Cloud Computing Assignment REPORT

Online power Aware coordinated virtual network embedding with 5G delay constraint

Team Members

Abhinav Gupta-(191CS201)
Anupam Lal-(191CS209)
Shubhanshu Singh-(191CS250)
Vamshikrishna M-(191CS261)

Submitted To

Dr. Sourav K. Addya
Asst Professor
CSE Department
NITK Surathkal

INTRODUCTION

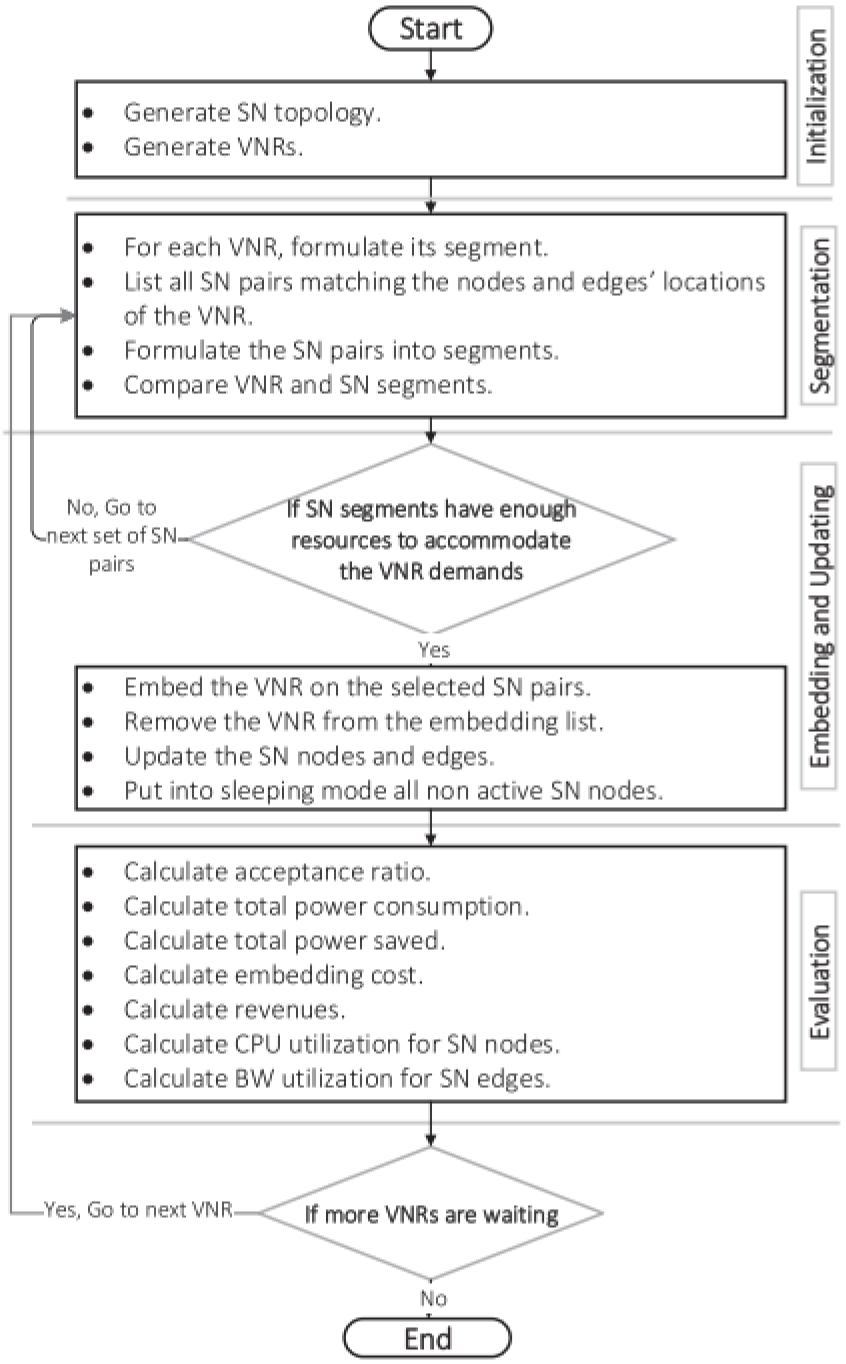
The paper "Online Power Aware Coordinated Virtual Network Embedding with 5G Delay Constraint" published in the Journal of Network and Computer Applications presents a novel algorithm for virtual network embedding (VNE) that aims to achieve power-aware coordination while considering the delay constraints of 5G networks.

The algorithm proposes a strategy to minimize the energy consumption of physical network resources while ensuring that the delay requirements of virtual network requests are met. This paper proposes a new solution for the two subproblems of the virtual network embedding problem, by converting the structure of any virtual network request into a collection of pairs of virtual nodes and their edge, where the demands of each pair will be listed together in a set format called a segment, and converting the resources of an exact similar number of substrate network pairs into segment format as well.

The proposed algorithm in the paper focuses on coordinating VNE with the aim of minimizing power consumption and meeting delay constraints of 5G networks.

The algorithm takes into account both the power consumption of physical nodes and the delay requirements of virtual network requests during the embedding process. It aims to optimize the embedding process by considering both power awareness and delay constraints.

The OPaCoVNE methodology is shown in the flowchart shown in Fig. At each time t if a VNR^r arrives, the code structures its set of segments, and based on the demanded locations for the virtual nodes, the code will build a similar SN topology to the VNR^r topology, then it formulates the SN set of segments and compares both sets to check if the candidate SN segments have enough resources to host all demands of VNR^r during the time interval. At each iteration, the algorithm keeps checking whether any VNR has expired, in order to remove its demands from the hosting resources, and it updates the whole SN accordingly. In parallel with that, at each iteration cycle, OPaCoVNE evaluates the power consumption of each SN node if it is idle or loaded, and turns it off, if it has zero utilization, to save the overall power consumption in the SN.


Start
Initialization
Generate SN topology.
Generate VNRs.
Segmentation
For each VNR, formulate its segment.
List all SN pairs matching the nodes and edges' locations of the VNR.
Formulate the SN pairs into segments.
Compare VNR and SN segments.
Embedding and Updating
No, Go to next set of SN pairs
If SN segments have enough resources to accommodate the VNR demands
Yes
Embed the VNR on the selected SN pairs.
Remove the VNR from the embedding list.
Update the SN nodes and edges.
Put into sleeping mode all non active SN nodes.
Evaluation
Calculate acceptance ratio.
Calculate total power consumption.
Calculate total power saved.
Calculate embedding cost.
Calculate revenues.
Calculate CPU utilization for SN nodes.
Calculate BW utilization for SN edges.
Yes, Go to next VNR
If more VNRs are waiting
No
End

WorkFlow

1. Input: The algorithm takes a set of virtual network requests and the physical network topology as input.
2. Initialization: The algorithm initializes the virtual network embedding process by selecting a virtual node from the request and a physical node from the physical network topology.
3. Power-aware Embedding: The algorithm embeds the selected virtual node onto the selected physical node in a power-aware manner, taking into account the energy consumption of the physical node.
4. Delay Constraint: The algorithm ensures that the delay constraints of the virtual network request are met by considering the communication delay between the virtual nodes and the physical nodes.
5. Virtual Link Embedding: The algorithm embeds the virtual links of the virtual network request onto the physical links of the physical network topology, taking into account the capacity constraints of the links.
6. Success Reporting: The algorithm reports the successful embedding of the virtual network request, including the energy consumption and the embedding results.
7. Loop: The algorithm repeats the above steps for all the virtual network requests.

Work Done

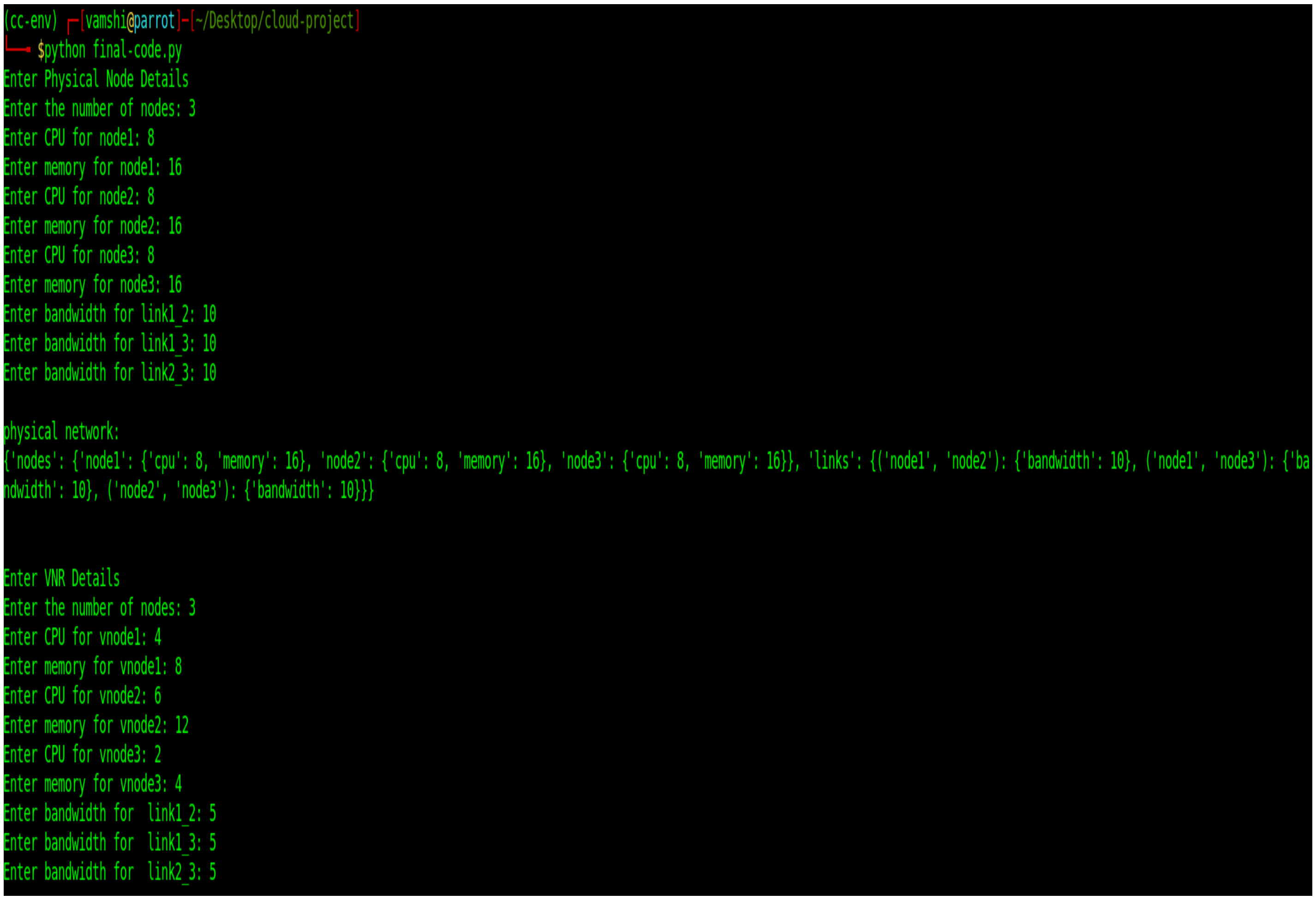
The VNE problem is about mapping a set of virtual networks onto a physical network. The physical network is represented as a graph where each node has CPU and memory resources, and each link has a bandwidth capacity.

The code defines the physical network topology and the virtual network requests. Then, it defines two cost functions, one for node mapping and another for link mapping.

1. Initialize an empty dictionary mapping to store the mapping of virtual nodes and links to physical nodes and links.
2. Iterate through each virtual node in the virtual network request.
3. For each virtual node, iterate through each physical node in the physical network.
4. Check if the physical node has enough resources (CPU and memory) to map the virtual node.
5. If yes, add the mapping of the virtual node to the physical node in the mapping dictionary, update the available resources in the physical node, and move to the next virtual node.
6. Iterate through each virtual link in the virtual network request.
7. For each virtual link, iterate through each physical link in the physical network.
8. Check if the physical link has enough bandwidth to map the virtual link.
9. If yes, add the mapping of the virtual link to the physical link in the mapping dictionary, update the available bandwidth in the physical link, and move to the next virtual link.
10. Return the mapping dictionary containing the mapping of virtual nodes and links to physical nodes and links.

RESULTS

Sample input of physical Resource and VNR

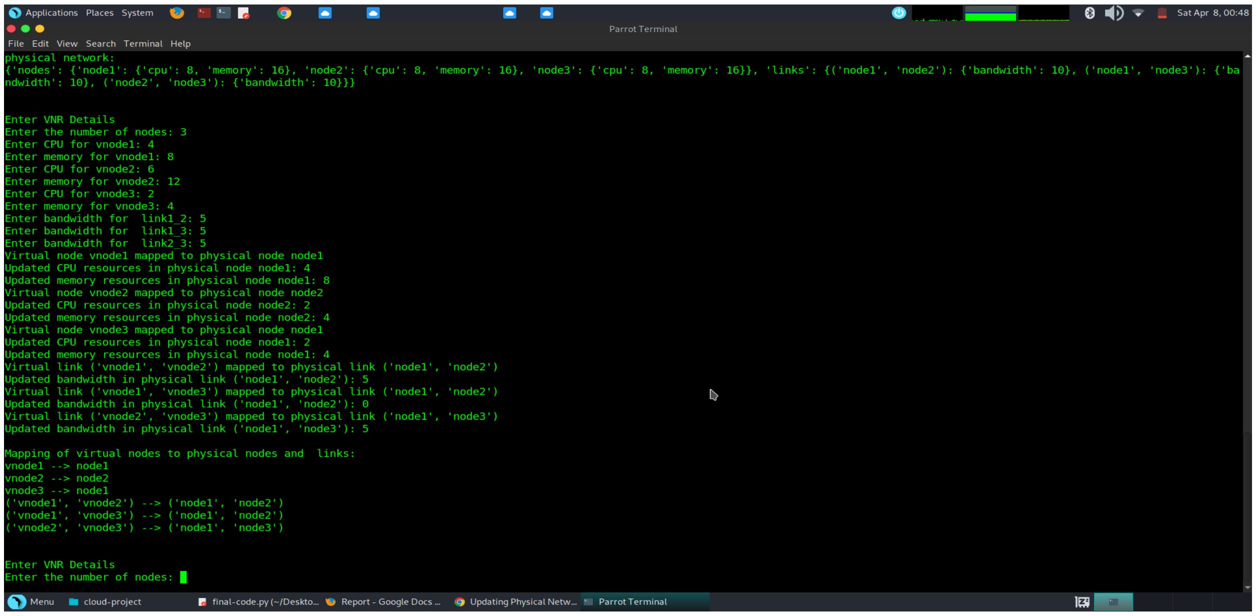


```
(cc-env) ┌─[vamshi@parrot]─[~/Desktop/cloud-project]
└──╼ $python final-code.py
Enter Physical Node Details
Enter the number of nodes: 3
Enter CPU for node1: 8
Enter memory for node1: 16
Enter CPU for node2: 8
Enter memory for node2: 16
Enter CPU for node3: 8
Enter memory for node3: 16
Enter bandwidth for link1_2: 10
Enter bandwidth for link1_3: 10
Enter bandwidth for link2_3: 10

physical network:
{'nodes': {'node1': {'cpu': 8, 'memory': 16}, 'node2': {'cpu': 8, 'memory': 16}, 'node3': {'cpu': 8, 'memory': 16}}, 'links': {('node1', 'node2'): {'bandwidth': 10}, ('node1', 'node3'): {'ba
ndwidth': 10}, ('node2', 'node3'): {'bandwidth': 10}}}


Enter VNR Details
Enter the number of nodes: 3
Enter CPU for vnode1: 4
Enter memory for vnode1: 8
Enter CPU for vnode2: 6
Enter memory for vnode2: 12
Enter CPU for vnode3: 2
Enter memory for vnode3: 4
Enter bandwidth for  link1_2: 5
Enter bandwidth for  link1_3: 5
Enter bandwidth for  link2_3: 5
```

Sample output of above input and Resource utilization:



```
physical network:
{'nodes': {'node1': {'cpu': 8, 'memory': 16}, 'node2': {'cpu': 8, 'memory': 16}, 'node3': {'cpu': 8, 'memory': 16}}, 'links': {('node1', 'node2'): {'bandwidth': 10}, ('node1', 'node3'): {'ba
ndwidth': 10}, ('node2', 'node3'): {'bandwidth': 10}}}


Enter VNR Details
Enter the number of nodes: 3
Enter CPU for vnode1: 4
Enter memory for vnode1: 8
Enter CPU for vnode2: 6
Enter memory for vnode2: 12
Enter CPU for vnode3: 2
Enter memory for vnode3: 4
Enter bandwidth for  link1_2: 5
Enter bandwidth for  link1_3: 5
Enter bandwidth for  link2_3: 5
Virtual node vnode1 mapped to physical node node1
Updated CPU resources in physical node node1: 4
Updated memory resources in physical node node1: 8
Virtual node vnode2 mapped to physical node node2
Updated CPU resources in physical node node2: 2
Updated memory resources in physical node node2: 4
Virtual node vnode3 mapped to physical node node1
Updated CPU resources in physical node node1: 2
Updated memory resources in physical node node1: 4
Virtual link ('vnode1', 'vnode2') mapped to physical link ('node1', 'node2')
Updated bandwidth in physical link ('node1', 'node2'): 5
Virtual link ('vnode1', 'vnode3') mapped to physical link ('node1', 'node2')
Updated bandwidth in physical link ('node1', 'node2'): 0
Virtual link ('vnode2', 'vnode3') mapped to physical link ('node1', 'node3')
Updated bandwidth in physical link ('node1', 'node3'): 5

Mapping of virtual nodes to physical nodes and  links:
vnode1 --> node1
vnode2 --> node2
vnode3 --> node1
('vnode1', 'vnode2') --> ('node1', 'node2')
('vnode1', 'vnode3') --> ('node1', 'node2')
('vnode2', 'vnode3') --> ('node1', 'node3')


Enter VNR Details
Enter the number of nodes: 
```

The loop will go on until anything other than the integer is given as output.

CONCLUSION

Virtual Network Embedding (VNE) by mapping virtual nodes and links to physical nodes and links, taking into consideration the resource utilization of the physical network. This allows for a better understanding and management of the physical network's resource usage during virtual network embedding. The final mapping of virtual nodes and links to physical nodes and links is returned as a dictionary.

We implemented the algorithm in the alib tool in the Python language. and presented the resource allocation using proper PRINT statements.

REFERENCES

1. Hejja, Khaled, and Xavier Hesselbach. "Online power aware coordinated virtual network embedding with 5G delay constraint." *Journal of Network and Computer Applications* 124 (2018): 121-136.
2. Alib tool