LETTER

Node Splitting for Improved Virtual Network Embedding: A Feasibility Study*

Jihun HA[†], **Yongtae PARK**[†], **Byungjo KIM**[†], **Eunah KIM**[††], **Sunhee YANG**[††], ***Nonmembers***, ***and*** **Hyogon KIM**[†a)], ***Member***

SUMMARY When the residual resources on a virtualized substrate network (SN) are insufficient to meet the resource demands from the requested virtual network (VN) at specific locations, we can attempt to accommodate the VN by allocating resources at alternative locations and transparently serve the accesses to the VN by having them internally rerouted to the actually allocated locations. In this letter, we explore the feasibility of nodal resource splitting in such alternative allocation scenarios. We find that in order to facilitate such alternative allocations, we should first define the node-link resource dependencies. Once the dependencies are given, we demonstrate that the splitting can visibly improve the SN utilization and the request rejection rate for VN embedding requests under many network scenarios.

key words: *network virtualization, node resources, resource splitting, embedding performance*

1. Introduction

We can embed multiple virtual networks (VNs) on a virtualized substrate network (SN). However, finding the optimal solution to embed VNs on a given SN is known to be NP-hard [1], and many heuristic algorithms have been proposed [1]–[6]. The embedding problem can be divided into node and link embedding problems. In node embedding, we allocate nodal resources on an SN such as CPU cycles or disk space to VN nodes, whereas in link embedding, we allocate link resources like bandwidth for virtual links. In order to increase the number of VNs embedded on a given SN, we should devise methods to improve the successful embedding rate of the more constrained resource type.

When the residual resources on a virtualized substrate network (SN) are insufficient to meet the resource demands from the requested virtual network (VN) at specific locations, we can attempt to accommodate the VN by allocating resources at alternative locations and transparently serve the accesses to the VN by having them internally rerouted to the actually allocated locations. For a detailed discussion, we consider the system model shown in Fig. 1. The SN serves VN creation requests that arrive at rate λ, and the VNs leave the SN at rate μ after completing their mission. Note that to the SN provider, the desirable operating point for the SN is in the high-utilization regime (i.e., $\lambda \rightarrow \mu$) as over-provisioning the SN infrastructure can be costly. On the other hand, it has to minimize the rejection rate of VN creation requests for better client satisfaction. Consequently, the SN provider must frequently deal with a situation where the resource required for VN creation cannot be met by the residual capacity on the SN at specific locations. This letter focuses on this critical operating regime where *alternative* resource mapping is required.

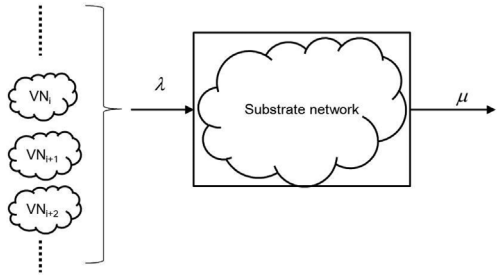


Fig. 1 System model.

There is a substantial body of work on alternative resource mapping [1], [2], [4], [5], [7]. While all these works pertain to link relocation or splitting, the work of Butt et al. considers node relocation [2], and that too without any splitting. The reason is that there are potentially complex issues when node splitting is considered. In case of CPU cycles and disk space, for instance, problems like scheduling can arise when using them in a distributed manner.

In our exploration of the potential value of node splitting, we leave the detailed investigation to future work and attempt to illustrate the most rudimentary aspects. Further, we approach the alternative *node* resource mapping problem as a generalization of the node relocation problem of Butt et al. [2]. In other words, we allow the allocation of not only the entire requested node resource at an alternative SN node, but also fractions at multiple alternative locations.

2. Split Embedding of Virtual Nodes

Suppose that a client requests the SN provider to allocate an amount of $Q_l(R_x)$ of a node resource of type x at location l on the SN. Let us denote the available capacity of type x

Manuscript received January 12, 2012.
Manuscript revised March 14, 2012.
[†]The authors are with the Korea University, Korea.
[††]The authors are with the Electronic Telecommunication Research Institute (ETRI), Korea.
*This research was supported by the Korea Communications Commission (KCC), Korea, under the Future Internet Testbed Construction for International Federation and Collaboration support program supervised by the Korea Communications Agency (KCA) under KCA-2011-10913-05003.
a) E-mail: hyogon@korea.ac.kr
DOI: 10.1587/transcom.E95.B.2463

Copyright © 2012 The Institute of Electronics, Information and Communication Engineers

resource at l by $C_l(R_x)$. If we have $C_l(R_x) \geq Q_l(R_x)$, the allocation succeeds; otherwise, the SN provider could take three approaches. First, it could simply decline the VN embedding request. Second, it could reallocate some of the already allocated VNs to accommodate the incoming VN that requests $Q_l(R_x)$. In this case, the notion of priority has to be defined for the VNs [2], and the derived overhead associated with the suboptimal allocation of the moved VN needs to be considered to determine if such relocation violates the resource requirements of existing VNs. Third, it could accommodate the embedding request at a set of locations K, where $\sum_{k \in K} C_k(R_x) > Q_l(R_x)$. This letter explores the third approach.

When alternative resource mapping occurs, a ripple effect can arise. Namely, the mapping can generate additional resource demand for other resource types $y \neq x$. For example, suppose the client requested $Q_l(R_{cpu})$. If the CPU resource was split into $R^{(1)}_{cpu}$ and $R^{(2)}_{cpu}$ and allocated on two SN nodes l and m, respectively, the computation supposed to be performed on l can be distributed over the two nodes. However, in order to move the input and the output of the computation, we will need additional bandwidth (R_{bw}) on the link between l and m. This derived $Q_{[l \to m]}(R_{bw})$ on path $l \to m$ depends on the type of application that the VN is created for and may affect other VNs that have been allocated bandwidths on the path. Unless we take into account this derived demand, not only the incoming VN but also some of the already allocated VNs may fail to meet their required resource allocation.

In order to avoid this problem, we need to compute the increase in the derived demand in case of an alternative node resource mapping. For the computation, we should know the relation between resource types before we commit to splitting the node resources. For example, the additional bandwidth cost in the above example for each link on the path $l \to m$ can be represented as a product of the amount of relocated node resource and the node-link resource dependency (NLRD), which we denote by $X_{bw \backslash cpu}$. Namely, for each link on the path $l \to m$, we need $R^{(2)}_{cpu} \cdot X_{bw \backslash cpu}$ additionally.

One problem in the above formulation is that it might be difficult for the SN provider to know *X a priori* for various types of VNs. As mentioned above, it is likely a facet of the application running on the VN. Therefore, it is desirable to have the NLRD be specified by the client, i.e., the VN requester. The following discussion is based on this premise, but in case the user does not specify it, the SN provider can set it to infinity, effectively disabling the split node allocation for the requested VN.

Algorithm 1 is a recursive algorithm that embeds the nodes $\{v_1, v_2, \ldots, v_n\}$ and the connecting links in the VN into the SN. It is executed every time a new request for VN embedding arrives at the system (see Fig. 1). We assume that the VN requester's original specification designates the SN node s_k for embedding v_k. We further assume that in addition to embedding at the originally specified location, one can accept the embedding at a "neighbor" node at distance d satisfying $d \geq 0$. We call them the candidate nodes, and *get_candiate_nodes*(s, d) below is the function that gets them.

Algorithm 1 embed_VNET(v_k).

```
 1: if k > n then
 2:     /* successfully allocated for all virtual nodes */
 3:     return 1 /* SUCCESS */
 4: end if
 5: /* sort substrate nodes by residual capacity */
 6: S_c = sort(get_candidate_nodes(s_k, d))
 7: for i = 1 to m do
 8:     /* m = # of elements in S_c */
 9:     if (EVN(v_k, S_c[i])) ∧ (EVL(v_k, S_c[i])) ∧ (embed_VNET(v_{k+1}))
        then
10:         return 1
11:     else
12:         disembed() /* rollback */
13:     end if
14: end for
15: /* need to split virtual node */
16: for i = 1 to m do
17:     S_cs = sort(get_candidate_nodes(S_c[i], d) − {S_c[i]})
18:     for j = 1 to l do
19:         /* l = # of elements in S_cs */
20:         if (EVN2(v_k, S_c[i], S_cs[j])) ∧ (EVL(v_k, S_c[i])) ∧ (EVL2(S_c[i],
            S_cs[j])) ∧ (embed_VNET(v_{k+1})) then
21:             return 1
22:         else
23:             disembed()
24:         end if
25:     end for
26: end for
27: return 0 /* FAILURE */
```

We begin by identifying the candidate SN nodes S_c in which to allocate the virtual node v_1 (i.e., invoke *embed_VNET* with $k = 1$). Then, the candidate nodes within radius d are sorted in decreasing order of their residual CPU resource (*sort*, line 6). For each candidate node $S_c[i]$, we check if its nodal resource can satisfy the demand for v_k (*EVN*, line 9). If so, we proceed to check if the resource demand for the incident virtual links of v_k toward other already embedded nodes v_z, $z < k$, can be satisfied at $S_c[i]$ (*EVL*, line 9). To check if v_k and v_z satisfy $Q_{[v_k \leftrightarrow v_z]}(R_{bw})$, we use the Edmonds-Karp algorithm [5]. If the maximum flow is larger than $Q_{[v_k \leftrightarrow v_z]}(R_{bw})$, *EVL* succeeds. The use of the Edmonds-Karp algorithm is simply for neutrality. If there is a specific routing algorithm adopted by the SN, it should be employed instead in the *EVL* check. When the *EVN* and *EVL* checks are successful, we move on to embed the next virtual node v_{k+1}.

In case the whole node embedding of v_k fails at all candidates, we roll back any partial resource allocation for v_k (line 12). Then we attempt to split the virtual node (lines 15-27). For each candidate node $S_c[i]$ (on which we could not allocate v_k), we obtain its neighbor set S_{cs}. For each element of S_{cs}, we check if we can split the node resource demand between $S_c[i]$ and $S_{cs}[j]$ (*EVN2*, line 21). Although

it can be implemented in a more sophisticated way, the *EVN2*($v_k, S_c[i], S_{cs}[j]$) function is implemented in a greedy way in this letter for simplicity. It first takes all available resource from node $S_c[i]$ and then takes the remaining amount from $S_{cs}[j]$. For example, suppose v_k requires 50 units of nodal resource. If $S_c[i]$ can provide only 40 units, while $S_{cs}[j]$ can provide 30 units, we take all 40 from $S_c[i]$ and the remaining 10 (out of 30) from $S_{cs}[j]$. After this splitting, $S_c[i]$ ends up with 0 units of the resource, and $S_{cs}[j]$ has 20 units. Then as mentioned above we check if the virtual links toward already embedded nodes can be provisioned (*EVL*, line 20). If so, we go on to check if the link resource between nodes $S_c[i]$ and $S_{cs}[j]$ is sufficient to support the derived link resource requirement, which arises because of the splitting (*EVL2*, line 20). Note that unless the NLRD X is given, this check cannot be performed. If all three checks are successful, the split allocation of v_k is complete, and we move on to v_{k+1}. But if even this split allocation fails, we declare the embedding of the given VN to be a failure.

3. Evaluation

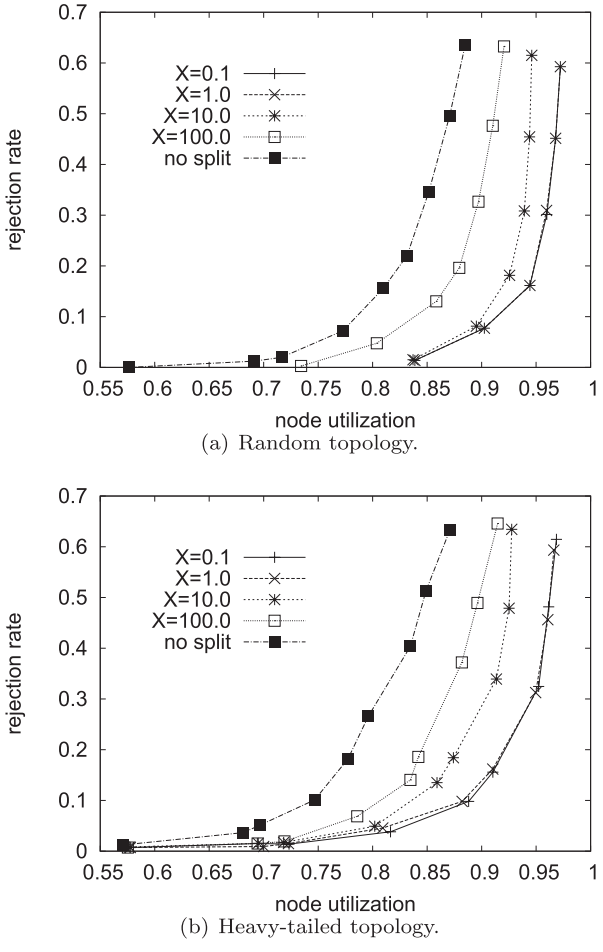
The parameter values used in the evaluation are presented in Table 1. Most of the parameter values are taken from the literature to put the results in perspective. First, the SN consists of 50 nodes, where the connectivity is given by either a heavy-tailed distribution corresponding to the power-law distribution with exponent $\gamma = 3$ [8] or a random distribution [9]. The former essentially reflects Barabási-Albert model where the preferential attachment toward more connected nodes is accounted for [10], whereas the latter does not have such preference. At each node, the available CPU resource is uniformly distributed between 50 and 100, and the available bandwidth on each link is also uniformly distributed between 50 and 100 [3]. The VN requested by the client has a random topology generated by the GT-ITM topology generator [9]. The size of the VN is relatively small, having 2 to 5 nodes where each node requires 10 to 20 units of CPU resources and 10 to 20 units of bandwidth. The inter-arrival time of the VN creation requests is exponentially distributed between 5 to 25 time units, and their lifetimes are also exponentially distributed with a mean of 1,000 time units [3].

Table 1 Experimental parameters.

Parameter	Value
No. of nodes in SN	50
SN connectivity	Power-law distribution, $\gamma = 3$
	GT-ITM random topology
Max. CPU in a SN node	$U[50, 100]$
Max. SN link bandwidth	$U[50, 150]$
No. of nodes in VN	$U[2, 5]$
VN topology	GT-ITM random topology
CPU req. for a VN node	$U[10, 20]$
Bandwidth req. for a VN link	$U[10, 20]$
Request inter-arrival time ($1/\lambda$)	[5, 25], exponential
VN lifetime ($1/\mu$)	1000, exponential
Allocation range (d)	1
CPU-bandwidth NLRD	$0.1 \le X_{bw\backslash cpu} \le 100$

From Little's Law, the average number of VNs in the steady state is given by $N = \lambda T$, so it ranges between 40 and 200 depending on λ. The range of candidacy d is set to 1, so only the directly connected neighbors are considered in addition to the originally specified substrate node when node embedding is attempted. Finally, we vary the NLRD between the CPU and bandwidth resources from 0.1 to 100. Note that as this coefficient increases, so does the overhead of split allocation; thus, it becomes more difficult.

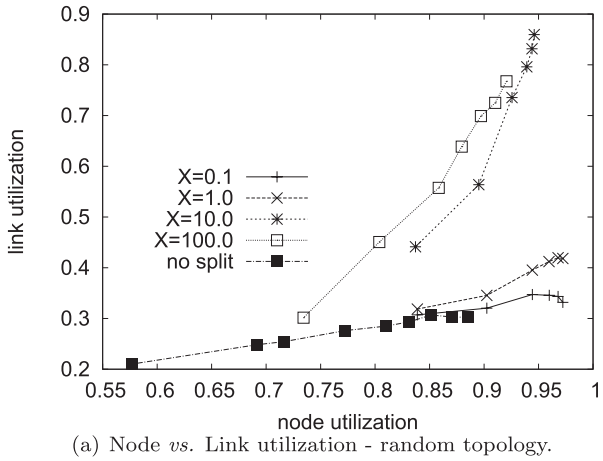
When we operate the SN with the given parameters, we obtain the results shown in Figs. 2 and 3. Figures 2(a) and 2(b) show a comparison of the embedding rejection rate as a function of the node resource utilization between the two SN topologies. The rejection rate is the probability of a client being rejected when it submits a VN creation request with its associated resource requirements, such as those given in Table 1. Node utilization refers to the fraction of the substrate node resource that is allocated to some VN. We can see that without split allocation, the rejection rate becomes visible when the node utilization exceeds 70%, and most requests are rejected well before the utilization reaches 90%. In contrast, split allocation delays the significant rejection events until later. For instance, with $X_{bw\backslash cpu} = 0.1$, more



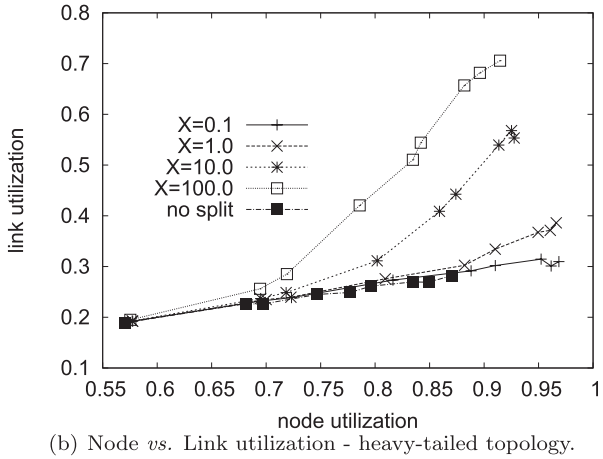
(a) Random topology.

(b) Heavy-tailed topology.

Fig. 2 Embedding rejection rate.



(a) Node *vs.* Link utilization - random topology.



(b) Node *vs.* Link utilization - heavy-tailed topology.

Fig. 3 Additional revenue and link cost.

than half of the requests are still accepted after 95% utilization. As $X_{bw\backslash cpu}$ increases, the rejection rate naturally approaches that of the no split case, for which we can regard $X_{bw\backslash cpu} = \infty$, serving as the bottom line for the node-splitting case. In conclusion, the results show that for a wide range of the NLRD values, the split allocation algorithm allows the SN provider to utilize the SN to a fuller degree, while accommodating client requests with less rejection; therefore, user satisfaction improves.

If Fig. 2 sheds light on the effect of split allocation on user experience, Fig. 3 gives the SN provider's perspectives. It essentially shows how far node utilization can be pushed and at what link cost, before the rejection rate becomes unacceptably high. Reiterating the results of Fig. 2, the figure demonstrates that node splitting increases the node utilization for all values of $X_{bw\backslash cpu}$. Namely, the curves for node splitting cases reach farther to the right. The difference will determine the additional *revenue* [7] for the node-splitting SN. However, the bandwidth toll for the split allocation also rises compared with the no-splitting case. Naturally, the cost incurred on the link increases as the NLRD value increases. For instance, the $X = 100.0$ curve lies on top. However, in case the SN is node-bound as we assume in this letter, that is, in case the node resource shortage is the dominant reason for rejecting client request, the NLRD is of less importance. Utilizing otherwise idling link capacity to accommodate more clients through node splitting should be the only reasonable option for the SN provider in this case.

4. Conclusion

In this letter, we explored the potential value of and the requirements for split node resource allocation. The results are encouraging. Using parameter values based on past studies, we can achieve significantly lower rejection rates for client requests for virtual network creation, and that at higher utilization of the substrate network. However, we find that the dependency relation between different types of resources should be given to help make the decision on accepting a request. The relation hinges on how the virtual network is used, and it can be given by the client who is likely to know better for what purpose the requested virtual network will be used. Our experiment with a wide range of possible dependency values shows that the split node resource allocation has a positive impact on both the user performance and provider performance, and this impact is worth investigating further. A possible direction in a follow-up study is to find ways to determine the inter-resource dependencies for specific uses of virtual networks.

References

[1] N.M.M.K. Chowdhury, M.R. Rahman, and R. Boutaba, "Virtual network embedding with coordinated node and link mapping," Proc. IEEE Infocom, 2009.
[2] N.F. Butt, N.M.M.K. Chowdhury, and R. Boutaba, "Topology-Awareness and reoptimization mechanism for virtual network embedding," Proc. Netw., 2010.
[3] I. Fajjari, N. Aitsaadi, G. Pujolle, and H. Zimmermann, "VNE-AC: Virtual network embedding algorithm based on ant colony metaheuristic," Proc. IEEE ICC, 2011.
[4] J. Lischka and H. Karl, "A virtual network mapping algorithm based on subgraph isomorphism detection," Proc. 1st ACM Workshop on VISA, 2009.
[5] C. Edmonds and R.M. Karp, "Theoretical improvements in algorithmic efficiency for network flow problems," J. Association for Computing Machinery (JACM), vol.19, no.2, pp.248–64, April 1972.
[6] X. Cheng, S. Su, Z. Zhang, H. Wang, F. Yang, Y. Luo, and J. Wang, "Virtual network embedding through topology-aware node ranking," ACM Comput. Commun. Rev., vol.41, no.2, pp.39–47, April 2011.
[7] M. Yu, Y. Yi, J. Rexford, and M. Chiang, "Rethinking virtual network embedding: Substrate support for path splitting and migration," Proc. ACM SIGCOMM, 2008.
[8] A. Medina, A. Lakhina, I. Matta, and J. Byers, "BRITE: An approach to universal topology generation," Proc. International Workshop on Modeling, Analysis and Simulation of Computer and Telecommunications Systems (MASCOTS), 2001.
[9] K. Calvert, M. Doar, and E.W. Zegura, "Modeling Internet topology," IEEE Commun. Mag., vol.35, no.6, pp.160–163, June 1997.
[10] A.L. Barabasi and R. Albert, "Emergence of scaling in random networks," J. American Association for the Advancement of Science, pp.509–512, Oct. 1999.