Survivable embedded Virtual Network Design to Survive a Facility Node Failure

I. System model and problem

In this section, we first introduce basic concepts on virtual network embedding, then present our survivable virtual network embedding problem.

A. Virtual network embedding

We represent a **virtual network** (VN) as an undirected graph $G(V,E)$ where V and E are the sets of virtual nodes and virtual links, respectively. Each virtual link e_{ij} has the bandwidth demand d_{ij}. Each virtual node v_i has the computation capacity demand d_i. For virtual node v_i, the virtual function required to be executed on the virtual node is denoted as $f(i)$. Fig.1(a) shows an example virtual network with virtual node set $V = \{v_1, v_2, v_3, v_4\}$ and virtual edge set $E = \{e_{12}, e_{13}, e_{14}, e_{23}\}$. The virtual functions required to be executed on these virtual nodes are $f(1) = f_1$, $f(2) = f_2$, $f(3) = f_3$, $f(4) = f_4$, respectively.

We model the **physical network** as an undirected graph $G(S,L)$, where S and L is the sets of physical nodes and physical links, respectively. For physical node s_i, we use $F(i)$ and c_i to respectively denote the set of feasible virtual functions can be executed on this node and the available computational capacity. Each physical link l_{ij} has an available bandwidth of b_{ij}. In the physical network in Fig.1(b), physical node set $S = \{s_1, s_2, s_3, s_4, s_5, s_6, s_7, s_8\}$, link set $L = \{l_{12}, l_{13}, l_{14}, l_{15}, l_{23}, l_{25}, l_{35}, l_{36}, l_{37}, l_{47}, l_{58}\}$, the feasible virtual function set of each physical node is $F(1) = \{f_1\}$, $F(2) = \{f_2, f_3\}$, $F(3) = \{f_3\}$, $F(4) = \{f_4\}$, $F(5) = \{f_1, f_2\}$, $F(6) = \{f_1, f_4\}$, $F(7) = \{f_2, f_3\}$,$F(8) = \{f_2\}$, respectively.

Given the VN request $G(V,E)$, the problem of ***virtual network embedding*** aims to map this request onto the physical network $G(S,L)$ while providing enough resource as demanded. A feasible mapping should satisfy three constraints including node capacity constraint, link bandwidth constraint, and function type constraint.

For a virtual node v_i, a physical node s_j can hold this virtual node only when both node capacity constraint and function type constraint (i.e., $f_i \in F_j$) are satisfied, that is, the node capacity request should be satisfied by the physical node with $d_i \leq c_j$, the virtual function required to be executed on virtual node v_i can be executed on physical node s_j with $f_i \in F_j$. If a physical node s_j satisfies both two constraints, the node mapping is feasible, and we denote such mapping as $\phi(v_i) = s_j$.

For a virtual link e_{ij} with its two virtual nodes are mapped to two physical nodes $s_{i'}$ and $s_{j'}$ (i.e., $\phi(v_i) = s_{i'}$ and $\phi(v_j) = s_{j'}$). Under link bandwidth constraint, if $d_{ij} \leq b_{i'j'}$ where $b_{i'j'}$ is the available bandwidth of the path connecting the physical nodes $s_{i'}$ and $s_{j'}$, virtual link e_{ij} can be mapped to the physical path $p_{\phi(v_i)\phi(v_j)}$, we denote the feasible link mapping as $\rho(e_{ij}) = p_{\phi(v_i)\phi(v_j)}$.

Obviously, to find a feasible virtual network embedding, we should find two mapping functions ϕ and ρ to map all the virtual nodes to physical nodes, and all the virtual links to physical paths.

Fig.1(c) shows a feasible virtual network embedding to embed virtual network in Fig.1(a) to the physical network in Fig.1(b) where virtual node v_1 is embedded in physical node s_1, virtual node v_2 is embedded in physical node s_2, virtual node v_3 is embedded in physical node s_3, and virtual node v_4 is embedded in physical node s_4, respectively. In Fig.1(C), we also show the resource occupied and available in the physical network after such mapping. For example, for physical node s_1, its occupied computation capacity is 2 and the available capacity is 5.

Given the VN request $G(V,E)$ and the physical network $G(S,L)$, for a feasible mapping, we denote the mapping physical graph as $G\left(\hat{S}, P\right)$ where $\hat{S}$ is the physical node set that hold the virtual nodes with $\hat{S} = \{s_{i'} : \phi(v_i) = s_{i'}, for\ all\ v_i \in V, s_{i'} \in S\}$ and P is the path set in which each holds one virtual link with $P = \{p_{\phi(v_i)\phi(v_j)} : \rho(e_{ij}) = p_{\phi(v_i)\phi(v_j)}, for\ all, e_{ij} \in E\}$. As each virtual link corresponds a physical network path which consisting of multiple physical links, we also denote $G\left(\hat{S}, \hat{L}\right)$ as the occupying physical network with $\hat{L} = \{l_{pg} : l_{pg} \in p_{s_{i'}s_{j'}}, \rho(e_{ij}) = p_{\phi(v_i)\phi(v_j)}, for\ all\ e_{ij} \in E, l_{pg} \in L\}$.

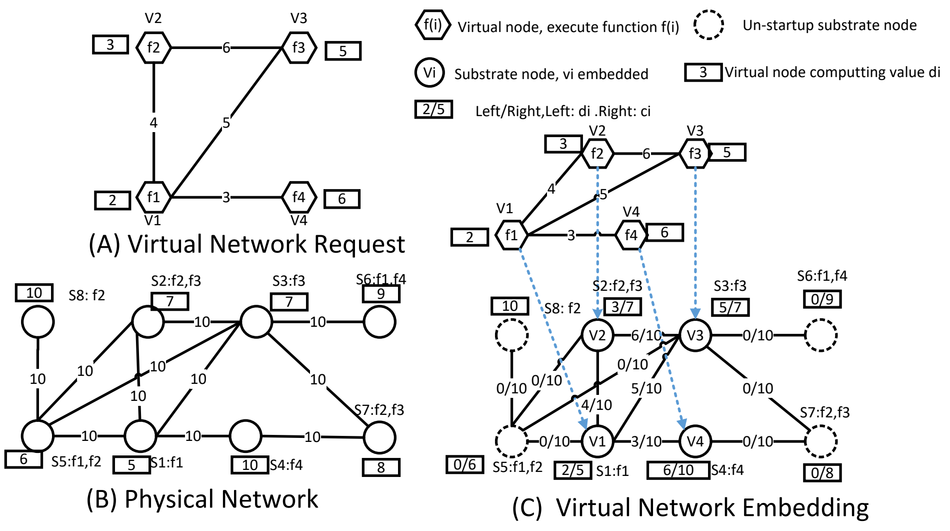
Lots of studies investigate the virtual network embedding problem [1], [2], as the focus of this paper is not virtual network embedding, we adopt algorithm in [lischka2009virtual] as the basic virtual network embedding algorithm.

B. Survivable virtual network embedding

Due to malicious attacks, natural disasters, unintentional cable cuts, planned maintenance, equipment malfunctioning, physical nodes that host the virtual nodes may suffer unavoidable fail. Failure in the VN can happen when single or multiple physical network components failures, which results in financial losses. In general, the multiple physical node's simultaneous failure is mutual independent, a single node failure happen at most of time[3]. In this paper, we study the survivable virtual network embedding problem with single node failure. In section, we will discuss how our algorithm can be extended to the scenario with multiple node failure.

For a VN request $G(V,E)$ and a physical network $G(S,L)$, given a feasible mapping with its occupying physical network $G\left(\hat{S}, \hat{L}\right)$, this paper wants to add minimum backup physical resources to provide survivable network service when any one physical node fails.

Node failures not only affect the visualized services running on the failed physical node, but also would terminate all the communications which traverse through this node. The fail of



Virtual node, execute function f(i)
Un-startup substrate node
Substrate node, vi embedded
Virtual node computting value di
Left/Right,Left: di .Right: ci
(A) Virtual Network Request
(B) Physical Network
(C) Virtual Network Embedding
S8: f2
S2:f2,f3
S3:f3
S6:f1,f4
S7:f2,f3
S5:f1,f2
S1:f1
S4:f4

Fig. 1. (A) Virtual Network Request $G(V,E)$, $V = \{v_1,v_2,v_3,v_4\}$, $E = \{e_{12},e_{23},e_{13},e_{14}\}$, $f(i) = \{f_1,f_2,f_3,f_4\}$, $d_i = \{2,3,5,6\}$, $d_{ij} = \{4,5,3,6\}$.(B) Physical Network $G(S,L), S = \{s_1,s_2,s_3,s_4,s_5,s_6,s_7,s_8\}, L = \{l_{12},l_{13},l_{14},l_{15},l_{23},l_{25},l_{35},l_{36},l_{37},l_{47},l_{58}\}, F(i) = \{\{f_1\},\{f_2,f_3\},\{f_3\},\{f_4\},\{f_1,f_2\},\{f_1,f_4\},\{f_2,f_3\},\{f_2\}\}, c_i = \{5,7,7,10,6,9,8,10\}, b_{ij} = \{10,10,10,10,10,10,10,10,10,10,10\}$ (C) Vertex mapping $v_1 \rightarrow s_1, v_2 \rightarrow s_2, v_3 \rightarrow s_3, v_4 \rightarrow s_4$, link mapping $e_{12} \rightarrow l_{12}, e_{23} \rightarrow l_{23}, e_{13} \rightarrow l_{13}, e_{14} \rightarrow l_{14}$

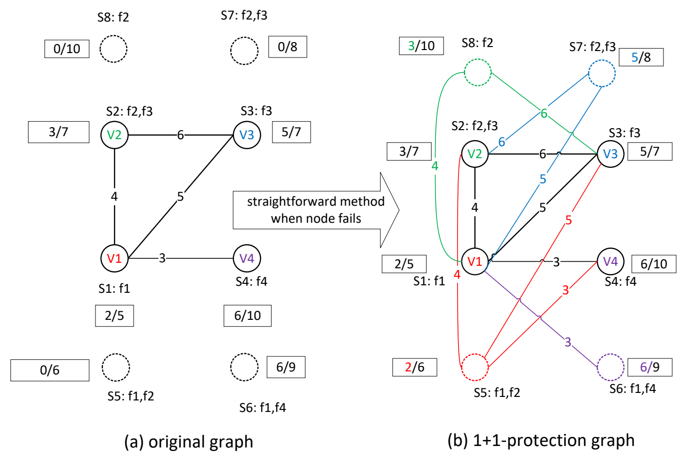


Fig. 2. 1+1-protection scheme

physical node $s_i \in S$ results in the fail of physical links in $L_i = \{l_{ik} : k \in N(i)\}$ where $N(i)$ is the neighbor nodes of node s_i.

As we can not predict which node will fail even though we know multiple nodes will not simultaneously fail, to handle single node failure, one straightforward way is to provision dedicated backup resource for each virtual node in a VN request, also known as the 1 + 1-protection scheme. Fig.2 utilizes an example to illustrate this straightforward way. In the example, a virtual network with 4 virtual nodes are mapped to a physical network, where 4 physical nodes are involved in such embedding. To provide 1 + 1-protection scheme, 4 backup nodes, 8 backup links are added in the graph.

Although 1 + 1-protection scheme is simple to implement, it requires large amount of backup resource. The aim of this paper is to provide survival network service with minimum backup resource cost.

II. Graph Decomposition Based Problem Formulation

Survivable virtual network embedding requires to add backup resources to guarantee that when any one physical node fails, the remaining physical resource plus the backup resources can still support a feasible mapping. To facilitate finding feasible mapping when node fails, this section first decomposes the virtual network and physical network into star based local components. Based on which, a novel bipartite graph is proposed and the problem of survivable virtual network embedding with minimum backup resource is formulated as a virtual star assignment problem based on the well defined bipartite graph.

A. *Star based graph decomposition*

The virtual network is decomposed into virtual local stars. Each virtual local star associates with a virtual node. Given virtual node v_i, the corresponding virtual local star is defined as an attributed, single-level, rooted tree and expressed as

$$VirtualStar(v_i) = (v_i, \phi(v_i), d_i, f_i, D_i, N_i) \qquad (1)$$

where N_i is the neighbor node set of v_i in the virtual network and D_i denotes the bandwidth demand set associate with node v_i with $D_i = \{d_{ij} | v_j \in N_i\}$. Note that, VirtualStar(v_i) includes the node mapping information $\phi(v_i)$. As we want to minimize the backup resource to provide the survivable network service, re-use the mapping before node failure may be a good choice to reduce the additional resource and make the system remain stable. In the virtual star structure, edges exist between the root node and its neighbor nodes, and no edge exists among its neighbor nodes.

The physical network is decomposed into physical local stars. Similarly, each physical local star associates with a physical node. Given physical node s_j, the corresponding physical local star is defined as an attributed, single-level, rooted tree and expressed as

$$PhysicalStar(s_j) = (s_j, \phi^{-1}(s_j), c_j, F(j), \phi(N(\phi^{-1}(s_j))), a) \tag{2}$$

where $\phi^{-1}(s_j)$ is the virtual nodes that map to physical node s_j, $N(\phi^{-1}(s_j))$ is the neighbor nodes of the virtual nodes in $\phi^{-1}(s_j)$, $\phi(N(\phi^{-1}(s_j)))$ is the physical nodes that hold these neighbors, c_j is the node capacity, $F(j)$ is the virtual functions supported by s_j, a is a one-bit single with its value being 0 or 1 to indicate whether this physical node has setup the virtual machine. Similarly to virtual star, in the physical star structure, edges exist between the root node and its neighbor nodes, and no edge exists among its neighbor nodes.

Virtual star and physical star defined in (1) and (2) well capture the local structures hidden in the VN request to preserve the relationship of node with its adjacent.

Based on the virtual local star and physical virtual star, the virtual network and physical network can be decomposed into multiple components.

B. Bipartite graph

We build a bipartite graph $G = \{V_1, V_2, E\}$ to present the relationship between the virtual network and physical network. V_1 and V_2 are the vertex sets representing the set of virtual stars and the set of physical stars, respectively. If VirtualStar(v_i)'s virtual function f_i can be executed by a physical node s_j with $f_i \in F_j$, an edge $e(i,j)$ is added to the edge set E to connect the VirtualStar(v_i) and PhysicalStar(s_j).

Our goal is to minimize the backup resource to provide survivable service. To serve the goal, given edge $e(i,j)$, we define the edge weight $w(i,j)$ as the backup resource cost to map the virtual star(v_i) to the physical star (s_j) when node failure happens. According to whether the virtual node v_i is mapped to the physical node s_j before node fails, we define the edge weight in two different cases.

$$w(i,j) = \begin{cases} \alpha \sum_{\phi(v_k) \notin \phi(N(\phi^{-1}(s_j)))} d_{ik} & v_i \in \phi^{-1}(s_j), v_k \in N(i) \\ \alpha \sum_{k \in N(i)} d_{ik} + \beta M_m + \lambda c_i + \theta & v_i \notin \phi^{-1}(s_j), v_k \in N(i) \end{cases} \tag{3}$$

In (3), θ is defined as follows.

$$\theta = \begin{cases} C_s & a = 0 \\ 0 & a = 1 \end{cases} \tag{4}$$

In the first case ($v_i \in \phi^{-1}(s_j)$), as the virtual node v_i is mapped to the physical node s_j before node fails, node capacity demand is satisfied already, thus only bandwidth backup cost is needed when map the virtual star (v_i) to the physical star (s_j). For each neighbor $v_k \in N(i)$, if the physical node that holds virtual node v_k fails with $\phi(v_k) \notin \phi(N(\phi^{-1}(s_j)))$, new path with bandwidth d_{ik} should be added as the backup resource. Therefore, in this case, the backup cost only includes bandwidth cost and expressed as $\alpha \sum_{\phi(v_k) \notin \phi(N(\phi^{-1}(s_j)))} d_{ik}$ where α is the unit bandwidth cost.

In the second case ($v_i \notin \phi^{-1}(s_j)$), as the virtual node v_i is not mapped to the physical node s_j before node fails, virtual node v_i needs to be migrated to physical node s_j. Therefore, the edge weight $w(i,j)$ includes nodes capacity cost, paths bandwidth cost, and migration cost to migrate a virtual node from a physical node to another physical node when physical nodes fail. Moreover, if the backup physical node s_j does not hold any virtual node before, migrating a virtual node to this physical node also introduces a virtual machine booting cost denoted as C_s.

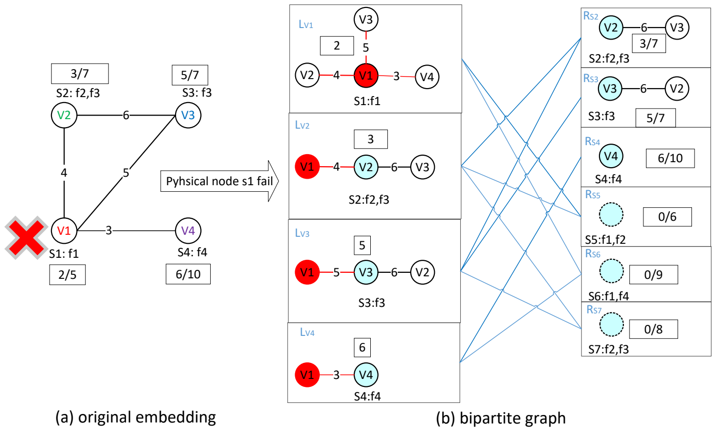


Fig. 3. Components of VirtualStar(v_i) and PhysicalStar(s_j) when physical node s_1 fail

Fig.3 shows one example of such bipartite graph when physical node s_1 fails. The edge weight of this bipartite graph can be shown in Eq(II-B).

$$\begin{array}{ccccccc} & R_{S_1} & R_{S_2} & R_{S_3} & R_{S_4} & R_{S_5} & R_{S_6} \\ L_{V_1} & \infty & \infty & \infty & \infty & \boxed{C_s + M_m\text{+(2)+12}} & C_s + M_m + (2) + 12 \\ L_{V_2} & \infty & \boxed{4} & \infty & \infty & C_s + M_m + (3) + 10 & \infty \\ L_{V_3} & \infty & M_m + (5) + 11 & \boxed{5} & \infty & \infty & \infty \\ L_{V_4} & \infty & \infty & \infty & \boxed{3} & \infty & C_s + M_m + (6) + 3 \end{array}$$

For the virtual node v_i, if its virtual function can not be executed in physical node s_j with $f(i) \notin F(i)$, no edge is connected with $VirtualStar(v_i)$ in V_1 and $PhysicalStar(s_j)$ in V_2. To facilitate problem formulation in following section, we set the edge weight to be ∞. For example, as v_1's virtual function is f_1, it can not be executed in physical node s_2. Therefore, no edge is added to connected with $VirtualStar(v_1)$ in V_1 and $PhysicalStar(s_2)$ in V_2 and we set the $w(1,2) = \infty$.

For $VirtualStar(v_2)$, as v_2 is originally hold by s_2, however, when physical node s_1 (which holds v_1 originally) fails, the virtual link e_{12} fails to be mapped to the physical network. We should find a new path connecting $\phi(v_2)$ with $\phi(v_1)$ satisfying bandwidth demand $d_{12} = 4$. Therefore, the edge weight connecting $VirtualStar(v_2)$ and $PhysicalStar(s_2)$ is 4.

When node s_1 fails, it directly impact virtual node v_1. As s_5 can execute virtual function f_1, we can add a edge to connect $VirtualStar(v_1)$ and $PhysicalStar(s_5)$. However, in this example, as s_5 does not setup virtual machine before, there also introduces virtual machine's start cost C_s. Therefore, the link weight is C_s (start cost)+ Mm (migrating cost)+ 3 (node capacity cost)+ 10 (bandwidth cost).

C. Problem formulation

To provide survivable service, each virtual star (both the root virtual node and the virtual links connecting root node and its adjacent nodes) should be mapped to a physical star. Assume virtual network consists of n virtual nodes, thus n virtual stars. The physical network consists of m physical nodes, thus m

physical stars. In Eq(5), we use bit binary M_{ij} to denote whether the i-th virtual star is mapped to the j-th physical star.

$$M_{ij} = \begin{cases} 1 & map\ virtualstar(i)\ to\ physicalstar(j) \\ 0 & otherwise \end{cases} \tag{5}$$

When a physical node fails, to minimize the backup resource cost, the survivable virtual network embedding problem can be defined as follows.

$$\begin{aligned} \min_{M_{ij}} \quad & \sum_{i=1}^{n}\sum_{j=1}^{m} M_{ij} w_{ij} \\ s.t., \quad & \sum_{i=1}^{n} d_i M_{ij} \le c_j \\ & \sum_{i=1}^{m} M_{ij} \le 1 \\ & M_{ij} = \{0,1\} \end{aligned} \tag{6}$$

where $\sum_{i=1}^{n}\sum_{j=1}^{m} M_{ij} w_{ij}$ denotes the total backup resource cost to map all the virtual stars to physical stars when physical nodes fails. In (6), $\sum_{i=1}^{n} d_i M_{ij} \le c_j$ is the physical node's capacity constraint, that is, even though multiple virtual nodes are allowed to be mapped to a physical node, the total capacity demand should not be larger than the capacity of the physical node. $\sum_{i=1}^{m} M_{ij} \le 1$ indicates that one virtual star can be mapped to only one physical star.

Obviously, problem defined in (6) is an ILP problem, which is general an NP-complete problem.

Theorem 1 Problem defined in (6) is an NP-complete problem.

Proof. If there is only one physical node with $m = 1$, our survivable virtual network embedding problem can be degenerate into single knapsack problem, which is NP-complete problem. In practice, m is usually larger than 1, according to the reducibility theorem[4] in computer complexity field, it is easy to conclude that our defined in (6) is also NP-complete. □

D. Dynamic Programming Mehtod

Although solving the ILP formulation will result in a minimum cost survival virtual network embedding, its exponential time complexity makes such approach impractical to embed a virtual network to a large physical network. In this section, we propose a dynamic programming based algorithm that has only a polynomial time complexity and, thus, is feasible for practical network systems.

To describe our dynamic programming based algorithm, we define $dp[i][x_1][x_2]\ldots[x_m]$ to denote the best virtual star placement with the minimum backup resource cost to place the first i $(0 \le i \le n)$ virtual stars to the m physical stars with their capacity limit being x_1, x_2, …, x_m in the physical network.

The i-th virtual node star has the options to be placed onto any one of the physical stars that are alive. Let $\theta(i,j)$ denote the backup resource cost to place the i-th virtual node star to the j-th physical node after the first $i-1$ virtual nodes are best placed. $\theta(i,j)$ is expressed as follows.

$$\theta(i,j) = \begin{cases} dp[i-1][x_1][x_2]\ldots[x_j - d_i]\ldots[x_m] + w_{ij} & (x_j \ge d_i, f_i \in F_j) \\ \infty & otherwise \end{cases} \tag{7}$$

In (7), if the capability limit of the physical node c_j is large than the capacity demand d_i and the virtual function f_i can be executed in the physical node $f_i \in F_j$, $\theta(i,j)$ is the sum of the cost under best virtual star placement to place the first $i-1$ $(0 \le i \le n)$ virtual stars to the m physical stars (i.e., $dp[i-1][x_1 - d_i][x_2]\ldots[x_m]$) and the cost that mapping the virtual star (v_i) to physical star (s_1) (i.e., w_{i1}).

Based on $\theta(i,j)$, $dp[i][x_1][x_2]\ldots[x_m]$ can be calculated through following dynamic programming function.

$$dp[i][x_1][x_2]\ldots[x_m] = min\{\theta(i,1), \theta(i,2), \ldots, \theta(i,j), \ldots, \theta(i,m)\} \tag{8}$$

Algorithm 1 Dynamic Programming Based Algorithm fff

Require: $dp[i][x_1][x_2]\ldots[x_m] = 0(1 \le i \le n, 0 \le x_1 \le c_1, 0 \le x_2 \le c_2, \ldots, 0 \le x_m \le c_m)$ is firstly assigned as infinity ∞, $dp[0][x_1][x_2]\ldots[x_m] = 0(0 \le x_1 \le c_1, 0 \le x_2 \le c_2, \ldots, 0 \le x_m \le c_m)$, m is the number of physical nodes. $M[x_1][x_2]\ldots[x_m] = \begin{bmatrix} 0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0 \\ 0&0&0&0&0&0&0 \end{bmatrix}$ is placement of every virtual node.

Ensure: obtaining minimum weight sum when node up-bound capacity of physical nodes is $c_1, c_2, \ldots, c_m$, respectively. obtaining optimal virtual node's placement.

```
1: for all $i$ such that $1 \le i \le n$ do
2:     for all $x_1, x_2, \ldots, x_m$ such that $c_1 \ge x_1 \ge d_i$, $c_2 \ge x_2 \ge d_i$,$c_3 \ge x_3 \ge d_i$,$c_m \ge x_m \ge d_i$ do
3:         $dp[i][x_1][x_2]\ldots[x_m] = min\{\theta(i,1), \theta(i,2), \ldots, \theta(i,j), \ldots, \theta(i,m)\}$
4:         $j' = \arg\min_j\{\theta(i,1), \theta(i,2), \ldots, \theta(i,j), \ldots, \theta(i,m)\}$, .
5:         $M[x_1][x_2]\ldots[x_m] = M[x_1][x_2]\ldots[x_{j'} - d_i]\ldots[x_m]$
6:         $M[x_1][x_2]\ldots[x_m]_{ij'} = 1$
7:     end for
8: end for
9: return $dp[i][c_1][c_2]\ldots[c_m]$ and $M[c_1][c_2]\ldots[c_m]$
```

The pseudo-code of the dynamic programming based algorithm is shown in Alg.II-D. We further use an example in Fig.4 to illustrate the algorithm.

For clear presentation, in this example, three virtual stars needed to be placed to two available physical stars to achieve the minimum backup resource cost. The available capacity under physical nodes are $c_1 = 5$ and $c_2 = 4$, respectively. The capacity demands of these three virtual stars are $d_1 = 1$, $d_2 = 2$, and $d_3 = 1$. The virtual functions required to be executed in these virtual stars are $f(1) = f_1$, $f(2) = f_1$, $f(3) = f_2$. The functions supported by these two physical nodes are $F(1) = \{f_1, f_2\}$ and $F(2) = \{f_1\}$.

In Fig.4, x and y axis denote the capacity limit of physical star s_1 and s_2, respectively. The weight of edge connecting virtual stars to physical stars are $w_{11} = 1$,$w12 = 2$,$w_{21} = 3$,$w22 = 1$, $w_{31} = 1$, $w32 = \infty$, respectively.

Initially, in Fig.4(a), as no virtual star is placed to any physical stars, the backup cost under all the capacity limit cases (x_1=0,1,2,3,4 and x_2=0,1,2,3,4,5) are all 0. Specially, even though capacity limits are set 4 and 5, dp[0][4][5]=0.

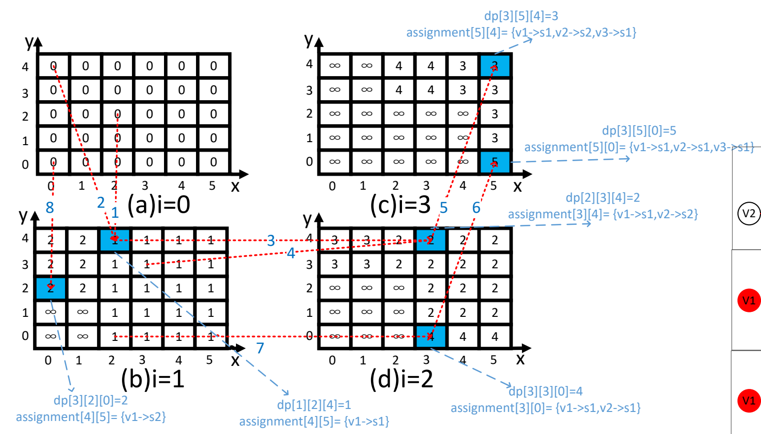


Fig. 4. Three virtual node, the first virtual node could be placed to first and second physical node, the second virtual node could be placed to first and second physical node, the third virtual node could only be placed to first physical node. The node computation capacity of first physical node is 5, the node computation capacity of second physical node is 4. w_{ij}=[1,2;3,1;1,∞].

In Fig.4(b), when place the first virtual node v_1 with capacity demand $d_1 = 1$ to these two physical nodes, as f_1 can be executed in both physical nodes. Therefore, we have

$$dp[1][x_1][x_2] = \min\{\theta(1,1), \theta(1,2)\} \tag{9}$$

Specially, if the capacity limit of these two physical nodes are x_1=2 and x_2=0, we have $dp[1][2][0] = dp[1][0][0] + w_{11} = 2$. If the capacity limit of these two physical nodes are x_1=2 and x_2=4, we have $\theta(1,1) = dp[0][4][0] + w_{11})$ and $\theta(1,2) = \infty$ and thus $dp[1][2][4] = min\{dp[0][4][0] + w_{11}), dp[0][2][2] + w_{12}\} = 1$.

Similarly, when place the second virtual node with capacity demand $d_2 = 2$ to these two physical nodes, the cost results under all capacity limits is shown in Fig.4(c). As f_2 can be executed in both physical nodes, thus v_2 can be placed into both physical nodes, we have

$$dp[2][x_1][x_2] = \min\{\theta(2,1), \theta(2,2)\} \tag{10}$$

Specially, if the capacity limit of these two physical nodes are x_1=3 and x_2=4, we have $dp[2][3][4] = min\{dp[1][2][4] + w(2,1), dp[1][3][3] + w(2,2)\} = 2$. If the capacity limit of these two physical nodes are x_1=3 and x_2=0, we have $dp[2][3][0] = dp[1][2][0] + w(2,2) = 4$.

Fig.(d) shows the minimum resource cost result when place the third virtual node with capacity demand $d_3 = 1$ to these two physical nodes. As f_3 can only be executed in physical node s_1, we have $\theta(3,2) = \infty$. Specially, if the capacity limit of these two physical nodes are x_1=5 and x_2=0, we have $dp[3][5][0] = dp[2][3][0] + w(3,1) = 4$. As the node capability of these two physical nodes are 5 and 4, respectively, we have $dp[3][5][3] = dp[2][3][4] + w(3,1) = 3$, as indicted in Fig.4(d), the best placement is achieved at dp[3][5][4]=3 with the assignment $v_1 \rightarrow s_1, v_2 \rightarrow s_2, v_3 \rightarrow s_1$.

For example, as shown in Fig.3 and Equ.(II-B), the optimal

$$M_{ij} = \begin{bmatrix} 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}.$$

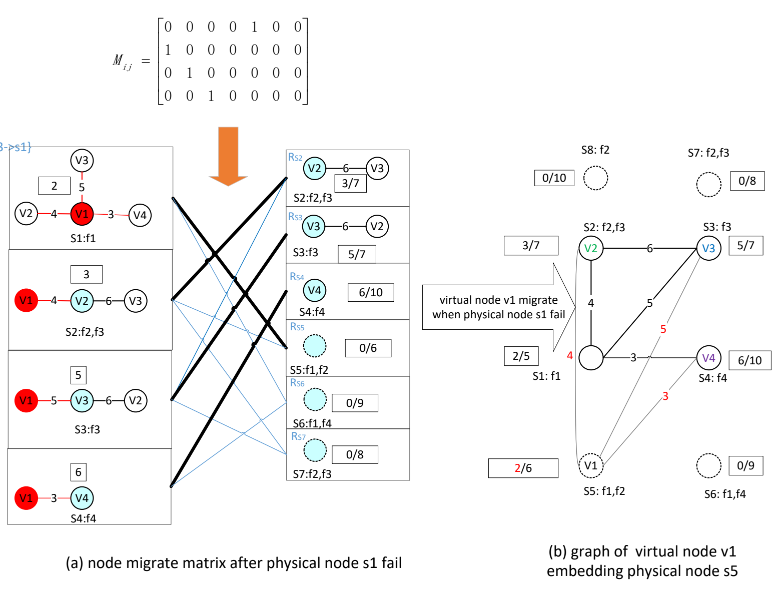

(a) node migrate matrix after physical node s1 fail
(b) graph of virtual node v1 embedding physical node s5
virtual node v1 migrate when physical node s1 fail

Fig. 5. Node1Failure

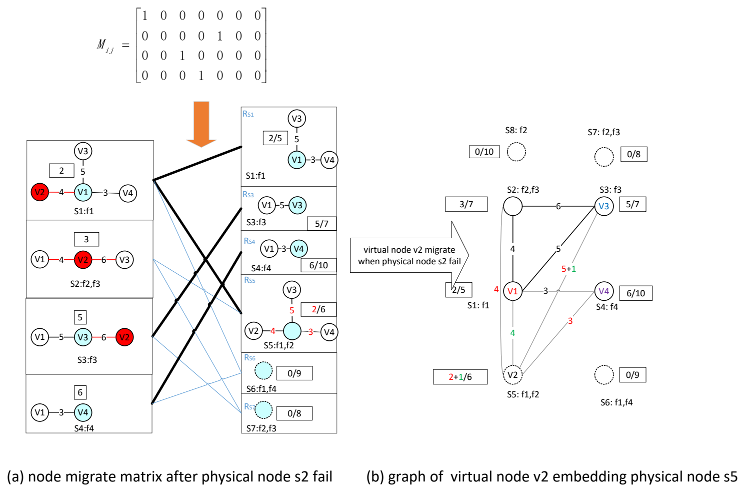

(a) node migrate matrix after physical node s2 fail
(b) graph of virtual node v2 embedding physical node s5
virtual node v2 migrate when physical node s2 fail

Fig. 6. Node2Failure

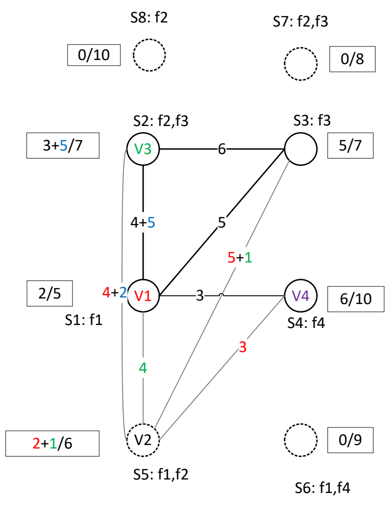


Fig. 7. Node3Failure

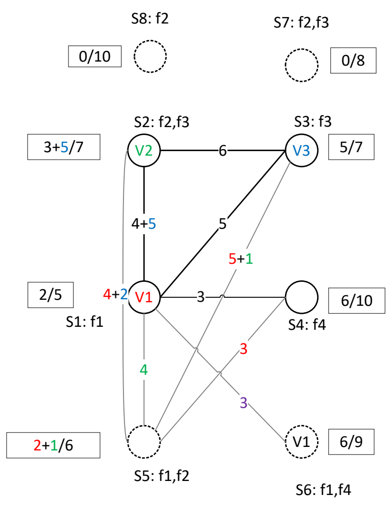


Fig. 8. Node4Failure

III. Avoid duplicately adding backup resource

Given one node fails, after applying Algorithm II-D to calculate the minimum additional resource, new physical node and physical path is added and involved to maintain the network service. If a physical node that does not hold any virtual node before node fail is added to be the backup node, we should set the physical node with $a = 1$ to indicate that this node is involved to hold virtual already.

As our survivable virtual network embedding problem wants to minimize the backup resource added when any one node fails instead of given one node failure, we should test each node fail one by one and add sufficient backup resource. As the backup resource only need when node fails, the backup resource should be shared when different node fails. If we directly apply cost defined in (3) to calculate the backup resources when another physical node fails, as cost defined in (3) does not consider the backup resource sharing, it will result in a problem of duplicately adding backup resource.

To solve this problem, we add the $M(j)$ to the physical star structure to denote the set of virtual nodes migrated into physical node s_j in the case of node failure.

$$PhysicalStar(s_j) = (s_j, \phi^{-1}(s_j), c_j, F(j), \phi(N(\phi^{-1}(s_j))), a, M(j)) \tag{11}$$

As the backup resource should be added to only once, to facilitate expressing this constraint, we define following constraint.

$$\mu(x) = \begin{cases} x & x > 0 \\ 0 & x \leq 0 \end{cases} \tag{12}$$

When we test another node fails, instead of cost defined in (3), we define a new cost function by taking consideration of the backup resource sharing when different node fails.

As shown in eq(13), the resource cost is setting according to two cases. If virtual node v_i is originally hold by physical server s_j or v_i is migrated to the physical server, that is, $v_i \in \left(\phi^{-1}(s_j) \cup M(j)\right)$, when its neighbor $v_k \in N(i)$ fails with $\phi(k) \notin \phi\left(\phi^{-1}(s_j)\right)$, the path bandwidth cost is $\sum_{k \in N(i) \cap N(i')} f(d_{ik} - d_{i'k}) + \sum_{k \in (N(i) - N(i'))} d_{ik}$.

Otherwise, the virtual node v_i is not hold by the physical server and not migrated to the server in previous node fail test, besides the bandwidth cost $\sum_{k \in N(i) \cap N(i')} f(d_{ik} - d_{i'k}) + \sum_{k \in (N(i) - N(i'))} d_{ik}$, the node capacity cost $\lambda f(d_i - \max_{v_{i'}}(d_{i'}))$, VM migrate cost βM_m and θ should be counted for mapping the virtual star (v_i) to the physical server (s_j). In $f(d_i - \max_{v_{i'}}(d_{i'}))$, if the backup capacity already allocated $\max_{v_{i'}}(d_{i'})$ is larger than amount needed in current mapping (i.e., d_i), no resource is needed, otherwise, the backup resource gap $d_i - \max_{v_{i'}}(d_{i'})$ is needed.

As shown in Fig.4 Line 1, when optimal weight sum of placing the first node v_1 with node computation capacity of the first and second physical node is 0 and 2 respectively is calculated from not placing any virtual node with node computation capacity of the first and second physical node is 0 and 0 respectively.

As shown in Fig.4 Line 2 and Line 3, when optimal weight sum of placing the first node v_2 with node computation capacity of the first and second physical node is 2 and 3 respectively is calculated from the minimum of between placed former 1 node with node computation capacity of the first and second physical node is 1 and 3 respectively, and placed former 1 node with node computation capacity of the first and second physical node is 2 and 2 respectively.

As shown in Fig.4 Line 4, when optimal weight sum of placing the first node v_2 with node computation capacity of the first and second physical node is 3 and 0 respectively is only calculated from the minimum of between placed former 1 node with node computation capacity of the first and second physical node is 2 and 0 respectively.

As shown in Fig.4 Line 6, when optimal weight sum of placing the first node v_3 with node computation capacity of the first and second physical node is 4 and 4 respectively is only calculated from the minimum of between placed former 2 nodes with node computation capacity of the first and second physical node is 2 and 4 respectively, because virtual node v_3 must be placed onto first physical node $(f_1 \in F_1, f_1 \notin F_2)$.

there are n+1 level's dynamic programming functions and there are $\prod_{i=1}^{m} C^i$ dynamic programming functions in every level. The time complexity of calculating every dynamic programming function is $O(m)$. Therefore, overall time complexity of the dynamic programming method is $n * \prod_{i=1}^{m} C^i * O(m) = O(n * m * \prod_{i=1}^{m} C^i)$. Additionally, for recording every level's virtual node's placement, the overall space complexity is $O[(n+1) * \prod_{i=1}^{m} C^i]$, which is, however, could be optimized to $O[\prod_{i=1}^{m} C^i]$, because when updating every level's dynamic programming function, the dynamic programming function only use back step level's dynamic programming function.

As shown in Fig.9 and Equ.(III), the optimal $M_{ij} = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix}$.

For $VirtualStar(v_2)$, as v_3 is originally hold by s_3, however, when physical node s_1 fails, we could migrate virtual node v_3 onto physical node s_2, as s_2 is setup virtual machine before, therefore, cost include the virtual link bandwidth cost

$$w(i,j) = \begin{cases} \sum_{k\in N(i)\cap N(i')} f(d_{ik} - d_{i'k}) + \sum_{k\in(N(i)-N(i'))} d_{ik} & v_i \in \left(\phi^{-1}(s_j)\cup M(j)\right), v_{i'} \in M(j), k \notin \phi\left(\phi^{-1}(s_j)\right) \\ \sum_{k\in N(i)\cap N(i')} f(d_{ik} - d_{i'k}) + \sum_{k\in(N(i)-N(i'))} d_{ik} + \lambda f(d_i - \max_{v_{i'}}(d_{i'})) + \beta M_m + \theta & v_i \notin \left(\phi^{-1}(s_j)\cup M(j)\right), v_{i'} \in M(j) \end{cases} \tag{13}$$

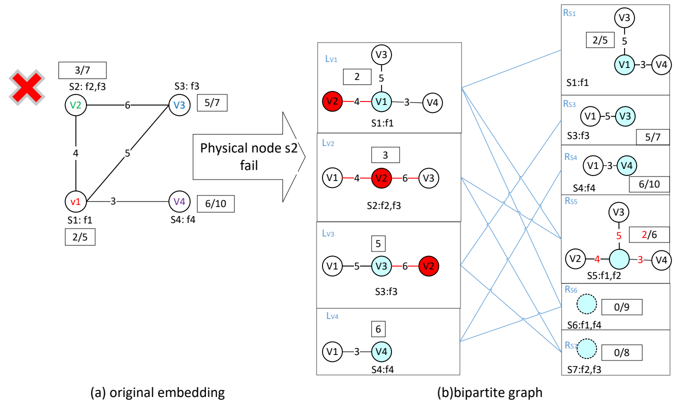


Fig. 9. Components of VirtualStar(v_i) and PhysicalStar(s_j) when physical node s_2 fail

11, and node capacity cost 5, and virtual machine migration cost M_m.

Based method in Sec.V-D1, obtaining the optimal node mapping when physical node s_1 fails: $v_1 \rightarrow s_5$, $v_2 \rightarrow s_2$, $v_3 \rightarrow s_3$, $v_4 \rightarrow s_4$. The physical node s_5 begin setup and apply 2 node computing for virtual node v_1, and find three physical path corresponding virtual link $(v_1, v_2), (v_1, v_3), (v_1, v_4)$ with bandwidth constraints 4,5,3, respectively.

$$\begin{array}{cccccccc} & R_{S_1} & R_{S_2} & R_{S_3} & R_{S_4} & R_{S_5} & R_{S_6} & R_{S_7} \\ L_{V_1} & \boxed{4} & \infty & \infty & \infty & M_m & C_s + M_m + (2) + 12 & \infty \\ L_{V_2} & \infty & \infty & \infty & \infty & \boxed{M_m+(1)+5} & \infty & C_s + M_m \text{[illegible]} \\ L_{V_3} & \infty & \infty & \boxed{5} & \infty & \infty & \infty & C_s + M_m \text{[illegible]} \\ L_{V_4} & \infty & \infty & \infty & \boxed{3} & \infty & C_s + M_m + (6) + 3 & \infty \end{array}$$

Fig.9 shows one example of such bipartite graph with physical node s_2 fails after physical node s_1 failed. The edge weight of this bipartite graph can be shown in Eq(III).

For $VirtualStar(v_1)$, as v_1 is originally hold by s_1, however, when physical node s_2 fails, we could find a alternate physical node s_5 as migrated node for virtual node v_1, and the physical node s_5 own backup node computation and edge bandwidth resource because former physical node s_1 failure situation, so that when physical node s_2 fail the edge that connect $VirtualStar(v_1)$ and $PhysicalStar(s_5)$ with the edge weight being M_m .

For $VirtualStar(v_2)$, as v_2 is originally hold by s_2, however, when physical node s_2 fails, we could find a alternate physical node s_5 as migrated node for virtual node v_2, and the physical node s_5 own backup node computation and edge bandwidth resource because former physical node s_1 failure situation, so that when physical node s_2 fail the edge that connect $VirtualStar(v_1)$ and $PhysicalStar(s_5)$ with the edge weight being $M_m+(1)+5$, the physical node s_5 just apply more 1 node computation resource for migrating virtual node v_2 for original backup node computation(2 node computation) resource of physical node s_5. There are a path

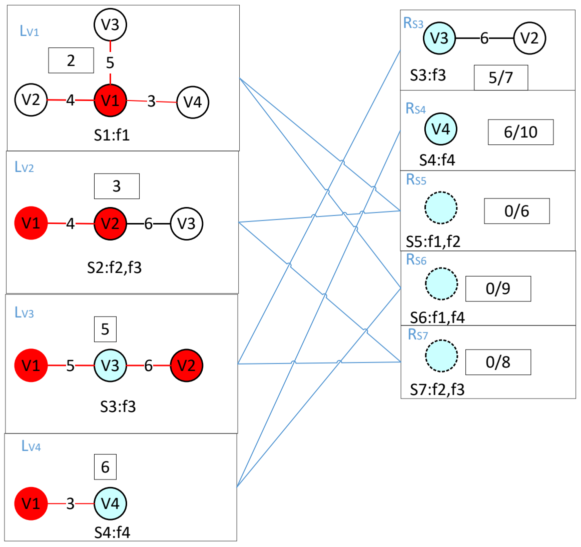


Fig. 10. Components of VirtualStar(v_i) and PhysicalStar(s_j) when physical node s_1 and s_2 fail simultaneously

between physical node s_5 and physical node s_3 mapping virtual node v_3, therefor, just apply 1 edge bandwidth to add the path physical node s_5 to physical node s_3 for satisfying the edge between virtual node v_2 and virtual node v_3, finally the cost of augmented path bandwidth is 1+4=5.

Based method in Sec.V-D1, obtaining the optimal node mapping when physical node s_2 fails with physical node s_1 had been failed: $v_1 \rightarrow s_1$, $v_2 \rightarrow s_5$, $v_3 \rightarrow s_3$, $v_4 \rightarrow s_4$. The physical node s_5 had been setup and remain 2 backup node computing because of physical node s_1 failure, therefore, just apply more 1 node computing for virtual node v_2, and find three physical path corresponding virtual link $(v_1, v_2), (v_1, v_3), (v_1, v_4)$ with bandwidth constraints 4,5,3, respectively.

$$\begin{array}{cccccccc} & R_{S_1} & R_{S_2} & R_{S_3} & R_{S_4} & R_{S_5} & R_{S_6} & \\ L_{V_1} & \infty & \infty & \infty & \infty & \boxed{C_s + M_m+(2)+12} & C_s + +M_m + (2) + 12 & \\ L_{V_2} & \infty & \infty & \infty & \infty & C_s + M_m + (3) + 10 & \infty & \boxed{C_s + M} \\ L_{V_3} & \infty & \infty & \boxed{5} & \infty & \infty & \infty & C_s + M \\ L_{V_4} & \infty & \infty & \infty & \boxed{3} & \infty & C_s + M_m + (6) + 3 & \end{array}$$

Fig.10 shows one example of such bipartite graph when physical node s_1 and s_2 fail simultaneously. The edge weight of this bipartite graph can be shown in Eq(III).

1) Dynamic Programming Mehtod: **Although solving the ILP formulation will result in an optimal node mapping, its exponential time complexity makes such approach impractical for large quantity of virtual nodes situation. In this section, we describe an alternative using dynamic programming heuristic that has only a polynomial time complexity and, thus, is feasible for large quantity of virtual nodes situation. To describe the proposed $STAR$ algorithm based**

on dynamic programming, let us number the virtual nodes from 0 to n, where n is the number of virtual network request node. In addition, let $dp[i][x_1][x_2]\dots[x_m]$ be the best node's placement (place virtual node onto qualified physical node so as to minimize total weight sum) to place the i-th virtual node with the former i-1 virtual nodes were optimal to be placed and node computation up-bound capacity (the node computing sum of all virtual node which is mapped onto this physical node less than the physical node's node computing capacity) of every physical node is $x_1, x_2, \dots, x_n$, respectively. The pseudo-code of the dynamic programming based algorithm is shown in Alg.II-D, and take out a simple example as shown in Fig.4.

As shown in Fig.4 Line 1, when optimal weight sum of placing the first node v_1 with node computation capacity of the first and second physical node is 0 and 2 respectively is calculated from not placing any virtual node with node computation capacity of the first and second physical node is 0 and 0 respectively.

As shown in Fig.4 Line 2 and Line 3, when optimal weight sum of placing the first node v_2 with node computation capacity of the first and second physical node is 2 and 3 respectively is calculated from the minimum of between placed former 1 node with node computation capacity of the first and second physical node is 1 and 3 respectively, and placed former 1 node with node computation capacity of the first and second physical node is 2 and 2 respectively.

As shown in Fig.4 Line 4, when optimal weight sum of placing the first node v_2 with node computation capacity of the first and second physical node is 3 and 0 respectively is only calculated from the minimum of between placed former 1 node with node computation capacity of the first and second physical node is 2 and 0 respectively.

As shown in Fig.4 Line 6, when optimal weight sum of placing the first node v_3 with node computation capacity of the first and second physical node is 4 and 4 respectively is only calculated from the minimum of between placed former 2 nodes with node computation capacity of the first and second physical node is 2 and 4 respectively, because virtual node v_3 must be placed onto first physical node ($f_1 \in F_1, f_1 \notin F_2$).

there are n+1 level's dynamic programming functions and there are $\prod_{i=1}^{m} C^i$ dynamic programming functions in every level. The time complexity of calculating every dynamic programming function is $O(m)$. Therefore, overall time complexity of the dynamic programming method is $n * \prod_{i=1}^{m} C^i * O(m) = O(n * m * \prod_{i=1}^{m} C^i)$. Additionally, for recording every level's virtual node's placement, the overall space complexity is $O[(n+1) * \prod_{i=1}^{m} C^i]$, which is, however, could be optimized to $O[\prod_{i=1}^{m} C^i]$, because when updating every level's dynamic programming function, the dynamic programming function only use back step level's dynamic programming function.

Placing the i-th virtual node where node computation capacity of every physical node is x_1, x_2, x_3, the optimal result $dp[i][x_1][x_2][x_3]$ is inferred from the optimal result of the former i-1 virtual nodes are succeed to be placed. For example, for requesting $dp[i][x_1][x_2][x_3]$, the state is inferred from $dp[i-1][x_1-d_i][x_2][x_3]$ when former i-1 virtual nodes are succeed to be placed and node computation capacity of every physical node is $x_1-d_i, x_2, \dots, x_n$. The state of $dp[i-1][x_1-d_i][x_2][x_3]$ is optimal result when calculating $dp[i-1][x_1-d_i][x_2][x_3]$, and first physical node's node computation capacity $x_1-d_i+d_i$ of $dp[i][x_1][x_2][x_3]$ add d_i because the i-th virtual node is placed into the first physical node.

The formulation of our survivable virtual network embedding problem is standard multiple knapsack problem's formulation, we propose a dynamic programming method for solving the multiple knapsack problem. Assume $C_1, C_2, \dots, C_n$, C_i are strictly positive integers. Define dynamic programming function $dp[i][C_1][C_2]\dots[C_m] = 0 (C_1 \le c_1, C_2 \le c_2, \dots, C_m \le c_m)$ to be the minimum value that can be attained with n capacity variables which less than or equal to C_i using items up to knapsack i.

Initial state of dynamic programming function $dp[i][C_1][C_2]\dots[C_m] = 0$ is firstly assigned as infinity ∞. Then when there is no any virtual node which is confirmed to map into another physical node so that dynamic programming function $dp[0][C_1][C_2]\dots[C_m] = 0 (C_1 \le c_1, C_2 \le c_2, \dots, C_m \le c_m)$.

We define dynamic transfer equation of $dp[i][C_1][C_2]\dots[C_m]$ recursively as in Equation 15. when put i-th virtual node into other physical node, in this situation, the current dynamic programming function $dp[i][C_1][C_2]\dots[C_m]$ is calculated from former state $dp[i-1][C_1][C_2]\dots[C_m]$ where i-1 virtual nodes are succeeded to be mapped.

$$dp[i][C_1][C_2]\dots[C_m] = \min \begin{cases} dp[i-1][C_1-d_i][C_2]\dots[C_m] + M_{i1} \\ dp[i-1][C_1][C_2-d_i]\dots[C_m] + M_{i2} \\ \dots \\ dp[i-1][C_1][C_2]\dots[C_m-d_i] + M_{im} \end{cases} \tag{14}$$

when every virtual node is available in all physical nodes, the formulation of our problem is degenerated into simple multiple knapsack problem, according to the reducibility theorem in computer complexity field, it is easy to conclude that our problem is also NP-complete.

A. Proof

Theorem 1. *Two graphs are isomorphic if there is a correspondence between their vertex sets that preserves adjacency. Thus $G = (V, E)$ is isomorphic to $G^* = (V^*, E^*)$ if there is a bijection $\phi : V \to V^*$ such that $e_{ij} = v_i v_j \in E$ iff $\phi(v_i)\phi(v_j) \in E^*$.*

Denote the augmented graph of last step operation in Section... for protecting all physical node embedded with virtual node as $G^*(\hat{S}, \hat{L})$, when a physical node s_i fail. We proof, as follow, that any physical node which is embedded virtual node fail and the former graph $G(\hat{S}, \hat{L})$ is subgraph isomorphic with the remove graph $G^*\left(\hat{S} - \{s_i\}, \hat{L} - L_f\right)$.

Our algorithm $STAR$ find another alternate node of graph $G^*\left(\hat{S} - \{s_i\}, \hat{L} - L_f\right)$ for migrating the failure node s_i based method in Section .., the alternate node of failure node s_i is either a physical node embedded virtual node or a new startup physical node, as shown in Fig.**??** failure node s_1 migrate to node s_2 and in Fig.**??** failure node s_1 migrate to node s_4, respectively.

The other physical nodes except failure node s_i are able to keep their former physical node position in graph $G^*\left(\hat{S}-\{s_i\}, \hat{L}-L_f\right)$ as shown in Fig.**??** node s_2 and s_3, also migrate onto a physical node embedded virtual node or a new startup physical node, as shown in Fig.**??** node s_2 is migrated onto node s_3 and node s_3 is migrated onto node s_4.

That is to say our algorithm $STAR$ guarantee a bijection $\phi : S \rightarrow S^*$ that one to one map all nodes of graph $G(\hat{S}, \hat{L})$ onto nodes of graph $G^*\left(\hat{S}-\{s_i\}, \hat{L}-L_f\right)$ when physical node s_i fail.

The link between two physical nodes which are both not failure node s_i keep former position when the endpoint of this link keep former position as shown in Fig.**??** the link between node s_2 and node s_3, otherwise migrate onto new augmented link as shown in Fig.**??** the link between node s_2 and node s_3 migrate onto link between node s_3 and node s_4, exist "implicit" link in one node as shown in Fig.**??** the link between s_2 and s_3 exist in node s_2.

The link between two physical nodes whose one is failure node s_i augment a new link for alternating former existed link as shown in Fig.**??**,**??**,**??**,**??**, or exist "implicit" link in one node internally as shown in Fig.**??** the link between node s_1 and s_2 exist in node s_2. Ultimately exist all $e_{ij} = s_i s_j \in L$ iff $\phi(s_i)\phi(s_j) \in L^*$.

That is, the graph $G(\hat{S}, \hat{L})$ is subgraph isomorphic to graph $G^*\left(\hat{S}-\{s_i\}, \hat{L}-L_f\right)$.

IV. SCRIPT

Based on the discussion above, the computation problem of minimal graph alignment cost is transformed to solving optimal multiple knapsack problem, which is one of the fundamental combinational optimization problems. In our case, there are two sets of vertices corresponding to the two sets of star structure of $STAR_L$ and $STAR_R$ respectively, and the weight of the edge between star structures of $STAR_L$ and $STAR_R$ is the alignment cost between the corresponding two stars.

Firstly, we proposed the formalization of the multiple knapsack problem as follow:

$Cost_{ij}(1 \leq i \leq n, 1 \leq j \leq m)$ represent that the cost of i-th virtual node map the j-th substrate node. ∞ represent that there is not map relationship.

$c_i(1 \leq i \leq n)$ represent that the i-th virtual node demand computational resource, $C_i(1 \leq i \leq m)$ represent that the i-th substrate network maximum useful computation resource.

Objection: minimum $Cost_{ij} * M_{ij}$

constraints: $\sum\limits_{1\leq j\leq m} M_{ij} = 1$, $\sum\limits_{1\leq i\leq n} c_i * M_{ij} \leq C_j$

The cost should include and bandwidth cost due to the additional resource

To provide survivable virtual network embedding, we should add backup resources to guarantee that when a physical node $s_i \in S$ fails, the remaining physical resource plus the backup resources (i.e., $G\left(\hat{S}+S_b-\{s_i\}, \hat{L}+L_b-L_f\right)$) can still support a feasible mapping. That is, we can find both node mapping function and link mapping function to map the each virtual node and virtual link to physical network $G\left(\hat{S}+S_b-\{s_i\}, \hat{L}+L_b-L_f\right)$.

To facilitate finding node mapping function and link mapping function, we propose a graph decomposition solution to divide both virtual network and the physical network resource into small local stars which capture the local structure information of the original large networks. Then based on the stars, we formulate the network embedding problem as a multiple knapsack problem and propose dynamic programming based algorithm to solve the problem.

Theorem 2. *Given the VN request $G(V,E)$ and the physical network $G(S,L)$ when a physical node $s_i \in S$ fails, if we can find a sub-graph $G(S_v, P_v) \subset G\left(\hat{S}+S_b-\{s_i\}, \hat{L}+L_b-L_f\right)$ where $G\left(\hat{S}+S_b-\{s_i\}, \hat{L}+L_b-L_f\right)$ is the remaining physical resource plus the backup resources, such that, for any $v_i \in V$ and $e_{ij} \in E$, we can find $s_i \in S$ satisfying $v_i \rightarrow s_i$ and $f(i) \in F(i)$, and the survivable network service can be provided.*

Theorem 3. *Given the VN request $G(V,E)$ and the physical network $G(S,L)$ when a physical node $s_i \in S$ fails, if we can find a sub-graph $G(S_v, P_v) \subset G\left(\hat{S}+S_b-\{s_i\}, \hat{L}+L_b-L_f\right)$ where $G\left(\hat{S}+S_b-\{s_i\}, \hat{L}+L_b-L_f\right)$ is the remaining physical resource plus the backup resources, such that $G(S_v, P_v)$ is isomorphic to $G\left(\hat{S}, P\right)$, the survivable network service can be provided.*

Proof. When a physical node $s_i \in S$ fails, if we can map this request onto the physical network $G\left(\hat{S}+S_b-\{s_i\}, \hat{L}+L_b-L_f\right)$ while providing enough resource as demanded, the survivable service can be provided with the help of additional resource S_b and L_b. We denote the mapping physical graph as $G(S_v, P_v)$. Obviously, $G(S_v, P_v) \subset G\left(\hat{S}+S_b-\{s_i\}, \hat{L}+L_b-L_f\right)$ where $G\left(\hat{S}+S_b-\{s_i\}, \hat{L}+L_b-L_f\right)$ is the remaining physical resource plus the backup resources.

As both $G\left(\hat{S}, P\right)$ and $G(S_v, P_v)$ are the mapping physical graphs that satisfy the VN request, that is, for each v_i, there exists $s_j \in \hat{S}$ satisfying $v_i \rightarrow s_j$ and $s_{j'} \in S_v$ satisfying $v_i \rightarrow s_{j'}$; for each virtual link $e_{ij} \in E$, we can find $p_{i'j'} \in P$ with $v_i \rightarrow s_{i'}, v_j \rightarrow s_{j'}$ satisfying $e_{ij} \rightarrow p_{i'j'}$ and $p_{i''j''} \in P_v$ with $v_i \rightarrow s_{i''}, v_j \rightarrow s_{j''}$ satisfying $e_{ij} \rightarrow p_{i''j''}$.

Therefore there is a bijection $\phi : S_v \rightarrow \hat{S}$ such that $p_{ij} \in P_v$ iff $\phi(s_i)\phi(s_j) \in P$, that is, $G(S_v, P_v)$ is isomorphic to $G\left(\hat{S}, P\right)$. □

Clearly, isomorphic graphs have the same order and size. ϕ preserves the adjacency and non-adjacency of the vertices. To provide the survivable virtual network embedding under single node failure scenario, the backup resource added should support us to find a sub-graph $G(S_v, P_v)$ with the bijection ϕ to preserve adjacency information in $G\left(\hat{S}, P\right)$.

To find the surviviale service when node fails, Theorem 3 provides us a basic principle to find the backup VN mapping.

Let $G(V,E)$ and $G^*(V^*, E^*)$ be graphs. G and G^* are said to be isomorphic if there exist a pair of functions $f : V \rightarrow V^*$ and $g : E \rightarrow E^*$ such that f associates each element in V with

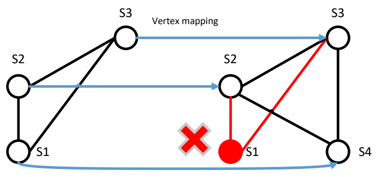


Fig. 11. Mapping

exactly one element in V^* and vice versa; g associates each element in E with exactly one element in E^* and vice versa, and for each $v \in V$, and each $e \in E$, if v is an endpoint of the edge e, then $f(v)$ is an endpoint of the edge g(e).

Theorem 4. *Two graphs are isomorphic if there is a correspondence between their vertex sets that preserves adjacency. Thus $G = (V, E)$ is isomorphic to $G^* = (V^*, E^*)$ if there is a bijection $\phi : V \to V^*$ such that $e_{ij} \in E$ iff $\phi(v_i)\phi(v_j) \in E^*$.*

Clearly, isomorphic graphs have the same order and size. ϕ preserves the adjacency and non-adjacency of the vertices.

When a physical node $s_i \in S$ fails, there exists a survivable virtual network embedding only when there exists a sub-graph of the the remaining occupying physical network plus the backup network which is isomorphic to $G\left(\hat{S}, P\right)$.

For a VN request $G(V, E)$ and a physical network $G(S, L)$, given a feasible mapping with occupying physical network $G\left(\hat{S}, \hat{L}\right)$, to provide the survivability to handle single node failure problem, some backup nodes and links should be added. We denote the additional backup node set and link set as S_b and L_b. Survivable virtual network embedding should guarantee that when a physical node $s_i \in S$ fails, the remaining physical resource plus the backup resources can still support a feasible mapping.

We can use $G(V, E, g)$ and $G^*(V^*, E^*, g^*)$ to represent two graphs. V represents the vertices, E the edges and g the rule linking edges with vertices. Using this notation it is possible to dene isomorphic graphs as follows:

Isomorphic Graph: Two graphs $G(V, E, g)$ and $G^*(V^*, E{\cdot}g^*)$ are isomorphic if there are bijections $f_1 : V \to V^*$ and $f_2 : E \to E^*$ such that for each edge $e_{ij} \in E$, $g(e_{ij}) = v_i - v_j$if and only if $g^*[f_2(e_{ij})] = f_1(v_i) - f_1(v_2)$:

Given the VN request $G(V, E)$ When a node fails, we denote the added node and link physical resources To guarantee survivable virtual network embedding when a node fails, obviously,

Given a VN request $G(V, E)$ and its embedding , computer the minimum additional resources

Given a VN request $G(V, E)$, for each node v_i, denote its attached physical node as $a(v_i)$. The node resource for virtual network embeddi

On the one hand, node failures directly. On the other hand, a failure of a physical node

two Survivability is the ability of a system such as a computer or a communication network to operate correctly in the presence of faults. From these studies, we can conclude that .

Given

One extreme case for the VN protection approach physical node may suffer the problem of instance fail.

Given the present-day importance of communications systems and infrastructures in general, networks should be designed and operated in such a way that failures can be mitigated. Network nodes might for instance fail , and so forth. Survivable have been used by the networking community to capture the ability of a communications system to maintain operation when confronted with network failures.

Therefore, in this paper, we just design a single node failure situation, but we will discuss that our algorithm extend for adapting multiple node failure situation.

A. Survivable

Survivability is guaranteed on the set of critical nodes of a VN through redundant virtual nodes with backup nodes set $B(V, F)$, we suppose all virtual node as critical node in this paper. In Fig.14, node set V of backup node $B(V, S)$ is $\{s_5, s_6, s_7\}$, function type set F of node is $\{\{f_1, f_2\}, \{f_1, f_4\}, \{f_2, f_3\}\}$. A backup (redundant) node b_i may not be able to assume full execution function of a failed critical node. Hence, the backup node may not have sufficient resources in terms of computation resource.

B. Survivable embedded Virtual Network Request

In this subsection, we define the SeVN design problem as follows, for a given VN request with $|V|$ nodes, the VN had been already embedded in substrate network SN and every node running function $f_i(f_i \in F_i^V)$, protect the VN with some augmented backup nodes and a set of appropriate links to connect these virtual nodes, and reserve sufficient computing and communication resources in these nodes and links to guarantee the restorability of VN request after a facility/substrate node failure.

There are different combinations of function type with respect to every nodes of embedded virtual network, but there is only one type of function type running onto substrate node which corresponding one embedded virtual node at one moment.

There exist many backup virtual nodes $B(V, S)$ which are abstracted from un-startup substrate network's node. When one fault node v_i appeared in virtual network request $G^V(V^V, E^V, f^V, C^V, B^V)$, Solving the survivable request of embedded virtual network is of approximately equivalence with asking 1-FNFT(G, B) of graph G and backup nodes set B without computation and bandwidth's limitation.

There exist popular and easy to understand method[3] as show in Fig.16, when every virtual nodes fail iteratively because the corresponding the failure substrate node occur failure, then directly startup new node, connect link among nodes $V \cup B$, augment node computing or augment demand bandwidth of existing link as shown in 16. The rude method demand startup 4 new nodes, 8 new edges, 16 node computing and 36 edge bandwidth.

After a failure, even an unaffected virtual node may be migrated from a working host node to its corresponding backup host node, the former method[3] do not consider the situation of node's migration.

We supposed a novel method STAR algorithm, which augment resource as shown in Fig.15, our method demand startup 2 new nodes, 5 new edges, 11 node computing and 23 edge bandwidth.

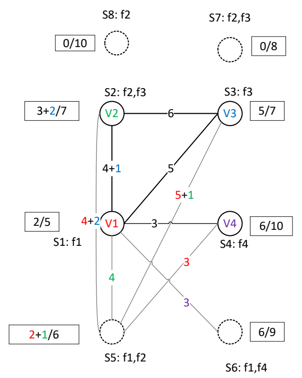


Fig. 12. FD

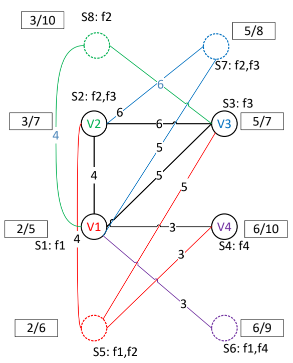


Fig. 13. FI

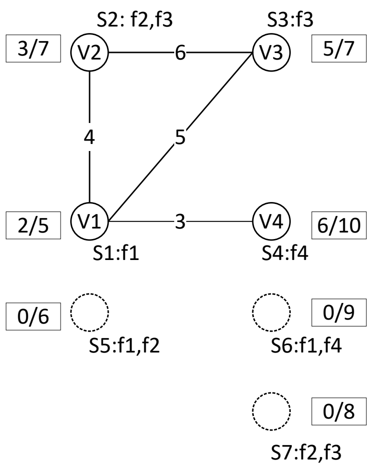


Fig. 14. embedded Virtual Network $G^V(V^V, E^V, f^V, F^V, C^V, B^V, M_V^V)$, $V^V : \{s_1, s_2, s_3, s_4, s_5, s_6, s_7\}$, $E^V = \{e_{12}, e_{23}, e_{13}, e_{14}\}$, $f^V : \{f_1, f_2, f_3, f_4\}$, $F^V = \{\{f_1\}, \{f_2, f_3\}, \{f_3\}, \{f_4\}, \{f_1, f_2\}, \{f_1, f_4\}, \{f_2, f_3\}\}$, $C^V = \{5, 7, 7, 10, 6, 9, 8\}$, $B^V = \{4, 6, 5, 3\}$, $M^V = \{s_1, s_2, s_3, s_4\}$

There exist many approaches is modeled as the VN embedding (VNE) problem which attracts a broad interest recently[2]. VNE is extremely important in order to maximize the number of coexisting VNs and increase the utilization of substrate infrastructures. Virtual Network Embedding deals with the allocation of virtual resources both in nodes and links. Therefore, it can be divided in two sub-problems: Virtual Node Mapping (VNoM) where virtual nodes have to be allocated in physical nodes and Virtual Link Mapping (VLiM) where virtual links connecting these virtual nodes have to be mapped to paths connecting the corresponding nodes in the substrate network. There has existed most studies about virtual network embedding method listed in a survey paper[2]. After all, we randomly choose a virtual network embedding method with considering three constraints: node computing, edge bandwidth and node function type from existed virtual network embedding method. As shown in Fig.**??**(a), we embed virtual network request as shown in Fig.**??** into the substrate network, node v_1 is embedded in node s_1, node v_2 is embedded in node s_2, node v_3 is embedded in node s_3, node v_4 is embedded in node s_4. demand node computing of every virtual node is not more than these virtual nodes corresponding substrate node's remain node computing. Every link of virtual network request exist a corresponding path consisted of substrate node and remain bandwidth of all links of corresponding path is more than demand bandwidth of the virtual network link.

C. Embedded Virtual Network

When a virtual network request has been inserted in to substrate network, augmented resource is attached into virtual network, embedded virtual network denote as $G^V(V^V, E^V, f^V, F^V, C^V, B^V, M^V)$ as shown in Fig.14. F^V denote a set of function type set $F_i^V(f_i^V \in F_i^V)$ which corresponding to every VN nodes v_i. C^V denote node's computational resource's capacity. B^V denote edge's bandwidth resource's capacity. M^V denote node's mapping relationship of virtual network embedding algorithm, $M_i = j$ represent that the i-th VN's node had been embedded at j-th substrate network node. Besides, with respect to the node embedding, we suppose a assumption that all virtual nodes of VN should be mapped on isolated physically substrate nodes.

D. Node Failure

Given the present-day importance of communications systems and infrastructures in general, networks should be designed and operated in such a way that failures can be mitigated. Network nodes might for instance fail due to malicious attacks, natural disasters, unintentional cable cuts, planned maintenance, equipment malfunctioning, and so forth. Survivable have been used by the networking community to capture the ability of a communications system to maintain operation when confronted with network failures.

In general, the substrate nodes simultaneous failure is mutual independent , a single node failure happen at most of time[3]. Therefore, in this paper, we just design a single node failure situation, but we will discuss that our algorithm extend for adapting multiple node failure situation.

E. Survivable

Survivability is guaranteed on the set of critical nodes of a VN through redundant virtual nodes with backup nodes set $B(V, F)$, we suppose all virtual node as critical node in this paper. In Fig.14, node set V of backup node $B(V, S)$ is $\{s_5, s_6, s_7\}$, function type set F of node is $\{\{f_1, f_2\}, \{f_1, f_4\}, \{f_2, f_3\}\}$. A backup (redundant) node b_i may not be able to assume full execution function of a failed critical node. Hence, the backup node may not have sufficient resources in terms of computation resource.

F. Survivable embedded Virtual Network Request

In this subsection, we define the SeVN design problem as follows, for a given VN request with $|V|$ nodes, the VN had been already embedded in substrate network SN and every node running function $f_i(f_i \in F_i^V)$, protect the VN with some augmented backup nodes and a set of appropriate links to connect these virtual nodes, and reserve sufficient computing and communication resources in these nodes and links to guarantee the restorability of VN request after a facility/substrate node failure.

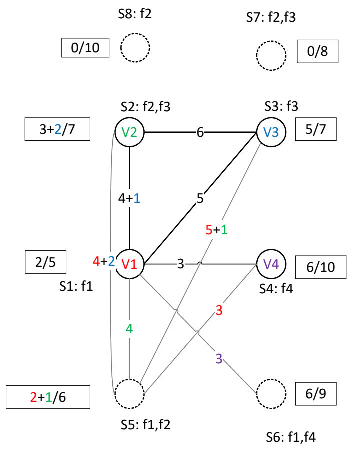


Fig. 15. *STAR* algorithm protection scheme

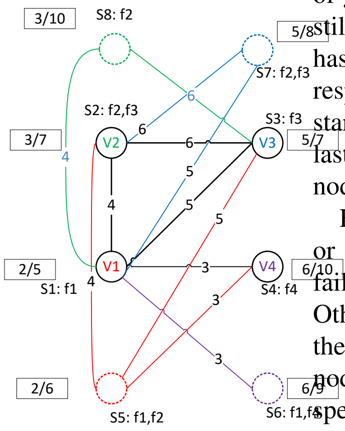


Fig. 16. 1+1-protection scheme

There are different combinations of function type with respect to every nodes of embedded virtual network, but there is only one type of function type running onto substrate node which corresponding one embedded virtual node at one moment.

There exist many backup virtual nodes $B(V,S)$ which are abstracted from un-startup substrate network's node. When one fault node v_i appeared in virtual network request $G^V(V^V,E^V,f^V,C^V,B^V)$, Solving the survivable request of embedded virtual network is of approximately equivalence with asking 1-FNFT(G,B) of graph G and backup nodes set B without computation and bandwidth's limitation.

There exist popular and easy to understand method[3] as show in Fig.16, when every virtual nodes fail iteratively because the corresponding the failure substrate node occur failure, then directly startup new node, connect link among nodes $V \cup B$, augment node computing or augment demand bandwidth of existing link as shown in 16. The rude method demand startup 4 new nodes, 8 new edges, 16 node computing and 36 edge bandwidth.

After a failure, even an unaffected virtual node may be migrated from a working host node to its corresponding backup host node, the former method[3] do not consider the situation of node's migration.

We supposed a novel method STAR algorithm, which augment resource as shown in Fig.15, our method demand startup 2 new nodes, 5 new edges, 11 node computing and 23 edge bandwidth.

V. Heuristics Algorithms

A. Heuristics Algorithms Design Formulation of SeVN

With the discussion above, for a given N-nodes VNR, SeVN is designed within augment graph G^* of N+k nodes through properly selecting the necessary nodes or links between these nodes and dimensioning the resources requirements associated with these nodes and links.

Our design objective is to minimize the total amount of such resources(startup node number, node computing, edge bandwidth) while still guaranteeing that if a node fails (that is, the node and its adjacent links are removed in the graph G^*) virtual network request's topological graph still keep integration of graph G topological structure and insist normal work, we can still assign each node/link of VN to a node/link of the SeVN, that has sufficient node computing and edge's bandwidth resources respectively. It is our objection that firstly minimize number of start-up backup nodes, then minimize node computing resource, last minimize edge bandwidth resource to construct function node fault tolerant graph G^*.

Furthermore, there are two different cases. If re-embedding or migrating the unaffected node(un-failed node) is allowed for failure recovering, it is known as the FD-SeVN design problem. Otherwise, after each failure, if the failed node is restored in the only one backup node without migrating other unaffected node, it is referred to as the FI-SeVN design problem. Generally speaking, designing FD-SeVN is a combinatorial optimization problem and needs to be investigated in depth, while FI-SeVN exists exclusively and could be figured out easily.

B. Design Procedure

In this section, we propose a heuristic algorithms for FD-SeVN problem, as well as an SeVN problem's algorithm with resources sharing consideration fitting for both FD-SeVN and FI-SeVN.

Decompose topological graph of virtual network VN into N star structure $STAR$, construct match relationship between star structures. This match relationship is corresponding node transform cost relationship.

However, as a matter of fact, computing the minimum additional resources needed to convert one attributed graph to another (hereafter called graph alignment problem, GAP) is an NP-complete problem which could be reduced from the Graph Edit Distance problem [5]. Therefore, we propose a heuristic algorithms in detail, and we first decompose a graph to a set which contains star structures which retains certain structural information of the graph of embedded virtual network. Then, the graph alignment cost could be approximated by the matching cost(transform cost) between two graphs based on their star representations. This approach is elaborated as follows.

The general idea of the heuristic for FD-SeVN design is to consider the failure of primary nodes in virtual network node's label sequentially, and in each step, compute the minimum additional resources needed to reassign the task graph based on an incremental approach (i.e., recovering from the current node failure should take not only the survived primary nodes/links resources into consideration, but also the survived redundant resources reserved for previous node failure). After iteratively execute all the node failures, SeVN is constructed within the Star Structure with the added redundant resources in each step. Generally speaking, select a optimal node v_i to minimum additional resource with respect to every steps.(Min value for every step)

Redistribute physical resource corresponding to augment graph G^i with additional computational and bandwidth resource. concrete step is that apply for computational resource for virtual nodes and complete path around bandwidth for virtual edges.

C. FD-SeVN Algorithm

1) Graph Decomposition: Star Structure: A star structure s is an attributed, single-level, rooted tree which can be represented by a 7-tuple $star = (v^*, c^*, C^*, f^*, F^*, L^*, B^*)$ as shown in Figure 3, where v^* is the root node, c^* is the node's demand computing, C^* is the node's remain computing capacity could be resigned again, f^* is the node's function type, F^* is function type set of the node, L^* is other all nodes adjacent with nodes v^*, B^* is the bandwidth of each link e_{ij} associated with the root node v^*. Edges exist between the root node and its adjacent nodes, and no edge exists among its adjacent nodes.

More exactly, for node v_i in an attributed graph $G^V(V^V, E^V, f^V, F^V, L^V, C^V, B^V, M^V)$, we can generate a star structure $star_i$ corresponding to v_i in the following way, $star_i = (v_i^V, c_i^V, C_i^V, f_i^V, F_i^V, L_i^V, B_i^V)$ where f_i^V is a function type f_i running in node v_i, $F_i^V = \{f_j|$ for all f_j belong node v_i function set$\}$. $L_i^V = \{v_j|$ for all nodes v_j adjacent with node $v_i\}$, $B_i^V = \{b_{i,j}|$ for all $e_{i,j} \in L_i^V\}$. Accordingly, we can derive N star structures from topological graph of substrate network with embedded virtual network of N nodes (we uniformly discuss all N virtual nodes fail). In this way, topological graph of embedded virtual network can be transformed to a star structure. A quintesfntial example should be cited that construct $star_2$ corresponding to node v_2, $star_2 = (v_2, 3, 7, f_2, \{f_2, f_3\}, \{v_1, v_3\}, \{b_{21} = 4, b_{23} = 6\})$, virtual node v_1 correspond star structure $star_1 = (v_1, 2, 5, f_1, \{f_1\}, (v_2, v_3, v_4), \{b_{12} = 4, b_{13} = 5, b_{14} = 3\})$.

2) Match Items: Based on the topological graph of the embedded virtual network, we construct star structure set $STAR^L$, which erase some star structure corresponding to one substrate node without embedding virtual node. when calculating the virtual node v_i which map to substrate node s_i failure, constructing star structure set $STAR^R$ with erasing the node v_i and the edge adjacent with node v_i as shown in Fig.3.

3) Alignment Cost Matrix: Due to the particularity of star structure, the alignment cost between two star structure set can be computed easily as below. For the two star structures $star_i^L$ and $star_j^R$ which is from $STAR^L$ and $STAR^R$ respectively, if node f_i^V of $star_i^L$ is not included in F_j^V, the alignment cost of $star_i^L$ to $star_j^R$ is ∞, otherwise the alignment cost of $star_i^L$ to $star_j^R$ is

$$\lambda(star_i^L, star_j^R) = \sum_{v_u \in L_i \cap L_j} \gamma |b_{i,u} - b_{j,u}|_0 + \sum_{v_u \in L_i - L_j} \gamma b_{i,u} + \beta |c_i - c_j|_0 + \delta(v_i, v_j) + \theta(v_j)$$

where δ, θ and $|x|_0$ is defined as follows:

$$\delta(v_i, v_j) = \begin{cases} \mathrm{M_m} & v_i \neq v_j \\ 0 & otherwise \end{cases}$$

$$\theta(v_j) = \begin{cases} \mathrm{C_s} & v_j \ is \ free \\ 0 & otherwise \end{cases}$$

$$|x|_0 = \begin{cases} x & if \ x \geq 0 \\ 0 & if \ x \leq 0 \end{cases}$$

$\delta(v_i, v_j) \neq 0$ indicates that root node v_i should be reallocated to node v_j. M_m is migration cost, C_s is startup new node cost.we define $\lambda(star_i, star_j)$ according to graph edit distance[6]. In mathematics and computer science, graph edit distance (GED) is a measure of similarity (or dissimilarity) between two graphs.

In this way, both the topological graph of embedded virtual network and the residual graph are composed to two star structure sets. With such a graph decomposition once a specific virtual node failure, an alignment matrix of the two star structure set could be constructed based on the alignment cost definition above. Therefore, the proposed GAP could be transformed to a (multiple knapsack problem) which will be investigated in the following part. For example, the alignment cost matrix when virtual network v_1 fail as follow.

$$\begin{array}{ccccccc}
 & R_{S_2} & R_{S_3} & R_{S_4} & R_{S_5} & R_{S_6} & R_{S_7} \\
L_{V_1} & \infty & \infty & \infty & \boxed{C_s+(2)+12} & C_s + (2) + 12 & \infty \\
L_{V_2} & \boxed{4} & \infty & \infty & C_s + (3) + 10 & \infty & C_s + (3) + 10 \\
L_{V_3} & M_m + (5) + 11 & \boxed{5} & \infty & \infty & \infty & C_s + (5) + 11 \\
L_{V_4} & \infty & \infty & \boxed{3} & \infty & C_s + (6) + 3 & \infty
\end{array}$$

To elaborate this approach, Figure 3 shows how the 4-nodes VN be decomposed into a set of four star structures (shown as $STAR_L$), and each containing a primary task node (also called root node which is highlighted and color as blue), its adjacent links and its neighboring nodes.

D. Multiple Knapsack Problem

Based on the discussion above, the computation problem of minimal graph alignment cost is transformed to solving optimal multiple knapsack problem, which is one of the fundamental combinational optimization problems. In our case, there are two sets of vertices corresponding to the two sets of star structure of $STAR_L$ and $STAR_R$ respectively, and the weight of the edge between star structures of $STAR_L$ and $STAR_R$ is the alignment cost between the corresponding two stars.

Firstly, we proposed the formalization of the multiple knapsack problem as follow:

$M_{ij} = 1 (1 \leq i \leq n, 1 \leq j \leq m)$ represent that whether the i-th virtual node map the j-th substrate node, n and m denote virtual node's size and substrate node's size respectively.

$Cost_{ij} (1 \leq i \leq n, 1 \leq j \leq m)$ represent that the cost of i-th virtual node map the j-th substrate node. ∞ represent that there is not map relationship.

$c_i (1 \leq i \leq n)$ represent that the i-th virtual node demand computational resource, $C_i (1 \leq i \leq m)$ represent that the i-th substrate network maximum useful computation resource.

Objection: minimum $Cost_{ij} * M_{ij}$
constraints: $\sum_{1 \leq j \leq m} M_{ij} = 1$, $\sum_{1 \leq i \leq n} c_i * M_{ij} \leq C_j$

1) Dynamic Programming Equation: We propose a dynamic programming equation method for solving the multiple knapsack problem. Assume $C_1, C_2, \ldots, C_n$, C are strictly positive integers. Define $dp[i][C_1][C_2] \ldots [C_m] = 0$ to be the minimum value that can be attained with n capacity variables which less than or equal to C_i using items up to knapsack i.

Initial state $dp[i][C_1][C_2] \ldots [C_m] = 0$ is firstly assigned as infinity ∞. $dp[i][C_1][C_2] \ldots [C_m] = 0$ when there is no any virtual node which is confirmed to map into another node. The total cost of current state is zero. $dp[i][C_1][C_2] \ldots [C_m]$ when i nodes is succeeded to be mapped another nodes. The total cost of current state is minimal cost from optimal node location.

We can define $dp[i][C_1][C_2] \ldots [C_m]$ recursively as in Equation 15, when put i-th node into another node, in this situation, the current state is calculated from former i-1 nodes are succeeded to be mapped.

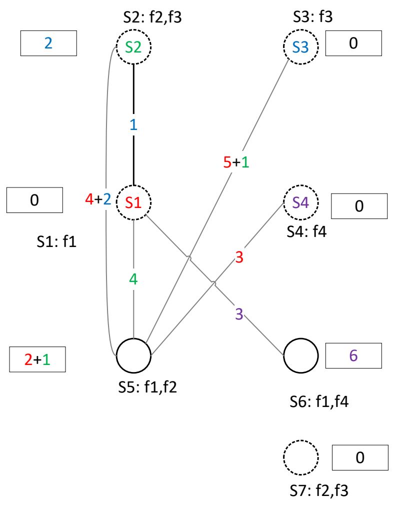


Fig. 17. Augmented Resource

$$\min\begin{cases} dp[i-1][C_1-c_i][C_2]\ldots[C_m]+\lambda(star_i^L, star_1^R) \\ dp[i-1][C_1][C_2-c_i]\ldots[C_m]+\lambda(star_i^L, star_2^R) \\ \cdots \\ dp[i-1][C_1][C_2]\ldots[C_m-c_i]+\lambda(star_i^L, star_n^R) \end{cases} \tag{15}$$

dynamic programming method could be applied to solve the multiple knapsack problem whose time complexity is $O[(n+b)*n*\prod_{i=1}^{n+b}C^i]$, which is pseudo-polynomial time. Space complexity is $O[n*\prod_{i=1}^{n+b}C^i]$.

VI. SeVN augment resource embed formulation

In this section, we refer that design procedure of augmented resource allocation of SeVN graph, we also give out resource demand allocation method fitting for both FD-SeVN and FI-SeVN embedding, since the difference between FD-SeVN and FI-SeVN in embedding approach could be reconciled by a general resources sharing constraint. More details would be elaborated in the following part.

Besides, with respect to the node embedding even and virtual links embedding, because not all the virtual links or not all their bandwidth would be employed simultaneously under single node failure, some virtual links could share substrate resources if they are embedded on the same substrate link, which would reduce the total substrate bandwidth needed.

A. Embed Augmented Backup Resource

From Sec.V-D1, we have obtained map relationship with respect to node v_i failure, then compute out that every node should to be reallocated how much computation and every link should to be reallocate how much bandwidth in substrate network SN as shown in Fig.17. As most virtual network embedding algorithm, node mapping phase had been completed with respect to our algorithm, the next procedure give out edge mapping phase. we use standard shortest path algorithm, dijkstra algorithm, for requesting path of substrate node with respect to every virtual network's node, then reallocate some bandwidth to this path of substrate network for this virtual network survivable request.

Finally, we repeat the procedure of graph decomposition and multiple knapsack problem for each following node failure to construct the graph of final FD-SeVN. As shown in Fig.15, partial augmented resource which is labeled consecutively with different color of the SeVN for each node failure is presented. It is worth noting that, every step of partial augmented resource is based on the previous step.

B. Algorithm procedure

In this section, we describe the whole our algorithm procedure of SeVN problem in Alg.**??**

Algorithm 2 survivable embedded virtual network request algorithm

```
Require: $G^V(V^V, E^V, f^V, C^V, B^V)$: topological graph of virtual network's request; $G^S(V^S, E^S, F^S, C^S, B^S)$, topological graph of substrate network .
Ensure: generate SeVN and embed SeVN augment resource into substrate network.
 1: embed VN $G^V$ into substrate network $G^S$.
 2: extract embedded virtual network eVN $G^*$ from SN corresponding to this VN embedding request
 3: for all $v_i$ such that $v_i \in$ virtual node do
 4:     decompose eVN into two star structure sets $STAR^L$ and $STAR^R$ from graph $G^*$.
 5:     construct items based star structure set $STAR^L$.
 6:     construct knapsacks based star structure set $STAR^R$ from embedded virtual network eVN.
 7:     construct alignment cost matrix based graph edit distance method[6]
 8:     solve multiple knapsack problem through dynamic programming.
 9:     add new nodes ,connect new edge, allocate node computing and edge's bandwidth into $G^*$ to construct new graph $G^*$.
10: end for
11: embed new additional(augment) boot up resource or startup new node from SeVN $G^*$ into substrate network $G^S$
12: return  SeVN,$G^S$
```

1 node+2 node computaion+12 bandwidth 0 node+1 node computaion+5 bandwidth 1 node+5 node computaion+11 bandwidth 1 node+6 node computaion+3 bandwidth =3 node+14 node computaion+31 bandwidth

1 node+2 node computaion+12 bandwidth 0 node+1 node computaion+5 bandwidth 0 node+2 node computaion+3 bandwidth 1 node+6 node computaion+3 bandwidth =2 node+11 node computaion+23 bandwidth

VII. Evaluation

We first describe the simulation setup, then present the simulation results.

A. Simulation setup

1) experimental parameter: For any VN request, the number of VN nodes is randomly determined by a uniform distribution between 3 and 8 and each pair of virtual nodes is randomly connected with probability 0.8. The computing requirements on

VN nodes follow a uniform distribution from 5 to 10, as well as the bandwidth on VN links from 10 to 20.

VN requests arrive randomly over a timespan of 600 time slots and the inter-arrival time is assumed to follow the Geometric distribution at a rate of 1.0 per time slot. The resource lease times of each VN request follows the uniform distribution as well at a rate range from 1 and 100 per time slot. A high request rate and long lease times ensures that the physical infrastructure is operating at high utilization. we suppose a assumption that all virtual nodes of VN should be mapped on isolated physically substrate nodes.

We assume the relative cost of computing compared with bandwidth is 3[7], [8], which means $\theta = 3$. The SN topologies used are randomly generated with 30 nodes using the GT-ITM tool[9]. The computing(bandwidth) resources of the substrate nodes (links) are real numbers uniformly distributed between 10 and 30 (100 and 200), respectively. To model the facility node failure scenario, we choose all substrate facility nodes of substrate network to fail one by one, and statistically calculating migration frequency in every 2000 time units.

In our proposed two-step method for $SeVN$ problem, we employ a proactive failure dependent protection(FD) based $SeVN$ problem design with sufficient redundancy before embedding process in order to conquer the limitation of existing work, which consider the redundancy within the embedding process and failure independent protection based migration approach in resource efficiency.

For performance comparisons, besides our scheme (named by $STAR$), we also implemented another one resilient algorithms RVN as follows: RVN algorithms connect links between backup and critical nodes, and backup and backup nodes, but not connect links between critical and critical nodes. Moreover, We further compare that to a virtual network embedding request algorithm , where VN do not have survivability requirement, and the virtual network embedding request (labeled bVN), as a baseline to gauge the amount of augmented resources consumed for survivability. We suppose that when a virtual network is embedded in substrate network, the survivable guarantee has been successful with respect to algorithm bVN.

Based on the resource of substrate network whether shared or non-shared[10], we independently implement the two situation with respect to resource distribution pattern.

Every virtual network's request exist a lease time span, statistically note virtual or substrate network's performance metric in real-time situation. Because the experimental data of every algorithm performance metric fluctuate, we incrementally note down experimental data of every virtual or substrate network performance metric, then average the experimental data by successful virtual network embedding as denominator.

2) performance metric: In this section, we evaluate the performance of the SeVN problem heuristic method $STAR$ when allocating resources with considering all virtual nodes. In particular, we focus on the resource usage of the physical infrastructure and virtual network request, and the admission rates of survivable eVN requests.

Different measures[2] of heuristic algorithms are presented in this section in terms of average acceptance ratio, embedding stress, and other measures as described in the follow.

- **Acceptance Ratio**: Opposite to the concept of Blocking Probability. Ratio of a virtual network request is successful embedded into substrate network . Ratio of embedded virtual networks that were successfully augmented for survivable guarantees and embedded into the substrate topology. Ratio of a embedded virtual network is successful survivable when the virtual network had been embedded in substrate network.
- **Active Node**, represent that number of substrate network booted nodes and number of virtual network request demand nodes. This metric is especially useful in the context of energy efficient VNE algorithms.
- **Path Length**: the path length metric measures the number of links between two interconnected virtual nodes, the number of links between two substrate nodes that are mapped to two interconnected virtual nodes.
- **Cost, Revenue, and Cost/Revenue**: Cost: Sum of node computing ability or link bandwidth resources being used for the embedding. Revenue: Sum of node computing ability or link bandwidth demands realized for the virtual networks. Cost/Revenue: The ratio indicates the virtualization overhead.
- **Stress**: Average number of virtual links/nodes that have been assigned to the substrate links/nodes or substrate links/nodes.
- **Migration Frequency Ratio**: The number of migrations refers to the number of virtual nodes that need to be migrated to new facility nodes in case corresponding substrate nodes fail.
- **utilization**: Bandwidth utilization or computing utilization of substrate links/nodes, in each SN entity (node or link), the sum of the spent substrate resources due to the mapping of virtual entities divided by the total amount of resources.
- backup nodes number: Number of backups The number of backups metric counts the number of backup resources that are set up for a virtual entity. Several additional substrate entities can be reserved to serve as a replacement in case the entity hosting the virtual entity fails.

All simulations are ran on a common PC, which is equipped with one Intel (R) i5-5200U CPU(2.20GHz) (4Cores) and 8.00G-B RAM.

B. Accepted VNRs and SeVNRs

Fig.18 show that number of accepted virtual network request and accepted survivable embedded virtual network request.

C. Acceptance Ratio

In this section, acceptance ratio of the embedded VN's survivable request with respect to $SeVN$ problem. Moreover, the acceptance ratio of VN without survivability requirement is presented as a baseline to gauge the impact of additional amount of resources consumed for survivability on the service provisioning capability of SN.

In Fig.19, acceptance ratio drops quickly before 200 time units because there are sufficient substrate resources for the arrived VN requests, and when the amount of running VN requests in system are stable (after 400 time units), acceptance ratio would keep consistent. Fig.19 also depicts that $STAR$ lead to higher

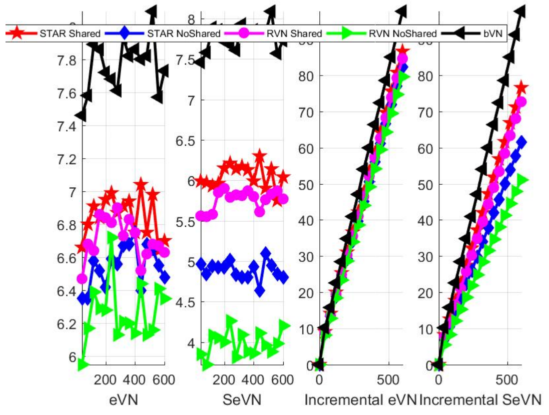


Fig. 18. Accepted VNRs and Accepted SeVNRs

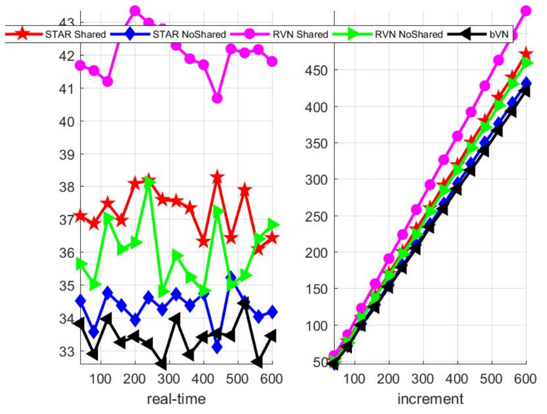


Fig. 20. Virtual Network Active Node

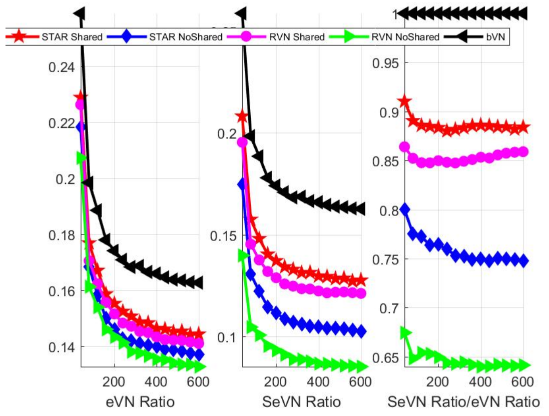


Fig. 19. Acceptance Ratio

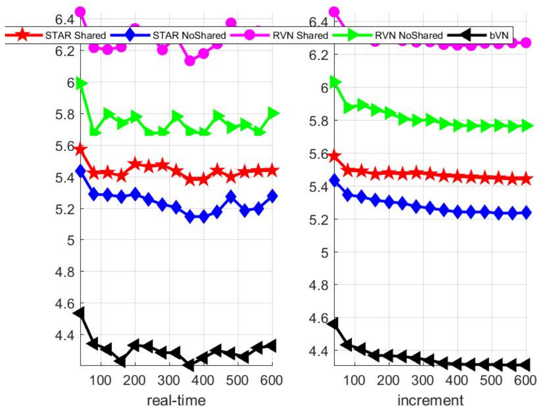


Fig. 21. Virtual Network Average Active Node

acceptance ratio than RVN algorithm needless of resources sharing. In particular, comparing with RVN-Shared, $STAR$-Shared approach achieves almost 6% improvement of acceptance ratio in the long run. Intuitively, it is the result of fact that, for $STAR$-NoShared embedding, the backup node is associated with a lot of resources since it has to emulate every primary node after it fails, and so, embedding of backup node is vulnerable. However, the acceptance ratio suffers at least 6% losses than $STAR$-Shared caused by redundant resources consumption for survivability purpose.

D. Active Node

1) Virtual Network Active Node: Number of nodes that need to be active in order to host all the virtual networks.

Virtual Network Active Node Fig.20:

Virtual Network Average Active Node Fig.21:

2) Substrate Network Active Node: The number of active substrate nodes is related to the average length of substrate paths,because additional nodes are used to forward communication data between end nodes. Therefore, the probability that previously switched off nodes are selected to forward data rises. Regarding energy efficiency, the number of nodes that need to be turned for an embedding can be a rough estimation of energy consumption. In general, the ratio between running nodes and the total number of substrate nodes should be taken into account.

Substrate Network Active Node Fig.22:

Substrate Network Average Active Node Fig.23:

3) Substrate Network Active Node/ Virtual Network Active Node : Substrate Network Active Node/ Virtual Network Active Node Fig.24:

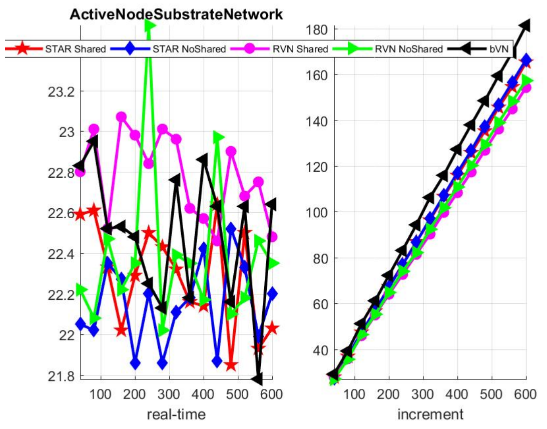


Fig. 22. Substrate Network Active Node

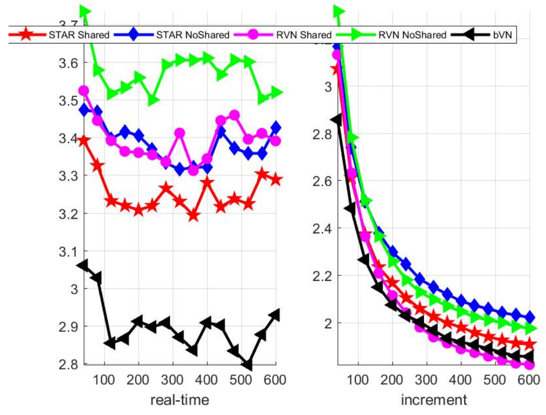


Fig. 23. Substrate Network Average Active Node

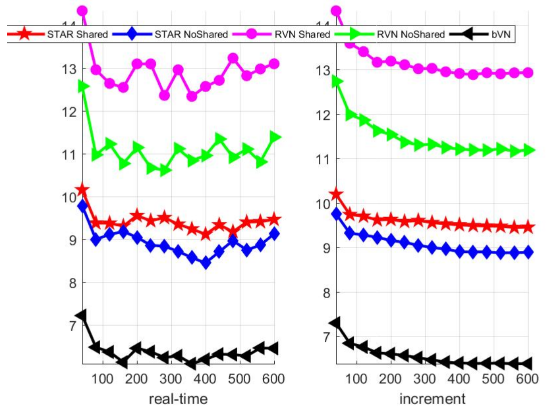


Fig. 26. Virtual Network Average Path Length

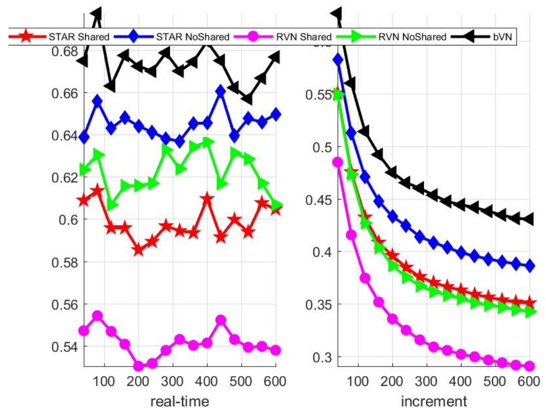


Fig. 24. Substrate Network Active Node / Virtual Network Active Node

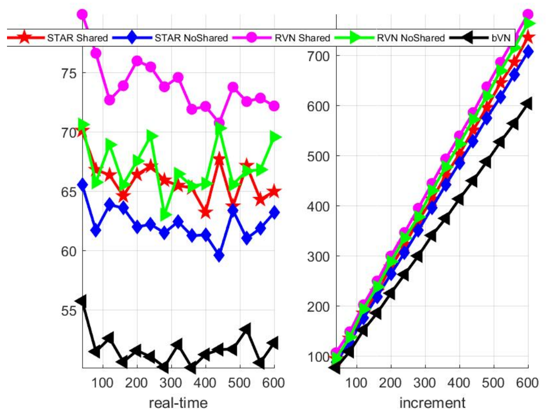


Fig. 27. Substrate Network Path Length

E. Path Length

1) Virtual network Path Length: Virtual Network Path Length Fig.25

Virtual Network Average Path Length Fig.26:

2) Substrate network Path Length: The longer a corresponding path, the more resources had to be reserved for the embedding of the virtual link. Since every substrate node (except the receiving node) that is part of a path will take some time to forward packages sent via this path, the quality of service is influenced by the path length. In general, the package delay increases in connection with the length of a path.

Substrate Network Path Length Fig.27:

Substrate Network Average Path Length Fig.28:

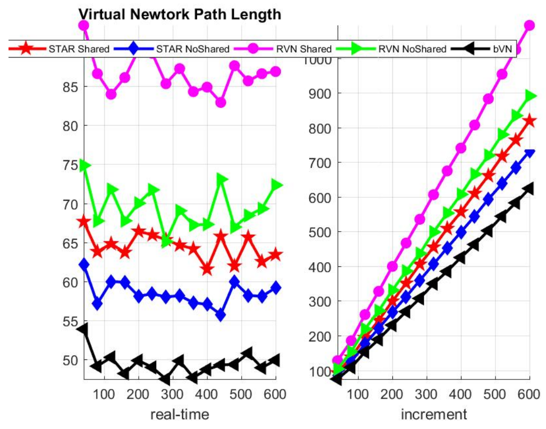


Fig. 25. Virtual Network Path Length

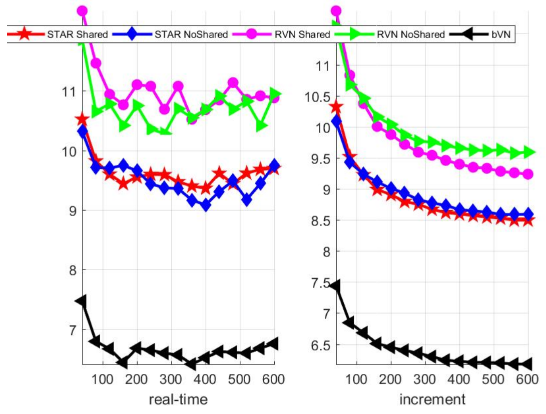


Fig. 28. Substrate Network Average Path Length

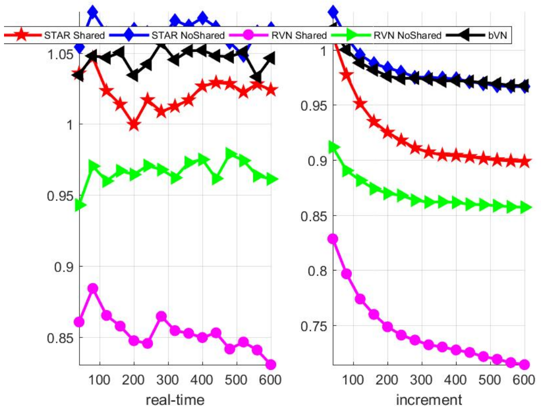


Fig. 29. Substrate Network Path Length/ Virtual Network Path Length

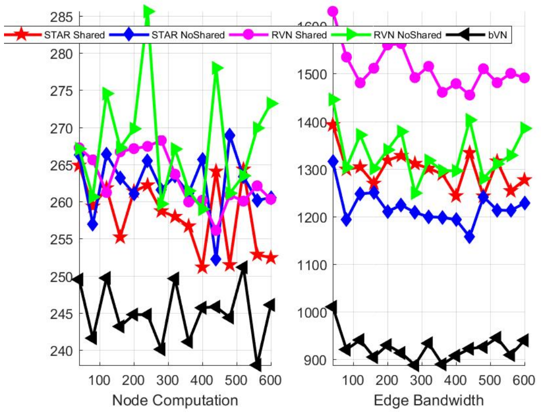


Fig. 30. Substrate Network Real-Time Cost

3) Substrate Network Path Length/ Virtual Network Path Length: Substrate Network Path Length/ Virtual Network Path Length Fig.29:

F. Cost, Revenue and Cost/Revenue

1) Cost: In this work, the embedding cost is calculated as the cost of the substrate resources (i.e. cost of node computing ability on all facility nodes, edge bandwidth on all fiber links) consumed to satisfy the $SeVN$ resource requirements.

Substrate Network Real-Time Cost Fig.30:

Substrate Network Increment Cost Fig.31:

Substrate Network Average Real-Time Cost Fig.32:

Substrate Network Increment Average Cost Fig.33:

2) Revenue: Virtual Network Real-Time Revenue Fig.34:

Virtual Network Increment Revenue Fig.35:

Virtual Network Average Real-Time Revenue Fig.36:

Virtual Network Average Increment Revenue Fig.37:

3) Cost/Revenue: By dividing Cost by Revenue, varying Cost values are balanced. The higher the value, the more resources were needed to embed the SeVNs.

Cost/Revenue Fig.38:

Increment Cost to Increment Revenue Fig.39:

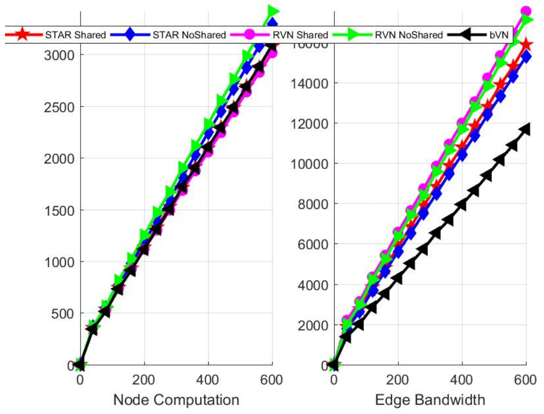


Fig. 31. Substrate Network Increment Cost

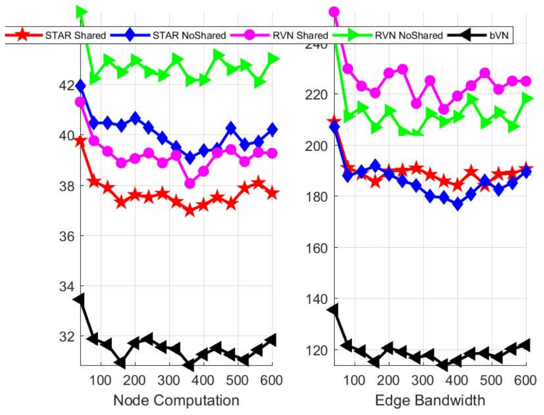


Fig. 32. Substrate Network Average Real-Time Cost

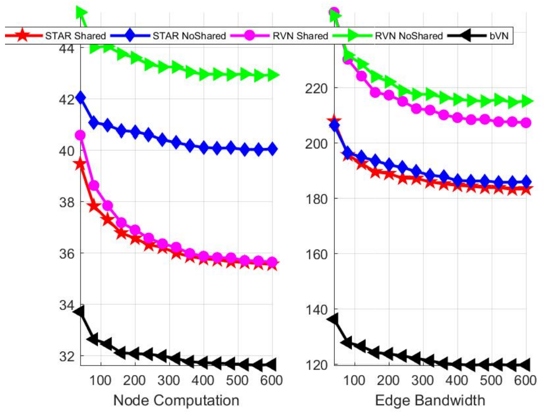


Fig. 33. Substrate Network Average Increment Cost

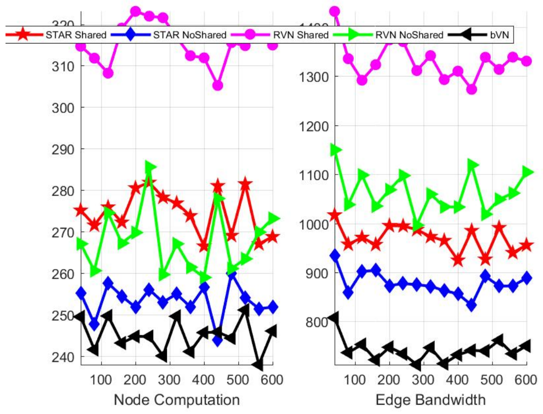


Fig. 34. Virtual Network Real-Time Revenue

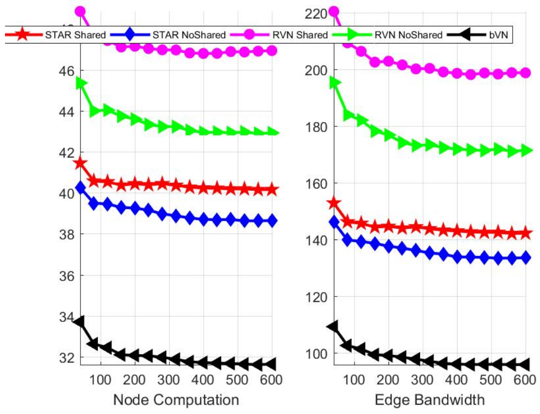


Fig. 37. Virtual Network Average Increment Revenue

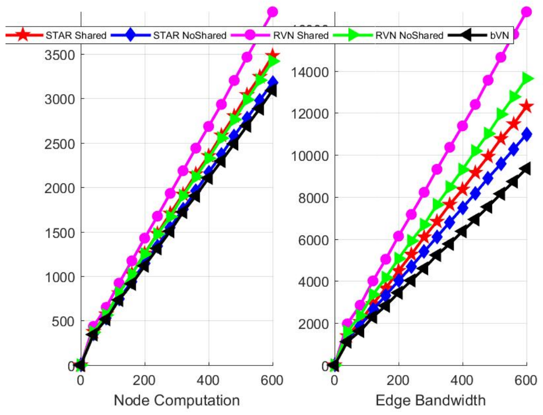


Fig. 35. Virtual Network Increment Revenue

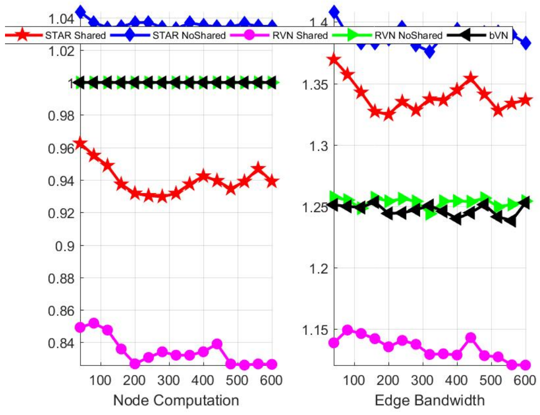


Fig. 38. Cost/Revenue

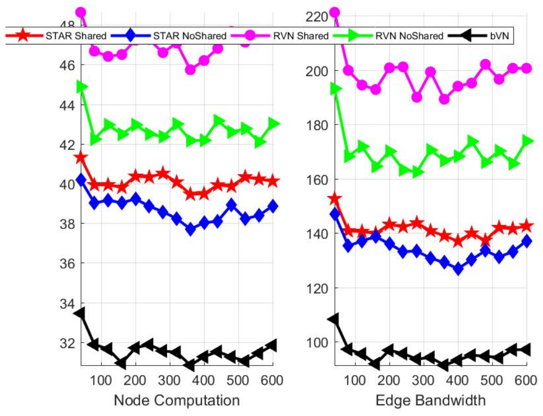


Fig. 36. Virtual Network Average Real-Time Revenue

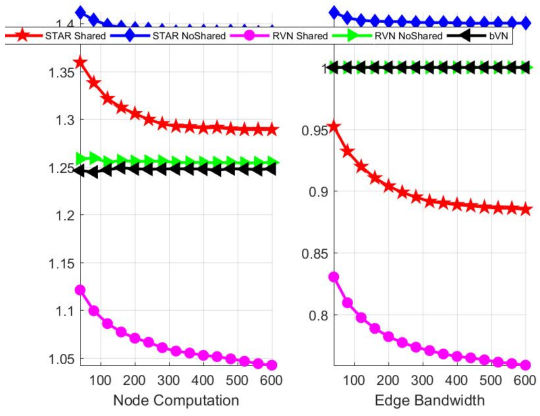


Fig. 39. Increment Cost/Increment Revenue

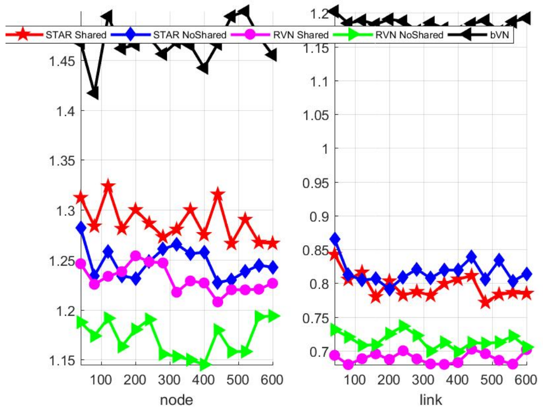


Fig. 40. Migration Frequency Ratio

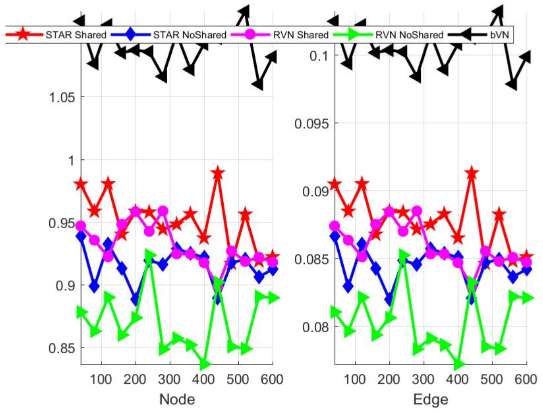


Fig. 41. Stress

G. Migration Frequency

Typically, at least the virtual nodes hosted on the failing substrate nodes have to be moved. Other constraints, like maximum path length, might however trigger even more migrations. Since migrations are resource intensive, they should be kept to a minimum. Fig.40

H. Stress

The stress level of a substrate entity reflects the number of virtual entities that are mapped onto it. The more virtual enti ties use the same substrate resource, the higher the impact regarding possible side effects. For example, mapping many virtual nodes onto a single substrate node keeps the CPU of the host operating system busy. A high substrate link stress might result in some additional packet delay because the resources of the substrate link and the host interfaces communicating through this link have to be shared between virtual entities. The stress of a SN element (node or link)is the number of virtual instances mapped on top of it. Stress Fig.41:

I. Utilization

Utilization measures, in each SN entity (node or link), the sum of the spent substrate resources due to the mapping of virtual entities divided by the total amount of resources. This metric is a more precise measurement tool than the stress level metric, because it also takes into account the magnitude of the resource usage instead of simply counting the number of virtual entities that use a resource. For example, to measure the utilization of a substrate node, the sum of mapped computing resources divided by the node computing capacity of the node could be used. The usage of a substrate link could be based on the sum of mapped bandwidth resources divided by the total link bandwidth capacity.

Utilization Fig.42:

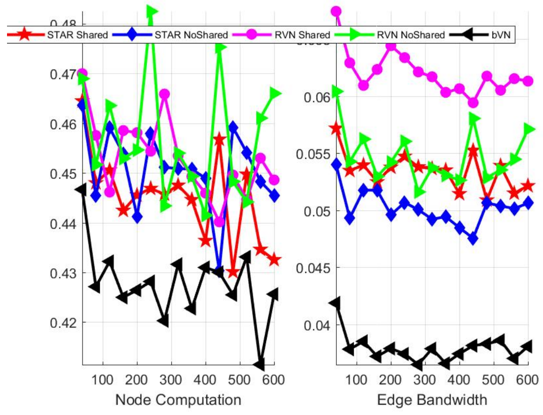


Fig. 42. Utilization

References

[1] J. Lischka and H. Karl, "A virtual network mapping algorithm based on subgraph isomorphism detection," in *Proceedings of the 1st ACM workshop on Virtualized infrastructure systems and architectures*, pp. 81–88, ACM, 2009.

[2] A. Fischer, J. F. Botero, M. T. Beck, H. De Meer, and X. Hesselbach, "Virtual network embedding: A survey," *IEEE Communications Surveys & Tutorials*, vol. 15, no. 4, pp. 1888–1906, 2013.

[3] W.-L. Yeow, C. Westphal, and U. C. Kozat, "Designing and embedding reliable virtual infrastructures," *ACM SIGCOMM Computer Communication Review*, vol. 41, no. 2, pp. 57–64, 2011.

[4] D. Wood and D. Wood, "Theory of computation," 1987.

[5] D. Justice and A. Hero, "A binary linear programming formulation of the graph edit distance," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 28, no. 8, pp. 1200–1214, 2006.

[6] A. Sanfeliu and K.-S. Fu, "A distance measure between attributed relational graphs for pattern recognition," *IEEE transactions on systems, man, and cybernetics*, no. 3, pp. 353–362, 1983.

[7] M. Armbrust, A. Fox, R. Griffith, A. D. Joseph, R. H. Katz, A. Konwinski, G. Lee, D. A. Patterson, A. Rabkin, I. Stoica, *et al.*, "Above the clouds: A berkeley view of cloud computing," tech. rep., Technical Report UCB/EECS-2009-28, EECS Department, University of California, Berkeley, 2009.

[8] H. Yu, C. Qiao, V. Anand, X. Liu, H. Di, and G. Sun, "Survivable virtual infrastructure mapping in a federated computing and networking system under single regional failures," in *Global Telecommunications Conference (GLOBECOM 2010), 2010 IEEE*, pp. 1–6, IEEE, 2010.

[9] E. W. Zegura, K. L. Calvert, and S. Bhattacharjee, "How to model an internetwork," in *INFOCOM'96. Fifteenth Annual Joint Conference of the IEEE Computer Societies. Networking the Next Generation. Proceedings IEEE*, vol. 2, pp. 594–602, IEEE, 1996.

[10] J. Lu and J. Turner, "Efficient mapping of virtual networks onto a shared substrate," 2006.