet is destined to the application layer (T=1), notify it to the upper layer; else process the packet at the network layer

2.6 Commands

The "CODE" field of a network packet (Sec. 2.2) identifies the command or message type associated to the packet. Depending on the "T" field in the header, different meaning is associated to codes in packets destined to the network layer (T=0) or to the application layer (T=1).

Tab. 2.3 lists all the defined commands. The *e*, *r* and *n* fields indicate fields that must be present at the beginning of the payload in packets that feature them, their meaning being:

- *e* is the error code in case of NACK
- *r* is the register number to be set or get
- *n* is the limit on the acceptable reply size

CODE	T=0	T=1
000	ACK	ACK
001	NACK *e*	NACK *e*
010	GET *r*	GET *r*
011	TRACE	GET *r n*
100	MSG	MSG
101	PING	MSG *n*
110	SET *r*	SET *r*
111	CONFIG	SET *r n*

Table 2.3: Command codes.

In the case of T=0, commands are processed at the network layer and are not notified to the upper layers.

In the case of T=1, packets are sent to the upper layer. Generally speaking, the application layer makes no assumption about the meaning of these codes, but the recommended implementation in this standard for the application layer (see Sec. 4) assigns a semantics to the command codes, where the user application can be stated in terms of a register machine, whose registers can be read and written (GET and SET commands), and may generate messages (MSG command) to alert other nodes of particular events.

This section explains the commands for the network layer, except the GET, SET and MSG commands, which are described at the application layer (Sec. 4); this is because their format and meaning are the same, only applied at a different layer. Registers and messages at the network layer are application-dependent and can be used to configure parameters pertaining the communication interfaces.

See Sec. 4 for the other application-layer commands (T=1).

2.6.1 ACK Code

Description

Sent in reply to commands that mandate it, to indicate a successful result.

Payload

Optional, application-specific. Null if not specified.

2.6.2 NACK Code

Description

Sent in reply to commands that mandate it, to indicate an error in the processing of the previous request.

Payload

- Error code (1 byte)
- Optional error-specific additional information

2.6.3 CONFIG Code

Description

Generated by the master node only, to configure nodes during setup of the network. The CONFIG command can use network packets with "DST + MAC/DEPTH" addressing mode, other than the other addressing modes. See Sec. 2.4 for details about the configuration process.

Payload

A sequence of 0 or more records, ad shown in Tab. 2.4, representing the entries to be added, deleted or modified in the routing table of the recipient node.

Interface	Length	Span	Network Address
1 byte	1 byte	2 byte	2 byte

Table 2.4: Records composing the payload of the CONFIG command.

For each entry, the rule type is inferred from the "Length" field. If it's greater than or equal to 16, the rule is span-based, otherwise it is mask-based.

For mask-based rules, the "Network Address" and "Length" fields represent the `XXXX` and `L` parts of the rule, and the "Span" field is not present.

For span-based rules, the "Network Address" field is the starting address (`XXXX`), and the span extent (`YYYY` − `XXXX`) is computed as follows:

- if "Length" is lesser than 255, the extent is Lenght − 16, and the "Span" field is not present
- otherwise the extent is given in the "Span" field

The "Interface" field is the `II` field in rule notation, but it can assume special meanings as explained below.

Before further processing, the routing table is searched for rules equivalent to the one being examined, and if one or more of such rules exist, they are removed from the table; "equivalent" has different meanings according to the rule type:

- for mask-based rules, an old rule is equivalent to the new one if the address of the old rule, logically ANDed with the mask implied by the new rule, equals the address of the new rule (that is, the subnetwork denoted by the old rule is a subset of the one in the new rule)
- for span-based rules, an old rule is equivalent to the new one if the address interval in the old rule is a subset (or the same set) of the new rule

After that, if the "Interface" value is lesser than 0x7F, the new rule is added to the table, otherwise if it is 0x7F nothing is done, giving the value 0x7F (DEL character in ASCII) the special role of deleting a rule. Other values are undefined.

It can be seen that, according to the case, a rule can effectively be added, modified or deleted with proper "CONFIG" commands.

Note that the direction of rules expressed in the "CONFIG" command is always assumed +1, except when the interface is 0 (not normally used), in which case the direction is also 0.

Reply

ACK or NACK.

2.6.4 PING Code

Description

Used to test the network connectivity.

Payload

A sequence of 0–64 arbitrary bytes.

Reply

ACK, with the same payload as the request.

2.6.5 TRACE Code

Description

Generated by the master node only, to test the signal strength and quality of the physical links forming the network.

A node accepting a packet with TRACE command, either for routing or as the final recipient, checks that the node depth is greater than or equal to the minimum one indicated in the payload, and lesser or equal to the limit imposed by the packet size.

If this is the case, the node updates the entry in the payload corresponding to its depth and to the packet direction with the quality data regarding the reception of the packet itself. Note that the last node receiving the packet can update the first half of the record only, corresponding to the "+1" direction.

Whether the node updates or not the payload, it continues the normal processing, routing the packet to another node or accepting it locally and replying.

This command is therefore special, because a node parses the command, and possibly modifies its payload, even if it's not the actual recipient of the packet.

Payload

A 2-byte header, followed by one or more 8-byte records, as shown in Tab. 2.5, where the last record has only the first 4 bytes.

	direction = +1				direction = -1			
start depth	attempts	delay	stren.	qual.	attempts	delay	stren.	qual.
16 bit	1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	1 byte	1 byte

Table 2.5: Header and records composing the payload of the TRACE command.

TODO: document the 4 parameters.

Reply

TRACE (with updated payload) or NACK

2.7 Extended Network Layer (Master Node)

This section lists the additional functions that the extended network layer, typically running on the master node only, must perform for the correct implementation of the network.

2.7.1 Network Formation

If the tree structure of the network, together with the routing tables of every node, is not fixed and precomputed, the master node must provide the proper algorithms for the formation of the network.

In order to do this, the master node needs several input data describing the features and collocation of every node and the nature of every physical medium involved in the communication.

First, every node must be represented with the following (non-exhaustive) list of parameters:

- MAC address
- list of physical interfaces available for communication
- where meaningful (e.g. wireless interfaces), the geographical position of the node for every interface
- optionally, a user-defined name for the node, to be used in higher-level commands

Then every physical interface, coherently with those listed in the previous list, must be defined in terms of the following list of parameters, whose meaning can be different for the various types of interfaces, due to the different nature of the transmission media:

- the "cost" of every hop of the interface, depending on the parameters that the application needs to optimize more
- the "range" of a given interface, that is a measure of how much the total cost of a transmission increases with the distance between the nodes; this can be given in number of hops or in actual distance
- the timing information of data transmission through the interface, in order to compute the timeouts in case of packet loss

The information on the physical interfaces allows to define a cost function for the transmission of data through a given interface; this can be used by well-known algorithms (e.g. Dijkstra's), to extract an optimal tree from the graph describing the physical topology. The graph itself is described by the previously listed information on the nodes and the interfaces.

2.7.2 Packet Loss

The network must be robust to the likely case of a packet lost or corrupted along one of the several links needed to reach the recipient node from the source one. This is easily detected when sending a command that mandates a reply. In this case, both the command or the reply can be lost.

The standard implementation of the network layer already manages the case of a duplicated command request (see Sec. 2.5), in a way dependent on the command's destination layer.

On the other hand, the master node, which typically is the originator of a request or command, must handle the case of a reply not received, by defining a timeout interval after which the packet can be considered lost. The timeout duration must be computed on the basis of several parameters:

- the timing information of all the links the command and the reply packets must travel, according to the routing for the destination node and the physical interface characterization previously explained
- the time needed by the destination node to process the request, depending on the application
- additional possible latencies, depending on the implementation

After a reply has not been received in the given time, the master node must try to retransmit the packet, up to a desirable number of times, before reporting an error to the application.

2.7.3 Sequence Number Generation

As said (see Sec. 2.5), every new packet transmitted must be assigned a suitable sequence number to distinguish it from the previously transmitted packets.

Given that the possible different numbers are limited, and that a duplicated sequence number can lead to a node mistakenly interpreting the command as a duplicated one and generating an incorrect reply[4], care must be taken to generate those numbers effectively.

This is specially true for the master node, that generates packets destined to every other node in the network. In this case the master node must track the sequence numbers it used for (virtually) every destination address, to avoid the risk of generating the same number for two consecutive packets to the same node, even after having generated different numbers for the other transmissions in between.

Given that there can be several thousands nodes in the network, the master node must necessarily implement optimized techniques to store this kind of information, like remembering a limited number of the last packets, and freeing their memory when safe.

Generating the sequence number in a proper sequence can also be done with optimized algorithms, in order to minimize the cost of storing the needed information.

[4] As previously seen, this behavior is implemented for commands pertaining to the application layer only.

Chapter 3

Presentation Layer

Chapter 4

Application Layer

This chapter describes a recommended semantics for the commands of the application layer, that can ensure a better interoperability between different systems, while keeping the maximum flexibility for the implementation of the specific applications.

Tab. 4.1 lists the command codes associated to packets ("CODE" field), already described in Sec. 2.6, but here listing only the commands directed to the application layer (T=1). As for the other cases, the *e*, *r* and *n* fields indicate fields that must be present at the beginning of the payload in packets that feature them, their meaning being:

- *e* is the error code in case of NACK
- *r* is the register number to be set or get
- *n* is the limit on the acceptable reply size

CODE	T=0	T=1
000		ACK
001		NACK *e*
010		GET *r*
011		GET *r n*
100		MSG
101		MSG *n*
110		SET *r*
111		SET *r n*

Table 4.1: Command codes (application layer only).

The reference implementation organizes the user application in terms of a register machine, whose registers can be read and written (GET and SET commands), and may

bits	bytes	format							
7	1	0**bbbbbbb**							
14	2	10**bbbbbb**	**NN**						
21	3	110**bbbbb**	**NN**	**NN**					
28	4	1110**bbbb**	**NN**	**NN**	**NN**				
35	5	11110**bbb**	**NN**	**NN**	**NN**	**NN**			
42	6	111110**bb**	**NN**	**NN**	**NN**	**NN**	**NN**		
49	7	1111110**b**	**NN**	**NN**	**NN**	**NN**	**NN**	**NN**	
56	8	11111110	**NN**	**NN**	**NN**	**NN**	**NN**	**NN**	**NN**
64	9	11111111	**NN**	**NN**	**NN**	**NN**	**NN**	**NN**	**NN**	**NN**

Figure 4.1: Variable-length binary coding of register addresses, showing the available address bits and the needed coding bytes for every case. Boldface text represents the register address.

generate messages (MSG command) to alert other nodes of particular events. MSG commands can be generated by any node, and directed to any other node, in asynchronous way, without explicit request from the master node.

The side-effects of reading or writing a register, or receiving a message, are entirely application-dependent and represent how the specific user application works.

For the description of the ACK and NACK commands, see the corresponding sections in the chapter about the network layer (Sec. 2.6).

4.1 Register Address Format

The address of the register to be read or written (r field in the payload of some of the commands) is represented using a custom integer compressive coding, with a variable number of bytes (1–9), allowing the coding of up to 2^{64} addresses, but keeping a more compact notation in case of addresses in a narrower range.

In the simplest case of addresses in the range 0–127, the address is represented naturally by a single byte; this case is denoted by the most significant bit of the byte set to 0.

Higher addresses are coded by appending up to 8 additional bytes to the first one, whose quantity is indicated by the number of most significant bits set to 1 in the first byte. In this case the first byte may or may not contain some of the most significant bits of the address, separated by a bit 0 from the bits set to 1 in the most significant places.

Fig. 4.1 illustrates the binary coding. Bits indicated with 'b', or hexadecimal digits indicated with 'N', compose the register address.

4.2 Commands

4.2.1 GET Code

Description

Reads a register. Available in the two versions "GET *r*" and "GET *r n*".

Payload

- *r*: register address (1–9 bytes)
- (second version only) *n* (1 byte): limit on acceptable reply size

Reply

ACK with payload containing the reading result, or NACK.

4.2.2 SET Code

Description

Writes a register. Available in the two versions "SET *r*" and "SET *r n*".

Payload

- *r*: register address (1–9 bytes)
- (second version only) *n* (1 byte): limit on acceptable reply size
- optional: data to be written to register

Reply

ACK with application-dependent payload, or NACK.

4.2.3 MSG Code

Description

Notifies an event. Can be sent from/to any node, without explicit request from the master node. Available in the two versions "MSG" and "MSG *n*".

Payload

- (second version only) *n* (1 byte): limit on acceptable reply size
- optional: data associated to event

Reply

ACK with application-dependent payload, or NACK.