This full text paper was peer reviewed at the direction of IEEE Communications Society subject matter experts for publication in the ICC 2008 proceedings.

A Distributed Virtual Network Mapping Algorithm

Ines HOUIDI, Wajdi LOUATI, Djamal ZEGHLACHE
Institut TELECOM
TELECOM & Management SudParis
9 rue Charles Fourier 91011 EVRY Cedex, France
{ines.houidi, wajdi.louati, djamal.zeghlache}@int-edu.eu

***Abstract*—Network Virtualization is a promising concept to diversify the Future Internet architecture into separate Virtual Networks (VN) that can support simultaneously multiple network experiments, services and architectures over a shared substrate network. To take full advantage of this paradigm this paper addresses the challenge of assigning VNs to the underlying physical network in a distributed and efficient manner. A distributed algorithm responsible for load balancing and mapping virtual nodes and links to substrate nodes and links has been designed, implemented and evaluated. A VN Mapping Protocol is proposed to communicate and exchange messages between agent-based substrate nodes to achieve the mapping. Results of the implementation and a performance evaluation of the distributed VN mapping algorithm using a Multi-agent approach are reported.**

I. Introduction

Network Virtualization has recently received considerable attention in both academia and industry as an important enabler for designing the Future Internet architecture [1], [2]. The Network Virtualization concept has been proposed as a potential solution for diversifying the Future Internet architecture into separate Virtual Networks (VN). The VNs can support simultaneous network experiments, services and architectures over a shared substrate network [3], [4], [5]. A VN is a group of *Virtual Nodes* (e.g. virtual routers) interconnected via dedicated *Virtual Links* over a substrate network. As depicted in the figure 1, multiple virtual networks may share the same underlying physical network. The set of virtual nodes and virtual links forming the VN should be assigned to a specific set of *Substrate Nodes* and *Substrate Paths*, respectively. A Substrate Path is a logical path between two substrate nodes which may be a single substrate link or a sequence of substrate links.

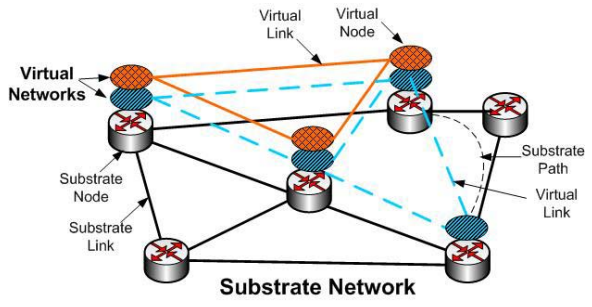

Fig. 1. Mapping of Virtual Networks to a shared substrate network

The VN set up is achieved by selecting the best and optimal network topology from the physical substrate comprising both virtual nodes and links. Mapping multiple VNs into a shared physical infrastructure represents a significant challenge that has been addressed in many research studies. A number of proposals and algorithms have been put forward to address the VN assignment/mapping problem [6], [7], [8], [9]. The objective is to make efficient use of the substrate network by providing effective methods and algorithms for mapping a VN to the substrate network.

In these proposals, the VN mapping algorithms are carried out in a centralized manner. A central entity is responsible for receiving VN requests and for assigning a set of virtual nodes to a set of substrate nodes. This entity should maintain up to date information about the substrate network to make the appropriate VN mapping decisions. However, maintaining up to date information about the substrate network in a centralized way suffers from scalability limitation, high latency and serious delays in making decisions especially when the underlying physical network is a highly dynamic and changing environment (e.g. node joining/leaving). This is exactly the motivation of this paper whose goal is to provide the design, implementation and evaluation of a *distributed* algorithm for mapping VNs to substrate network resources. The objective is to enable and guarantee a balanced load-sharing among all substrate nodes and links during the VN mapping.

The design and implementation of the proposed VN mapping algorithm relies on the Multi-Agent based approach [10] to ensure distributed negotiation and synchronization between the substrate nodes. These nodes handle autonomous and intelligent agents which exchange messages and cooperate to carry out the distributed VN mapping algorithms. A VN Mapping Protocol is proposed to enable communications between the agent-based substrate nodes in a distributed manner. Performance and scalability results of the distributed VN mapping algorithm developed using Multi-agent based approach are reported.

The remainder of this paper is organized as follows. Section II of this paper summarizes related work. Section III presents the network model and the VN mapping problem. The design of a distributed VN mapping algorithm is addressed in Section IV. The implementation and evaluation of the proposed algorithm based on multi-agent based approach is reported in Section V.

II. Related Work and Motivation

A number of heuristic greedy algorithms [6], [9], [11], customized algorithms [9] and iterative mapping processes [8] have been put forward to efficiently assign VN to substrate resources. In [6] and [9], authors proposed heuristic greedy algorithms to maintain low and balanced stress among all substrate nodes and links during the VN assignment process. The aim of such greedy algorithms is to assign virtual nodes to substrate nodes with maximum available resources. The overall objective is to achieve near optimal VN mapping solutions. However, the proposed VN mapping algorithms are carried out in a centralized manner. A central coordinator is required to maintain global substrate information. Using this information, the centralized VN mapping algorithm may avoid conflicts and inconsistencies during resource allocation and can easily obtain the globally optimal solution. However, maintaining a global view and knowledge about the substrate network requires efficient message-passing mechanisms which can impose a large overhead on the central entity. The centralized mapping approach presents limitations in resilience and scalability when the number of substarte/virtual nodes and VN requests increases. This is especially challenging for highly dynamic substrate networks that can change not only in state but also in composition (node failures, node joining/leaving...). The central coordinator has to process a significant number of signalling messages which may require complex algorithms for synchronization with substrate state. This can increase complexity and induce costly and time-consuming mapping.

The objective of our work is to decentralize the VN mapping algorithm among the substrate nodes. These nodes should be able to decide which mapping actions to undertake. The main advantages of the decentralized VN mapping approach are the ability to: (1) reduce communication costs and the number of messages exchanged with the central coordinator, (2) ensure robustness since no central entity is used avoiding single point of failure, (3) achieve high-speed parallel processing of multiple VN requests, and (4) deal with partial failures that require automatic and runtime reparation. When a node/link failure occurs, there is no need to consult a central entity since localized control can react quickly to local failures.

III. VN Mapping Model and Problem formulation

A. Substrate Network Model

The substrate network can be represented by a weighted undirected graph $G_s = (N_s, L_s)$, where N_s is the set of substrate nodes and L_s is the set of substrate links between nodes of the set N_s. Each substrate node $n_s \in N_s$ is associated with the capacity weight value $C(n_s)$ which denotes the *available* capacity of the physical node n_s. Each substrate link $l_s(i,j) \in L_s$ between two substrate nodes *i* and *j* is associated with two non-negative values: 1) an additive weight value $w(l_s(i,j))$ which denotes a set of parameters including the cost and delay of the substrate link and 2) a capacity weight value $C(l_s(i,j))$ which denotes the *available* bandwidth capacity associated to the substrate link.

Let ψ be a set of substrate paths in the substrate network G_s. The available bandwidth capacity $C(P)$ associated to a substrate path $P \in \psi$ between two substrate nodes can be evaluated as the minimal residual bandwidth of the links along the substrate path:

$$C(P) = \min_{l_s(i,j) \in P} C(l_s(i,j)) \quad (1)$$

The weight (i.e. cost, delay, jitter) of the substrate path $w(P)$ can be measured as the sum of all link weights along the path P:

$$w(P) = \sum_{l_s(i,j) \in P} w(l_s(i,j)) \quad (2)$$

B. Virtual Network Model

Users/customers should send requests to the substrate provider to set up on-demand VN topologies with different capacity parameters over the shared substrate. The VN request can be represented by a weighted undirected graph $G_v = (N_v, L_v)$, where N_v is the set of virtual nodes and L_v is the set of virtual links between nodes of the set N_v. Each virtual node $n_v \in N_v$ is associated with a minimum required capacity denoted by $C(n_v)$. Each virtual link $l_v \in L_v$ between two virtual nodes is associated with a capacity weight value $C(l_v)$ which denotes the minimum required bandwidth capacity of the virtual link l_v.

In this work, the VN request is represented by the quadruple $Req = (Reqid, G_v, V_v, M_v)$ where ***Reqid*** represents the unique identifier for the request ***Req***. $G_v = (N_v^{Reqid}, L_v^{Reqid})$, where N_v^{Reqid} and L_v^{Reqid} denote the set of virtual nodes and links, respectively, associated to the request ***Req***. In this work, V_v and M_v denote a node capacity vector and a link capacity matrix, respectively, associated to the graph G_v such that:

- V_v=[$C(n_v^i)$] is the minimum required capacity vector for virtual nodes n_v^i, where $1 \leq i \leq |N_v^{Reqid}|$.
- M_v=[$C(l_v(i,j))$] is the minimum required bandwidth capacity matrix for virtual links $l_v \in L_v^{Reqid}$ between nodes n_v^i and n_v^j, where $1 \leq i,j \leq |N_v^{Reqid}|$.

C. Virtual Network Mapping Problem Description

Based on the substrate and VN models, the challenge is to find the best mapping between the virtual graph G_v and the substrate graph G_s. This mapping, denoted by ***MAP***, should achieve the best load balancing in the substrate network resources, with respect to the capacity constraints, while reducing the cost.

1) Node Mapping: Let $MAP_N : N_v^{Reqid} \rightarrow N_s^{Reqid} \subseteq N_s$ denotes a mapping function between virtual nodes and substrate nodes, where N_s^{Reqid} represents the set of substrate nodes capable of supporting at least one virtual node of a request ***Req***, i.e. $N_s^{Reqid} = \{n_s \in N_s \mid C(n_s) \geq \min_{n_v \in N_v^{Reqid}}\{C(n_v)\}\}$.

2) Link Mapping: Let $MAP_L : L_v^{Reqid} \rightarrow \phi \subseteq \psi$ denotes a mapping function between virtual links and substrate paths, where $\phi = \{P \in \psi \mid C(P) \geq C(l_v), \forall\, l_v \in L_v^{Reqid}\}$

3) Objectives: Finding the optimal VN mapping solution that satisfies multiple objectives and constraints can be formulated as an NP-hard problem. The objective of this work is to design, implement and evaluate a distributed heuristic algorithm to efficiently assign VNs to specific substrate nodes and links, while minimizing the network cost. To simplify the problem, we assume that the substrate network resources (e.g. CPU, bandwidth) are unlimited to accept and handle all VN requests. We also suppose that all VN requests are defined and specified in advance, i.e. Offline problem.

IV. Distributed Localized VN Mapping Algorithm

Since VN topologies can become quite large, mapping the requested VN to the entire substrate at once is not feasible for latency and complexity reasons. One well known and viable solution is to subdivide the entire VN topology into a set of elementary clusters (e.g. path, tree and star clusters). This decomposition may reduce the complexity of mapping the entire VN topology and can provide also an efficient solution to deal with the dynamic aspect of a substrate.

This work focuses on the star based VN decomposition. The VN topology can be viewed as interconnected star (or hub-and-spoke) clusters. The hub-and-spoke cluster is composed of a central node (i.e *hub*) to which multiple adjacent nodes (i.e. *spokes*) are connected. Spokes may also represent the hubs of other clusters. The mapping of a VN topology to a substrate network is achieved by assigning sequentially the hub-and-spoke clusters constituting the VN. The selection and mapping of these clusters are in general performed by using centralized heuristic algorithms [6], [9]. This paper addresses how to decompose and map the star topologies in a *collaborative* and *decentralized* manner.

The selection of the hub-and-spoke clusters from the entire VN topology is performed as follows:

1) Find the virtual node with the highest capacity as the hub of the cluster.
2) Determine the neighboring virtual nodes directly connected to the hub node to represent the spoke nodes.
3) Remove the hub and spoke nodes as well as their corresponding links from the VN topology
4) Select the next hub-and-spoke cluster: Go to (1)

The mapping of the hub-and-spoke cluster to the substrate is executed as follows:

1) Find the substrate node, called *root*, with the maximum available resources to be assigned to the hub node.
2) Determine the set of substrate nodes able to support the spoke nodes based on shortest path algorithms.
3) Remove the substrate nodes and paths already assigned to the hub-and-spoke cluster.
4) Select the next root node: Go to (1)

In this work, the VN mapping algorithm used for selecting and assigning the hub-and-spoke clusters to the substrate is distributed. Each substrate node designated as "root node" will be responsible for selecting and mapping one cluster to the substrate. The root nodes should communicate, collaborate and interact with each other to plan collective VN mapping decisions. This enables *distributed localized mapping* of VNs. The proposed distributed algorithm can be viewed as a cooperative task executed jointly by all root nodes via messages exchange. The distributed algorithm relies on a communication protocol, called *Virtual Network Mapping Protocol*, to exchange messages and control/signaling information between all substrate nodes.

A. Virtual Network Mapping Protocol

The VN mapping protocol is based on five types of messages: *MSG*, *START*, *NOTIFY*, *NEXT* and *STOP*. These messages are sent and received asynchronously between the substrate nodes to exchange information and organize the distributed algorithm iterations. The protocol messages are described as follows:

- **MSG** (n_s, $C(n_s)$): This message is used to exchange node capacities among all substrate nodes. Each node broadcasts its capacity to all other nodes.
- **START** (*Req*): This message is sent from a synchronizer (e.g. network provider) to all substrate nodes to trigger and start the distributed mapping algorithm for the request *Req*.
- **NOTIFY** (*Reqid*, $\{n_v, n_s\}$, l_v): After assigning a virtual node n_v (or a virtual link l_v) of a request *Req*, the root node n_s should notify the other nodes about this mapping.
- **NEXT**(*Reqid*): Once the hub-and-spoke cluster of a request *Req* has been assigned, the root node supporting the hub should send a NEXT to all substrate nodes to solicit a successor root node for supporting the new cluster.
- **STOP** (*Reqid*): This message is used to stop the execution of the distributed algorithm once the entire request *Req* is mapped successfully onto the substrate.

B. Distributed VN Mapping Algorithm

This section presents the proposed distributed VN mapping algorithm to honor different VN requests on the shared substrate. This distributed algorithm should be able to process multiple VN requests simultaneously.

In this work, each substrate node n_s^i, $(1 \leq i \leq |N_s|)$ executes three distributed algorithms running in parallel: a *Capacity-Node-Sorting* algorithm (*see Algorithm 1*), a *Shortest Path Tree (SPT)* algorithm and a *main VN mapping* algorithm (*see Algorithm 2*). The Capacity-Node-Sorting and SPT algorithms are executed continuously and periodically to maintain up to date information about the substrate network (e.g. node and link capacities). This information is communicated to the main VN mapping algorithm at run time during the VN assignment process.

Two predefined functions are used in this paper: *Sort* and *Head* function. *Sort* is a sorting function that sorts a vector of nodes (e.g. N_s) based on their capacities by ordering them from higher to lower capacity. The *Head* function returns the first element (node identifier) of the vector.

1) The Capacity-Node-Sorting algorithm: As depicted in the *Algorithm1*, the substrate nodes exchange their node capacities via the MSG message (see *P1* and *P2*). Based on this exchange of capacities, each node constructs (*step a*) progressively the capacity vector (V_s): V_s=[$C(n_s^i)$],($1 \leq i \leq |N_s|$) which represents the available capacity vector for all substrate nodes. The vector V_s should be updated dynamically when the available capacity of a given substrate node changes (e.g. node failure, capacity update after a mapping process, etc). The *Sort* function is used to order N_s from higher to lower capacity. The Capacity-Node-Sorting algorithm maintains up to date information about all substrate node capacities and returns a sorted N_s vector to the main VN mapping algorithm.

Algorithm 1 : Capacity-Node-Sorting Algorithm for each Substrate Node $n_s^i \in N_s, (1 \leq i \leq |N_s|)$

Input/Output: N_s; $V_s \Leftarrow NULL$

P1. n_s^i Send **MSG**$(n_s^i, C(n_s^i))$ to all nodes $n_s^j \in N_s \setminus \{n_s^i\}$

P2. **Upon** receiving **MSG**$(n_s^j, C(n_s^j))$ **do**

a. $V_s[n_s^j] \Leftarrow C(n_s^j)$ // *Create/Update the capacity vector* V_s

b. **Sort** (N_s) // *Sort* N_s *based on their capacities in* V_s

2) The Shortest Path Tree (SPT) algorithm: The SPT algorithm computes a path from node n_s^i to each node n_s^j $(i \neq j)$ so that the weight between node n_s^i and all other nodes is minimum. In this work, the SPT is implemented by using the distributed Bellman-Ford algorithm which is known as the fastest distributed algorithm for solving shortest path problems for general network topologies [12]. We assumed that each substrate node n_s^i maintains all parameters (e.g. capacity and weight) of the links directly connected to its network interfaces. Let P_j denotes the shortest path between node n_s^i and a substrate node n_s^j. The substrate path P_j is associated with the minimum path weight $w(P_j)$ between n_s^j and n_s^i (see equations 2). Consequently, the SPT algorithm returns a set *T* associated to the node n_s^i such that: $T = \{(P_j, w(P_j))\}$. The set *T* is maintained up to date to be used by the main VN mapping algorithm.

Due to the important number of messages exchanged between the substrate nodes in both the Capacity-Node-Sorting and the SPT algorithms, a separate signalling/control network is required to support these messages. This can reduce time delay and message overload in the substrate network.

3) The main VN mapping algorithm: Upon receiving a VN request, each substrate node $n_s^i \in N_s$ executes *Algorithm 2* which in turn calls a Hub-Spokes-Mapping procedure described further in this paper. The VN mapping algorithm is started upon receiving a START message (*S1*) handling the VN request (*Req*). A N_s^{Reqid} vector should keep an up to date copy of N_s maintained by the Capacity-Node-Sorting algorithm. The main function of (*S1*) is summarized as follows:

- Check if the node n_s^i is the root node: In the case when node n_s^i represents the highest substrate node capacity (i.e. Head(N_s^{Reqid})), the node n_s^i will be designated as the root node (*S1.b.i*).

Algorithm 2 : The Main VN Mapping Algorithm for each Substrate Node $n_s^i \in N_s, (1 \leq i \leq |N_s|)$

Input: $Req = (Reqid, G_v, V_v, M_v)$ // $G_v = (N_v^{Reqid}, L_v^{Reqid})$

(S1) **Upon** receiving **START**(*Req*) **do**

// *Check if* $n_s^i \in N_s^{Reqid}$

if $C(n_s^i) < \min_{n_v \in N_v}\{C(n_v)\}$ **then Exit else**

a. $N_v'^{Reqid} \Leftarrow N_v^{Reqid}$; $N_s^{Reqid} \Leftarrow N_s$ // Initialisation

b. **if** n_s^i = **Head**(N_s^{Reqid}) **then**

i. $root \Leftarrow n_s^i$ // n_s^i *is the root*

// root node determines the first hub and spoke cluster

ii. **Sort**(N_v^{Reqid}), $hub \Leftarrow$ **Head** (N_v^{Reqid})

iii. $Spokes \Leftarrow \{n_v \in N_v^{Reqid} | \exists l_v(hub, n_v) \in L_v^{Reqid}\}$

iv. **Hub-Spokes-Mapping** (*hub*, *Spokes*)

v. **if** $N_v^{Reqid} = \emptyset$ **and** $L_v^{Reqid} = \emptyset$ **then**

Send **STOP**(*Reqid*) to all nodes in N_s^{Reqid}

else Send **NEXT**(*Reqid*) to all nodes in N_s^{Reqid}

(S2) **Upon** receiving **NOTIFY**(*Reqid*, $\{n_v, n_s\}$, l_v) **do**

a. $N_v^{Reqid} \Leftarrow N_v^{Reqid} \setminus \{n_v\}$, $N_s^{Reqid} \Leftarrow N_s^{Reqid} \setminus \{n_s\}$

b. $L_v^{Reqid} \Leftarrow L_v^{Reqid} \setminus \{l_v\}$

// *M: Set of* $\{n_v, n_s\}$ *pair already assigned*

c. $M \Leftarrow M \cup \{n_v, n_s\}$

(S3) **Upon** receiving **NEXT**(*Reqid*) **do**

// Virtual Nodes are still not assigned

a. **if** $N_v^{Reqid} \neq \emptyset$ **then**

i. $hub \Leftarrow$ **Head** (N_v^{Reqid})

ii. **Synch** (N_s^{Reqid}, N_s)

iii. **if** n_s^i = **Head**(N_s^{Reqid}) **then** $root \Leftarrow n_s^i$

iv. $Spokes \Leftarrow \{n_v \in N_v'^{Reqid} \mid \exists l_v(hub, n_v) \in L_v^{Reqid}\}$

// *A: Set of Spokes Already Assigned*

v. $A \Leftarrow Spokes \cap \{n_v \mid \{n_v, n_s\} \in M\}$

vi. $Spokes \Leftarrow Spokes \setminus A$

vii. **Hub-Spokes-Mapping**(*hub*, *Spokes*)

viii. Send **NEXT**(*Reqid*) to all nodes in N_s^{Reqid}

// Virtual Links are still not assigned

b. **if** $N_v^{Reqid} = \emptyset$ **and** $L_v^{Reqid} \neq \emptyset$ **then** $hub \Leftarrow M[n_s^i]$

i. $\forall \{n_v, n_s\} \in M$, **if** $\exists l_v(hub, n_v) \in L_v^{Reqid}$ **then**

1. $MAP_L(hub, n_v) \Leftarrow P_{n_s}$

2. $L_v^{Reqid} \Leftarrow L_v^{Reqid} \setminus \{(hub, n_v)\}$

3. Send **NOTIFY**(hub, n_v) to all nodes

// All virtual nodes and links are assigned

c. **if** $N_v^{Reqid} = \emptyset$ **and** $L_v^{Reqid} = \emptyset$ **then**

Send **STOP**(*Reqid*) to all nodes in N_s^{Reqid}

- Next, the root node determines the first hub and spoke cluster: The hub node is determined by searching the highest virtual node capacity. The *Sort* and *Head* functions applied successively to N_v^{Reqid} provide the hub node (*S1.b.ii*). The set *Spokes* of spoke nodes directly connected to the hub is determined in step (*S1.b.iii*).
- The root node assigns the hub and spoke nodes to the substrate (*S1.b.iv*) (using the Hub-Spokes-Mapping procedure).

The Hub-Spokes-Mapping procedure is responsible for assigning hub and spoke nodes to the substrate nodes. As depicted in the *Procedure 3*, the procedure assigns the hub node to the root node (*see a*), updates the capacity vector V_s in the Capacity-Node-Sorting Algorithm by sending a MSG message handling the new root node capacity (*see b*), removes the hub and root nodes from the node lists (*see c*), and notifies all substrate nodes, via the NOTIFY message, to inform them about this mapping (*see d*). Upon receiving a NOTIFY message (*see algorithm 2. S2*), the substrate nodes should remove the virtual nodes and substrate nodes already assigned from their node lists and should also store the mapping information in the set *M* (M is the set of $\{n_v, n_s\}$ pair already assigned).

Procedure 3 : Hub-Spokes-Mapping Procedure

Input: hub, $Spokes$
a. $MAP_N(hub) \Leftarrow root$
b. Send **MSG**$(root, C(root))$ to all substrate nodes
c. $N_v^{Reqid} \Leftarrow N_v^{Reqid} \setminus \{hub\}$; $N_s^{Reqid} \Leftarrow N_s^{Reqid} \setminus \{root\}$
d. Send **NOTIFY** (*Reqid*, {*hub*, *root*}) to all nodes
// Define the set of n_s nodes to be assigned to the spoke nodes
e. **Sort** (*Spokes*)
f. $K \Leftarrow$ **Card**$(Spokes)$ // *The size of the set Spokes*
g. $T^{Reqid} \Leftarrow T$ // $T =$ ***SPT***$(root)$
h. $T'^{Reqid} = \{(P_j, w(P_j)) \in T^{Reqid} \mid C(n_s^j) \geq C(n_v), \forall\ n_v\}$
i. $Min_k(T'^{Reqid})$
j. $S = \{n_s^j \in N_s^{Reqid} \mid w(P_j) \in Min_k(T'^{Reqid})\}$; **Sort** (S)
// Assign spoke nodes and virtual links to the substrate.
k. **Repeat**
i. $MAP_N($**Head**$(Spokes)) \Leftarrow$ **Head**(S)
ii. Send **MSG**(**Head**$(S), C($**Head**$(S))$) to all substrate nodes
iii. $MAP_L(hub,$ **Head**$(Spokes)) \Leftarrow P_{Head(S)}$
iv. $M \Leftarrow M \cup \{$**Head**$(Spokes), Head(S)\}$
v. Send **NOTIFY**({**Head**(*Spokes*),**Head**(*S*)},(hub,**Head**(*Spokes*)))
vi. $Spokes \Leftarrow Spokes \setminus \{$**Head**$(Spokes)\}$
vii. $N_v^{Reqid} \Leftarrow N_v^{Reqid} \setminus \{$**Head**$(Spokes)\}$
viii. $L_v^{Reqid} \Leftarrow L_v^{Reqid} \setminus \{(hub,$ **Head**$(Spokes))\}$
ix. $N_s^{Reqid} \Leftarrow N_s^{Reqid} \setminus \{$**Head**$(S)\}$
Until $Spokes = \emptyset$

Once the hub is assigned to the root node, the challenge is to efficiently define the appropriate set of substrate nodes to be assigned to the spoke nodes. Our work does not only consider the link constraint but takes also into account the spoke node capacity requirements. The Hub-Spokes-Mapping procedure determines the substrate nodes that have the shortest path to the root node and, at the same time, have the maximum node capacity. The spoke nodes with higher required capacity should be mapped to the substrate nodes with higher available capacity. These steps are achieved as follows (steps: e to j): First, the procedure takes the set *T* from the SPT algorithm, all substrate paths that do not satisfy the bandwidth capacity constraints specified in the matrix M_v should be removed. Let $T^{Reqid} = \{(P_j, w(P_j)) \mid \forall\ l_v \in L_v^{Reqid}, C(P_j) \geq C(l_v)\}$, where $C(P_j)$ denotes the available bandwidth capacity associated to the substrate path P_j (equations 1).

Once T^{Reqid} is computed, the Hub-Spokes-Mapping procedure selects from the set T^{Reqid} the pairs $(P_j, w(P_j))$ in which their associated nodes n_s^j satisfy the capacity node constraints specified in V_v. Let T'^{Reqid} denotes this set of selected pairs $(P_j, w(P_j))$. Next, the predefined function $Min_k(T'^{Reqid})$ is used to select the k-minimum weight value $w(p_i)$ from the set T'^{Reqid}, where k is the size of the set *Spokes* computed in the Hub-Spokes-Mapping procedure. Finally, the Hub-Spokes-Mapping procedure determines the set *S* of selected substrate nodes to be mapped to the spoke nodes.

Once the two sets *S* and *Spokes* are determined, the *repeat-until loop* is used in the procedure (*step k*) to map, one by one, the spoke nodes and their virtual links to the substrate.

Let us come back to the main algorithm. Once the first hub-and-spoke cluster is mapped, the node n_s^i should send a NEXT message (*S1.b.v*) to all substrate nodes to solicit a successor root node to support the next hub-and-spoke cluster of the VN topology. Upon receiving a NEXT message (*see S3*), the new root node should first synchronize N_s^{Reqid} with the N_s vector (*S3.a.ii*). That is, the content of N_s^{Reqid} is re-sorted based on the order of N_s. Next, the same steps as *S1* are repeated to determine the hub and spoke nodes. Step (S3.a.v) determines the set of neighboring spoke nodes already assigned to the substrate (*i.e. the set A*). Next, the hub-and-spoke procedure is called to map the spoke nodes which are not already assigned (*i.e. Spokes except A*). Finally, (*S3.b*) computes the shortest paths between the hub node and the spoke nodes already assigned to previous clusters. Once the entire request *Req* is mapped successfully onto the substrate, a STOP message is sent (*S3.C*) to stop the execution of the distributed algorithm.

V. Performance Evaluation

A. Multi-Agent based Distributed VN Mapping Algorithm

Since the Multi-Agent System approach [10] represents a good tool to build modular and autonomous systems capable of operating in dynamic and distributed environments, we propose to implement and evaluate the distributed VN mapping algorithm based on autonomous and intelligent agents. The Multi-Agent based algorithm ensures distributed negotiation and synchronization between the substrate nodes. These nodes handle autonomous and intelligent agents which exchange messages and cooperate to perform the distributed VN mapping algorithm. Declarative Agent Communication Language (ACL) [13] is used to define and specify the interactions and messages between the agent-based substrate nodes. The Distributed VN Mapping Algorithm is implemented and tested over the GRID5000 platform [14]. This platform can emulate a real substrate network where different topologies can be generated. An agent development framework [15] is used to implement the autonomous agents responsible for carrying out the distributed algorithm. These agents are deployed in the GRID5000 machines to emulate the substrate nodes and to handle the main VN mapping algorithm.

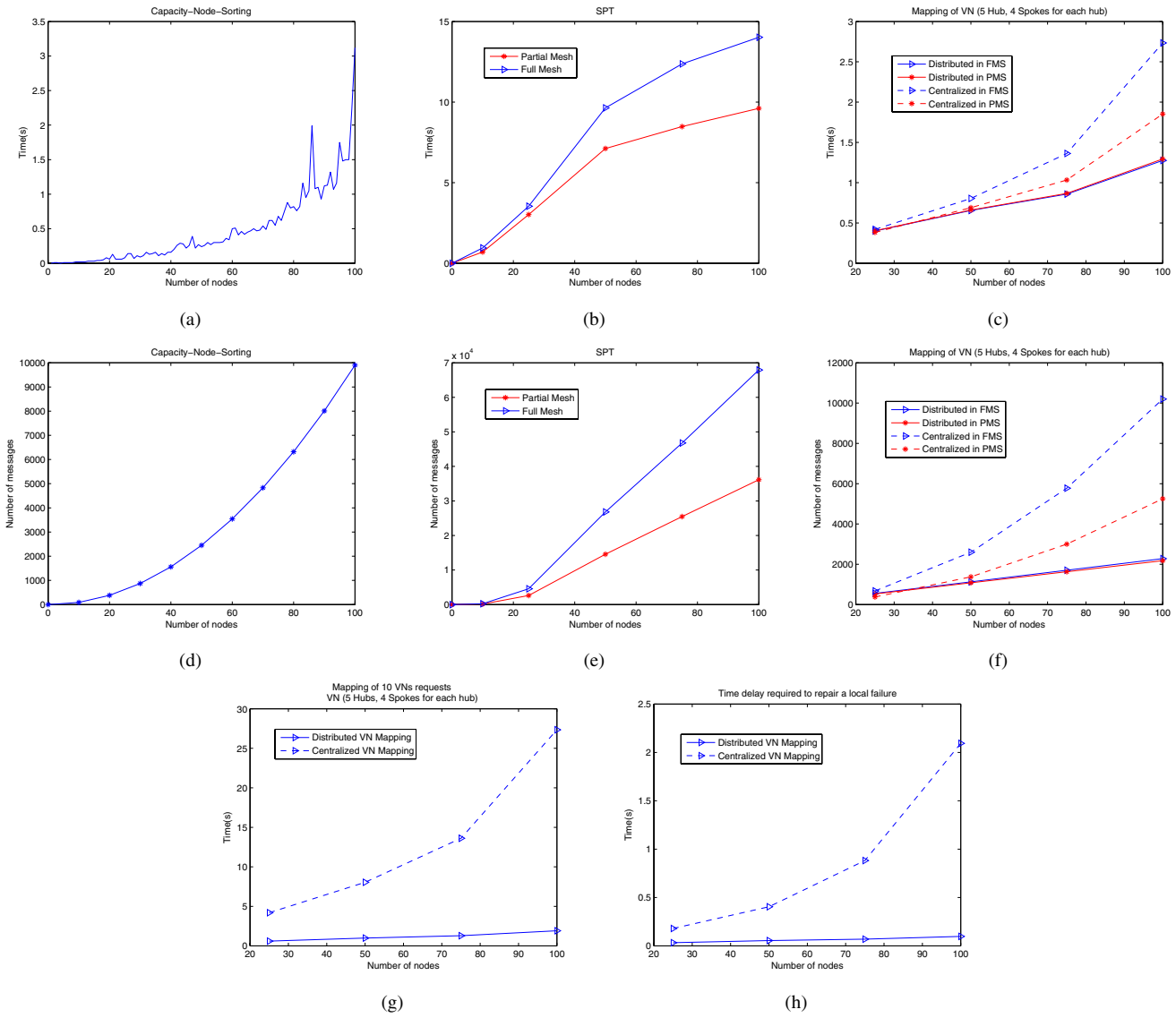

Fig. 2. (a) Capacity-Node-Sorting Algorithm evaluation: Time versus Nodes (b) SPT procedure evaluation: Time versus Nodes (c) The time delay taken by the mapping algorithm to assign one VN request in both FMS and PMS topologies: centralized vs distributed (d) Capacity-Node-Sorting Algorithm evaluation: Messages versus Nodes (e) SPT procedure evaluation: Messages versus Nodes (f) Number of messages used by the mapping algorithm to map one VN request: centralized vs distributed (g) The time delay taken by the mapping algorithm to assign ten VN requests: centralized vs distributed (h) The time delay taken by the mapping algorithm to repair a node failure: centralized vs distributed

B. Performance Results

Our objective is to evaluate the distributed VN mapping algorithm performance in terms of time delay and number of messages required to map a given VN topology to a substrate. A number of experiments have been set up to assess the three distributed algorithms: Capacity-Node-Sorting, SPT and main mapping algorithms.

The Capacity-Node-Sorting and SPT algorithms require a significant time delay due to the important number of messages exchanged between substrate nodes in the signalling/control network. The first objective is, thus, to evaluate the performance of these two algorithms in terms of time delay and message overload in the substrate network.

A first experiment has been conceived to evaluate the Capacity-Node-Sorting algorithm. Several random substrate topologies with different sizes (from 0 up to 100 nodes) are generated over the GRID5000 platform. As shown in figure 2(a), the response time remains below 1 s when the number of nodes is in the range from 0 to 80 and exhibits less variability on the average. Beyond 80 nodes, the Capacity-Node-Sorting response time is more affected but is limited to a span of 1 to 2.7 s. Figure 2(d) depicts the number of messages exchanged between nodes that grows exponentially with increasing size of the substrate topology.

The second experiment concerns the performance evaluation of the SPT algorithm. The SPT algorithm has been deployed in two types of substrate topology (Full Mesh Substrate (FMS)

and Partial Mesh Substrate(PMS)). Figure 2(b) depicts two time delay curves. The lower and upper curves represent the time delay of the SPT algorithm running on a partial mesh and a full mesh topology, respectively. Our major observation is that the additional delay between the two curves increases from 0 to 4.5 s when the number of nodes increases from 0 to 100. This additional delay is induced by the increasing number of messages illustrated in figure 2(e).

A third experiment evaluates the time delay taken by the distributed algorithm to map one VN topology (with 25 virtual nodes) to both full and partial mesh topologies. The performance results are compared to those achieved by a centralized VN mapping algorithm. The time delay required to map a VN in centralized manner (the two upper curves) is higher to the time delay needed to map a VN in a distributed fashion (the two lower curves). The additional delay, in the case of the centralized approach, is due to the time delay needed for a central coordinator to gather all information about the substrate links. This delay depends on the substrate topology (i.e. FMS vs PMS topology). This is different from the distributed approach where each substrate node already maintains all parameters (e.g. capacity and weight) of the links directly connected to its network interfaces. As illustrated in figure 2(f), the number of messages exchanged to map a VN, in both centralized and distributed cases, corroborates the delay results shown in the figure 2(c).

A fourth experiment determines the time delay taken by the main algorithm to map simultaneously multiple VN topologies to a full mesh substrate. The time delay required to assign multiple VNs in both centralized and distributed mapping cases is depicted in the figure 2(g). The time delay required to map ten VN requests (with 25 virtual nodes) in a centralized manner (the upper curve) is higher compared to the time delay needed to map ten VN requests in a distributed manner (the lower curve). The decentralized VN mapping achieves high-speed parallel processing of several VN requests.

The last experiment concerns the evaluation of the time delay needed to repair/reassign a VN when a change occurs in the substrate or the VNs (e.g. node/link failures). As shown in figure 2(h), the time delay required by the distributed VN mapping algorithm to localize the affected cluster and reassign it (see upper curve) is lower compared to the time delay needed by a centralized VN mapping to react to local failures (lower curve).

In conclusion, the performance results conducted in this paper show that:

- In the distributed mapping approach, the number of signalling messages exchanged between the substrate nodes may be important (e.g. in the case of SPT and Capacity-Node-Sorting algorithms). This increases time delay and message overload in the signalling/control network.
- Compared to the centralized mapping approach, the proposed distributed algorithm can reduce the time delay to process multiple VN requests in parallel.
- Unlike the centralized approach, the distributed VN mapping algorithm can reduce the time delay required to repair partial failures in the substrate.

VI. Conclusion

This paper presented the design, implementation and evaluation of a *distributed* VN mapping algorithm. The substrate nodes handle autonomous and intelligent agents which exchange messages and cooperate to carry out the proposed algorithm. A VN Mapping Protocol is used to communicate and exchange messages between the agent-based substrate nodes. Results from performance evaluation of the distributed mapping suggest that the challenge of constructing VNs from a shared physical network can find viable solutions.

Future work will consist of implementing and evaluating more thoroughly a self-healing system for the VN mapping. This system should monitor, analyze, identify node/link failures and repair localized problems automatically in the substrate. The distributed VN mapping algorithm presented in this paper can be a potential starting point for self-provisioning and self-management of VNs in a shared substrate and certainly a decentralized VN provisioning and management framework [16] for further investigation.

References

[1] Global Environment for Network Innovations (GENI). http://www.geni.net/.
[2] Future Internet Network Design (FIND). http://find.isi.edu/.
[3] T. Anderson, L. Peterson, S. Shenker, and J. Turner,. "Overcoming the Internet impasse through virtualization". IEEE Computer Magazine, vol. 38, no. 4, pp. 34-41, 2005.
[4] A. Bavier, N. Feamster, M. Huang, L. Peterson, and J. Rexford,. "In VINI veritas: Realistic and controlled network experimentation". in Proc. ACM SIGCOMM, Pisa, Italy, September 2006.
[5] N. Feamster, L. Gao, and J. Rexford,. "How to lease the Internet in your spare time". SIGCOMM Comput. Commun. Rev., vol. 37, no. 1, pp. 61-64, 2007.
[6] Y. Zhu and M. Ammar,. "Algorithms for assigning substrate network resources to virtual network components". in Proc. IEEE INFOCOM, 2006.
[7] J. Fan and M. Ammar,. "Dynamic topology configuration in service overlay networks: A study of reconfiguration policies". in Proc. IEEE INFOCOM, 2006.
[8] J. Lu and J. Turner,. "Efficient mapping of virtual networks onto a shared substrate". Washington University, Technical Report WUCSE-2006-35, 2006.
[9] M. Yu, Y. Yi, J. Rexford, and M. Chiang,. "Rethinking virtual network embedding: Substrate support for path splitting and migration". Princeton University, Technical Report TR-788-07, July 2007.
[10] Tesauro G., et al.," A multi-agent systems approach to autonomic computing ", Third International Joint Conference on Autonomous Agents and Multi Agent Systems, 2004, p. 464-471.
[11] R. ricci, et al. A solver for the network testbed mapping problem. *ACM Computer Communication Review*, 33(2):65–81, January 2003.
[12] R. Bellman, "Dynamic Programming". Princeton, N.J.: Princeton Univ. Press, 1957.
[13] FIPA (FOUNDATION FOR INTELLIGENT PHYSICAL AGENTS), http://www.fipa.org/.
[14] GRID 5000, https://www.grid5000.fr/
[15] JADE (Java Agent DEvelopment Framework), http://jade.tilab.com/
[16] I. Houidi, W. Louati and D. Zeghlache. "A Distributed and Autonomic Virtual Network Mapping Framework". The Fourth International Conference on Autonomic and Autonomous Systems, ICAS 2008, March 16-21, Gosier, Guadeloupe