1 The Master and Dog Paradigm

Topology discovery is a distributed algorithm that is at the core of several routing algorithms, such as link-state routing. It is the problem of each node in a network discovering and maintaining information on the network topology. The problem is challenging as the network can change rapidly, indeed more rapidly than the nodes can track and account for the changes.

The topology discovery algorithm we develop generalizes of the "master and dog" paradigm. Here is the story. A master rides a motorbike while his dog, running behind him, tries to get to him. If the master reduces the speed of the motorbike, then the dog come a bit closer to him. However, if the master accelerates, then the distance between the two increases. But, certainly enough, if the master stops for a sufficiently long period of time then the dog will reach eventually his master.

In our problem, the master is a graph representing the communication structure of a network. A move of the master corresponds to a modification of this graph. What makes our case different from the basic paradigm is that we have many dogs, each of them being a node in the graph. The distance between each dog and the master is represented by the difference between the physical graph and the image of it that each node constructs as soon as it gets some information about the "position" of the master. By definition, we say that a dog-node has reached the master-graph when the local image of the graph in this node corresponds exactly to the real situation of the graph.

In the case of the simple "master and dog" story, the dog reacts to the moves of the master because he can see the master. In our case, the situation is a bit more complicated. Only certain nodes can "see" certain moves of the graph. In other words, the nodes as a community get all information about the moves of the graph, but each individual node has only a partial direct access to these moves. More precisely, a node \( n \) gets direct information about the modification of a link \( l \) of the physical network if \( n \) is the arriving node of link \( l \).

Besides the limited information that it can get directly on the graph, each node can acquire more information about the graph from its neighbors (note that these neighbors changes over times as the graph evolves), which themselves either get some direct information on the graph or more information from their neighbors, and so on.

We would like to prove a result analogous to the one we have between the master and the dog, namely that the dog can reach his master if the latter stays still for a while. In our case, we shall prove that, under certain conditions, all dog-nodes can reach the master-graph if the latter stays still long enough.

2 Requirements for the Topology Discovery Problem

In this section, we define the requirements of our problem with great care. Note that we do not state any solution, rather we express the constraints that must be guaranteed by a potential solution for it to be considered correct. In the formal development of our solution in subsequent sections, we shall record the place where we take accounts (totally or partially) of these requirements.

We are given a finite set of nodes connected by oriented links which can be added or removed as time goes, thus forming a dynamic graph.

**REQ-1**
Each node builds an image of the graph either by getting direct information from it (cf. REQ-3) or indirect information from its neighbors (cf. REQ-4).

When a link from node $a$ to node $b$ is added to or removed from the graph then node $b$ is directly made aware of it.

The neighbors of a node $b$ are the nodes that are connected to $b$ by a link entering in $b$. Neighbors of $b$ send to $b$ their local networks.

Nodes communicate by means of messages sent on links of the graph.

A message sent from $a$ to $b$ is lost if the link from $a$ to $b$, which existed when the message was sent, is broken before the message reaches $b$.

We must prove that under certain conditions the images of the graph built by nodes are all identical and equal to the graph itself.

### 3 Development Strategy

Our development comprises an initial model followed by seven refinements. Here is a brief description of each of them:

- The initial model only takes care of the first requirement defining the physical network.
- In refinement 1, we introduce the logical network. It corresponds to the idea that the physical network is transmitted to all nodes as a community.
- In refinement 2, we introduce explicitly the local networks of each node. This is done together with a very abstract way of gradually transmitting the contents of the logical network to the local networks.
- In refinement 3, we introduce modification times of the logical network as well as modification times of the local networks. This allows us to simplify the formalization.
- Refinement 4 is technical: it concerns the merging of various events.
- In refinement 5, we formally define the neighbors of a node and thus take full account of the fourth requirement.
- In refinement 6, we prove our main result, corresponding to the seventh requirement.
- In the last refinement, we take account of the message handling, corresponding to requirements five and six.
4 Formal Development

4.1 Initial Machine

This initial machine takes account of requirement REQ-1.

We are given a finite set of nodes connected by oriented links that can be added or removed as time goes, thus forming a dynamic graph

Communications channels between nodes of the physical network form a graph, which can be represented by a finite binary relation. At this stage however, we shall not formalize our graph by means of a binary relation. In fact, it is more convenient to represent the graph by a subset of the set $L$ of all possible links between two different nodes:

sets: $L$  

axm0_1: finite($L$)

The physical network $NET$ is thus a variable defined as a simple subset of $L$:

variables: $NET$  

inv0_1: $NET \subseteq L$

At this stage, we have two events: adding or removing a link from the physical $NET$. This corresponds to the dynamic evolution of the connections between the nodes. Our topology discovery algorithm has no influence on these external evolutions.

4.2 First Refinement

In this refinement, we take account of requirements REQ-2 and REQ-3 in a very abstract way. More will be said on these requirements in the next refinement.

Each node builds an image of the graph either by getting direct information from it (cf. REQ-3) or indirect information from its neighbors (cf. REQ-4)
When a link from node $a$ to node $b$ is added to or removed from the graph then node $b$ is directly made aware of it

To this end, we introduce the *logical network* $\text{net}$. It represents the abstract knowledge that the nodes as a community may obtain "somehow" on the overall situation of the physical network. The variable $\text{net}$ is thus a subset of $L$ similar to $\text{NET}$. Notice again that the logical network $\text{net}$ is a convenient abstraction which will be given later a concrete definition in refinement 5.

**variables:** $\text{net}$  
**inv1:** $\text{net} \subseteq L$

Besides the two previous events, we formalize two more events adding or removing a link from the logical $\text{net}$ following the evolution of $\text{NET}$.

\[
\text{discover_up} \quad \text{status} \quad \text{convergent}
\]

\[
\text{any} \quad l
\]

\[
\text{where} \quad l \in \text{NET} \setminus \text{net}
\]

\[
\text{then} \quad \text{net} := \text{net} \cup \{l\}
\]

\[
\text{end}
\]

\[
\text{discover_dn} \quad \text{status} \quad \text{convergent}
\]

\[
\text{any} \quad l
\]

\[
\text{where} \quad l \in \text{net} \setminus \text{NET}
\]

\[
\text{then} \quad \text{net} := \text{net} \setminus \{l\}
\]

\[
\text{end}
\]

Note that these events do not make any node aware of the link that is added to or removed from the graph as is mentioned explicitly in requirement \text{REQ-3}. This will be done in the next refinement. For the moment, they only update the logical network $\text{net}$. These events are convergent: this is proved by means of the following variant denoting the symmetric difference between the sets $\text{NET}$ and $\text{net}$:

\[
\text{variant1: } (\text{NET} \setminus \text{net}) \cup (\text{net} \setminus \text{NET})
\]

This convergence means that if the physical network $\text{NET}$ is not modified for a certain time then the logical network $\text{net}$ will eventually be the same as the physical network $\text{NET}$ (master and dog).

### 4.3 Second Refinement

In this refinement, we take further account of requirement \text{REQ-3}: an arbitrary node is made aware of the change when a modification in the graph has occurred. Moreover, we also take further account of requirement \text{REQ-2} dealing with the image of the graph that is built by each node.

To this end, we define the *local* networks. Each of them represents the individual knowledge that each node acquires about the physical network. It corresponds to the gradual spreading of the logical network $\text{net}$ to each node of the physical network. In order to formalize this, we introduce first the finite set of nodes $N$, which we do not yet relate to the links $L$. 
Here is the definition of the local networks $l_{\text{net}}$: it is a binary relation from $N$ to $L$. More precisely, the set of links associated with node $n$ is the image $l_{\text{net}}[\{n\}]$ of the singleton set $\{n\}$ under the relation $l_{\text{net}}$.

When a link modification is detected by the events discover_up or discover_dn, then some messages to all nodes (except the node that is directly made aware of the modification) are put in two big reservoirs called $m_{\text{net}_{\text{up}}}$ (for establishing a new link) and $m_{\text{net}_{\text{dn}}}$ (for removing a link). Notice that the message handling done in these reservoirs does not correspond at all to the final message handling that we have in the real algorithm (presented in refinement 7). This is only a convenient abstraction which we use for the moment.

Moreover at this stage, a modification of one reservoir is done "magically" by removing old messages that can be still waiting in the other (see the updates of the events discover_up and discover_dn below). As a consequence, $m_{\text{net}_{\text{up}}}$ and $m_{\text{net}_{\text{dn}}}$ are disjoint:

> $m_{\text{net}_{\text{up}}} \cap m_{\text{net}_{\text{dn}}} = \emptyset$

Besides the event init, the events discover_up and discover_dn are updated as follows:

As can be seen, a node $n$ is chosen (arbitrarily for the moment) and the local network in that node is updated. This arbitrary node will be made precise in refinement 5. A similar behavior can be observed for the next event:
In this refinement, which is still very abstract, nodes have free access to the reservoirs \( m_{\text{net\_up}} \) and \( m_{\text{net\_dn}} \). Thus they can update their local knowledge. This is done by two additional events named change_link_up and change_link_dn. Notice that these events have the status anticipated, meaning that their convergence must be proved in further refinements.

What has been presented so far is illustrated in figure 1.

<table>
<thead>
<tr>
<th>change_link_up</th>
<th>change_link_dn</th>
</tr>
</thead>
<tbody>
<tr>
<td>\textbf{status}</td>
<td>\textbf{status}</td>
</tr>
<tr>
<td>anticipated</td>
<td>anticipated</td>
</tr>
<tr>
<td>\textbf{any}</td>
<td>\textbf{any}</td>
</tr>
<tr>
<td>( n )</td>
<td>( n )</td>
</tr>
<tr>
<td>( l )</td>
<td>( l )</td>
</tr>
<tr>
<td>\textbf{where}</td>
<td>\textbf{where}</td>
</tr>
<tr>
<td>( n \mapsto l \in m_{\text{net_up}} )</td>
<td>( n \mapsto l \in m_{\text{net_dn}} )</td>
</tr>
<tr>
<td>\textbf{then}</td>
<td>\textbf{then}</td>
</tr>
<tr>
<td>( l_{\text{net}} := l_{\text{net}} \cup {n \mapsto l} )</td>
<td>( l_{\text{net}} := l_{\text{net}} \setminus {n \mapsto l} )</td>
</tr>
<tr>
<td>( m_{\text{net_up}} := m_{\text{net_up}} \cup (N \setminus {n}) \times {l} )</td>
<td>( m_{\text{net_dn}} := m_{\text{net_dn}} \setminus (N \times {l}) )</td>
</tr>
<tr>
<td>\textbf{end}</td>
<td>\textbf{end}</td>
</tr>
</tbody>
</table>

![Fig. 1. Handling Abstract Messages](image_url)

All four events of this refinement maintain the following invariants between \( \text{net}, l_{\text{net}}, m_{\text{net\_up}} \) and \( m_{\text{net\_dn}} \):
Figure 2 gives an informal explanation of refinement inv2_5. When a link \( l \) is present in \( \text{net} \) and absent from \( \ell_{\text{net}} \) for node \( n \), then it must be present in the message reservoir \( m_{\text{net}_\text{up}} \) for node \( n \).

\[
\text{inv2}_5: \quad \forall n, l \cdot l \in \text{net} \land n \mapsto l \notin \ell_{\text{net}} \Rightarrow n \mapsto l \in m_{\text{net}_\text{up}}
\]

Figure 3 gives a similar explanation for invariant inv2_6. When a link \( l \) is present in both \( \text{net} \) and \( \ell_{\text{net}} \) for node \( n \), then it must be absent from the message reservoir \( m_{\text{net}_\text{dn}} \) for node \( n \).

\[
\text{inv2}_6: \quad \forall n, l \cdot l \in \text{net} \land n \mapsto l \in \ell_{\text{net}} \Rightarrow n \mapsto l \notin m_{\text{net}_\text{dn}}
\]

We have similar explanations for invariants inv2_7 and inv2_8. Finally, we add the following new anticipated event, which will be explained in the next refinement. At this stage, we just ensure that this event maintains invariants non-deterministically:
4.4 Third Refinement

This refinement is not concerned with any requirement; it is instead a data-refinement. First we formally define the concept of parity for natural numbers:

\[
\text{constants: } \text{parity}
\]

\[
\text{axm}_3.1: \quad \text{parity} \in \mathbb{N} \rightarrow \{0, 1\}
\]

\[
\text{axm}_3.2: \quad \text{parity}(0) = 1
\]

\[
\text{axm}_3.3: \quad \forall x \cdot \text{parity}(x + 1) = 1 - \text{parity}(x)
\]

Now the magic abstract behavior of the variables \(m_{\text{net\_up}}\) and \(m_{\text{net\_dn}}\) of previous refinement is refined. For this, we add a variable \(age\), denoting the "date" when the logical network \(net\) has been modified:

\[
\text{variables: } \text{age}
\]

\[
\text{inv}_3.1: \quad \text{age} \in L \rightarrow \mathbb{N}
\]

\[
\text{inv}_3.2: \quad \forall l \cdot l \in L \Rightarrow (l \in \text{net} \Leftrightarrow \text{parity}(\text{age}(l)) = 1)
\]

Invariant \(\text{inv}_3.2\) might seem a bit strange. It is due to the fact that the presence or absence of a link in \(net\) changes simultaneously when the \(age\) variable is incremented (see the updates of events \(\text{discover\_up}\) and \(\text{discover\_dn}\) below). The parity of the variable \(age\) thus determine the absence or presence of a link in \(net\): this allows us to remove the variable \(net\) since we can easily reconstruct it from the variable \(age\). Similarly, we introduce a variable \(l_{\text{age}}\), recording the local age of a link in the variable \(l_{\text{net}}\):

```plaintext
change_link_2
status
  anticipated
any
ln
where
\(ln \in N \leftrightarrow L\)
\(\forall n, l \cdot l \in \text{net} \land n \rightarrow l \notin ln \Rightarrow n \rightarrow l \in m_{\text{net\_up}}\)
\(\forall n, l \cdot l \notin \text{net} \land n \rightarrow l \in ln \Rightarrow n \rightarrow l \in m_{\text{net\_dn}}\)
\(\forall n, l \cdot l \notin \text{net} \land n \rightarrow l \notin ln \Rightarrow n \rightarrow l \notin m_{\text{net\_dn}}\)
then
\(l_{\text{net}} := ln\)
end
```
variables: \( l_{\text{age}} \)

\textbf{inv3.3:} \( l_{\text{age}} \in N \times L \rightarrow \mathbb{N} \)

\textbf{inv3.4:} \( \forall l \cdot l \in L \Rightarrow l_{\text{age}}[N \times \{l\}] \subseteq 0 \ldots \text{age}(l) \)

\textbf{inv3.5:} \( \forall n, l \cdot n \in N \land l \in L \Rightarrow (n \mapsto l \in l_{\text{age}}[N \times \{l\}] \subseteq 0 \ldots \text{age}(l)) = 1 \)

Invariant \textbf{inv3.4} states that local ages are less than or equal to ages. In other words, \( \text{age}(l) \) is the maximum of all local ages concerning link \( l \). Moreover, invariant \textbf{inv3.5} expresses a property for local ages and local networks similar to the one that we expressed for the variables \( \text{age} \) and \( \text{net} \) in invariant \textbf{inv3.2}. As a consequence and for the same reason as for variable \( \text{net} \), we can remove variable \( l_{\text{net}} \). Finally, we have also to introduce ages in the messages. For this we define a variable \( m_{\text{net}} \).

\textbf{variables:} \( m_{\text{net}} \)

Variable \( m_{\text{net}} \) is subject to the following invariants:

\textbf{inv3.6:} \( m_{\text{net}} \in N \times L \leftrightarrow \mathbb{N} \)

\textbf{inv3.7:} \( \forall l \cdot l \in L \Rightarrow m_{\text{net}}[N \times \{l\}] \subseteq 0 \ldots \text{age}(l) \)

\textbf{inv3.8:} \( \forall n, l \cdot n \in N \land l \in L \Rightarrow (n \mapsto l \in m_{\text{net}} \Leftarrow n \mapsto l \mapsto \text{age}(l) \in m_{\text{net}} \land \text{parity}(\text{age}(n \mapsto l)) = 1) \)

\textbf{inv3.9:} \( \forall n, l \cdot n \in N \land l \in L \Rightarrow (n \mapsto l \in m_{\text{net}} \Leftarrow n \mapsto l \mapsto \text{age}(l) \in m_{\text{net}} \land \text{parity}(\text{age}(n \mapsto l)) = 0) \)

Invariants \textbf{inv3.8} and \textbf{inv3.9} establish the connection between \( m_{\text{net}} \) and both \( m_{\text{net}}\_\text{up} \) and \( m_{\text{net}}\_\text{dn} \). They explain that a link in \( m_{\text{net}}\_\text{up} \) or \( m_{\text{net}}\_\text{dn} \) corresponds to the most recent link in \( m_{\text{net}} \) together with the right parity. As a consequence, variables \( m_{\text{net}}\_\text{up} \) and \( m_{\text{net}}\_\text{dn} \) can be removed. The next two invariants say that \( \text{age}(l) \) is either in \( l_{\text{age}} \) or in \( m_{\text{net}} \), but not in both:

\textbf{inv3.10:} \( \forall n, l \cdot \text{age}(l) = l_{\text{age}}(n \mapsto l) \lor n \mapsto l \mapsto \text{age}(l) \in m_{\text{net}} \)

\textbf{inv3.11:} \( \forall n, l \cdot n \mapsto l \mapsto \text{age}(n \mapsto l) \notin m_{\text{net}} \)

The next invariant states that if a local age is strictly less than an age, then intermediate ages are waiting in \( m_{\text{net}} \):

\textbf{inv3.12:} \( \forall n, l \cdot \text{age}(l) = l_{\text{age}}(n \mapsto l) \land n \mapsto l \mapsto \text{age}(l) \notin m_{\text{net}} \)
\textbf{inv3\_12:} \quad \forall n, l : \text{age}(n \mapsto l) < \text{age}(l) \\
\Rightarrow \\
\text{l\_age}(n \mapsto l) + 1 \ldotp \text{age}(l) \in m\_net[n \mapsto l]

Next are the updates of the events:

\textbf{init}
\begin{align*}
NET & := \emptyset \\
\text{age} & := L \times \{0\} \\
\text{l\_age} & := N \times L \times \{0\} \\
m\_net & := \emptyset
\end{align*}

\textbf{discover\_up}
\begin{align*}
\text{any} \\
l \\
n \\
\text{where}
\begin{align*}
l & \in NET \\
\text{parity}(\text{age}(l)) & = 0 \\
n & \in N
\end{align*}
\text{then}
\begin{align*}
\text{age}(l) & := \text{age}(l) + 1 \\
m\_net & := m\_net \cup ((N \setminus \{n\}) \times \{l\} \times \{\text{age}(l) + 1\}) \\
l\_age(n \mapsto l) & := \text{age}(l) + 1
\end{align*}
\text{end}

\textbf{discover\_dn}
\begin{align*}
\text{any} \\
l \\
n \\
\text{where}
\begin{align*}
l & \not\in NET \\
\text{parity}(\text{age}(l)) & = 1 \\
n & \in N
\end{align*}
\text{then}
\begin{align*}
\text{age}(l) & := \text{age}(l) + 1 \\
m\_net & := m\_net \cup ((N \setminus \{n\}) \times \{l\} \times \{\text{age}(l) + 1\}) \\
l\_age(n \mapsto l) & := \text{age}(l) + 1
\end{align*}
\text{end}

We notice that the actions of these events are identical. Hence we shall be able to merge them in the next refinement.

\textbf{change\_link\_up}
\begin{align*}
\text{status} & := \text{convergent} \\
\text{any} \\
n \\
l \\
x \\
\text{where}
\begin{align*}
x & = \text{age}(l) \\
n \mapsto l \mapsto x & \in m\_net \\
\text{parity}(x) & = 1
\end{align*}
\text{then}
\begin{align*}
l\_age(n \mapsto l) & := x \\
m\_net & := m\_net \setminus \{n \mapsto l \mapsto x\}
\end{align*}
\text{end}

\textbf{change\_link\_dn}
\begin{align*}
\text{status} & := \text{convergent} \\
\text{any} \\
n \\
l \\
x \\
\text{where}
\begin{align*}
x & = \text{age}(l) \\
n \mapsto l \mapsto x & \in m\_net \\
\text{parity}(x) & = 0
\end{align*}
\text{then}
\begin{align*}
l\_age(n \mapsto l) & := x \\
m\_net & := m\_net \setminus \{n \mapsto l \mapsto x\}
\end{align*}
\text{end}

Likewise, we notice that the actions of these events are identical and the same as the actions of the next event. Hence we shall merge the three of them in the next refinement. The next event gives an explanation
to abstract event change_link_2: it corresponds to old messages being stored locally, although they are not up-to-date but still with their "age" greater than the local ages (node $n$ cannot know that they are not the most recent ones).

\begin{verbatim}
change_link_2
  status
  convergent
  any
  n
  l
  x
  where
  x \neq \text{age}(l)
  n \mapsto l \mapsto x \in m_{\text{net}}
  x > l_{\text{age}}(n \mapsto l)
  with
  (\text{parity}(x) = 0 \Rightarrow l_{\text{n}} = l_{\text{net}} \setminus \{n \mapsto l\}) \land
  (\text{parity}(x) = 1 \Rightarrow l_{\text{n}} = l_{\text{net}} \cup \{n \mapsto l\})
  then
  l_{\text{age}}(n \mapsto l) := x
  m_{\text{net}} := m_{\text{net}} \setminus \{n \mapsto l \mapsto x\}
end
\end{verbatim}

Notice the implicative definition of the witness for abstract parameter $l_{\text{n}}$: its definition depends on the parity of parameter $x$.

Event discard_1 below is new. It corresponds to messages that should not be stored as they are definitely too old. This event is also convergent.

The last event discard_2, also new, corresponds to dummy messages already received but still sent and thus received again. Notice that this event is not convergent. This is the only event that is not convergent. As a result, our fundamental theorem (proved in refinement 5) concerning the eventual identity between the local networks and the real network will depend on the fairness of this event. In other words, it must allow other events to eventually "execute" and thus converge.

\begin{verbatim}
discard_1
  status
  convergent
  any
  n
  l
  x
  where
  n \mapsto l \mapsto x \in m_{\text{net}}
  x \leq l_{\text{age}}(n \mapsto l)
  then
  m_{\text{net}} := m_{\text{net}} \setminus \{n \mapsto l \mapsto x\}
end
discard_2
  any
  n
  l
  x
  where
  n \mapsto l \mapsto x \notin m_{\text{net}}
  x \leq l_{\text{age}}(n \mapsto l)
  then
  m_{\text{net}} := m_{\text{net}} \setminus \{n \mapsto l \mapsto x\}
end
\end{verbatim}

These two events have identical actions. Thus they will be merged in the next refinement.

Events which are said to be convergent in this refinement use the following obvious variant following our last invariant, which says that variable $m_{\text{net}}$ is finite.
4.5 Fourth Refinement

In this refinement, we just merge events, as announced in the previous section:

\[
\begin{align*}
\text{init} & \quad NET := \emptyset \\
& \quad age := L \times \{0\} \\
& \quad l_{\text{age}} := N \times L \times \{0\} \\
& \quad m_{\text{net}} := \emptyset
\end{align*}
\]

\[
\begin{align*}
\text{discover} & \quad \text{refines} \\
& \quad \text{discover}_\text{up} \\
& \quad \text{discover}_\text{dn}
\end{align*}
\]

\[
\begin{align*}
\text{any} & \quad l \quad n \\
\text{where} & \quad l \in NET \Leftrightarrow \text{parity}(age(l)) = 0 \\
& \quad n \in N \\
\text{then} & \quad age(l) := age(l) + 1 \\
& \quad m_{\text{net}} := m_{\text{net}} \cup ((N \setminus \{n\}) \times \{l\} \times \{age(l) + 1\}) \\
& \quad l_{\text{age}}(n \mapsto l) := age(l) + 1 \\
\text{end}
\end{align*}
\]

\[
\begin{align*}
\text{change\_link} & \quad \text{refines} \\
& \quad \text{change\_link\_up} \\
& \quad \text{change\_link\_dn} \\
& \quad \text{change\_link\_2}
\end{align*}
\]

\[
\begin{align*}
\text{any} & \quad n \quad l \quad x \\
\text{where} & \quad n \mapsto l \mapsto x \in m_{\text{net}} \quad x > l_{\text{age}}(n \mapsto l) \\
\text{then} & \quad l_{\text{age}}(n \mapsto l) := x \\
& \quad m_{\text{net}} := m_{\text{net}} \setminus \{n \mapsto l \mapsto x\}
\end{align*}
\]

\[
\begin{align*}
\text{discard} & \quad \text{refines} \\
& \quad \text{discard\_1} \\
& \quad \text{discard\_2}
\end{align*}
\]

\[
\begin{align*}
\text{any} & \quad n \quad l \quad x \\
\text{where} & \quad x \leq l_{\text{age}}(n \mapsto l) \\
\text{then} & \quad m_{\text{net}} := m_{\text{net}} \setminus \{n \mapsto l \mapsto x\}
\end{align*}
\]

Variable \(m_{\text{net}}\) still occurs in the guard of the event \(\text{change\_link}\). In the next refinement however, we will eliminate the variable \(m_{\text{net}}\) from this guard and thus we will be able to remove the variable \(m_{\text{net}}\).

4.6 Fifth Refinement

In this refinement, we finally establish the connection between the set \(L\) of links and the set \(N\) of nodes. For this, we define two constant functions \(\text{fst}\) and \(\text{snd}\) projecting a link on its first and second node respectively. We also define a constant bijection \(\text{link}\) between pairs of nodes and links. Axioms \(\text{axm5\_4}\), \(\text{axm5\_5}\), and \(\text{axm5\_6}\) state the obvious properties relating these three constants.
Notice that when a link \( l \) belongs to the physical network (that is, \( l \in NET \)) then node \( fst(l) \) is a neighbor of node \( snd(l) \). This takes account of requirement \( \text{REQ-4} \):

The neighbors of a node \( b \) are the nodes that are connected to \( b \) by a link arriving in \( b \). Neighbors of \( b \) send to it their local networks

Now we are able to remove the message container \( m_{\text{net}} \). This is due to the fact that a link between two neighbors in the graph can play its role. More precisely, from invariant \( \text{inv3}_{12} \) established in the third refinement, which we copy below,

\[
\text{inv3}_{12}: \forall n, l \cdot \text{age}(n \mapsto l) < \text{age}(l) \Rightarrow l \_\text{age}(n \mapsto l) + 1 \_\text{age}(l) \in m_{\text{net}}[\{n \mapsto l\}]
\]

we can deduce the following theorem (with the help of invariant \( \text{inv3}_{4} \)):

\[
\text{thm5}_{1}: \forall n, l \cdot l \_\text{age}(n \mapsto l) < l \_\text{age}(m \mapsto l) \Rightarrow n \mapsto l \mapsto l \_\text{age}(m \mapsto l) \in m_{\text{net}}
\]

This theorem says that when the local age of link \( l \) at node \( n \) is strictly less than that at node \( m \) then the local age at node \( m \) is in a message for node \( n \). In other words, the difference in local ages induces a presence in \( m_{\text{net}} \). This will help us remove \( m_{\text{net}} \) from the guard of abstract event \( \text{change\_link} \), which we copy below:

\[
\begin{align*}
\text{change\_link} \\
\text{any} & \quad n \quad l \quad x \\
\text{where} & \quad n \mapsto l \mapsto x \in m_{\text{net}} \\
& \quad x > l \_\text{age}(n \mapsto l) \\
\text{then} & \quad l \_\text{age}(n \mapsto l) := x \\
& \quad m_{\text{net}} := m_{\text{net}} \setminus \{n \mapsto l \mapsto x\}
\end{align*}
\]
In fact, we replace now abstract parameter \( n \) by node \( \text{snd}(k) \), where \( k \) is a link arriving in \( n \) (thus, as said above, \( \text{fst}(k) \) is a neighbor of \( \text{snd}(k) \)). Moreover, we suppose that parameter \( x \) is less than or equal to the local age at node \( \text{fst}(k) \). This additional guard might seem strange: why have not we chosen \( \text{fst}(k) \) for \( x \)? The answer will be given in refinement 7. From all this, we obtain the following concrete version of the event change_link:

\[
\text{change_link} \\
\quad \text{any} \quad l \quad x \\
\quad \text{where} \\
\quad \quad k \in \text{NET} \\
\quad \quad x > \text{l_age}(	ext{snd}(k) \mapsto l) \\
\quad \quad x \leq \text{l_age}(	ext{fst}(k) \mapsto l) \\
\quad \text{with} \\
\quad \quad n = \text{snd}(k) \\
\quad \text{then} \\
\quad \quad \text{l_age}(	ext{snd}(k) \mapsto l) := x \\
\quad \quad \text{m_net} := \text{m_net} \setminus \{(\text{snd}(k) \mapsto l) \mapsto x\}
\]

But now we see that variable \( \text{m_net} \) does not occur any more in any guard. As a consequence, we can remove this variable which does not influence the behavior of our events. Next are the updates of the events. In the event discover we make now the last touch, by replacing the random choice of parameter \( n \) by the precise node \( \text{snd}(l) \). In doing this, we take full account of requirement \( \text{REQ-3} \).

<table>
<thead>
<tr>
<th>When a link from node ( a ) to node ( b ) is set into or removed from the graph then node ( b ) is directly made aware of it</th>
</tr>
</thead>
</table>

\[
\text{discover} \\
\quad \text{any} \\
\quad \text{any} \quad l \\
\quad \text{where} \\
\quad \quad l \in \text{NET} \iff \text{parity}(	ext{age}(l)) = 0 \\
\quad \text{with} \\
\quad \quad n = \text{snd}(l) \\
\quad \text{then} \\
\quad \quad \text{age}(l) := \text{age}(l) + 1 \\
\quad \quad \text{l_age}(	ext{snd}(l) \mapsto l) := \text{age}(l) + 1 \\
\quad \text{end}
\]

Notice that in the event discard, we perform a similar modification as the one we have done in the event change_link, namely replacing parameter \( n \) by node \( \text{snd}(k) \), where \( k \) is a link belonging to \( \text{NET} \).

14
4.7 Sixth Refinement

In this refinement, we take account of requirement REQ-7:

We want to prove that under certain conditions the images of the graph built by nodes are all identical and equal to the graph itself

In particular, we replace the variable NET by the variable G representing the physical graph. Here is the definition of this new variable and its connection to abstract variable NET:

<table>
<thead>
<tr>
<th>variables: G</th>
</tr>
</thead>
<tbody>
<tr>
<td>inv6_1: ( G \in N \leftrightarrow N )</td>
</tr>
<tr>
<td>inv6_2: ( \forall l \cdot l \in NET \leftrightarrow fst(l) \mapsto snd(l) \in G )</td>
</tr>
<tr>
<td>inv6_3: ( \forall l \cdot l_{age}(snd(l) \mapsto l) = age(l) )</td>
</tr>
</tbody>
</table>

Invariant inv6_3 will allow us to remove the variable age in the next refinement. We can now state the main result. Our two convergent events are discover and change_link. If these events cannot be "executed" any more (deadlock) and if the physical network is strongly connected at that time, then all local networks are identical to the physical network. Here are the new versions of these events:
Since we assume that they cannot be "executed" then their guards are false. Here are these guards:

\[ \exists l \cdot \text{fst}(l) \mapsto \text{snd}(l) \in G \iff \text{parity}(\text{age}(l)) = 0 \]

The negation of the first one can be simplified as follows:

\[ \neg \exists l \cdot \text{fst}(l) \mapsto \text{snd}(l) \in G \iff \text{parity}(\text{age}(l)) = 0 \]
\[ \iff \forall l \cdot \text{fst}(l) \mapsto \text{snd}(l) \in G \iff \text{parity}(\text{age}(l)) = 1 \]
\[ \iff \forall l \cdot l \in \text{NET} \iff \text{parity}(\text{age}(l)) = 1 \]

The strong connectivity of the graph \( G \) is expressed by means of the following statement (justified in the Appendix):

\[ \forall s \cdot s \neq \emptyset \land G[s] \subseteq s \Rightarrow N \subseteq s \]

We first prove the following theorem

16
To prove this theorem, we instantiate the quantified variable $s$ in the second hypothesis with the set \( \{ n \mid l_{\text{age}}(n \rightarrow l) = \text{age}(l) \} \). Proving that this set is not empty is easy: \( \text{snd}(l) \) belongs to it according to invariant inv6.3. \( G[s] \subseteq s \) is a consequence of the first hypothesis. The result then follows easily. The following theorem can be deduced from the previous one:

**thm6.2:** $\forall l, l \in \text{NET} \iff \text{parity}(l_{\text{age}}(l)) = 1$

$\forall l, a, b \in G \Rightarrow l_{\text{age}}(a \rightarrow l) \leq l_{\text{age}}(b \rightarrow l)$

$\forall s \cdot s \neq \emptyset \land G[s] \subseteq s \Rightarrow N \subseteq s$

$\Rightarrow$

$\forall l, n \cdot l \in \text{NET} \iff \text{parity}(l_{\text{age}}(n \rightarrow l)) = 1$

Here is our final result (with the help of invariant inv3.5).

**thm6.3:** $\forall l, l \in \text{NET} \iff \text{parity}(l_{\text{age}}(l)) = 1$

$\forall l, a, b \in G \Rightarrow l_{\text{age}}(a \rightarrow l) \leq l_{\text{age}}(b \rightarrow l)$

$\forall s \cdot s \neq \emptyset \land G[s] \subseteq s \Rightarrow N \subseteq s$

$\Rightarrow$

$\forall n \cdot l_{\text{net}}[\{ n \}] = \text{NET}$

This theorem can be restated in a less formal way as follows:

- Event discover deadlocks and
- Event change_link deadlocks and
- Physical graph is strongly connected

$\Rightarrow$

Local networks are all equal to physical network

Next are the updates of the three remaining non-convergent events:

**Modify_up**

any $l$

where $\text{fst}(l) \rightarrow \text{snd}(l) \not\in G$

then $G := G \cup \{ \text{fst}(l) \rightarrow \text{snd}(l) \}$

end

**Modify_dn**

any $l$

where $\text{fst}(l) \rightarrow \text{snd}(l) \in G$

then $G := G \setminus \{ \text{fst}(l) \rightarrow \text{snd}(l) \}$

end
4.8 Seventh Refinement

In this last refinement, we take account of requirements REQ-5 and REQ-6:

- Nodes communicates by means of messages sent on links of the graph
  - REQ-5

- A message sent from $a$ to $b$ can be lost if the link from $a$ to $b$, which existed when the message was sent, is broken before the message reaches $b$
  - REQ-6

To this end, we have an additional variable $m$ defined as follows (notice that variable $age$ has been removed). When $k \mapsto l \mapsto x$ belongs to $m$ it means that there exists a message concerned with link $l$ and age $x$ travelling on link $k$.

\[
\begin{align*}
\text{variables:} & \quad G, l_\text{age}, m \\
\text{init} & \quad G := \emptyset \\
& \quad age := L \times \{0\} \\
& \quad l_\text{age} := N \times L \times \{0\}
\end{align*}
\]

\[
\begin{aligned}
discard \\
\any \\
l & \mapsto x \\
k & \mapsto \\
\text{where} & \\
fst(k) \mapsto snd(k) \in G \\
x & \leq l_\text{age}(snd(k) \mapsto l) \\
\text{then} & \\
\skip & \\
\text{end}
\end{aligned}
\]

Invariant $\text{inv7}_2$ states that an age $x$ recorded in a message is always less than or equal to the local age at the sending point of the message. This is because the local age at the sending point can only be incremented after the message has been sent. This allows us to remove a guard from the event change_link, renamed below accept_message.

Invariant $\text{inv7}_3$ states that a message arriving at its destination $snd(k)$ is always such that the link from $fst(k)$ to $snd(k)$ is in the graph. This allows us to remove guards from the events accept_message and discard, renamed below to discard_message.

Here are all our final concrete events:
In the event `Modify_dn`, messages sent along the removed link \( l \) are lost as stipulated by requirement `REQ-6`:

\[
\text{A message sent from } a \text{ to } b \text{ is lost if the link from } a \text{ to } b, \text{ which existed when the message was sent, is broken before the message reaches } b
\]

The next event is new. It corresponds to sending a message. This can be done if a link indeed exists between the corresponding nodes as mentioned in requirement `REQ-5`. Note that this event is not convergent: a message can always be sent.

The last events are renaming of abstract events `change_link` and `discard`:
5 Concluding Remarks

The formal development was conducted using the Rodin Platform. Here is a summary of the proof statistics together with the corresponding percentage of automatic proofs:

<table>
<thead>
<tr>
<th>model</th>
<th>proofs</th>
<th>auto.</th>
<th>manual</th>
<th>% auto.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Ref. 1</td>
<td>3</td>
<td>3</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Ref. 2</td>
<td>47</td>
<td>47</td>
<td>0</td>
<td>100</td>
</tr>
<tr>
<td>Ref. 3</td>
<td>130</td>
<td>93</td>
<td>37</td>
<td>72</td>
</tr>
<tr>
<td>Ref. 4</td>
<td>6</td>
<td>4</td>
<td>2</td>
<td>67</td>
</tr>
<tr>
<td>Ref. 5</td>
<td>18</td>
<td>16</td>
<td>2</td>
<td>89</td>
</tr>
<tr>
<td>Ref. 6</td>
<td>13</td>
<td>9</td>
<td>4</td>
<td>69</td>
</tr>
<tr>
<td>Ref. 7</td>
<td>38</td>
<td>25</td>
<td>13</td>
<td>66</td>
</tr>
<tr>
<td>Total</td>
<td>255</td>
<td>197</td>
<td>58</td>
<td>77</td>
</tr>
</tbody>
</table>

As can be seen, Refinement 3 required by far the largest number of proofs: it is due to the large number of invariants (13), together with the number of events (10). It should be noted that the manual proofs were not difficult. It seems that many of them could be done automatically, which suggests that the automatic provers have to be improved!
APPENDIX: Strongly Connected Graph

Given a set $V$ of vertices and a binary relation $r$ from $V$ to itself, the graph represented by this relation is said to be *strongly connected* if any two nodes in $V$ are connected by a path built on $r$. This is illustrated in figure 4.

![Fig. 4. A Strongly Connected Graph](image)

This can be formalized as follows, where $r^*$ is the reflexive transitive closure of $r$:

$$\text{def}_1 : \quad r^* = V \times V$$

This definition is easy to understand: it simply says that every two points of $V$ are related through the reflexive transitive closure $r^*$ of $r$. But this definition is not very convenient to use in practice. Here is an *equivalent one* which is more convenient:

$$\text{def}_2 : \quad \forall s \cdot s \neq \emptyset \land r[s] \subseteq s \Rightarrow V \subseteq s$$

This definition is convenient because it gives us an *induction rule* to prove properties of the vertices of a strongly connected graph, which is the one we used in section 4.7. In order to prove the equivalence between these definitions, we need to have a formal representation of $r^*$:

$$\text{axm}_1 : \quad r^* \in V \leftrightarrow V$$

$$\text{axm}_2 : \quad \text{id} \subseteq r^*$$

$$\text{axm}_3 : \quad r^* ; r \subseteq r^*$$

$$\text{axm}_4 : \quad \forall p \cdot \text{id} \subseteq p \land p ; r \subseteq p \Rightarrow r^* \subseteq p$$
Proof of:

\[ \forall s \cdot s \neq \emptyset \land r[s] \subseteq s \Rightarrow V \subseteq s \]

\[ \Rightarrow r^* = V \times V \]

We prove the equivalent lemma

\[ \forall s \cdot s \neq \emptyset \land r[s] \subseteq s \Rightarrow V \subseteq s \]

\[ \Rightarrow \forall x \cdot V \subseteq r^*[\{x\}] \]

To prove this, we instantiate \( s \) in the hypothesis with \( r^*[\{x\}] \). The result follows easily according to axm_2 and axm_3.

Proof of:

\[ r^* = V \times V \]

\[ \Rightarrow \forall s \cdot s \neq \emptyset \land r[s] \subseteq s \Rightarrow V \subseteq s \]

First we replace \( r^* \) by \( V \times V \) in axm_2 to axm_4. Then we simplify the goal. Finally, we instantiate the quantified variable \( p \) of axm_4 with \((s \times s) \cup ((V \setminus s) \times V)\). The result follows easily.