Leader Election in IEEE 1394

Ersel Hengirmen
ehengirmen@hotmail.com

Wireless Systems, Networks and Cybersecurity Laboratory
Department of Computer Engineering
Middle East Technical University
Ankara Turkey

May 12, 2024

Outline of the Presentation

Agenda

The problem

Leader Election

IEEE 1394, also known as FireWire is a standard for high-speed serial bus comminication. It is commonly used in audio and video equipment. In such networks devices are hot-pluggable, meaning they can be added or removed at any time without disrupting the system's operation. However, such changes trigger a bus reset, and after the reset, all nodes within the network are restored to an equal status, requiring a new leader to be elected dynamically.

Agenda

What is the solution/contribution

- Implementation of Leader Election Algorithm on the AHCv2 platform.
- Anlysis of the algorithms within different topologies

Agenda

Motivation/Importance - 1

The bus resets that happen after an audio or video plug-in can be disruptive, causing interruptions in the system's operation. Think as when you are on your computer watching a video you unplug your headphone and your system freezes for 3-4 seconds. That is not a situation anyone wants.

Motivation/Importance - 2

The IEEE 1394 leader election protocol dynamically assigns leadership status after bus resets, the protocol creates uninterrupted communication and coordination among interconnected devices. This is crucial for the seamless exchange of digitized video and audio signals in various electronic devices.

Agenda

Background-1

The IEEE 1394 standard mentions about the problem under the concept of isochronous resource manager (IRM) selection. It also provides examples of processes for selecting the IRM within a backplane environment. However, it doesn't go into the algorithmic details of how the selection process should be implemented.

Background-2

Additionally, Devillers et al., formalizes a simple algorithm for this IEEE standard protocol. This paper addresses ambiguities and challenges encountered during the process. The authors of the paperesspecially focuses on the tree identify phase. This paper serves as formal verification of this distributed algorithm. We will also use their method in our implementation.

Agenda

Protocol

The protocol operates in a decentralized manner where nodes in the network coordinate to form a tree structure. When a node has received a parent request from all but one of its neighbors it sends a parent request to its remaining neighbor. If a node has received parent requests from all its neighbors, it knows that it is has been elected as the root of the tree. There are 3 main states for every node of the algorihtm

Algorithm-1

Leader Election in IEEE 1394

Listing 1: Leader Election algorithm in IEEE 1394.

```
Implements:
Uses:
    send_message,  # sends message to given neighboor for asking parentage
    send_ack_message,  # sends message to given neighboor for accepting parentage
    set_timer,  # set timer after the timer ends that uses the given function with given
 arguments.
    remove_timer,  # remove timer for that neighboor
Events:
    Init,
    MessageNeighboor,
    OnMessageFromNeighboor,
    OnReceivingAcknowledgementMessageFromNeighboor,
    OnReceivingParentageRequstFromNeighboor,
Needs:
    adjacent_nodes_set: ans,
    my_node_id: my_node_id

OnInit: () do
    parent = None
    message_queue_set = set()
    remaining_ans = ans.deep_copy()
    remaining_neighboor_count = remaining_ans.length()
    If remaining_neighboor_count == 1:  # becomes a leaf
```

(continues on next page)

Algorithm-2

(continued from previous page)

```
        last_neighboor = remaining_ans.first()
        MessageNeighboor(last_neighboor)

MessageNeighboor: ( x ) do
    If my_node_id < x:
        parent = x
    send_message(x)  # will u be my father
    set_timer(x, MessageNeighboor, x)
    message_queue_set.add(x)

OnMessageFromNeighboor: ( message, x ) do
    If message is 'acknowledgement':
        OnReceivingAcknowledgementMessageFromNeighboor(x)
    Else If message is 'parentage request':
        OnReceivingParentageRequstFromNeighboor(x)

OnReceivingAcknowledgementMessageFromNeighboor: (x) do
    remove_timer(x)
    message_queue_set.remove(x)
    parent = x # lifecycle of this node ends

OnReceivingParentageRequstFromNeighboor: (x) do
    neighboor_is_the_father = False
    If parent == x:  # parent chosen as x before thus contention
        neighboor_is_the_father = True
        send_message(x)
    Elif x in message_queue_set:  # other party whose id is lower chosen me as parent
  thus contention
        remove_timer(x)
        message_queue_set.remove(x)
        root = my_node_id  # root is chosen  if it is needed for everyone to know
  broadcase message can be added

    If not neighboor_is_the_father:
        remaining_ans.remove(x)
        remaining_neighboor_count -= 1
        send_ack_message(x)
        If remaining_neighboor_count == 1:  # becomes a leaf
            last_neighboor = remaining_ans.first()
            MessageNeighboor(last_neighboor)
```

States

- Waiting for Children
- Sending Parent Request
- Parent Contention(only for last 2 nodes)

- We know that for a graph to be fully connected and acyclic, it must have exactly N-1 connections
- If a graph has no non leaf nodes it means that it has at least 2N edges which will surely make it cyclic thus contradiction
- Since it is not cyclic some of the nodes has to be leafs at every iteraion.
- Only leafs sends request to possible parents.
- When a node chooses its parent(it has to be acknowledged) it is eliminated from this process.

Agenda

After every elimination there are 3 cases:

- Either there are at least one previous leaf remaining(in this case at least one of these chooses its parents)
- There is no previous leaf remaining but since these leaf nodes are removed from our calculation new leafs are created
- Parent chosen process ends

because of above reasons this algorithm is correct. Since for every node to choose its parent. There should be exactly one of its neighboors that has not chosen it as parent. Which means up to root contention the subtrees will merge and at last point one of the roots of these subtrees will become the leader.

In addition to above, for contention instead of using randomized waits I chose to appoint node with higher number as root. To do this:

- In case the current node having lower id. It first appoints the neighboor as its root then sends parentage request. In the case of receiveng a root request from said node, it resends root request to that neighboor forcing parentship until acknowledgement received.
- n case the current node having higher id. It sends parentage request. In the case of receiveng a root request from said node instead of acknowledgement, immediately accepts.

- In case of contention(last 2 nodes) since lower id forces parentage, higher id will receive parentage request at some point and acknowledge it ending the cycle.
- In case of no contention even if it is lower or higher id since the other node doesn't send a request back it will acknowledge the parentage request.

Agenda

Conclusions

- The leader election algorithm ensures uninterrupted communication and coordination among devices after a bus reset, which is crucial for tasks like streaming audio and video.
- The implemented algorithm achieves guaranteed termination and avoids cycles, ensuring efficient leader selection in $\mathcal{O}(n)$ message complexity and $\mathcal{O}(longestBranch)$ time complexity.

References

- **IEEE Standard for a High Performance Serial Bus**, IEEE Std 1394-1995, pp. 326-327, Aug. 1996. doi: 10.1109/IEEESTD.1996.81049.
- M. Devillers, D. Griffioen, J. Romijn, et al., ”Verification of a Leader Election Protocol: Formal Methods Applied to IEEE 1394,” Formal Methods in System Design, vol. 16, pp. 307–320, 2000.

Questions

THANK YOU

Leader Election in IEEE 1394

presented by Ersel Hengirmen
ehengirmen@hotmail.com

May 12, 2024