Home assignment for Computer Network EDA387

Group 24

October 14, 2025

Algorithm 1 Reliable Flooding for disseminating Link State Packets (LSPs) across a network.

```
1: Each node stores the latest Link State Packet (LSP) from every other node.
2: LSPs are forwarded to all nodes except the sender, preventing unnecessary retransmissions.
3: Periodic generation of new LSPs occurs, with an incremented Sequence Number (SEQNO). After a reboot, SEQNO starts at 0.
4: Upon hearing an LSP with SEQNO = $x$, a node sets its next SEQNO to $x + 1$.
5: Stored LSPs' Time-to-Live (TTL) decreases; LSPs expire when TTL reaches 0.
```

4 The Link State Algorithm and Self-stabilization

The Link State algorithm involves sending information not only to neighbors but to all network nodes. This question asks whether the link state algorithm is self-stabilizing or can become one. Algorithm 1 sketches the protocol for reliable flooding, as we learned in class. We clarify that the Link State algorithm also uses a local calculation. That is, the flooding method described in Algorithm 1 supplies the link state algorithm with a set of LSPs for each SEQNO. Using Dijkstra's shortest path algorithm, this link state algorithm enables each node to calculate the shortest path within a given graph locally.

Question 1.

- What is the model used by the Link-State algorithm? Please choose the most appropriate, correct, complete, and accurate answer. If you think there is more than one correct answer, select the ones that can help you simplify your answer to the rest of this question. Clearly justify your selection.

  1. Asynchronous shared memory network with the topology of the fully connected graph with fair communication.

2. Synchronous message-passing network with the topology of general graph and reliable communication channels.
3. Synchronous message-passing network with the topology of the general graph with fair execution.
4. Asynchronous message-passing network with the topology of the general graph with fair communication.
5. Synchronous shared memory network with topology of fully connected graph and fair communications.

- Please provide a detailed pseudo code for reliable flooding, which is used by the Link State Algorithm. Your pseudo-code should consider the model you selected above.
- Discuss the self-stabilization properties of the algorithm you have provided above. A simple 'yes' or 'no' response is insufficient for credit; your answer should demonstrate a comprehensive understanding of the topic. For example, you may analyze how the algorithm behaves when initiated from an arbitrary starting state (related to the link-state algorithm), highlighting potential disruptions in network operation without the ability to self-recover. Conversely, if you believe the algorithmis self-stabilizing, you can outline a correctness proof while emphasizing its key assumptions.

Proof. For the model choosing, the most appropriate model to chose is the model 4, *Asynchronous message-passing network with the topology of the general graph with fair communication.*, for the justifying:

- **Asynchronous** because in the Internet, all the nodes (or routers) act at their own pace, there is no a central clock for all these nodes. The time for a message from a node to another one is unpredictable, so we cannot use the synchronous one, because it contains restriction that is not correct in comparison with actual network model.
- **Messenger-passing** as stated before, because every nodes have their own clock, hence, own memory, it is impossible to have a shared memory between these nodes. Moreover, in a large network, nodes are added and removed quite frequently, a model that has a central memory will require more operation overhead to maintain, manage as well as to response to the network changes. All nodes in real topology communicate by sending and recieving messages over physical link.
- **General graph** in the real model, the network topology is not always clear, therefore, an algorithm and periodically flooding from neighbor nodes are require to discover the paths between these nodes.
- **Fair communication** is a realistic assumption. It states that if a message is sent infinitely, it will be received infinitely. On the other hand, even there are loss in communication channels, the Link State Protocol and algorithm are able to handle this loss, hence retransmit these packets.

Algorithm 2 Pseudocode for Reliable Flooding of LSPs

```
1: Initialization for node $P_i$:
2: $SEQNO_i \leftarrow 0$
3: $N(i) \leftarrow$ set of neighbors
4: $LSP[] \leftarrow \emptyset$
5: procedure PeriodicUpdate
6:     loop
7:         $SEQNO_i \leftarrow SEQNO_i + 1$
8:         $LSP_i \leftarrow generateLSP(i, SEQNO_i, TTL_i)$
9:         $LSP[i] \leftarrow LSP_i$
10:        for all neighbor $P_j \in N(i)$ do
11:            send $\langle i, LSP[i] \rangle$ to $P_j$
12:        end for
13:    end loop
14: end procedure
15: procedure OnReceive($\langle j, LSP_j \rangle$ from neighbor $P_j$)
16:    if $LSP_j \notin LSP[]$ or $SEQNO_j > LSP[j].SEQNO$ then
17:        $LSP[j] \leftarrow LSP_j$
18:        $SEQNO_i \leftarrow SEQNO_j + 1$
19:        for all neighbor $P_k \in N(i)$ such that $k \neq s$ do
20:            send $\langle j, LSP_j \rangle$ to $P_k$
21:        end for
22:    end if
23: end procedure
24: procedure DecrementTTL
25:    loop
26:        for all $\langle j, LSP_j \rangle$ in $LSP[]$ do
27:            $LSP_j.TTL \leftarrow LSP_j.TTL - 1$
28:            if $LSP_j.TTL \leq 0$ then
29:                remove $\langle j, LSP_j \rangle$ from $LSP[]$
30:            end if
31:        end for
32:    end loop
33: end procedure
```

Discussion on Self-Stabilization

Let's consider that the system starts with an arbitrary status, where nodes can send wrong Link-State package, or the packet is loss during transmission. In this algorithm, $SEQNO$ and TTL can be used to ensure the fault does not stay in the system forever.

In the pseudo code, when a node/processor P_i receives a LSP (line 16-19), considered that the packet contained fault information. But due to the flooding nature of this algorithm, P_i will receive a new packet from its neighbors, after a comparison, it keeps the packet with higher *SEQNO* (and plus 1, remarks that is an asynchronous round).

From the start, or restart event, every nodes have $SEQNO = 0$, means that they will accept any LSP packet with higher number. This ensure when a new node joins this network or a node that just recovered from fault can keep up with latest network topology.

Moreover, due to *fair communication*, the new, correct LSP will be eventually passed to all other nodes. And because LSP is periodically generated with a higher *SEQNO* (line 6-9), the correct (and newer) LSP is always accepted.

Finally, the *TTL* can ensure that even without new LSP, or the arbitrary status lasts longer than expected, node will drop expired LSP information. Every time it receives new LSP or completing a round, it can decrease the *TTL* of stored LSP information, hence remove the stale one when TTL drops to 0.

The safe configuration of this system is when all the nodes have the latest LSP information.

□

Question 2.

Please offer a self-stabilizing solution to the problem addressed by the Link State algorithm within the context of an asynchronous shared memory network, with a general graph topology and fair execution.

- Your solution should encompass its own pseudocode, which may leverage concepts and algorithms learned in class. In that case, please cite them clearly using the exact book chapter and section numbers.
- Additionally, provide a self-contained correctness proof, referencing relevant theorems and lemmas explicitly if utilized.

Algorithm 3 Token-Based Link-State Packet Dissemination

```
1: Structure to store Link State Packet:
       node {
           seqno;                                   ▷ Sequence number
           lsp;                                     ▷ Link state packet
           token;                                   ▷ Token presence indicator
       }
2: procedure OnReceiveToken
3:     seqno ← 0;                                   ▷ Node's sequence number is set to zero
4:     lsp ← GenerateLSP;
5:     token ← true;
6:     send token to next_node;
7: end procedure

8: procedure OnTimerExpired
9:     lsp ← GenerateLSP;
10:    seqno ← seqno + 1;
11:    for all each neighbor do
12:        send ⟨lsp, seqno⟩ to neighbor;
13:    end for
14: end procedure

15: procedure OnReceiveLSP(⟨lsp, seqno⟩ from neighbor)
16:    if seqno > stored_seqno then
17:        stored_seqno ← seqno;
18:        stored_lsp ← lsp;
19:        UpdateRoutingTable;
20:    end if
21: end procedure
```

Proof. Because these nodes (processors) use a shared memory, we have a *token* to ensure that only a node can enter the *critical selection* at a time. The main idea is that, a node can upon receiving a token, the node can generate *LSP* packet, sends to neighbors and also passes the token.

- **Safe configuration and legal execution** for this model, when all nodes keep the most recently data (the LSP packet that still has TTL and with the highest SEQNO), or with two neighbor nodes P_i and P_j, the LSP from each of these node should have the same SEQNO. From this state, only *expired* and *receiveToken* can happen.

- **Proof of correctness** when receiving the token, a node set its SEQNO to 0 and passed the token to neighbors, allow other nodes to generate LSP. In addition, if the system started on arbitrary state, the fault LSP packet will be erased because of the procedure *OnTimerExpired*, each node will generate new LSP and increase the SEQNO to 1.

On the receiver side, when it has a LSP packet from its neighbor, if the SEQNO of that packet is larger that the one it has, it will perform an update. This ensure that the LSP data will be frequently refreshed and only the newer is kept (line 16). Moreover, in the *OnReceiveToken*, we see that if the system is down, *token* will be passed to all nodes and they will begin a new procedure of generating LSP and broadcasting to all.

- **Convergence** because in this general graph, with fair execution, once a node sends a LSP, it will be propagated to all other nodes. With the constrain in the *SEQNO*, any stale LSP will be replaced.

□

Question 3: Comparison of Link-State Algorithms

Advantages

- **Original Link-State Algorithm:** It is efficient, scalable, and well-established, which has led to its wide use in production networks.
- **Self-Stabilizing Link-State Algorithm:** Its primary advantage is its inherent fault tolerance. The algorithm is self-stabilizing, meaning it can automatically recover from arbitrary transient faults without external intervention.

Disadvantages

- **Original Link-State Algorithm:** The main drawback is its lack of self-stabilization. It can be vulnerable to transient faults that corrupt routing tables, and it often requires significant manual configuration and careful management to operate reliably.
- **Self-Stabilizing Link-State Algorithm:** The robustness comes at a cost. These algorithms are typically more complex in their design and can be significantly harder to implement, verify, and debug compared to their non-stabilizing counterparts.