

**Individual Project SS-20 Master I.T. (Project 18)**

**Title: Multicast Based NFV Resource Information Synchronization for a Distributed NFV Orchestration**

**Submitted By: Devan Premchandrakumar Nair**

**Matriculation Number: 1274369**

**Supervisor: Mr. Gregor Frick**

**Examiner: Prof. Dr. Armin Lehmann**

**Date of Start: 19th June 2020**

Contents

[1 Introduction 3](#_Toc47365519)

[2 Theoretical Background 4](#_Toc47365520)

[3 Requirements Analysis 5](#_Toc47365521)

[3.1 General Objectives 5](#_Toc47365522)

[3.2 Clarifying the Requirements 5](#_Toc47365523)

[3.3 Target State 5](#_Toc47365524)

[3.4 Use Cases for the Prototype 7](#_Toc47365525)

[4 Realisation 10](#_Toc47365526)

[4.1 Messages and Fields 10](#_Toc47365527)

[4.2 Solution Architecture 11](#_Toc47365528)

[4.3 Transactions and Sessions 14](#_Toc47365529)

[4.4 Analysis of Use Cases 17](#_Toc47365530)

[5 Summary and Perspectives 17](#_Toc47365531)

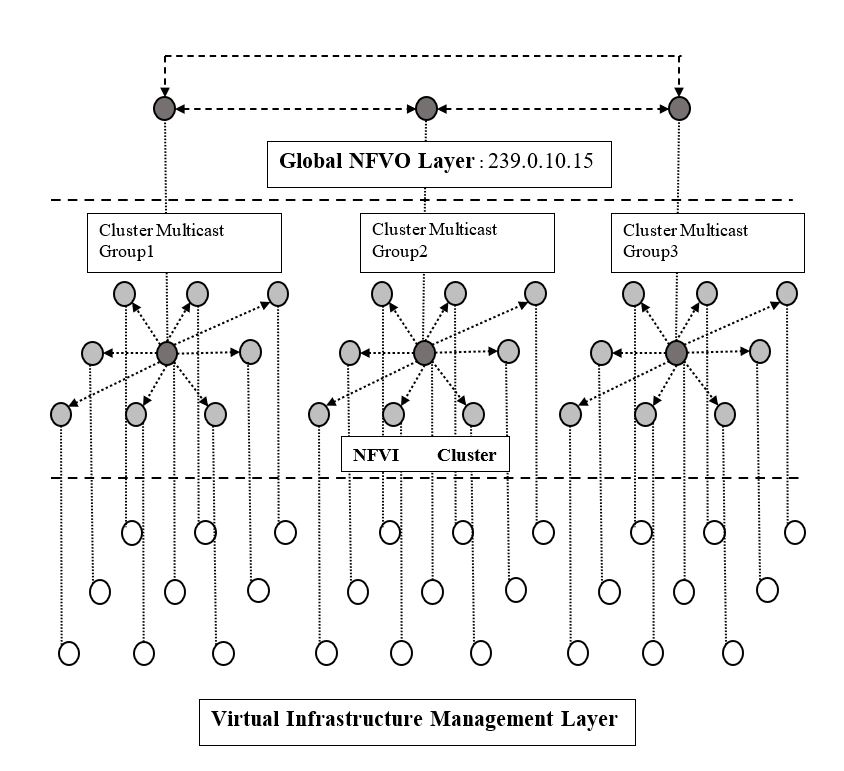
[6 Abbreviations 18](#_Toc47365532)

[7 References 18](#_Toc47365533)

[8 Appendix 18](#_Toc47365534)

# **Introduction**

# **Theoretical Background**



# **Requirements Analysis**

## General Objectives

An application level solution must be conceptualised and implemented to ensure that information of services and network resources available on individual orchestrators over a cluster network is synchronized across all the orchestrators. Also, the solution should be able to identify when instances of orchestrators fail and denote them under appropriate states. The network comprises of orchestrators which are connected to one multicast group which corresponds to a logical layer. The application solution should be periodically scheduled so that dynamical changes in resource information in one or more nodes at a given time is distributed across the whole network in a stable fashion. The application must provide suitable endpoints or API’s to access the resource and network information specific to an individual node and its knowledge about resources of all the other orchestrator nodes in the cluster, on each orchestrator node in the network. A suitable flooding strategy must be designed which would propagate updates/changes to network information/states to all orchestrator nodes in the network. At the network level there should be two sets of messages , one is communicated only to neighboring nodes and represents a keep alive message indicating validity of orchestrator node to the synchronization procedure with relevant information to represent the connection and the other is a reactive message that is flooded in the entire network when any orchestrator node makes an update to its resource information. The implementation solution must be agnostic of the underlying logical/physical topology and should track synchronization states of all nodes executing the application. There should be a common implementation of the services across all the orchestrator node in the network. The network level solution must ensure the orchestrators use only multicast messages to initiate a transaction. The application should be configurable at each node based on the network interfaces that are available and the network and service resources that the orchestrators have access to/can provision. The network must be capable of synchronizing any amount of resource and service information in the network, without any size restrictions of the synchronizing messages passed between the nodes .Each node must maintain its own database or a persistent solution to store synchronized resource information and state of synchronization of every node i.e. resource information should be synchronized even after an application reboot on an individual node (Not necessarily a simulation restart) .

## Clarifying the Requirements

The application will use CoAP for multicast message requests and responses. The resources associated with a node will be stored in a configuration file and user may be able to modify contents of these configuration files via API’s and the solution will ensure that changes to configuration file will be distributed across all the nodes in the network . The user may be able to remove any service or network resource add or update using these API’s on any node in the network .The implementation must flood the resource update/synchronize messages throughout the network, but send the keep-alive messages to the neighbors. Each orchestrator node must have two configuration sections *NETWORK SERVICES* and *APPLICATION SERVICES* referring to network resources and services that an individual orchestrator node can provide. The node must have a *HOSTID* section in configuration indicating a character string ID for the orchestrator node. The *HOSTID*  is unique in the entire cluster. The network topology will consist of multiple orchestrator nodes in a mesh configuration such that failure/collapse of one node will not lead to an instance of a network partition. In addition each node must be able to estimate the states of orchestrators as ‘synchronized’, ‘connected to failed nodes’ and ‘unknown’ or equivalent semantic representations symbolizing nodes having the capability to synchronize orchestrator resource/s, links to the synchronized sector that have failed and nodes partitioned from the present cluster of orchestrators and there is no network path to verify their resource synchronization, respectively. Each node which runs the application will have files to persist information about the resource information about each node in the cluster with the relation *HOSTID -> NETWORK SERVICES , APPLICATION SERVICES*. Each node will flood message whenever the resource information for any of the node changes. The solution will ensure that there are fail-safe transactions to maintain same version of information in a synchronized sector .The multicast messages in the whole network should be non-confirmable to prevent unnecessary repeated requests. The requests and response in synchronization message sequence should be JSON formatted .

## Target State

An example mesh network shown in the figure below was chosen to test different aspects of the solution. In the desired final state , each partition in the mesh should be synchronized with the latest resource information of all nodes in that sector. In case of a network partition, each node in the partition should mark the host ID’s of all nodes synchronized with its information , the host ID’s whose states cannot be synchronized because the network connectivity to those nodes have been lost and the list of nodes ,to which links of the synchronized sector has gone down thereby causing the partition. As a result of this state estimation each node in the network must be capable of determining the connectedness of each node with every other node whose resource information it is synchronizing.

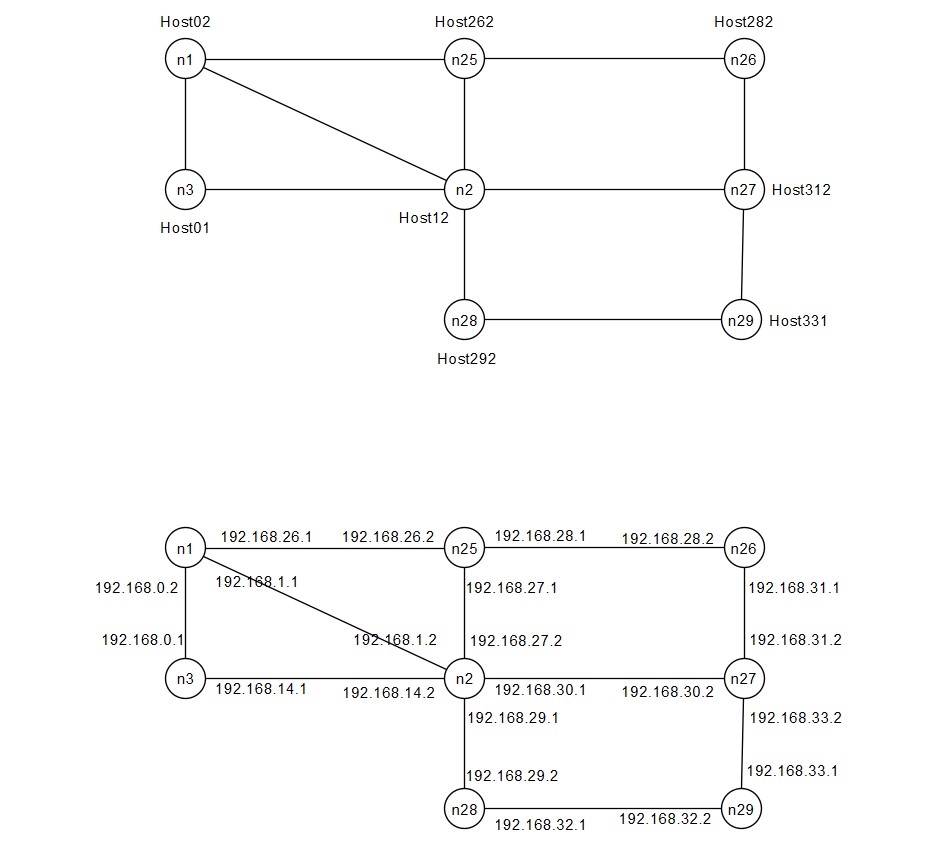


Figure 1:Example Topology used to analyze operation of the solution

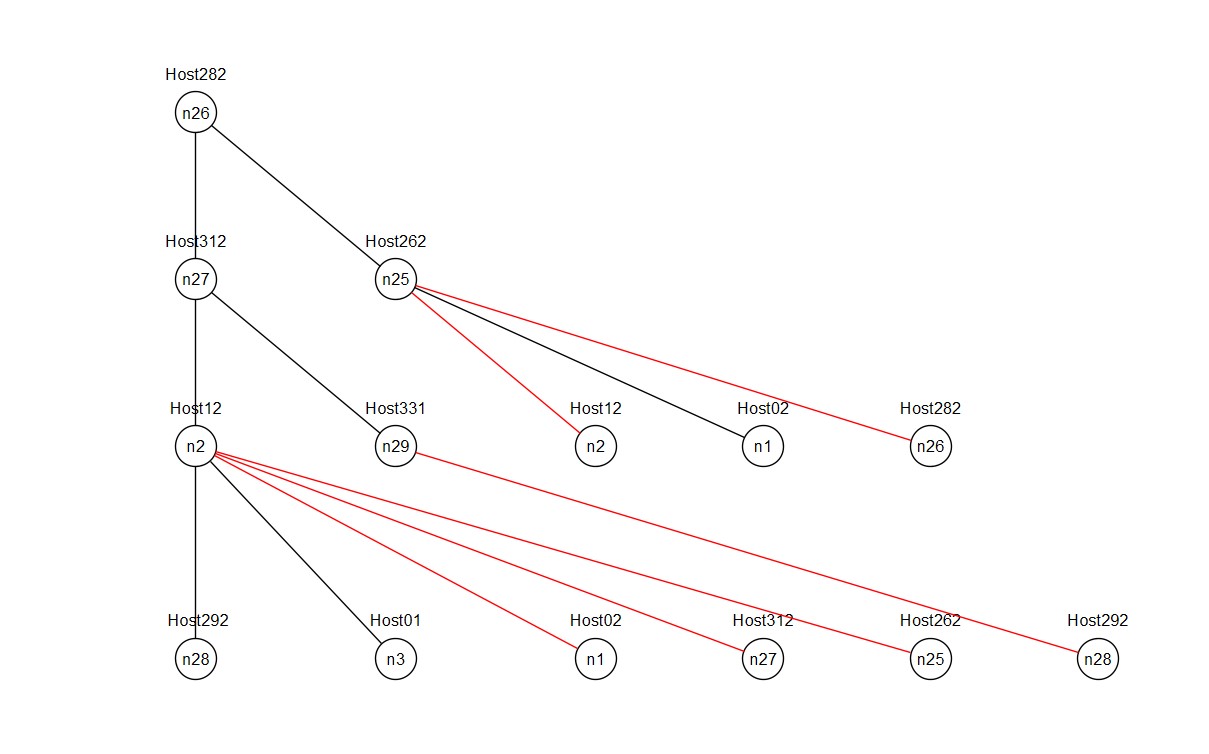


Figure 2: Estimating the connectedness of the topology for node n26 (as root node) using Breadth First Search Algorithm.

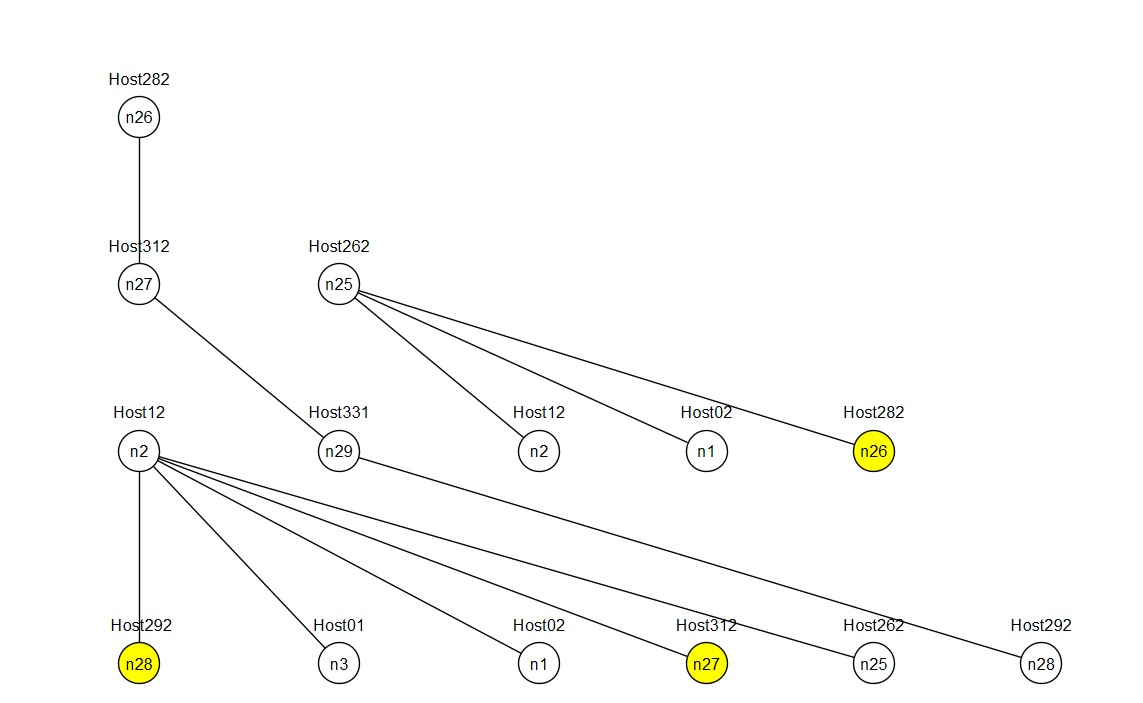


Figure 3:Estimating the connectedness of the topology for node n26 (as root node) during an instance of network partition using Breadth First Search Algorithm

## Use Cases for the Prototype

### Updating Resource Information of a node at runtime

In the given topology, the resource information of node n3 is changed at runtime and the rest of the nodes in the network receives the flooded Update message for the new resource information which has to be replaced instead of the old one. Node n3 sends each resource information with a version/sequence ID which should also be noted by all nodes to keep track of resource information and to make sure that nodes do not get desynchronized.

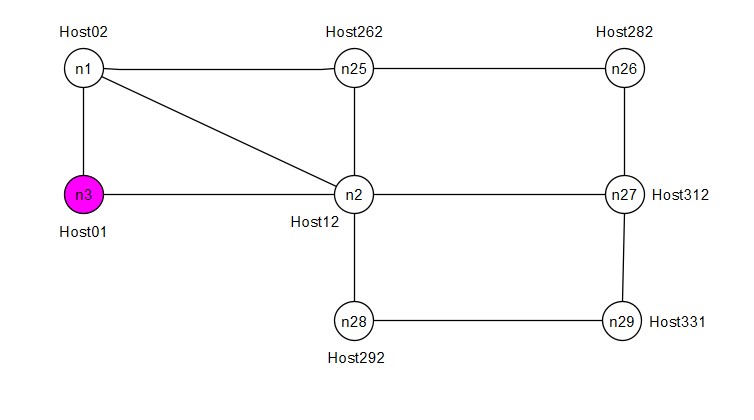


Figure 4:Node n3 changes resource information at runtime

### Node injected into an already synchronized network

The network initially starts up and synchronizes without the application running on node n29. So, the entire network except n29 synchronizes all its information. The network does not have knowledge of existence/connectedness to n29 at this point. After a while, the application is started on n29. Some time after this, n29 has resource information of all the nodes in the network and the rest of the network have updated their resource tables to include an added node n29/Host331 and estimates connectedness to the newly injected node into the network.

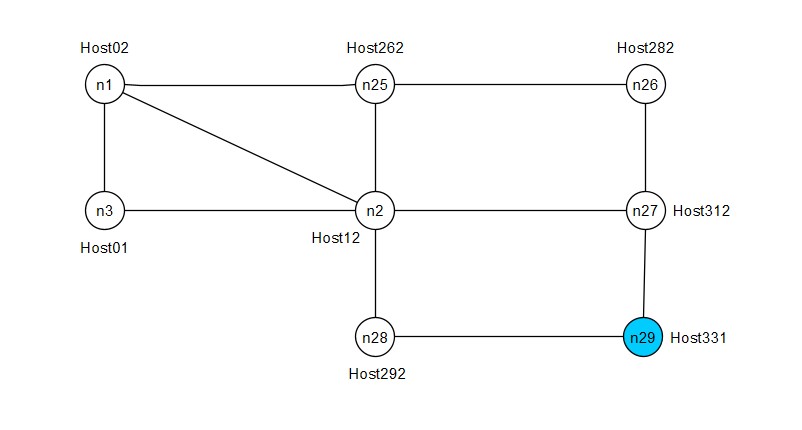


Figure 5:Node n29 injected into the network at runtime

### All Interfaces of a node failing without causing Network Partition

In this scenario, node n25 shuts down all its interfaces at runtime after the entire network has synchronized with resource information of all the nodes in the topology. After some time, all the nodes in the network estimate n25 to be unconnected in the topology and mark interfaces to it having failed or a node-failed state incapable of being synchronized with resource information updates of the rest of the network.

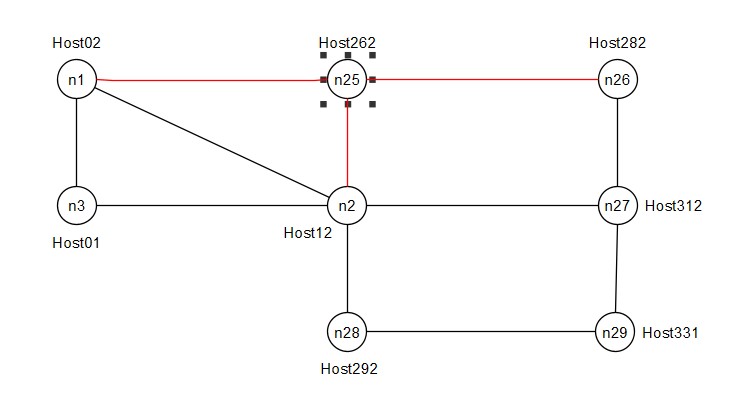


Figure 6:Instance of all interfaces of a node failing without causing a network partition

### Interfaces of multiple nodes failing causing Network Partition

In this case all the interfaces of nodes n25 and n2 fail resulting in a network partition scenario. The two partitions that can be observed are A(n1,n3) and B(n26,n27,n29 &n28). All nodes in each partition must mark its members as synchronized while the members of the other partition as unknown. It should also mark the host ID’s of nodes whose links have failed causing the synchronized segment to be unconnected from the unknown state nodes.

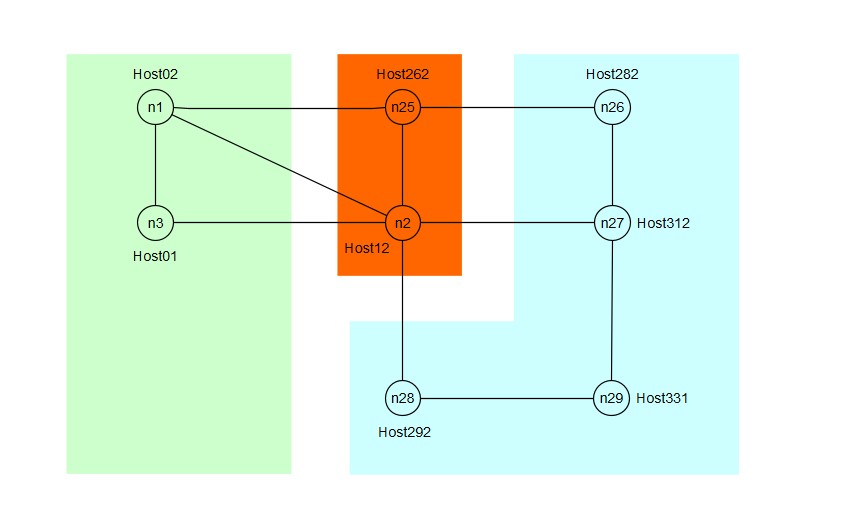


Figure 7:Figure 1:Possible case of a network partition to be dynamically detected by all nodes

### Interface of node fails without causing Network Desynchronization

In this scenario the interface of n29 connected to node n27 is brought down at runtime some time after the network has synchronized. The entire network still manages to synchronize the resource information/keep alive statuses of n27 from the other interface. All the resource information will be synchronized and none of the node will mark a network disconnect with n29.

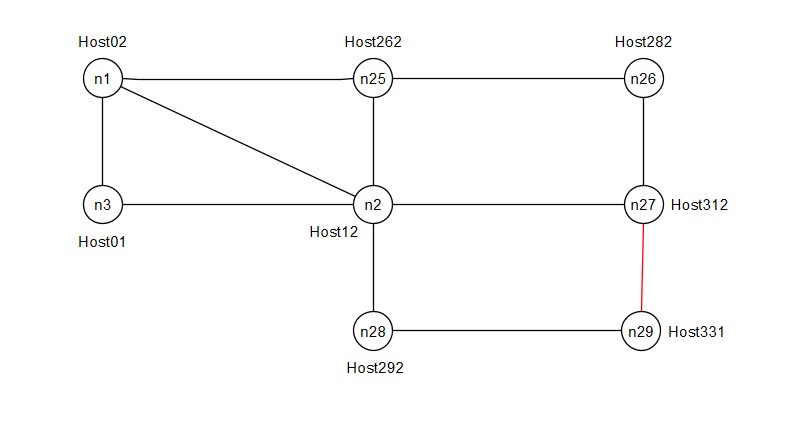


Figure 8:One of the interfaces of node n29 fails during runtime

# **Realisation**

## Messages and Fields

### NS-Hello (Network Synchronization-Hello)

NS-Hello is a POST like keep alive message used to inform a neighbor’s active state in synchronization process. Each node keeps track of the versions of resource information of all the nodes in the network using the NS-Hello message. It is basically a list of resource information version and neighborhood graph for all nodes in the network and the host ID of the node that is sending the message. So, each entry for a node will have primarily have a *hostID* field and a *globalTopologyLedger*  field. For all nodes in the *globalTopologyLedger*  field, there are primarily 3 subfields.

* *neighbor:* For all the nodes in the network , this will keep track of list of immediate neighbors. Only the node which is sending the NS-Hello message can modify its own entry in the field and update the nodes entire knowledge of the topology in this section and forward it to all its neighbors.
* *version:* This keeps track of the NS-Hello message version (abbr. as HV henceforth in the diagrams) for the given entry of host ID. Each node which receives this message from neighbor uses this version field to update its knowledge of topology of the nodes it is not directly connected to.
* *resVersion:* This field keeps track of the resource information version for that entry of host ID. Every time a node makes an update to its resource information ,it must update this entry in its local queues and also in this message so that other nodes in the network (e.g. newly added nodes to network) may keep track of the current version of resource information for each node.

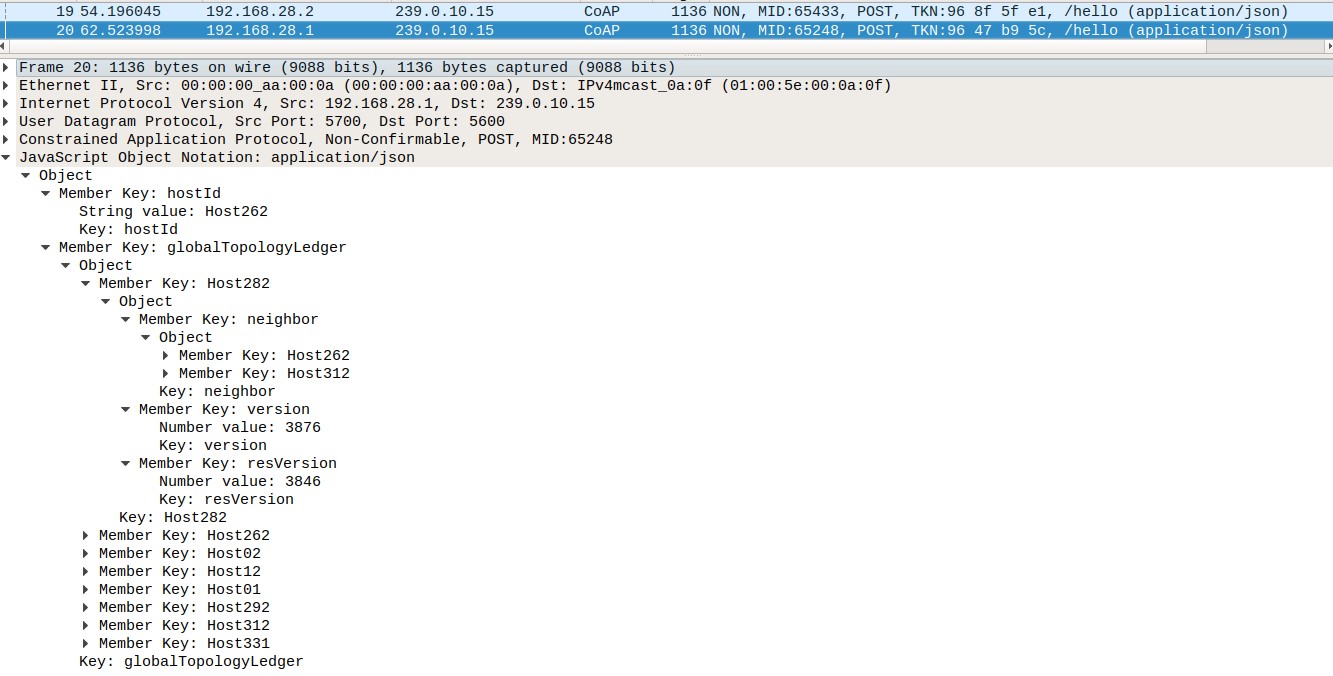


Figure 9:Example of a NS-Hello message sent by node n25 in Fig.1

### NS-Synchronize (Network Synchronization-Synchronize)

NS-Synchronize message is sent by a node only when it detects a change of resource information in the present node. This message is flooded throughout the network with the help of a flooding queue and a scheduler in each orchestrator node. So if a node receives a NS-Synchronize message then it checks its local queue of resource information and its version and if the received information is more recent then it updates its local resource information table and pushes that entry to the flooding queue, else it drops that particular message which is possibly a duplicate. The scheduler during its scheduled time-slice for firing will check flooding queue and if there are entries present it will forward it through all of its interfaces (The process is explained in subsequent sections in more detail). Primarily the message is a list of *hostID* and *nfvresources* for all the messages in queue of a node.

* *hostID:* Host ID of the node which fired the resource information update message.
* *nfvresources:* Resource Information of the node that must be distributed throughout the network.
* *version:* Version of the Resource Information being flooded/updated

In addition to the POST-like version shown in the figure below, there is also a request type version of this message, which is queried to all immediate neighbors at the multicast address, whenever a node detects an updated version for a particular node in the NS-Hello message but has a less recent version of the resource information in its local queue. This is primarily meant as a fail-safe transaction to prevent any potential desynchronizations arising in part due to possibility of resource information updates happening during an instance of network desynchronization.

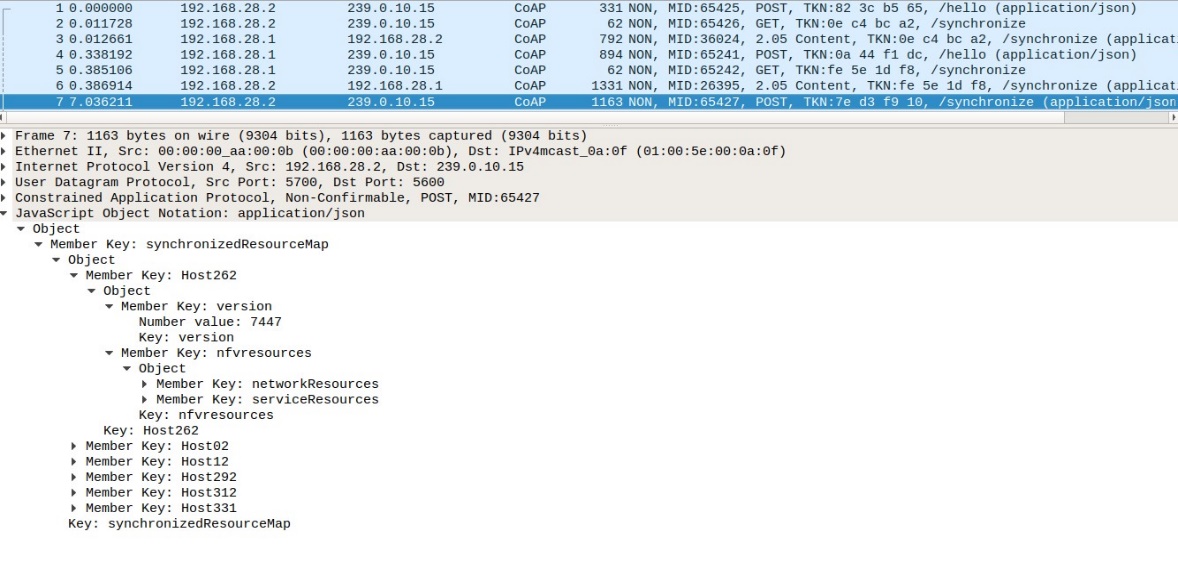


Figure 10:An NS-Synchronize message sent by node n26 in the Fig.1 topology at the interface 192.168.28.2

## Solution Architecture

### Description

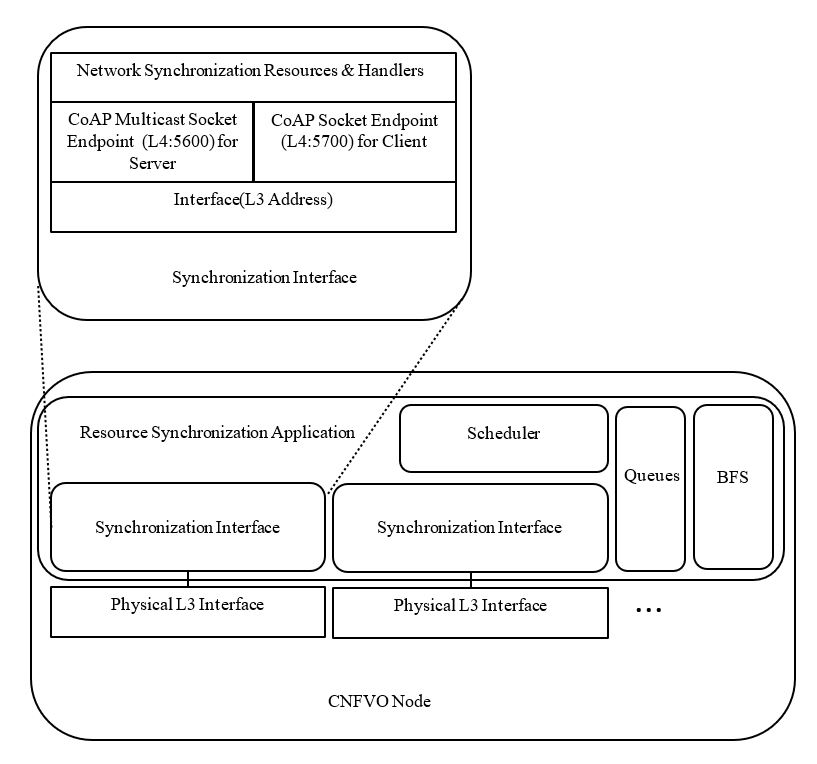


Figure 11:Software Architecture

The solution has Synchronization Interface(SI) module corresponding to the physical interfaces which must be enabled with the synchronization procedure. Each SI has a series of CoAP event handlers and triggers to initiate processing or sending of incoming and outgoing messages, respectively. The message sending and state estimation are triggered via a scheduler which fires up these modules within the application. The scheduler also periodically checks for updates to its local resource information file and fires up the flooding mechanism whenever it detects a change in its local resource information. The information regarding topology and resource information are stored in queues and maps within the application which are persisted by the scheduler during periodic intervals or whenever an update happens. The Breadth First Search Algorithm (BFS) module is used to compute the connectedness of the topology information available to the node and to estimate the states of different nodes within the network which are used to identify if a network partition has taken place or the list of nodes that are synchronized in the current partition.

### Flooding Mechanism

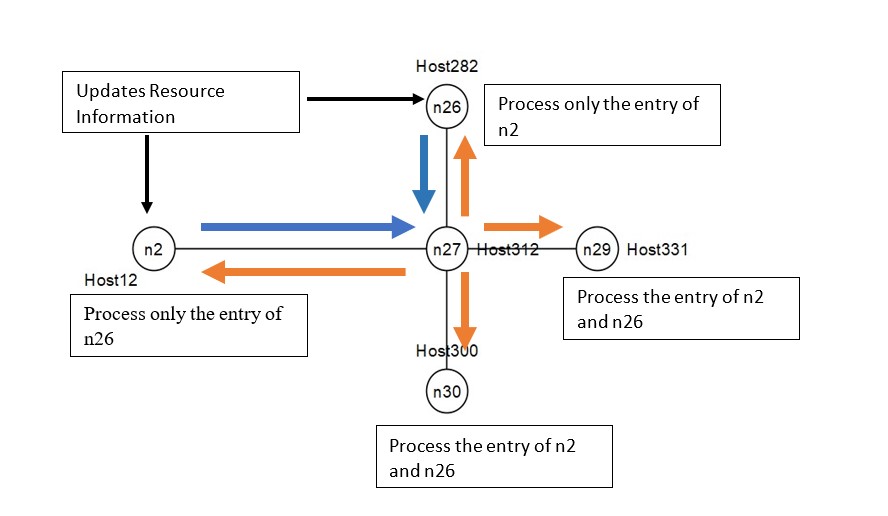


Figure 12:Mechanism and scheduling of flooding queue

The scheduler in a node fires up the flooding mechanism at periodic intervals(in the current implementation every ~12 sec) if any NS-Synchronize message is present in the flooding queue. Each node on an update of its local resource information updates its resource version ID and sends the updated resource information to all its neighbors. The receiving node in turn checks the resource information version and compares it with the version present in its own local map of resource information. If it finds that resource information with the given version is present ,it drops it. If the received packet is of a later version then, it updates its own local resource information queues and maps and pushes the information to the flooding queue. During the scheduled time-slice for flooding it will forward all the packets in the queue to all its neighbors. These neighbors will in turn check version of entries to determine if they should be pushed to the flooding queue or not. In the diagram shown above, we see the process of n27’s scheduler when it receives an NS-Synchronize update message from n26 and n2. It updates its own local information tables and floods two entries to all the nodes in its neighborhood. Nodes n26 and n2 receive duplicated copies of its own information which it drops, but updates the resource information of the other entry, meanwhile n27,n29 and n30 processes both the resource information update and floods as mentioned above .At steady state , all the nodes will have updated its resource information tables and would no longer flood the NS-Synchronize messages.

### Connectedness and State Estimation

Each node in the network estimates connectedness to other nodes using Breadth First Search Algorithm.

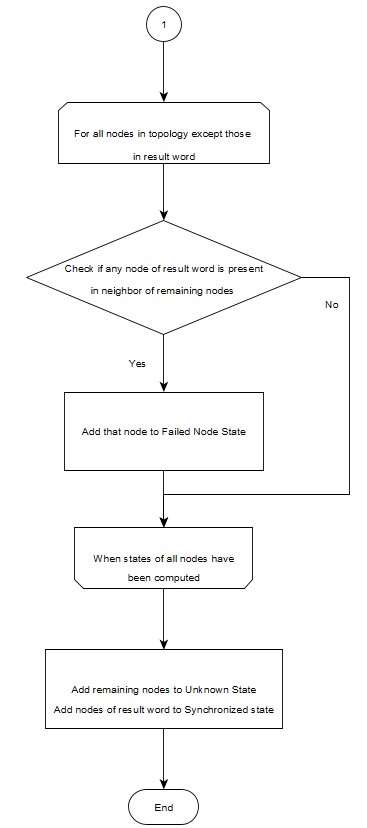
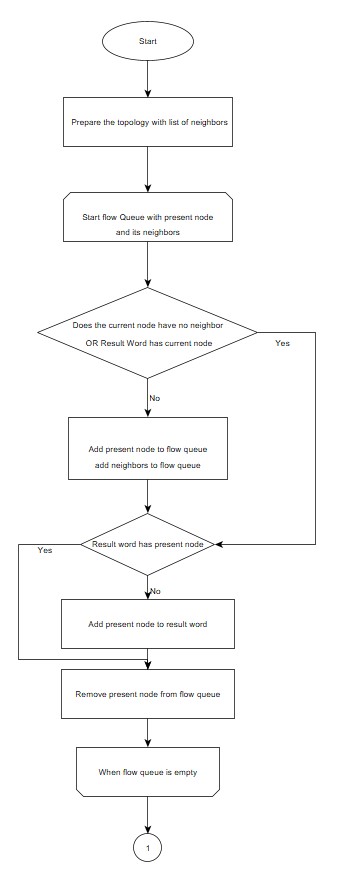


Figure 13:Breadth First Search Algorithm

## Transactions and Sessions

### NS-Hello

The figure below shows the interactions taking place to communicate the topology to all the nodes in the network.

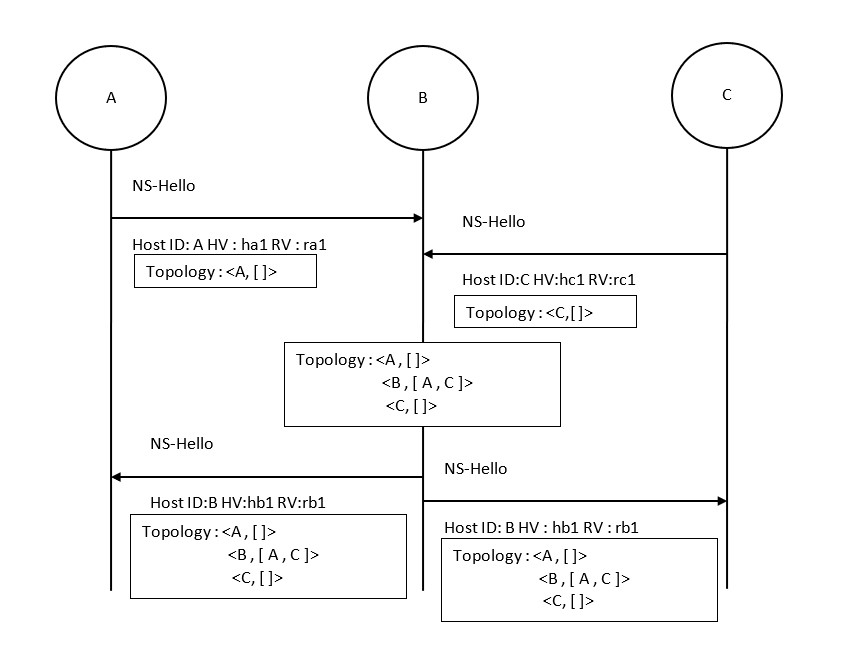


Figure 14:NS-Hello Operation

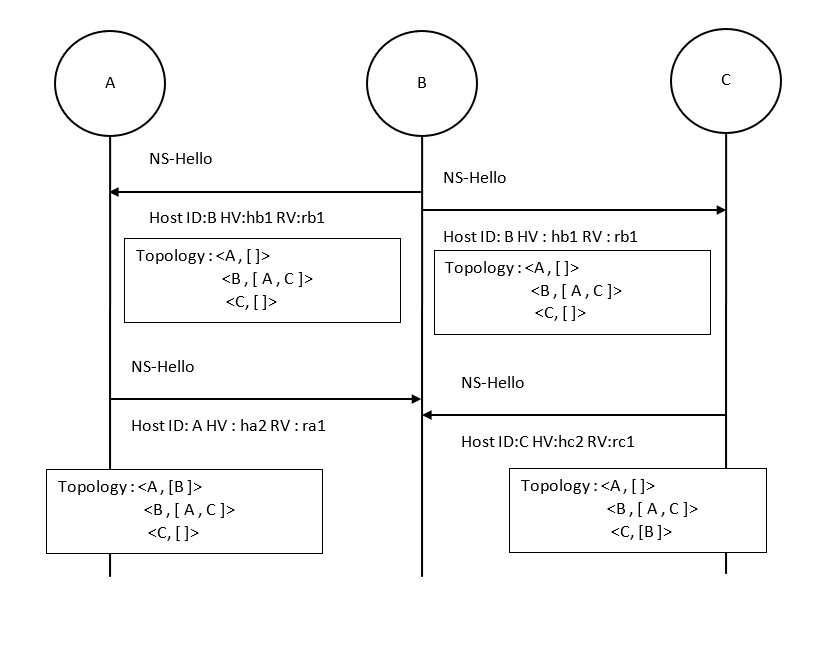


Figure 15:NS-Hello Operation

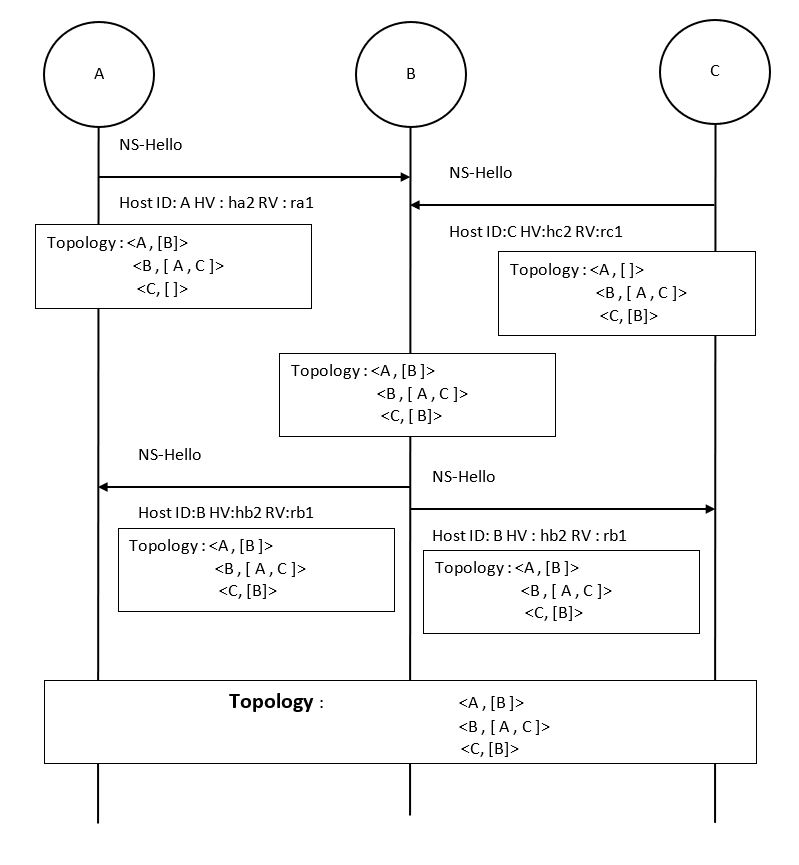


Figure 16:NS-Hello Operation

### NS-Synchronize

The figure below shows an example scenario explaining the messages exchanged when node D updates its resource information and the application attempts to synchronize the information to rest of the nodes.

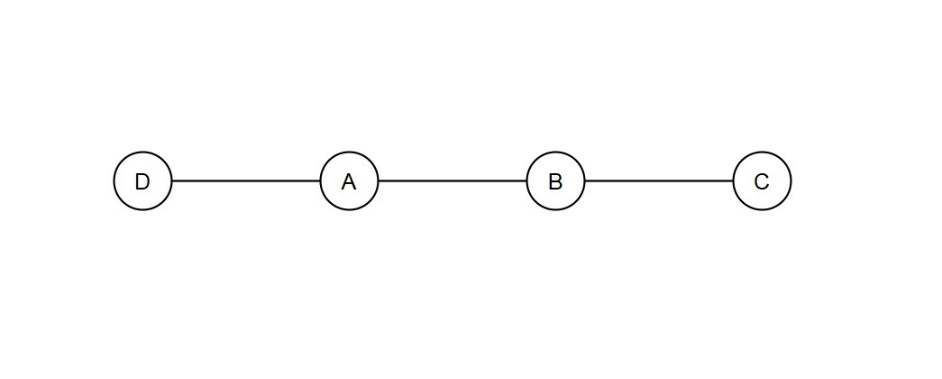


Figure 17:NS-Synchronize Operation Example topology

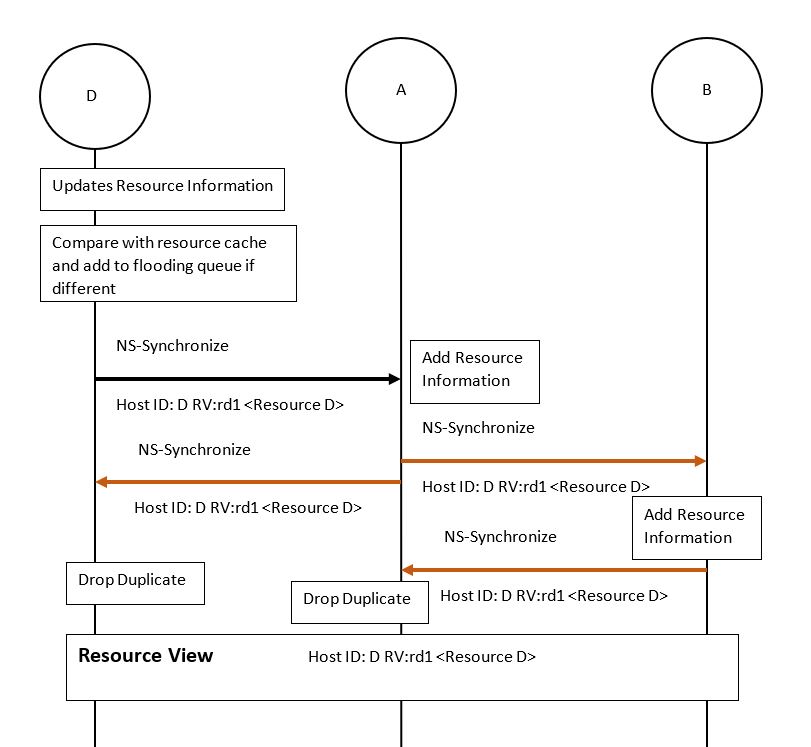


Figure 18:NS-Synchronize Operation

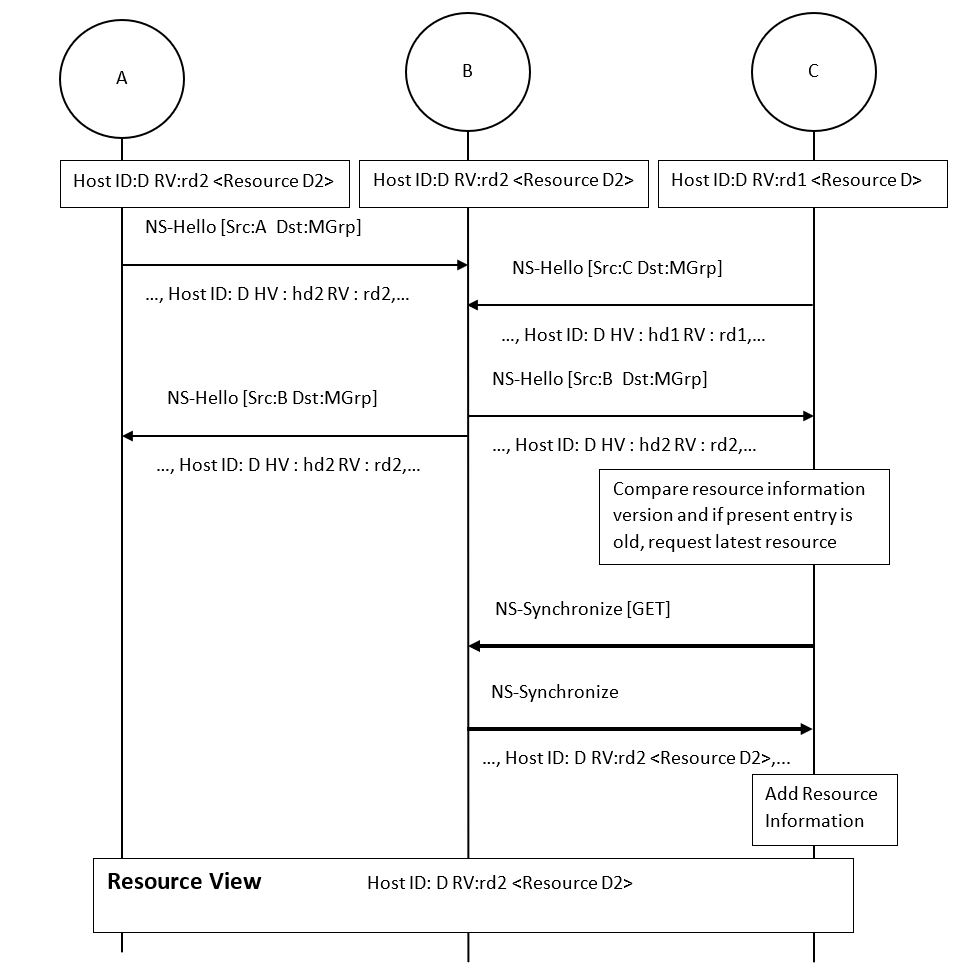


Figure 19:NS-Synchronize Operation

The figure above shows an alternate path which may be taken by application on a node to update its resource information, if the version update information is propagated via the NS-Hello transactions first and then the resource information happens via NS-Synchronize request type of messages.

## Analysis of Use Cases

### Updating Resource Information of a node at runtime

When n3 makes a change to its resource information, it floods it to the rest of the network using NS-Synchronize messages.

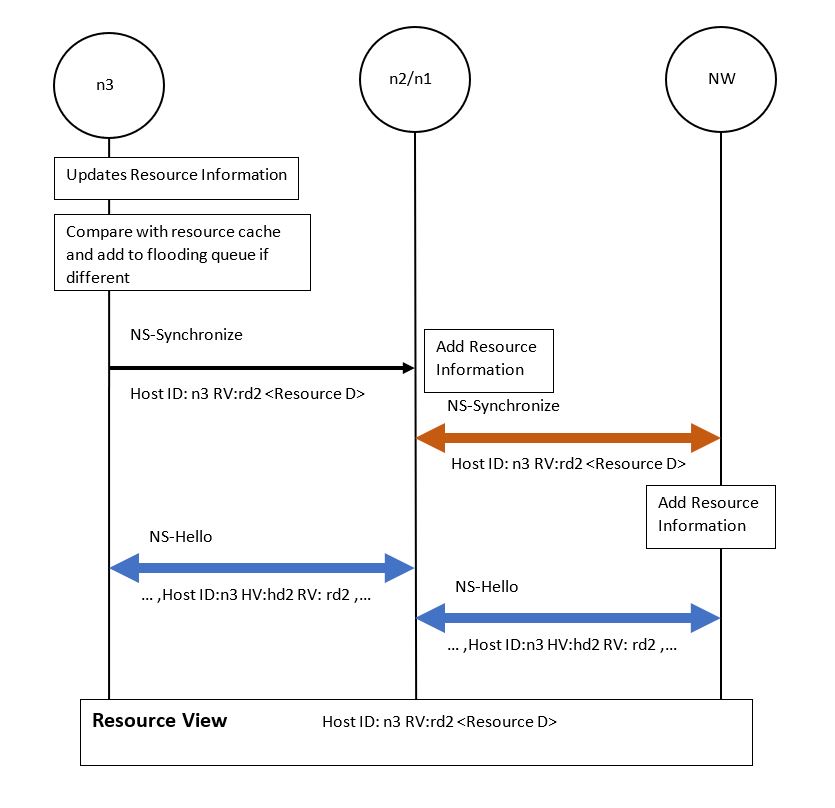


Figure 20:Transactions involved in a resource information update at runtime. (NW denotes rest of the network)

### Node injected into an already synchronized network

When a node n29 is injected into an already synchronized network, it floods the network with its resource information using NS-Synchronize messages. Node n29 gets information of the rest of the network from NS-Synchronize request type message sent to multicast address of its neighbor which was already synchronized with rest of network’s information. In the end the entire network is aware of n29’s resources and its neighborhood relationships with n27 & n28.

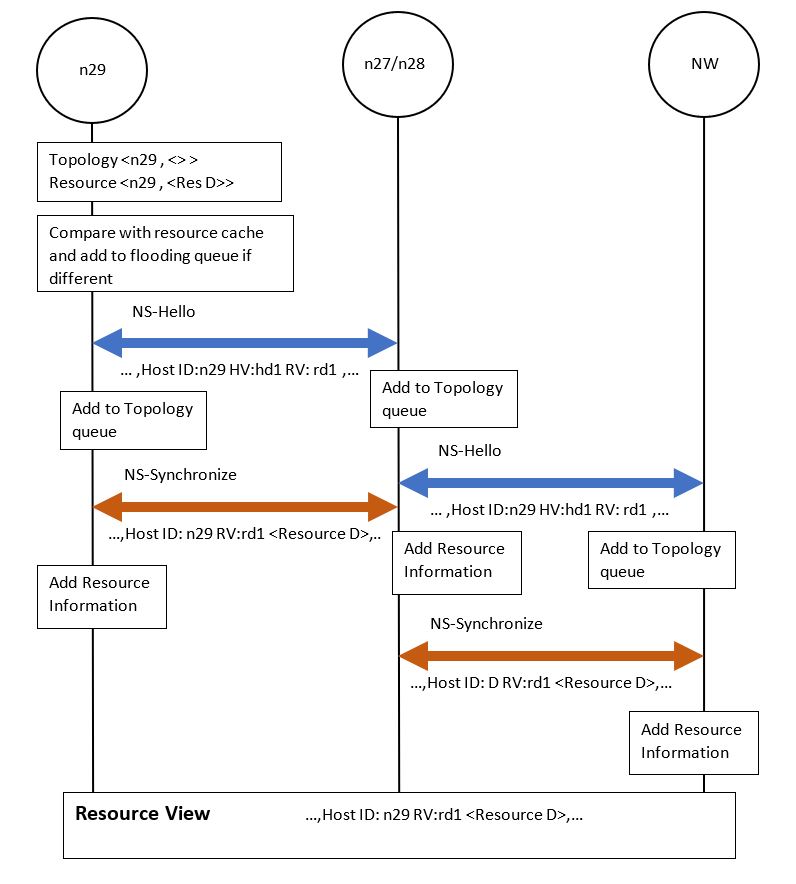


Figure 21:Transactions involved in injecting a new node into an already synchronized network. (NW denotes rest of the network)

### All Interfaces of a node failing without causing Network Partition

For the example topology consider when all interfaces of node n25 fail. Then we get a neighborhood relationship graph as shown below for the rest of the network. By applying the BFS Algorithm we can see the connected nodes include all nodes in network except n25. And since n25 could not communicate its own topology change to other nodes, the other nodes may estimate its desynchronization by checking the neighbors of any of the synchronized nodes. In this case the neighbor of n2,n1 and n26 i.e. n25 is desynchronized.

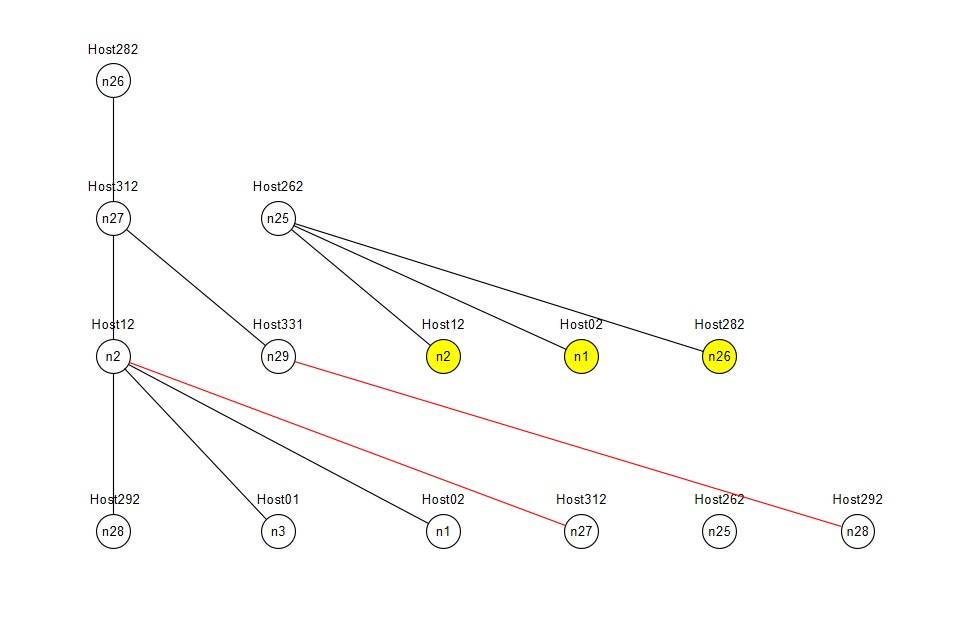
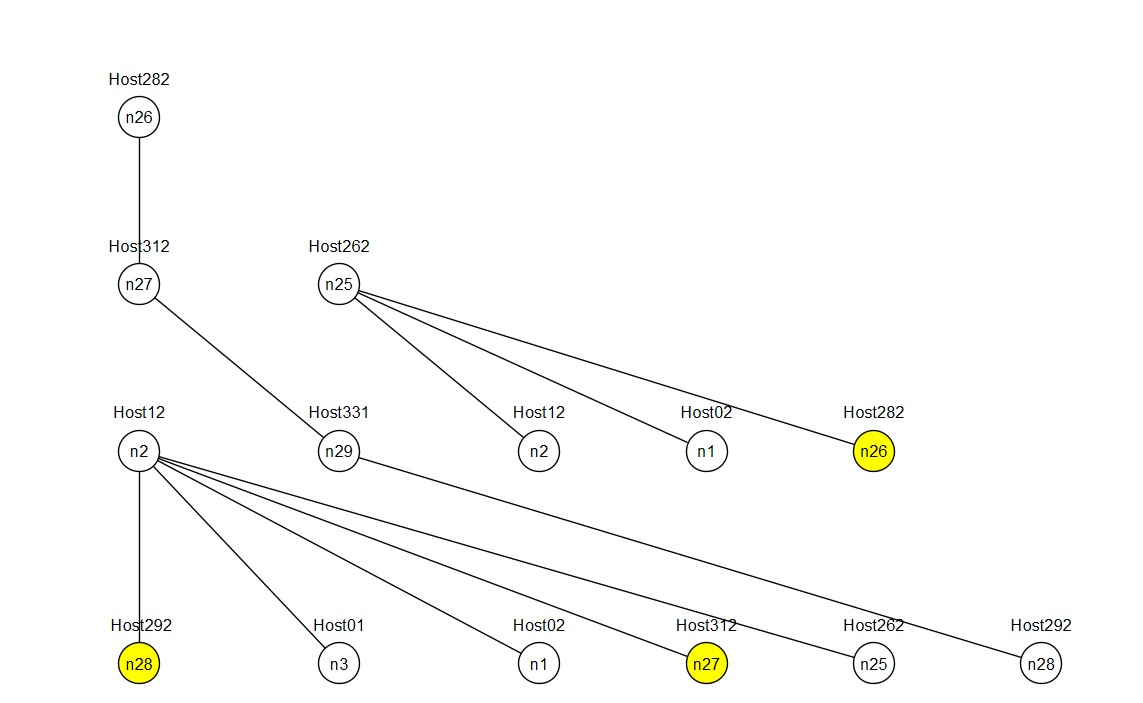


Figure 22:Network Graph for connectedness estimation.

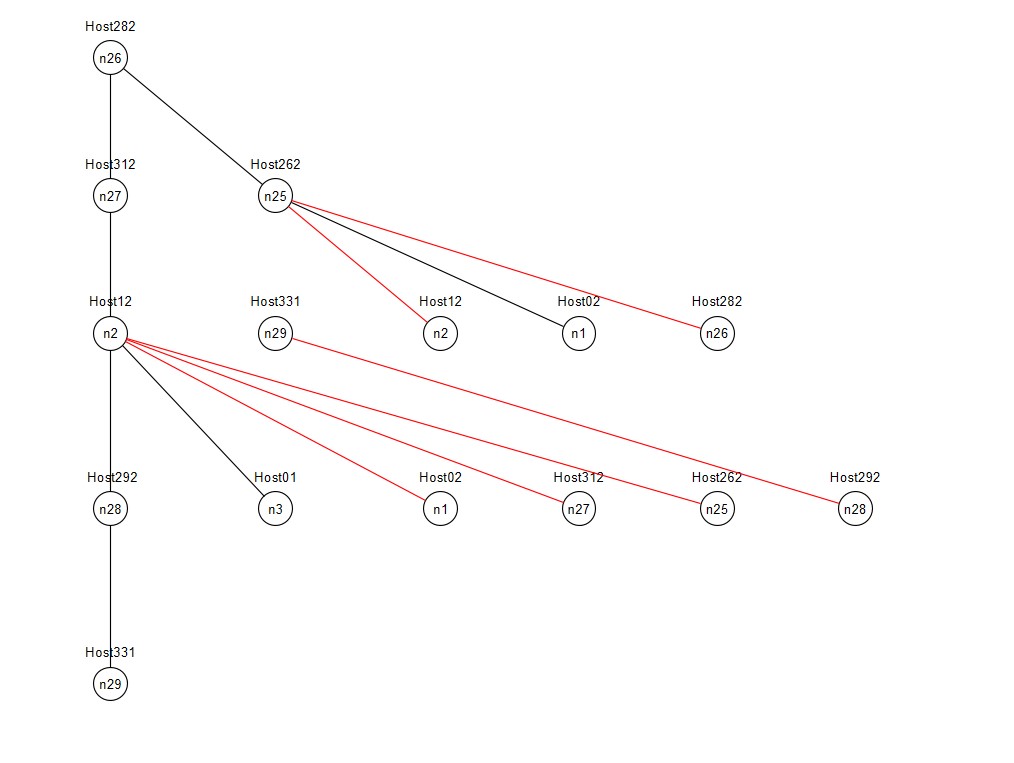
### Interfaces of multiple nodes failing causing Network Partition

The same scenario as mentioned above when coupled with all interfaces of n2 also failing demonstrates a case of network partition as shown below. The graph is calculated at n26, so the resulting connected nodes computation shows the partition which is synchronized with n26. In the same fashion as previous case if we compute the direct neighbors of the nodes of the synchronized sector, we ought to get the nodes whose interface failure has caused this network partition. The second connected neighbor to that would be the nodes whose states cannot be determined(hence marked unknown) because there is no UDP path to those nodes(In this case node n1 & n3).



### Interface of node fails without causing Network Desynchronization

Consider the scenario where one of the interfaces between node n27 and n29 collapses then, the two nodes deregister each other from the neighborhood relationships in NS-Hello’s topology sector. Then at any given node one may find equivalents of the graph below. Since there is another UDP path connecting n27 and n29 to the rest of the network, neither of the nodes get desynchronized from resource synchronization procedure and the connectedness estimation computes a fully connected network as shown below.



# **Summary and Perspectives**

# **Abbreviations**

**BFS:**

Breadth First Search (Algorithm)

**CoAP:**

Constrained Application Protocol

**CNFVO:**

Cluster Network Function Virtualization Orchestrator

**NFV:**

Network Function Virtualization

**NFVI:**

Network Function Virtualization Infrastructure

**NFVO:**

Network Function Virtualization Orchestrator

**VIM:**

Virtual Infrastructure Management

# **References**

# **Appendix**

## Screenshots of Solution Operation during various scenarios



Figure 23:NS-Synchronize message during a resource information update or new node injection

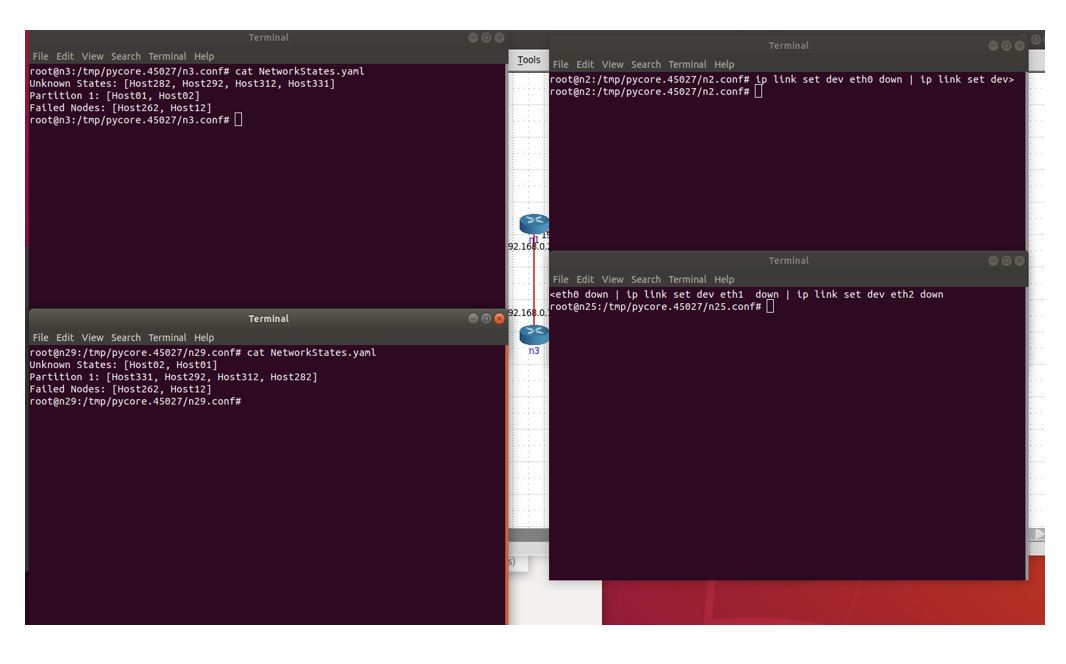


Figure 24:Demonstration of a network partition scenario identification.

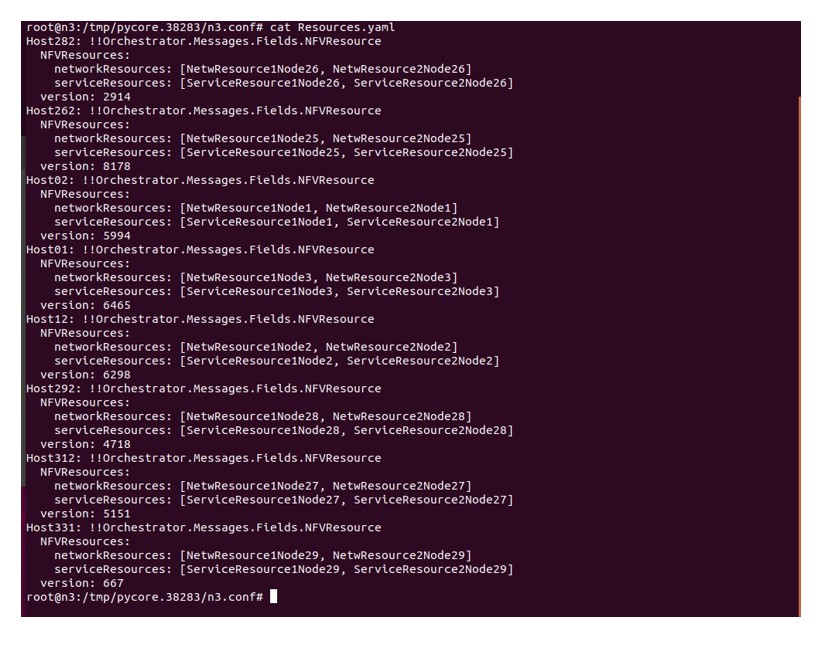


Figure 25:Example of global resource information view