Adaptive Virtual Network Provisioning

Ines Houidi
Institut Telecom
Telecom SudParis
Evry, France
ines.houidi@
it-sudparis.eu

Wajdi Louati
Institut Telecom
Telecom SudParis
Evry, France
wajdi.louati@
it-sudparis.eu

Djamal Zeghlache
Institut Telecom
Telecom SudParis
Evry, France
djamal.zeghlache@
it-sudparis.eu

Panagiotis Papadimitriou
Computing Dept.
Lancaster University
Lancaster, UK
p.papadimitriou@
lancaster.ac.uk

Laurent Mathy
Computing Dept.
Lancaster University
Lancaster, UK
l.mathy@lancaster.ac.uk

ABSTRACT

In the future, virtual networks will be allocated, maintained and managed much like clouds offering flexibility, extensibility and elasticity with resources acquired for a limited time and even on a lease basis. Adaptive provisioning is required to maintain virtual network topologies, comply with established contracts, expand initial allocations on demand, release resources no longer useful, optimise resource utilisation and respond to anomalies, faults and evolving demands.

In this paper, we elaborate on adaptive virtual resource provisioning to maintain virtual networks, allocated initially on demand, in response to a virtual network creation request. We propose a distributed fault-tolerant embedding algorithm, which relies on substrate node agents to cope with failures and severe performance degradation. This algorithm coupled with dynamic resource binding is integrated and evaluated within a medium-scale experimental infrastructure.

Categories and Subject Descriptors

C.2.1 [**Computer Communication Networks**]: [Network Architecture and Design]

General Terms

Design, Management

Keywords

Network virtualization, Virtual network provisioning, Fault-tolerant embedding

VISA 2010, September 3, 2010, New Delhi, India.

1. INTRODUCTION

Recently, network virtualization has received much attention [12, 5, 7, 20, 21, 13, 9, 11, 4], as it composes a promising way to diversify the Internet and ensure coexistence of heterogeneous network architectures on top of shared substrates. In addition, network virtualization enables the emergence of new actors and business roles to offer on-demand virtual networks (VNs) customized for particular service and user requirements [11, 4, 20, 17, 26]. For example, the so-called VN Providers offer the required level of abstraction and indirection between Service Providers and Physical Infrastructure Providers and are mainly responsible for constructing and delivering VN topologies.

Once allocated, virtual networks will be subject to dynamic variations due to changes in services demands, traffic loads, physical resources and infrastructures, and subject to mobility-induced variations. Hence, VN extensibility and elasticity are critical, especially in dynamically changing network environments. Existing work on network virtualization mainly focuses on initial VN provisioning including resource matching [19, 15], VN embedding [25, 24, 19, 18, 10, 14] and VN binding [7, 21, 13, 9, 20]. According to our knowledge, there are no research studies on adaptive provisioning of instantiated VNs to cope with dynamic changes in service demands and resource availability, especially when these changes enforce dynamic virtual node and link re-assignment.

In this context, we elaborate on adaptive VN provisioning with emphasis on the framework and algorithms in order to preserve VN topologies, established contracts and service level agreements in the event of resource failures or severe performance degradation. We rely on a multi-agent based framework to ensure distributed negotiation, adaptation and synchronization between substrate nodes that provide resources for VNs. This framework is composed of autonomous agents integrated into substrate nodes that carry out a decentralized fault-tolerant VN embedding algorithm to cope with (virtual) node and link failures. We use an implementation which couples the distributed VN embedding algorithm with dynamic VN binding to evaluate the performance and scalability of the adaptive VN provisioning within a medium-scale experimental infrastructure.

The paper is organized as follows. Section 2 presents the initial VN provisioning steps including resource matching, VN embedding and VN binding. Adaptive VN provisioning is introduced in

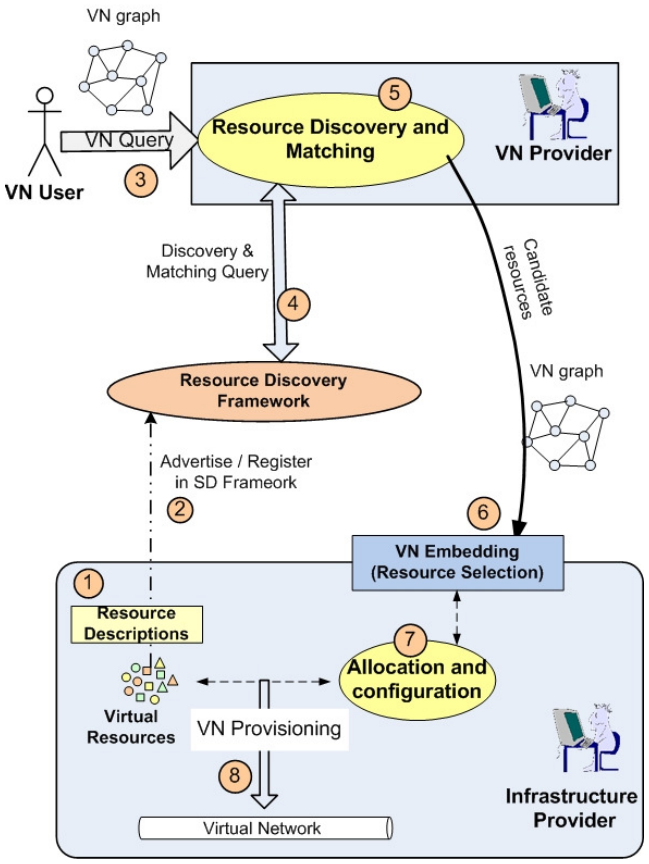


Figure 1: Virtual network provisioning

Section 3. Section 4 describes the multi-agent based adaptive embedding framework that integrates fault-tolerant provisioning and resource optimisation features. Our implementation for adaptive VN provisioning is discussed in Section 5. In Section 6, we provide experimental results on the performance and scalability of the fault-tolerant adaptive embedding. Finally, in Section 7 we highlight our conclusions and discuss directions for future work.

2. VIRTUAL NETWORK PROVISIONING: OVERVIEW

This section provides an overview of an initial provisioning process to set up virtual networks according to user demands. We assume the existence of *VN Providers* which essentially act as brokers that request, negotiate and acquire virtual resources from *Physical Infrastructure Providers* (InPs) on behalf of Service Providers, VN Operators or end-users. Upon receiving VN requests, the VN Providers are responsible for VN provisioning in cooperation with the InPs.

As shown in Fig. 2, VN provisioning includes four steps [15]: (i) resource description and advertisement, (ii) resource discovery and matching, (iii) VN embedding, and (iv) VN binding. Hereby, we briefly discuss these steps, since initial provisioning is extended to achieve adaptive embedding and handle run-time dynamic variations.

2.1 Resource Description and Advertisement

Prior to any VN provisioning operation, InPs describe and advertise physical resources they wish to offer (Fig. 2 - Step 1) to inform VN Providers on available substrate resources properties. In previous work [15], a resource description schema enables InPs to describe and customise their offered substrate resources. Physical resources are described in terms of functional attributes (i.e., static parameters such as node/link type) and non-functional attributes (corresponding to real-time parameters such as currently available capacity/bandwidth). Only functional attributes are advertised by the InP into the resource discovery framework (Step 2). The non-functional attributes are registered into local repositories in substrate nodes, since such dynamic attributes require decentralized real-time monitoring to be kept up-to-date and consequently stay in InP's realm and control. To simplify VN provisioning and enable coordinated matching and embedding, each node description includes key attributes of the links directly connected to the node.

2.2 Resource Discovery and Matching

Upon receiving a VN request from users (Fig. 2 - Step 3), resource discovery and matching consist of searching and finding resource candidates that comply with the requirements specified by the VN request. As depicted in Fig. 2 (Steps 4, 5), the VN Provider uses the discovery framework to search, discover and match the available substrate resources advertised by the InPs with the VN requests. To facilitate the overall discovery and matching process, in [15] we organized and classified the resource descriptions registered in the discovery framework using conceptual clustering. This groups substrate resources with similar concepts, descriptions and properties into clusters. Such clustering generates a hierarchy of clusters, called dendrogram, along with *conceptual descriptions* of each cluster. A similarity based matching algorithm [15] associates the required virtual nodes with the most similar cluster in the dendrogram. As a result, the initial matching step identifies for each required VN node (in the request) candidate resources (in the substrate) that fulfil the virtual node requirements.

2.3 VN Embedding

Once candidate resources have been identified by the matching step, the VN Provider sends to the InP the required VN topology as well as a list of candidate resource IDs related to each required virtual node (Fig. 2 - Step 6). The InP uses this data to find the optimal VN embedding, i.e., assigning the required virtual nodes and links (specified in the VN request) to a specific set of substrate nodes and substrate paths. VN embedding relies on a selection process within each InP. Finding the optimal VN embedding satisfying multiple objectives and constraints is a NP-hard problem that has been addressed in numerous research studies [25, 24, 19, 18, 10, 14]. The primary objective is to allow a maximum number of VNs to co-exist on top of the same substrate, while reducing the cost for users and increasing revenue for providers. In [14], we proposed a decentralized embedding algorithm across the substrate nodes.

2.4 VN Binding

Upon VN embedding, the selected substrate resources are allocated by the InPs in order to instantiate the requested VN (Fig. 2 - Step 7). VN binding essentially consists of virtual node and link setup. The substrate nodes handle incoming virtual node requests within their management domains, triggering the appropriate action (e.g., virtual node creation and configuration). Virtual link setup is dependant on the link virtualization technology and typically packet encapsulation/decapsulation is required. Eventually, the InP aggregates the allocated substrate resources to instantiate the VN (Fig. 2 - Step 8) and to provision the VN service to the

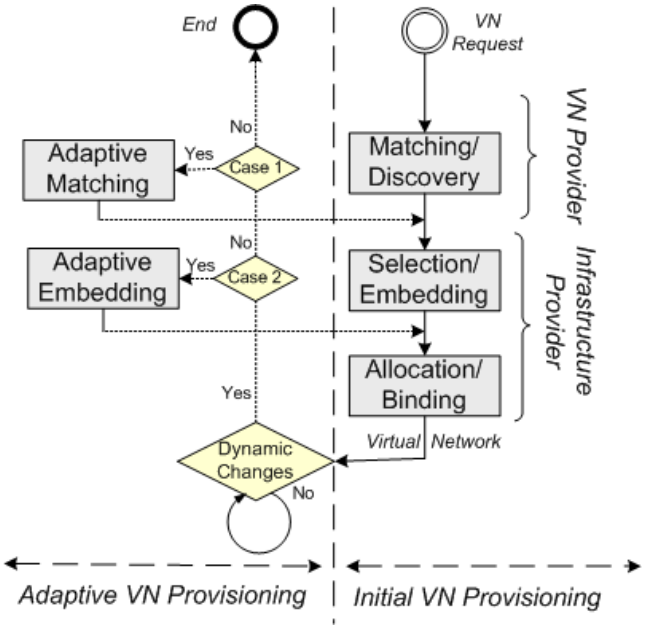


Figure 2: Adaptive virtual network embedding

user. The deployment of new architectures, services and applications in the offered VNs is achieved by the VN Operators and Service Providers. Further details on VN instantiation can be found in our previous work [20].

3. ADAPTIVE VIRTUAL NETWORK PROVISIONING

Once the VN has been provisioned and set up, adaptive provisioning and maintenance of the VN comes into play. Dynamic changes are induced by variations in the substrates and VNs and are related to resource failures, mobility, migration and maintenance needs. Dynamic and adaptive provisioning should handle changes in the topologies or resource restrictions of previously instantiated VNs and simultaneously cope with resource failures or performance degradation. The main goal of adaptive provisioning is to maintain VN topologies, established contracts and service level agreements.

The right side of Fig. 2 illustrates a diagram summarizing the initial provisioning steps, while the left side depicts adaptive provisioning which has to handle two cases of dynamic changes that call for adaptive matching and embedding:

- **Case 1**: It occurs when the VN user defines or expresses new requirements such as VN extensions (expansion of coverage) and new service requirements. These new demands may result in adding, removing and updating attributes/values to components of a previous VN request. New InPs may have to be involved and new additional resources provided to meet the new demands. The set of candidate resources defined in the initial matching should be updated to reflect the changes in virtual resources attributes/values and the need for new resources. An *adaptive matching* algorithm is required to identify new candidate resources with respect to the new request requirements. The adaptive matching algorithm starts by searching for virtual resources in the vicinity or neighbourhood of previous candidate resources (identified during the initial matching) instead of returning to the top level or root of the dendrogram to find a solution. The potential candidates are necessarily children, parents or siblings of prior matching. More precise and richer descriptions, with more attributes, move the matching deeper into the dendrogram. If the description is more abstract and high level, with fewer details and attributes, the matching moves up in the dendrogram. The adaptive matching algorithm is not discussed in this paper.

- **Case 2**: It corresponds to scenarios where virtual or physical resources allocated to VNs fail or suffer from anomalies (e.g., substrate or virtual node failure or performance degradation). The InP maintains the VN topologies (affected by the failures) by selecting new virtual or substrate resources to replace or compensate for the affected resources. Three resource failure scenarios are discussed in this paper: virtual node, substrate node and link failures. In this paper, we present and implement an *adaptive embedding* algorithm to deal with these scenarios.

Adaptive provisioning is concluded with dynamically binding and allocating new resources to maintain the VN. This procedure includes: (i) virtual node migration to the new host or new virtual node setup and configuration depending on the type of failure, and (ii) virtual link reassignment and setup to preserve the requested VN topology.

4. ADAPTIVE VIRTUAL NETWORK EMBEDDING

In this section, we elaborate on the fault-tolerant VN embedding algorithm which is the most critical component of the adaptive VN embedding procedure. The VN embedding algorithm is specifically designed to maintain VN topologies in the event of resource failures.

4.1 Multi-Agent Based Adaptive Embedding Framework

First, we discuss the design of an adaptive VN embedding framework to deal with dynamic changes requiring automatic and runtime reparation. The framework is responsible for:

1. Detecting and identifying local changes through monitoring (e.g., node/link failure, performance degradation).
2. Selecting new substrate resources to maintain VN topologies.
3. Instantiating a virtual node in the new selected substrate node or migrating virtual nodes from a substrate node to another.
4. Binding the virtual node along with the virtual links that are affected.

The framework relies on the multi-agent based approach to ensure distributed negotiation and synchronization between the substrate nodes [1]. As depicted in Fig. 3, the adaptive embedding framework is composed of autonomous agents which are integrated in substrate nodes. These substrate agents communicate, collaborate and interact with others to plan collective reselection of resources for adaptive VN embedding. The agents monitor, supervise and extract the dynamic (i.e., non-functional) attributes from the local repositories and decide locally which reselection actions to undertake.

The adaptive embedding framework relies on situation-awareness approaches. In fact, the conceptual clustering technique used during the matching phase can provide a generic model for a situated

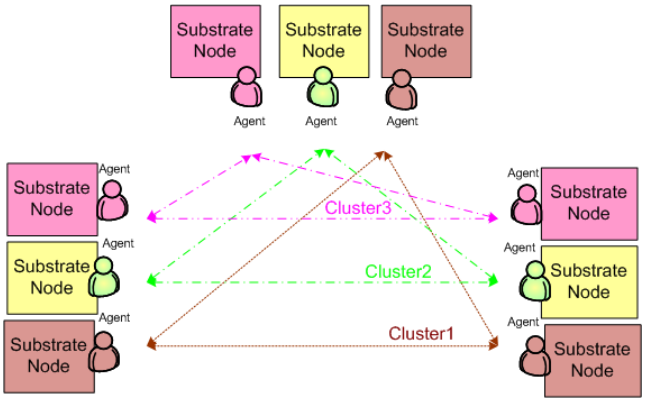


Figure 3: Multi-agent based adaptive embedding framework

multi-agent based infrastructure. The situated view of agents is determined based on similarities between substrate nodes. These similarities have been previously determined by the conceptual clustering algorithms, as introduced in Section 2.2. Only agents within the same cluster can negotiate and cooperate. The situation awareness signifies that each agent is aware of what is happening around it, i.e., to which cluster it belongs to and with which agents it should negotiate and synchronize to handle the distributed adaptive embedding decision making.

4.2 Distributed Fault-Tolerant Embedding Algorithm

We hereby propose a distributed fault-tolerant embedding algorithm to maintain VN topologies by reselecting new resources to replace the no-longer available ones. There is no need to consult a central entity upon failures, since distributed localized control can react quickly to local changes. The algorithm assumes the existence of supervision, monitoring and fault-diagnosis mechanisms to detect resource failures.

The multi-agent based framework introduced above is used to carry out the distributed fault-tolerant embedding algorithm. The cooperation between substrate agents relies on the matching results. Indeed, only substrate agents belonging to the same cluster (i.e., the set of candidate substrate nodes that match a given virtual node n_v) are responsible for executing the distributed fault-tolerant embedding algorithm. This algorithm can handle the following resource failure scenarios:

- **Virtual node failure**: When a substrate agent detects that a supported virtual node n_v has failed, it must either re-instantiate a new virtual node in the same substrate node or, if not possible, request other substrate nodes (i.e., the candidate resources matching the node n_v) to allocate resources for the virtual node. The virtual links associated with the affected virtual node should be also reallocated. The distributed adaptive VN embedding (Algorithm 1) is executed to repair the virtual node failure.
- **Substrate node failure**: When a substrate node n_s hosting multiple virtual nodes has failed, the agents that belong to the same cluster can detect this failure through keep-alive messages exchanged periodically. Only substrate node agents that belong to the same cluster are allowed to collaborate in order to choose alternative hosts where the affected virtual nodes as well as their associated links will be migrated [22] or allocated. Thus, the distributed adaptive VN embedding algorithm is executed for each virtual node hosted to the substrate node where failure was detected.
- **Link failure**: Upon link failure, the agent substrate nodes directly connected to this link collaborate to select an alternative link or path. Similarly to link failures, the fault-tolerant embedding algorithm can react to conditions of congestion or overload in the substrate links by monitoring bandwidth in the substrate nodes. In both cases, only *Step 6* of Algorithm 1 is executed to reassign the affected virtual links.

Fig. 4 illustrates a step-by-step scenario describing the distributed adaptive VN embedding in the event of virtual node failure. Each substrate node agent runs Algorithm 1. Upon detecting a virtual node failure, the substrate node agent hosting this virtual node sends a failure notification message ($ErrorMSG(n_v)$) to all substrate agents in the same cluster to notify them that the node n_v is no longer available (Step 1). Next, each agent should check its ability to support the request node on behalf of the failed one. To achieve that, each agent extracts the non-functional (NF) attributes of the affected VN request node n_v as well as the NF attributes of its own resources. The NF attributes are presented in the form of attribute-value pairs: *(att, x)*.

The NF attributes of an arbitrary request node n_v are represented as follows:

$$n_v = ((att_1, x_{v1}), (att_2, x_{v2}), ..., (att_f, x_{vf}), ..., (att_p, x_{vp})).$$

The NF attributes of the substrate node (agent host) n_s are expressed as:

$$n_s = ((att_1, x_{s1}), (att_2, x_{s2}), ..., (att_f, x_{sf}), ..., (att_p, x_{sp})).$$

The objective is to check how much the associated substrate node capabilities can respond to the virtual node n_v requirements. The substrate node n_s that should be selected should be as similar as possible to the virtual node n_v. To this end, each agent computes a dissimilarity metric *dism* between the NF attributes of request node n_v and the NF attributes of its associated substrate node (Step 2). The dissimilarity function expression is the following:

Dissimilarity function expression: The NF attributes may be of different types including binary, nominal and interval types. A node description may also include a mixture of the NF attributes types. To consider and combine all possible NF attributes types (binary, nominal or interval), the *dism* metric is computed as follows [23]:

$$dism(i,j) = \frac{\sum_{f=1}^{p} \delta_{ij}^{(f)} d_{ij}^{(f)}}{\sum_{f=1}^{p} \delta_{ij}^{(f)}} \quad (1)$$

where:

- p is the number of NF attributes of the request node i.
- Let $d_{ij}^{(f)}$ denote the dissimilarity of nodes i and j related to att_f. $d_{ij}^{(f)}$ is determined according to the att_f type:
  - if the type of att_f is binary or nominal:

$$d_{ij}^{(f)} = \begin{cases} 0 & if \ \ x_{if} = x_{jf} \\ 1 & if \ \ x_{if} \neq x_{jf} \end{cases}$$

  - if the type of att_f is interval, a normalized distance such as Euclidean distance and Manhattan distance can be used.

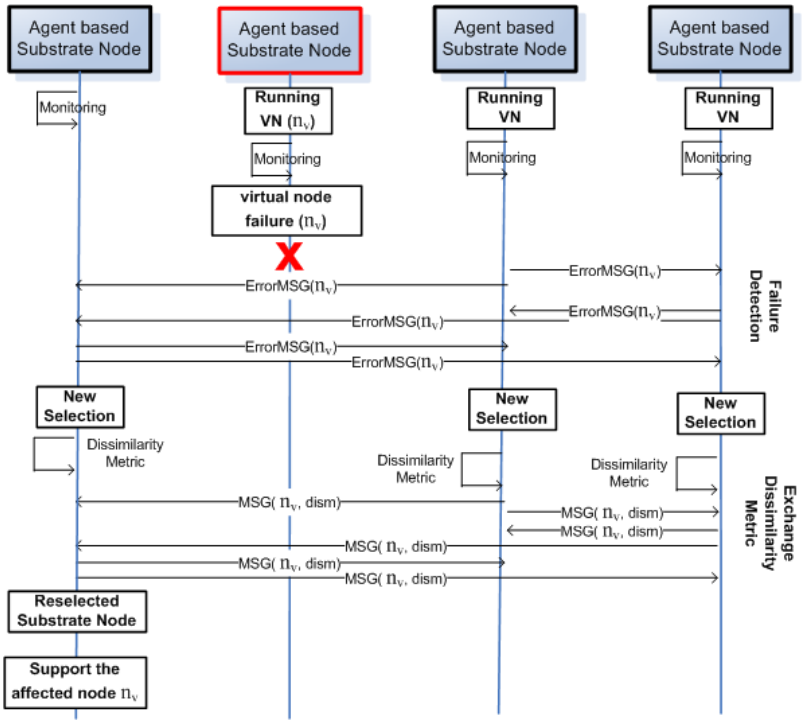


Figure 4: Message exchanges among agents for fault-tolerant embedding

- $\delta_{ij}^{(f)}$ is used to express the priority coefficients of the NF attribute att_f for nodes i and j. Consequently, multi-level priority is assigned to the NF attributes of each virtual node.

Algorithm 1 : Distributed adaptive VN embedding algorithm running by each agent based substrate node n_s

1) Send a notification message $(ErrorMSG(n_v))$ to all agents in the same cluster
2) Compute a dissimilarity metric between the NF attributes of the affected request node n_v and the NF attributes of the substrate node n_s
3) Exchange, via $(MSG(n_v, dism))$ messages, the computed dissimilarity metrics *dism* within the same cluster. The agent compares its dissimilarity metric of the request node n_v with all substrate nodes
4) **If** the dissimilarity metric of the request node n_v is the minimal one compared to the other computed dissimilarity metrics
5) **then** the substrate node hosting the agent will support the request node n_v.
6) Map the associated virtual links to the substrate paths between substrate nodes

Once the dissimilarity metrics *dism* are computed for the affected virtual node n_v, the agent exchanges these metrics via $(MSG(n_v, dism))$ message with all the agents belonging to the same cluster (Step 3). Each agent compares its dissimilarity metrics with those computed by the other agents. For the virtual node n_v, the (substrate node n_s, virtual node n_v) pair that has the minimum dissimilarity metric is selected for the adaptive embedding decision (Steps 4 and 5).

Once the virtual node n_v is assigned to the new substrate node, its associated virtual links are mapped to the substrate paths (Step 6) using a distributed shortest-path algorithm (see [14]).

5. IMPLEMENTATION

In this section, we provide an overview of our implementation for adaptive VN provisioning which utilizes the proposed fault-tolerant embedding algorithm.

5.1 Infrastructure and Software

We have implemented the adaptive VN embedding, as discussed in this paper, on the *Heterogeneous Experimental Network* [2], which comprises more than 110 computers interconnected by a single non-blocking, constant-latency Gigabit Ethernet switch. We mainly use Dell PowerEdge 2950 systems with two Intel quad-core CPUs, 8GB of DDR2 667MHz memory and 8 or 12 Gigabit ports. Our implementation synthesizes existing node and link virtualization technologies for adaptive VN provisioning. In particular, we use Xen 3.2.1 [6], Linux 2.6.19.2 and the Click modular router package 1.6 [16] (with patches eliminating SMP-based locking issues). We rely on XML to represent (virtual) network topologies and resource specifications within VN requests.

5.2 Functionality

A fixed number of HEN nodes compose the substrate, while we use a dedicated node for the VN Provider which receives VN requests and initiates VN provisioning, as exemplified in Section 2. Once a VN has been instantiated, we utilize the fault-tolerant VN embedding algorithm via autonomous agents that are deployed in the substrate nodes. Each substrate node exposes control interfaces allowing for remote procedure calls based on XML-RPC. Agents exchange messages depending on the cluster classification and are responsible for detecting (virtual) node failure and performance

degradation. Upon the detection of such an event, host discovery, virtual node migration or instantiation and virtual link setup are directly invoked. Within a substrate node, each virtual node request is handled by a separate thread, speeding up VN binding. For virtual link assignment, we modified and implemented a distributed version of the Bellman-Ford algorithm [8], which is known as the fastest distributed algorithm for solving shortest-path problems for generic network topologies.

To satisfy the node and link constraints of incoming VN requests, the substrate agents need updated resource information. To this end, the substrate nodes monitor CPU load and link bandwidth of adjacent links[1]. Each node maintains a local repository with its functional and non-functional attributes.

For the inter-connection of the virtual nodes, we currently use tunnels with IP-in-IP encapsulation. Each virtual node uses its virtual interface to transmit packets, which are captured by Click for encapsulation, before being injected to the tunnel. On the receiving host, Click demultiplexes the incoming packets delivering them to the appropriate virtual node. For packet forwarding, we use Click SMP with a polling driver. In all cases, Click runs in kernel space.

The substrate topologies are constructed off-line by configuring Virtual Local Area Networks (VLAN) in the HEN switch. This process is automated via a *switch-daemon* which receives VLAN requests and configures the switch accordingly.

To allow separation between multiple VNs, we use a globally unique identifier for VNs, namely *vnetID*. We also use the identifier *vmID* for the virtual nodes. The scope of *vmID* is restricted to a specific VN.

6. EVALUATION

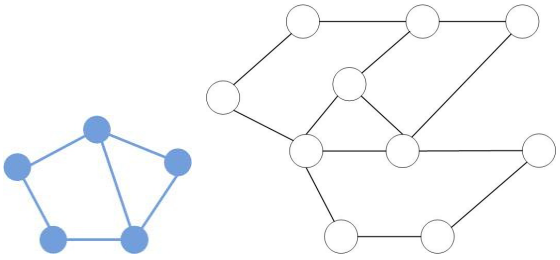
In this section, we provide experimental results to evaluate the efficiency of the distributed fault-tolerant embedding algorithm. According to our knowledge, there is no related algorithm in the literature; hence, we use a centralized adaptive embedding algorithm as a baseline for our performance study. The centralized algorithm combines local fault detection with centralized recovery. First, each substrate node detects and reports (virtual) node and link failures to a management entity responsible for the recovery decision. Subsequently, the management node exchanges messages with all participating substrate nodes for host discovery and virtual link reassignment. More precisely, the management entity sends the NF attributes of the failed virtual node to all substrate nodes which compute the dissimilarity metric and further communicate it to the centralized coordinator for the decision making. The subsequent steps (i.e., virtual node/link binding) are coordinated by the management node in cooperation with the involving substrate nodes.

First, we assess the performance of the fault-tolerant VN embedding and adaptive binding within HEN using our implementation. Next, we evaluate the scalability properties of the algorithm in GRID5000 experimental infrastructure [3].

6.1 Performance Evaluation

A first experiment has been set up to measure the delay incurred to adapt the VN when a virtual node performs poorly or fails. To this end, we initially provision the VN of Fig. 5(a) on top of a substrate topology which is composed of 10 nodes (Fig. 5(b)). Adaptive VN provisioning includes the following steps: (i) host discovery for the affected virtual node using the fault-tolerant algorithm, (ii) virtual node binding via migration or reinstantiation in the new host, and (iii) virtual link binding based on the distributed shortest-path algorithm.

[1]In our implementation, node and link capacities are the only non-functional parameters considered.



(a) VN topology (b) Substrate topology

Figure 5: VN and substrate topology in HEN for performance evaluation

Fig. 6 shows the corresponding results with a varying number of clusters. The time required to adapt the VN with the distributed embedding algorithm is always less than 2 sec and is decreased further in the presence of multiple clusters, as host discovery is restricted among fewer nodes. The improved performance with more clusters is also confirmed by Fig. 7, which depicts the number of messages exchanged during dynamic VN adaptation. This number includes the messages exchanged both for fault-tolerant embedding and VN binding.

In addition, Fig. 6 demonstrates the benefits of the distributed over centralized VN embedding. Even with a single cluster, the distributed algorithm needs less time to adapt the VN, since the required messages are exchanged directly between the nodes, without the need for centralized coordination. Efficient clustering speeds up host discovery and eventually enables dynamic VN adaptation at very short timescales.

We further measure the delay to adapt VNs in the event of a substrate node failure. In this case, a substrate node may host multiple virtual nodes which should be assigned to other hosts as determined by the fault-tolerant embedding algorithm. In this experimental scenario, we initially provision 3 VNs with the topology of Fig. 5(a) on top of the same substrate (Fig. 5(b)), which results in substrate nodes hosting up to 3 virtual nodes. The delay incurred during adaptation of the 3 VNs is illustrated in Fig. 8, uncovering the efficiency of the proposed embedding algorithm. In the presence of multiple clusters, VNs adaptation is concluded within less than 3 sec.

6.2 Scalability Evaluation

Hereby, we evaluate the scalability of the distributed fault-tolerant embedding algorithm in large-scale networks such as GRID 5000 [3]. The experimental facility GRID 5000 has been used to generate full mesh substrate topologies with different sizes (from 0 up to 100 nodes). Autonomous agents are deployed in the GRID 5000 machines to emulate the substrate agents to carry out the distributed fault-tolerant embedding algorithm. Our goal is to evaluate the delay and the number of messages required by the algorithm for dynamic host discovery in the event of virtual node failure. The binding step (i.e., virtual node and link setup) is not considered in this evaluation; we only measure the delay required for the distributed decision making.

Fig. 9 depicts the delay incurred for adaptive VN embedding with full-mesh substrate topologies. We present experimental results from a substrate without clustering (corresponding to the curve with one cluster in Fig. 9) versus a substrate with 2, 5 and 10 clus-

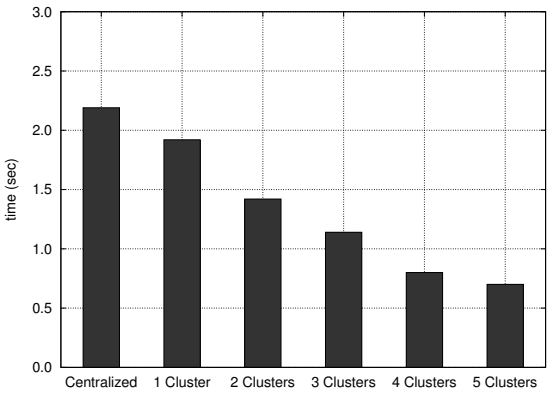


Figure 6: Average time required to adapt a VN in the event of virtual node failure

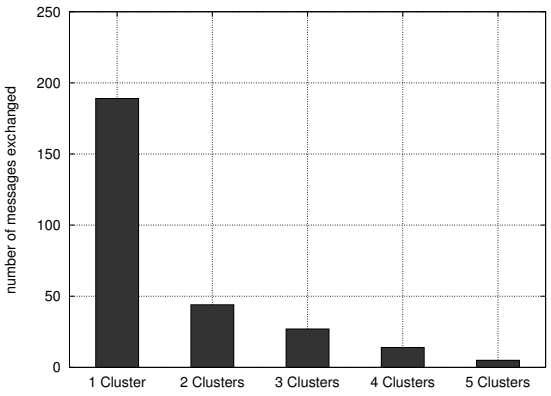


Figure 7: Number of messages exchanged to adapt a VN in the event of virtual node failure

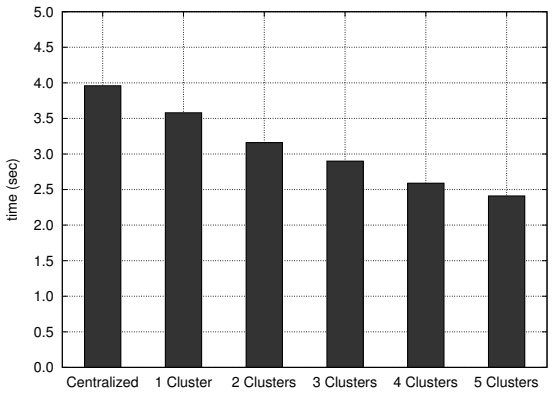


Figure 8: Average time required to adapt VNs in the event of substrate node failure

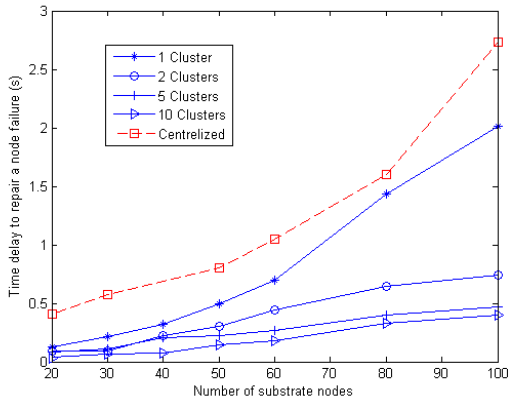


Figure 9: Average time required for adaptive VN embedding in the event of virtual node failure

ters. As shown in Fig. 9, the delay required to adapt a VN without clustering is in the order of 2 sec for as many as 100 substrate nodes. This delay does not exceed 0.75 sec when using clustering in substrate networks. The number of messages exchanged between substrate nodes decreases with clustering, as message exchange occurs only within the cluster that the failure has been detected.

Fig. 10 corroborates the delay results shown in the Fig. 9. A comparison with the centralized algorithm shows that the time required by our distributed fault-tolerant embedding algorithm to adapt a VN upon failures is much lower than the delay incurred with the centralized approach (the upper curve in Fig. 9). This is due to the increased number of messages exchanged between a centralized coordinator and substrate nodes.

7. CONCLUSION AND FUTURE WORK

In this paper, we presented the design, implementation and experimental evaluation of an adaptive fault-tolerant VN embedding algorithm to deal with dynamically changing network environments. The adaptive VN embedding algorithm relies on a multi-agent based framework to effectively repair resource failures and maintain VN topologies. The proposed algorithm has been implemented and evaluated within a medium-scale experimental infrastructure, where we are able to dynamically bind and allocate resources. Our experimental results show that our adaptive embedding algorithm and dynamic binding can react quickly and efficiently to resource failures.

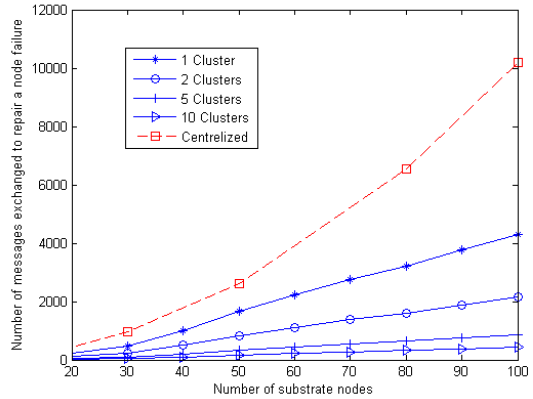


Figure 10: Number of messages exchanged during adaptive VN embedding in the event of virtual node failure

Future work will focus on adaptive matching in order to dynamically identify new candidate resources with respect to new requirements from VN users or Service Providers, which result in modifications to previous VN requests such as VN extensions/contractions.

Acknowledgments

Part of this work was performed within the 4WARD project, which is funded by the European Union in the 7th Framework Programme (FP7). We would like to thank our colleagues in the project for many fruitful discussions.

8. REFERENCES

[1] Foundation for intelligent physical agents, `http://www.fipa.org/`.

[2] Heterogeneous Experimental Network, `http://hen.cs.ucl.ac.uk`.

[3] GRID 5000, `https://www.grid5000.fr/`.

[4] 4WARD Project, http://www.4ward-project.eu.

[5] T. Anderson, L. Peterson, S. Shenker, and J. Turner. Overcoming the internet impasse through virtualization. In *Proceedings of ACM HOTNETS*, San Diego, CA, USA, 2004.

[6] P. Barham, B. Dragovic, K. Fraser, S. Hand, T. Harris, A. Ho, R.Neugebauer, I.Pratt, and A. Warfield. Xen and the art of virtualization. In *Proceedings of 19th ACM Symposium on Operating Systems Principles*. ACM Press, October 2003.

[7] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford. In VINI Veritas: Realistic and Controlled Network Experimentation. In *Proceedings of SIGCOMM '06*, pages 3–14. ACM, 2006.

[8] R. Bellman. Dynamic programming. *Princeton, N.J. Press*, 1957.

[9] S. Bhatia, M. Motiwala, W. Mühlbauer, Y. Mundada, V. Valancius, A. Bavier, N. Feamster, L. Peterson, and J. Rexford. Trellis: A platform for building flexible, fast virtual networks on commodity hardware. In *Proceedings of ACM CoNEXT ROADS Workshop*, 2008.

[10] N. Chowdhury, M. R. Rahman, and R. Boutaba. Virtual network embedding with coordinated node and link mapping. In *Proceedings of IEEE INFOCOM*, 2009.

[11] N. Feamster, L. Gao, and J. Rexford. How to Lease the Internet in Your Spare Time. *SIGCOMM CCR*, 37(1):61–64, 2007.

[12] GENI: Global Environment for Network Innovations. http://www.geni.net.

[13] J. He, R. Zhang-Shen, Y. Li, C.-Y. Lee, J. Rexford, and M. Chiang. DaVinci: Dynamically Adaptive Virtual Networks for a Customized Internet. In *Proceedings of ACM CoNEXT '08*. ACM, 2008.

[14] I. Houidi, W. Louati, and D. Zeghlache. A Distributed Virtual Network Mapping Algorithm. In *Proceedings of IEEE International Conference on Communications*, pages 273–286, Beijing, China, May 2008.

[15] I. Houidi, W. Louati, D. Zeghlache, and S. Baucke. Virtual Resource Description and Clustering for Virtual Network Discovery. In *Proceedings of IEEE ICC Workshop on the Network of the Future 2009*, Dresden, June 2009.

[16] E. Kohler, R. Morris, B. Chen, J. Jahnotti, and M. F. Kasshoek. The click modular router. *ACM Transaction on Computer Systems*, 18(3):263–297, 2000.

[17] W. Louati. On demand Virtual Network Service for Dynamic Networks. 2007.

[18] J. Lu and J. Turner. Efficient mapping of virtual networks onto a shared substrate. *TR.WUCSE-2006-35*, Washington University, 2006.

[19] R. Ricci, C. Alfeld, and J. Lepreau. A solver for the network testbed mapping problem. *ACM Computer Communication Review*, January 2003.

[20] G. Schaffrath, C. Werle, P. Papadimitriou, A. Feldmann, R. Bless, A. Greenhalgh, A. Wundsam, M. Kind, O. Maennel, and L. Mathy. Network virtualization architecture: Proposal and initial prototype. In *Proceedings of ACM SIGCOMM VISA*, Barcelona, Spain, August 2009.

[21] J. D. Touch. Dynamic Internet Overlay Deployment and Management Using the X-Bone. In *Proceedings of IEEE ICNP '00*. IEEE, 2000.

[22] Y. Wang, E. Keller, B. Biskeborn, J. van der Merwe, and J. Rexford. Virtual Routers on the Move: Live Router Migration as a Network-Management Primitive. *ACM SIGCOMM CCR*, 38(4):231–242, 2008.

[23] L. Xiao-hong, X. Yang, Q. Ke-yun, and P. Zheng. A clustering application method based on mix type variables in social system appraisement. In *Proceedings of IEEE International Conference on Man and Cybernetics Systems*, 3:2857–2862, 2003.

[24] M. Yu, Y. Yi, J. Rexford, and M. Chiang. Rethinking virtual network embedding: Substrate support for path splitting and migration. *ACM SIGCOMM Computer Communication Review*, April 2008.

[25] Y. Zhu and M. Ammar. Algorithms for assigning substrate network resources to virtual network components. In *Proceedings of IEEE INFOCOM*, 2006.

[26] Y. Zhu, R. Zhang-Shen, S. Rangarajan, and J. Rexford. Cabernet: Connectivity Architecture for Better Network Services. In *Proceedings of ReArch '08*. ACM, 2008.