**APPENDIX A**

**PROOF OF LEMMA 1**

**Lemma 1:** A node from a level-$i$ group can reach a node in any adjacent level-$i$ group in at most $3^i$ hops.

**Proof:** The proof is performed by induction.

**Basis:** $i = 0$. Let’s take two arbitrary adjacent level-0 groups: $G_A^0$ and $G_B^0$. From Property 1, $G_A^0 = \{A\}$ and $G_B^0 = \{B\}$. $G_A^0$ and $G_B^0$ are adjacent, thus $A$ and $B$ are neighbors, that is, $A$ can reach $B$ in $1 = 3^0$ hop. Since we chose $G_A^0$ and $G_B^0$ arbitrarily, the lemma is true for $i = 0$.

**Inductive step:** $i = k + 1$ (where $k \geq 0$). We assume that the lemma holds for all levels $\leq k$. Let’s take two arbitrary adjacent level-$i$ groups $G_A^i$ and $G_B^i$, and an arbitrary node $P \in G_A^i$. Let $R$ denote a node in $G_B^i$ that has a neighbor $Q$ such that $Q \in G_A^i$ (existence of $Q$ is guaranteed by the definition of adjacent groups).

Consider level-$i$ subgroups, that is, $G_C^{i-1}$, $G_D^{i-1}$, and $G_E^{i-1}$, such that $P \in G_C^{i-1}$, $Q \in G_D^{i-1}$, and $R \in G_E^{i-1}$.

We have the following three situations:

1): $C = D$ (group $G_C^{i-1}$ is adjacent to group $G_E^{i-1}$).

2): $G_C^{i-1}$ is adjacent to $G_D^{i-1}$.

In this case, from the inductive assumption $P$ can reach some node in $G_E^{i-1}$ (hence in $G_B^i$) in at most $3^{i-1} < 3^i$ hops.

**Proof:** The proof is performed by induction. Let $d(A,B)$ denote the distance in hops between nodes $A$ and $B$.

**Basis:** $i = 0$. Let’s take two arbitrary adjacent level-0 groups: $G_A^0$ and $G_B^0$. From Property 1, $G_A^0 = \{A\}$ and $G_B^0 = \{B\}$. $G_A^0$ and $G_B^0$ are adjacent, thus $A$ and $B$ are neighbors, that is, $d(A,B) = 1 = 3^0$. As we chose $G_A^0$ and $G_B^0$ arbitrarily, the lemma is true for $i = 0$.

**Inductive step:** $i = k + 1$ (where $k \geq 0$). We assume that the lemma holds for all levels $\leq k$. Let’s take two arbitrary adjacent level-$i$ groups $G_A^i$ and $G_B^i$. We have three possible situations:

1): $G_A^{i-1}$ is adjacent to $G_B^{i-1}$.
In this case, from the inductive assumption, \( d(A, B) \leq 3^{i-1} < 3^i \).

2) There exists \( G_{C,i} \) such that it belongs to \( G_A \) or \( G_B \) and it is adjacent to both \( G_{A,i} \) and \( G_{B,i} \).

In this case, \( d(A, B) \leq d(A, C) + d(B, C) \). From the inductive assumption \( d(A, C), d(B, C) \leq 3^{i-1} \), thus \( d(A, B) \leq 2 \cdot 3^{i-1} = 3^i \).

3) There exist \( G_{C,i} \) and \( G_{D,i} \) such that \( G_{C,i} \) belongs to \( G_A \) and \( G_{D,i} \) belongs to \( G_B \) and \( G_{c,i} \) is adjacent to \( G_{d,i} \). (From Property 4, \( G_{C,i} \) is adjacent to \( G_{A,i} \), and \( G_{D,i} \) is adjacent to \( G_{B,i} \).

In this case, \( d(A, B) \leq d(A, C) + d(C, D) + d(D, A) \). From the inductive assumption \( d(A, C), d(C, D), d(D, A) \leq 3^{i-1} \), thus \( d(A, B) \leq 3 \cdot 3^{i-1} = 3^i \).

Consequently, we have \( d(A, B) \leq 3^i \). Because \( G_A \) and \( G_B \) were chosen arbitrarily, the lemma is true for \( i = k + 1 \).

By applying mathematical induction to the basis and the inductive step, we proved the lemma for all \( i \). Moreover, \( 3^i \) is a tight bound, that is, it is reachable for some configurations.

**APPENDIX C**

**PROOF OF LEMMA 3**

**Lemma 3:** The distance between any two members of a level-\( i \) group is at most \( 3^i - 1 \) hops.

**Proof:** We choose an arbitrary group \( G_A \) and two arbitrary nodes \( P \) and \( Q \) that are members of this group. We add another node \( R \) to the system such that \( R \)'s only neighbor is \( Q \). \( R \) forms singleton groups \( G_i, G_i, \ldots, G_i \). From Lemma 2, \( P \) can reach \( R \) in at most \( 3^i \) hops. Since \( P \) can reach \( R \) only through \( Q, P \) can reach \( Q \) in at most \( 3^i - 1 \) hops, that is the distance between \( P \) and \( Q \) is at most \( 3^i - 1 \).

Because \( P \) and \( Q \) were chosen arbitrarily, the lemma holds for any members of group \( G_A \). Likewise, the arbitrary choice of \( G_A \) and \( i \) proves the lemma for all \( i \). Again, \( 3^i - 1 \) is a tight bound.

**APPENDIX D**

**PROOF OF LEMMA 4**

**Lemma 4:** Update propagation based on the responsibility rule and update vectors guarantees eventual consistency of node labels: in the absence of changes in the system, for any group \( G \), and any node \( A \), if \( A \) is eventually a member of \( G \) (eventually \( L(A) \) is \( X \)), then \( A \) and \( X \) eventually have equal labels starting from position \( i \) for all \( k \geq i \), \( L(A) = L(X) \).

**Proof:** Let \( \langle L(A) \rangle_k \) denote the \( i \)-th element of the label of node \( A \) in round \( r \). Moreover, we treat the first (starting from position 0) \( null \) value in the label as the end of the label. We observe that, from Property 1, at level 0 every node is always a member of its own singleton group. Therefore, in the remainder of the proof, when referring to eventual consistency at level \( i \), we always assume that \( i > 0 \).

After changes in the system have stopped, node \( A \) eventually becomes a member of \( G \) (\( i > 0 \)) iff:

\[
\exists A \forall r \geq r_A \left( \langle L(A) \rangle_r = \langle L(X) \rangle_r \right) \wedge \forall 0 \leq j < i \left( \langle L(A) \rangle_j = \langle L(A) \rangle_{r_A} \right) \wedge \forall 0 \leq j < i \left( \langle L(A) \rangle_j \neq \langle L(A) \rangle_{r_A} \right).
\]

In other words there exists a stabilization round for \( A \)'s label at level \( i \), \( r_A \), in and after which \( A \)'s label length is greater than \( i \), the \( i \)-th element of \( A \)'s label is always equal to \( X \), and \( A \)'s label and update vector do not change at lower levels.

All nodes that are eventually members of \( G \) constitute the following set: \( \{ A \mid A \ is \ eventually \ a \ member \ of \ G \} \). Since this set is finite, we can define its stabilization round, \( r^* \), as the maximum stabilization round over all its members. In other words, \( r^* \) denotes the round after which no nodes join or leave group \( G \). We will use \( \langle G \rangle \) to denote the stable set of nodes constituting group \( G \).

We will show that if a node belongs to \( \langle G \rangle \) its label and \( X \)'s label will eventually be equal at position \( i + 1 \). In other words, the two nodes from the same level-\( i \) group will eventually belong to the same level-\( i + 1 \) group. This is sufficient to prove eventual consistency. More formally, for an arbitrary node \( A \in \langle G \rangle \), we will show that (**\(*\**\)):

\[
\exists A \forall r \geq r_A \left( \langle L(A) \rangle_r = \langle L(X) \rangle_r \right) \wedge \forall 0 \leq j < i + 1 \left( \langle U(A) \rangle_j = \langle U(A) \rangle_{r_A} \right) \wedge \forall 0 \leq j < i + 1 \left( \langle L(A) \rangle_j \neq \langle L(A) \rangle_{r_A} \right).
\]

1. Although in a practical setting this may be impossible, it is perfectly valid from the graph theory perspective, and consequently, does not invalidate the proof.
Recall the responsibility rule and assume that, before the changes have stopped, the last update by $X$, acting as a level-$i$ head, was performed in round $r^i_X + 1$. The update had the sequence number $\Delta$ and corresponded to writing $\emptyset$ at the $i+1$-st position of $X$’s label. More formally, the following conditions hold for $X$’s label and update vector:

$$\forall r \geq r^i_X \left( (L(i+1))_r = \emptyset \right) \land \left( (U(i))_r = \Delta \right)$$

We will consider rounds $\geq \max(r^i_X, r^i_B + 1)$. We will show that, for an arbitrary node $A \in \langle G^i_X \rangle_{r^i_X}$, the following holds ($\exists \exists \exists$):

$$\exists r^i_A \geq \max(r^i_X, r^i_B + 1) \forall r \geq r^i_A \left( (L(A)(i+1))_r = \emptyset \right) \land (U(A))_r = \Delta,$$

which is sufficient to prove ($\ast \ast \ast$).

Let us observe that since the group membership is based on connectivity, all members of $\langle G^i_X \rangle_{r^i}$ form a connected graph. We can prove this by induction. For $i = 0$, $\langle G^0_X \rangle_{r^0} = \{X\}$, hence $\langle G^0_X \rangle_{r^0}$ is a connected singleton graph. For $i > 0$, our inductive assumption is that the connectivity property holds for all levels $j < i$. Let us then consider two arbitrary nodes: $P, Q \in \langle G^i_X \rangle_{r^i}$. Like in earlier proofs, we can have three possibilities. 1): $P, Q \in \langle G_{r^i-1}^i \rangle_{r^i}$ for some $Y$, in which case $P$ and $Q$ are connected from the inductive assumption. 2): $P \in \langle G_{r^i-1}^i \rangle_{r^i}$ and $Q \in \langle G_{r^i-1}^i \rangle_{r^i}$ (or vice versa) for some $Y$. In this case, from the inductive assumption, all nodes in $\langle G_{r^i-1}^i \rangle_{r^i}$ and all nodes in $\langle G_{r^i-1}^i \rangle_{r^i}$ are connected. Moreover, from Property 4, since $\langle G_{r^i-1}^i \rangle_{r^i}$ is the central subgroup of $\langle G^i_X \rangle_{r^i}$, it is adjacent to $\langle G_{r^i-1}^i \rangle_{r^i}$, that is, there exists at least one link between $\langle G_{r^i-1}^i \rangle_{r^i}$ and $\langle G_{r^i-1}^i \rangle_{r^i}$. Therefore, all nodes from $\langle G_{r^i-1}^i \rangle_{r^i}$ (in particular $Q$) are connected with all nodes from $\langle G_{r^i-1}^i \rangle_{r^i}$ (in particular $P$) and vice versa. 3): $P \in \langle G_{r^i-1}^i \rangle_{r^i}$ and $Q \in \langle G_{r^i-1}^i \rangle_{r^i}$ for some $Y$ and $Z$. Like in 2), in this case all nodes from $\langle G_{r^i-1}^i \rangle_{r^i}$ are connected with all nodes from $\langle G_{r^i-1}^i \rangle_{r^i}$ through the central subgroup $\langle G_{r^i-1}^i \rangle_{r^i}$. Consequently, $P$ and $Q$ are connected. Since we chose $P$ and $Q$ arbitrarily, the connectivity condition is true for all nodes that belong to $\langle G^i_X \rangle_{r^i}$. By applying mathematical induction we proved that nodes in any $\langle G^i_X \rangle_{r^i}$ form a connected graph.

Consequently, we can prove ($\exists \exists \exists$) by induction over the number of hops, $n$, from $X$.

**Basis:** $n = 0$. The only node, $A \in \langle G^i_X \rangle_{r^i}$, that is only zero hops away from $X$ is $X = X$, so $r^i_X + 1 = \max(r^i_X, r^i_B + 1)$. Consequently, ($\exists \exists \exists$) holds.

**Inductive step:** $n = k + 1$ (where $k \geq 0$). We assume that ($\exists \exists \exists$) holds for all nodes in $\langle G^i_X \rangle_{r^i}$, that are $\leq k$ hops away from $X$. Let us take an arbitrary node, $A \in \langle G^i_X \rangle_{r^i}$, that is $k + 1$ hops away from $X$. A has a neighbor, $B \in \langle G^i_X \rangle_{r^i}$, which is $k$ hops away from $X$ and for which ($\exists \exists \exists$) holds:

$$\exists r^i_B \geq \max(r^i_X, r^i_B + 1) \forall r \geq r^i_B \left( (L(B)(i+1))_r = \emptyset \right) \land (U(B))_r = \Delta,$$

Assume that there is no message loss. In round $r^i_B + 1$, $A$ receives a gossip message from $B$ and compares its label against $B$’s label to find the minimal common level (see Sect. 4.3.2). Since $A$ and $B$ are members of $\langle G^i_X \rangle_{r^i}$, their minimal common level, $j$ is $\leq i$. $A$ then compares its update vector with $B$’s update vector starting from position $j$. Since both $A$ and $B$ belong to $\langle G^i_X \rangle_{r^i}$, the first position at which their update vectors can differ is $i$, otherwise one of them would have to change its update vector in the present round at position $< i$, which precludes membership in $\langle G^i_X \rangle_{r^i}$. If the update vectors do not differ at position $i$ ($U(B))_r = U(A))_r$, then $\langle L(A)\rangle_{r^i_B + 1} = \emptyset$, then $\langle L(A)\rangle_{r^i_B + 1} = \emptyset$ because the responsibility rule ensures the following invariant:

$$\forall P, Q, i, r \left( (L(P))_r = (L(Q))_r \right) \land \left((U(P))_r = (U(Q))_r \right) \Rightarrow \left((L(P)(i + 1))_r = (L(Q)(i + 1))_r \right)$$

If the update vectors differ at position $i$, in turn, then $\langle U(A)\rangle_{r^i_B + 1} < \Delta$ as $\Delta$ is the last update performed by $X$ acting as a level-$i$ head. Consequently, following our consistency enforcement algorithm $A$ copies $B$’s label and update vector starting from position $i$. As a result, $\langle L(A)\rangle_{r^i_B + 2} = \emptyset$ and $\langle U(A)\rangle_{r^i_B + 2} = \Delta$. In both cases, we have shown that:

$$\exists r^i_A \geq \max(r^i_X, r^i_B + 1) \left( (L(A)(i+1))_r = \emptyset \right) \land (U(A))_r = \Delta,$$

Thus, to prove ($\exists \exists \exists$), we still have to show that $A$’s label element at level $i+1$ and $A$’s update vector element at level $i$ do not change in any round $r \geq r^i_B + 1$. To this end, assume the opposite, that is, in some round $r \geq r^i_B + 1$, $A$’s label or update vector change at the mentioned levels. The change cannot be a result of a local update by $A$ at some level $< i$ because this would violate $A \in \langle G^i_X \rangle_{r^i}$. Therefore, the change must be a result of copying the label of some neighbor, $C$, after a gossip message from the neighbor has been received. Again, the copying cannot be performed starting from any level $< i$, as this would violate $A \in \langle G^i_X \rangle_{r^i}$. Therefore, the copying occurred starting from level $i$. This means that $\langle L(A)\rangle_{r^i} = \langle L(C)\rangle_{r^i} = X$ and $\langle U(A)\rangle_{r^i} = \Delta < \langle U(C)\rangle_{r^i}$. We have a contradiction, because $\Delta$ was the sequence number of the last update performed by $X$ acting as a level-$i$ head, and thus it must be the case that: $\Delta \geq \langle U(C)\rangle_{r^i}$. Therefore, we have proved ($\exists \exists \exists$). Since $A$ was chosen arbitrarily, the inductive step holds for any node that belongs to $\langle G^i_X \rangle_{r^i}$ and is $k + 1$ hops away from $X$.

By applying mathematical induction to the basis and the inductive step, we proved ($\exists \exists \exists$) for all $n$, that is, for all nodes that constitute $\langle G^i_X \rangle_{r^i}$. Since for any $A \in \langle G^i_X \rangle_{r^i}$, ($\ast \ast \ast$) is a direct consequence of ($\exists \exists \exists$), we proved ($\ast \ast \ast$) for all nodes that belong to $\langle G^i_X \rangle_{r^i}$. In other words, we proved that, for any level $i$, any two nodes that eventually have equal labels at level $i$, also eventually have their labels equal at level $i+1$.

Because the number of levels is finite, the proof implies that, for any level $i$, any two nodes that have their label equal at level $i$, also have their labels equal at all levels $k > i$. This ends the proof of Lemma.
APPENDIX E
PROOF OF LEMMA 5

Lemma 5: Assume that the slot size is longer than the number of rounds it takes to propagate information between the heads, X and Y, of two adjacent “top-level” groups $G'_X$ and $G'_Y$. In this case, with probability $\geq \frac{1}{4}$, $G'_X$ will be able to join $G_Y^{i+1}$ or vice versa.

Proof: Consider two arbitrary nodes, X and Y, that are heads of adjacent “top-level” groups $G'_X$ and $G'_Y$ respectively. X and Y must potentially spawn level-i+1 groups.

Let $r_X$ and $r_Y$ denote the round in which X and Y respectively choose their virtual slots, as described in Sect. 4.4.1. Note that this implies that in round $r_X$, X has learned about Y, and similarly, in round $r_Y$, Y has learned about X. Let the number of slots $S = 2$. Moreover, assume that the slot size, $R$, meets the requirements of the lemma, that is, it is longer than the number of rounds necessary to propagate information between X and Y.

We have the following four possible slot selection configurations, each obtained with probability $\frac{1}{4}$:

<table>
<thead>
<tr>
<th></th>
<th>I</th>
<th>II</th>
<th>III</th>
<th>IV</th>
</tr>
</thead>
<tbody>
<tr>
<td>$s_X$</td>
<td>0</td>
<td>0</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>$s_Y$</td>
<td>0</td>
<td>1</td>
<td>0</td>
<td>1</td>
</tr>
</tbody>
</table>

Without the loss of generality assume that $r_X \geq r_Y$, that is, X selects its slot in the same or later round than Y. Consider configuration III, in which X selects slot $s_X = 1$ and Y selects slot $s_Y = 0$. Let $r_X' = r_X + s_X \cdot R + 1$ denote the round in which X potentially spawns group $G_X^{i+1}$, as specified by the algorithm. Likewise, let $r_Y' = r_Y + s_Y \cdot R + 1$ denote the round in which Y potentially spawns group $G_Y^{i+1}$. We will show (by contradiction) that by the time it spawns group $G_X^{i+1}$, X discovers that Y has spawned $G_Y^{i+1}$. Consequently, X can make $G'_X$, a subgroup of $G_Y^{i+1}$, decreasing the number of groups at level i+1.

To this end, assume that X spawns $G_X^{i+1}$ in round $r_X'$ and Y spawns $G_Y^{i+1}$ in round $r_Y'$. Consider value $r_X' - r_Y'$ which denotes how many rounds after $r_X$ has spawned group $G_X^{i+1}$, node X spawns group $G_Y^{i+1}$.

$$r_X' - r_Y' = (r_X + s_X \cdot R + 1) - (r_Y + s_Y \cdot R + 1) = r_X - r_Y + (1 - s_X) \cdot R = r_X - r_Y + (1 - 0) \cdot R = r_X - r_Y + R = r_X - r_Y + R \geq (\text{from: } r_X \geq r_Y) \geq R = R.$$

From the above calculation X spawns group $G_X^{i+1}$ at least R rounds after Y has spawned group $G_Y^{i+1}$, Contradiction!, because within at most R rounds, X would have learned that Y spawned $G_Y^{i+1}$, and consequently, would have made $G_X$ a subgroup of $G_Y^{i+1}$. Therefore, with probability at least $\frac{1}{4}$, $G_X$ and $G_Y$ will be subgroups of a common group $G^{i+1}_{XY}$. Because X, Y, $G'_X$, and $G'_Y$ were chosen arbitrarily, Lemma 5 holds for any head node at any level. In practice, the aforementioned probability is higher than $\frac{1}{4}$.

APPENDIX F
THE MAINTENANCE ALGORITHM

Each node running PL-Gossip reacts to two types of events: reception of a gossip message and periodical timeouts. Below, we describe these events in detail. We present the simplest version of the algorithm, without any optimizations.

F.1 Gossip Message Reception

Receiving a message (see Listing 1) allows a node to discover changes in the hierarchy and to update its routing table. A gossip message contains the label of the sender node with the corresponding update vector and the sender’s routing table, plus some possible additional information piggybacked by lower layers (e.g., link quality information of the sender’s neighbors). First, the node that received the message searches for the minimal common-level group it shares with the sender of the message (listing lines 3-7, see also Sect. 4.3.2). If such a group exists (ll. 9), the node compares its update vector with the sender’s update vector to determine which of the two labels is more fresh (ll. 13-17, see also Sect. 4.3.2). If both the labels are fresh (ll. 19), the node only updates its routing table with the entries contained in the gossip message (ll. 20-21). If, however, the sender’s label is more fresh (ll. 22), before updating its routing table (ll. 27-28), the node adopts that label as explained in Sect. 4.3.2 (ll. 23-26). Finally, if the sender’s label is stale, the node can still use parts of the sender’s routing table to update its own routing table (ll. 30-32).

If the node and the sender of the gossip message do not share any group (ll. 35), the node has just discovered a violation of Property 2 (see Sect. 4.4.1). To propagate the information about this violation to the head of its top-level group, the node adds appropriate entries to its routing table, as explained in Sect. 4.2 (ll. 36-43). These entries will allow the head to react to the violation.

F.2 Periodic Timeout

The timeout event (see Listing 2) gives a node the opportunity to react to the changes in the system that occurred since the last timeout. First, the node removes stale entries from its routing table (listing lines 49-50), which enables detecting disruptive failures. More specifically, if the node, being a level-i head (where $i \geq 0$), is not the top-level head (ll. 55), it must check whether the central subgroup of its level-1+i group is still reachable and adjacent to the node’s level-i group (ll. 56-61), as explained in Sect. 4.4.2. If these conditions are not met (a violation of Property 4 occurred), the node cuts its label down to level i (ll. 62-65), as described in Sect. 4.4.2. Otherwise, from the node’s perspective, there were no disruptive failures in the system.

Second, if the node is the top-level head (ll. 72, possibly as a result of an earlier label cut), it must check whether the hierarchy construction is complete. To this end, the node first determines if its routing table contains entries for a level-1+i group it could join (ll. 73-76), as explained in Sect. 4.4.1. If this is the case (ll. 76), the node joins its level-i group to the level-1+i group, by extending its label with the identifier of
the head of this level-i+1 group (ll. 77-81). It also cancels any possible pending suppression of label extension which corresponded to spawning a new supergroup (ll. 82-83). As explained in Sect. 4.4.1, if the level-i+1 group is itself a member of some higher-level groups, all members of the node’s level-i group will gradually extend their labels when exchanging gossip messages (ll. 22-28).

Even if an appropriate level-i+1 group to join could not be found, it is still possible that the hierarchy is not complete. More specifically, the node must check whether its routing table contains any entries for other groups starting from level i (ll. 88). If so the node activates a suppression counter to defer spawning a new level-i+1 group (ll. 99-103), as explained in Sect. 4.4.1. The suppression counter, once activated, is decremented during each timeout (ll. 107-108). When it reaches zero and the level-i+1 group still has to be spawned (ll. 90), the node extends its label and cancels the counter (ll. 91-97), effectively spawning a new level-i+1 group (with itself as the head of that group).

Finally, when the node reacted to all changes in the system, it broadcasts a gossip message (ll. 117-118), such that its neighbors can adopt any label updates and update the routes.

Listing 1. The handler of the gossip message reception.

```c
HANDLE onGossipReceived(msg) {  
  if (msg.type == Gossip) {  
    int i = 0;  
    for (; i < msg.uvec.len; msg.uvec.len++) {  
      if (this.uvec[i] == msg.uvec[i]) break;  
    }  
    if (i < msg.uvec.len) {  
      // we found a node that shares a group with us,  
      // so determine who has a more recent label  
      // find the minimal differing position  
      int j = i;  
      for (; j < msg.uvec.len; msg.uvec.len++) {  
        if (this.uvec[i] < msg.uvec[j]) break;  
      }  
      if (i >= msg.uvec.len) {  
        // we both have the same labels  
        this.rt.mergeWith(msg.rt, i - 1, msg.rt.topRow);  
      } else if (this.uvec[i] < msg.uvec[j]) {  
        // we are not up to date, so  
        // change our label and update vector  
        this.lab.copyFrom(msg.lab, j);  
        this.uvec.copyFrom(msg.uvec, j);  
        // merge routing tables  
        if (this.rt.mergeWith(msg.rt, i - 1, msg.rt.topRow));  
      } else {  
        // the other guy is not up to date, but we  
        // can still use a part of his routing table  
        this.rt.mergeWith(msg.rt, i - 1, j);  
      }  
    }  
    if (this.rt) this.scnt = ++this.scnt;  
  }  
}
```

Listing 2. The periodical timer handler.
Listing 3. The initialization handler.

```java
122   HANDLER onNodeBoot() {
123       // initialize
124       this.lab = {this.NODE_ID};
125       this.cver = {0};
126       this.ttl = {1};
127       this.cpos = -1;
128       this.ucnt = restore("UPDATE_CNT");
129   }
130   // set timer handler
131   setTimer(ΔT, &onTimeout);
132 }
```

Fig. 1. An example of hierarchical routing in the hierarchy from Fig. 1 of the paper.

F.3 Remarks

When a node repaired after a failure rejoins the system, its membership decisions (label updates) made before the failure may still be present in the labels of other nodes. Therefore, it is crucial to ensure that any decision made by this node after the failure is perceived by other nodes as later than any decision made by this node before the failure. Otherwise, the ordering of label updates is not preserved, which disrupts the consistency enforcement algorithm. In that case, we cannot predict the behavior of the system.

To this end, whenever a node performs a label update it stores the new value of the update counter persistently, for instance, in the local flash memory (ll. 114-115). During reboot, the node restores the last value of the counter from the persistent storage (see Listing 3 line 126), which ensures correct ordering of any subsequent membership decisions. Alternatively, a node rejoining the system obtains a new unique identifier which eliminates the problem completely.

APPENDIX G

Hierarchical Suffix-Based Routing

Routing is performed by resolving consecutive elements of the destination label starting from the maximal-position element differing at the sender (see Fig. 1). The main routing method, executed by a node on each hop, is presented in Listing 4.

Upon reception of an application message (which is different from a gossip message used by PL-Gossip to maintain the network structure), a node decrements the time-to-live (TTL) counter associated with the message and examines this counter to decide whether the message should be dropped (listing line 2-6). TTL is a mechanism for dropping messages that cannot be delivered to their receivers due to network dynamics (e.g., receiver failures). The TTL counter of a message is set by the originator of this message based on Lemma 3 (see Listing 5). More specifically, the originator resolves the minimal-level group it shares with the destination node (ll. 41-46; ), suppose the level of this group is i, and sets the TTL counter accordingly to $3^i - 1$.

If the message has not been dropped, the node determines how many elements of the destination label are left to be resolved (ll. 8-12). If there are no such elements left, then the present node is the destination and thus, it accepts the message (ll. 14-17). Otherwise, the message must be forwarded. As an optimization, the node first checks whether one of its neighbors is the destination node and if so, it forwards the message to this neighbor (ll. 19-22). If there are no such neighbors, the next hop is determined based on the routing table. More specifically, the present node looks up an entry for the next unresolved element of the destination, and forwards the message to the next hop neighbor associated with this

Listing 4. The main routing function.

```java
35   FUNCTION getNextHop(msg) {
36       // change TTL of the message
37       msg.ttl--;
38       if (msg.ttl <= 0) return null;
39   }
40   // determine if we share any group
41   int cpos = 0;
42   for (; cpos < msg.dstLab.len; cpos++) ++cpos
43   if (msg.dstLab == msg.dstLab[cpos]) break;
44   for (; cpos <= msg.dstLab.len; cpos++)
45       if (msg.dstLab[cpos] == msg.dstLab[cpos] + 1) {
46           Entry entry = this.resize(cpos - 1);msg.dstLab[cpos - 1];
47           return entry.nextHop = null;
48       }
49   return null;
50 }
```

Listing 5. The message initialization function.

```java
51   FUNCTION initMessage(dstLab, data) {
52       // create a new message
53       Message msg = new Message();
54       msg.dstLab = dstLab;
55       msg.data = data;
56   }
57   // compute TTL based on Lemma 3
58   int i;
59   for (i = 0; i < Math.min(msg.dstLab.len, this.lab.len); ++i) {
60       if (dstLab[i] == this.lab[i]) break;
61   }
62   msg.ttl = Math.min(intpow(3, i) - 1, MAX_PATH);
63 }
```
entry (ll. 24-37). Finally, it may happen that due to hierarchy
disturbance, the next hop cannot be resolved. In this case,
the node drops the message (the main routing method returns
`null`).