Pro-Red: Prognostic Redesign of Survivable Virtual Networks for Cloud Data Centers

Sara Ayoubi, Yiheng Chen, and Chadi Assi

***Abstract*—This paper deals with the problem of proactive survivability of Virtual Networks (VNs) residing in a cloud data center. In all of the previous work, the protection schemes consists of augmenting the VNs with a pre-determined number of backup nodes. Further, to reduce the amount of provisioned resources, various backup resource sharing schemes are introduced, which are mainly employed when mapping the augmented VN onto the substrate network. This renders the existing redesign techniques agnostic to the backup resource sharing in the substrate network, and highly dependent on the efficiency of the adopted mapping approach. In this paper, we swerve from this dogmatic approach, and introduce Pro-Red, a novel prognostic redesign technique that is capable of foretelling (encouraging) the backup resource sharing in the substrate network, prior to the embedding phase. Our numerical results prove that this redesign technique achieves lower-cost mapping solutions and greatly enhances the attainable backup sharing, boosting the overall network's admissibility.**

I. Introduction

Network virtualization is a key enabler of the multi-tenancy concept [1], where multiple network architectures and services can run on top of the same physical infrastructure. With network virtualization, the problem of allocating resources to the various tenants emerges as a challenging problem. This problem is formally known as the Virtual Network Embedding problem (VNE), which is proven to be NP-Hard [2]; therefore, numerous efforts have been devoted towards inaugurating effective heuristics for solving it [3]–[7]. The main weakness in these suggested approaches, in addition to the lack of a guarantee on the quality of the obtained solution, is that they assume that the physical infrastructure is available at all times, which renders most of the work in the area of VNE inapplicable in scenarios where network component failures can occur. Failures in the physical infrastructure are common due to a multitude of reasons [8]. In fact, the year 2013 has witnessed multiple cloud outages [9]; one of which got hold of the famous Amazon's EC2 cloud, causing 5 million dollars in revenue loss for a single hour of offline time. With millions of dollars at stake, attention converged towards solving the Survivable Virtual Network Embedding problem (SVNE) [10]–[17]. Given that the SVNE problem is a variation of the VNE problem, it is also NP-Hard. Hence, most of the relevant literature relax the problem by targeting one network component failure type: facility node failures, network node failures, or link failures. Some further simplify the problem by considering that a single network component can fail at any given point in time. In this paper, we consider the case of single facility node failures. When a facility node fails, the hosted virtual node(s) needs to migrate to a backup facility node, as well as its associated connections to other virtual nodes belonging to the same virtual network. One way to achieve this failure recovery is by redesigning the VN request into a Survivable VN (SVN), and then mapping the resultant SVN onto the physical network. This redesign consists of augmenting the original VN with backup nodes. Each backup node is in charge of protecting one or many primary nodes. Hence, backup virtual links must be established between each backup node and the neighbors of the primary nodes it protects. Upon the failure of a facility node which hosts a virtual node v, v will migrate to its associated backup node, which will then resume the communication with v's neighbors. The augmented backup virtual nodes and links need to be provisioned with sufficient computing and bandwidth capacity to recover from any facility node failure.

The survivable redesign technique encloses multiple challenges. Chief among these challenges is deciding how many backup nodes to use and how to allocate these backup nodes to the primary nodes in each VN, such that we minimize the amount of reserved resources in the substrate network. This problem is of paramount importance since these provisioned resources will remain idle until failures occur. Hence, over-provisioning can greatly impact the network's ability to admit future requests. Indeed, the cost-efficient survivable redesign problem against single facility node failures has recurred multiple times in the literature [11], [15]–[17]. However, in all of the previous contributions, the number of backup nodes is fixed to either 1 or k, k being the number of critical nodes in a given VN. In addition, to circumvent the inconvenience of idle resources, these latter introduce various backup resource sharing opportunities which can be exploited in the substrate network upon mapping the resultant SVN. In this paper, we argue that fixing the number of backup nodes to either 1 or k could yield infeasible or even costly mapping solutions. We provide several motivational examples to support our proclamation. Moreover, we observe that all of the aforementioned redesign techniques are agnostic to the backup resource sharing in the substrate network, where this responsibility is delegated to the adopted mapping algorithm. This is problematic, since given that the SVNE is NP-Hard, adding more constraints for backup resource sharing will surely yield a more complex model. Hence, the existing literature solve this problem by relaxing the SVNE algorithm [10]–[17]. For instance, by solving the virtual node mapping and virtual link mapping disjointly [10]–

This work was made possible by the NPRP 5-137-2-045 grant from the Qatar National Research Fund (a member of the Qatar Foundation), and the NSERC discovery grant. The statements made herein are solely the responsibility of the authors.

978-0-9836283-9-2

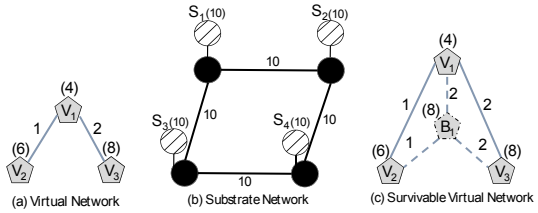


Fig. 1. Substrate Network and Virtual Network Representation

[12], [14], [16], [17] or by performing the primary and backup mapping in a sequential fashion [10], [13], [15], [16]. Multiple other decomposition schemes can be applied; however, it is these very same relaxation techniques that sacrifice the quality of the obtained solution. This results in costly embedding solutions that are incapable of exploiting backup resource sharing in the substrate network, and lead to a substantial amount of idle resources that limit the cloud provider's long term revenue.

In light of the above, we introduce Pro-Red; a novel prognostic redesign approach that explores the space between 1 and k and promotes backup resource sharing at the VN level. Hence, it alleviates this concern from the embedding algorithm and achieves cost-efficient SVNs using abridged mapping techniques. Pro-Red adopts a unique approach for the redesign; not only does it determine the augmented number of backup nodes and their connections to the primary nodes, but also their actual positioning in the VN such that it minimizes the provisioned cost at the substrate level. Hence, its prognostic property lays in its ability to foretell the backup resource sharing at the VN level, prior to the embedding phase. Our numerical results prove that our suggested approach yields significant gain in terms of increasing the substrate network's admission rate, decreasing the amount of idle bandwidth in the substrate network, and boosting the overall revenue of the cloud provider.

The rest of this paper is organized as follows: In Section II, we formally present the SVN redesign problem for single facility node failure. Section III presents firm motivational examples that prove the misfits of conventional redesign techniques. In Section IV, we introduce the theocratical foundation of Pro-Red, and then present its step-by-step procedural details. Section V introduces our SVN embedding model that complements the features of Pro-Red. Section VI is dedicated for the numerical results. We conclude the paper in Section VII.

II. Problem Definition

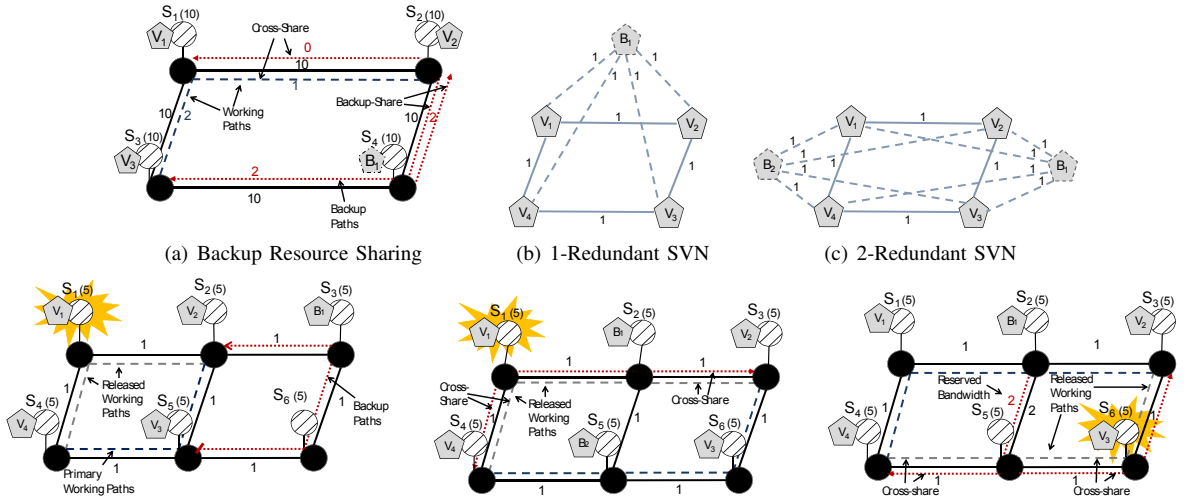
1. **The Substrate Network :** We represent the substrate network as an undirected graph denoted by G^s = (N,L), where N is the set of substrate facility nodes, and L is the set of substrate links. Facility nodes are connected to the network via network nodes (routers/switches). Each substrate facility node $n \in N$ is associated with a finite computing capacity, denoted by c_n. Similarly, each substrate link $l \in L$ has a finite bandwidth capacity, denoted by d_l. Figure 1 illustrates a substrate network with 4 facility nodes, each with a CPU capacity of 10 units (number in parenthesis above each facility node). Similarly, we observe that the substrate links interconnecting the network nodes exhibit 10 units of bandwidth capacity each (number in parenthesis above each substrate link).
2. **The Virtual Network (VN) :** A Virtual network represents a client's request to deploy an application in a cloud data center. It consists of a set of virtual nodes (virtual machines), interconnected with virtual links. The virtual links correspond to the communication requirements between the virtual nodes in a given VN request. We denote a VN as a virtual graph G^v = (V,E), where V represents the set of virtual nodes, each with a CPU demand of c_v, and e is the set of virtual links, each with a bandwidth demand of d_e. Figure 1 shows an example of a VN request with 3 virtual nodes and links, in addition to their associated CPU and bandwidth demands, respectively.
3. **Problem Definition :** Given the VN request, the SVNE problem aims to map this request onto the substrate network while providing survivability against single facility node failures. This can be done by redesigning the VN request into an SVN, which consists of augmenting the VN with backup nodes and provisioning enough bandwidth and CPU resources to recover from any facility node failure. The problem of designing reliable VNs encloses two major concerns: First, deciding how many backup nodes are needed to protect a given VN, and second, determining which backup node will be in charge of protecting which set of critical nodes. These two concerns highly depend on the substrate network capacity. On one hand, provisioning a high number of backup nodes and links greatly decreases the substrate network's admission rate, since these resources will remain idle until failure occurs. On the other hand, limiting the number of backup nodes to a pre-determined constant may yield infeasible mapping solutions. Hence, finding the optimal design of reliable VNs consists of finding the tradeoff between the amount of backup resources provisioned and the efficient utilization of the substrate network. The SVN redesign problem can thus be formulated as follows:

   Problem Definition 1. *Given a substrate network G^s = (N, L), and a VN request G^v = (V,E), Find the optimal redesign D of the given VN request G^v into a reliable VN, such that the amount of idle resources in the substrate network is minimized.*

III. The SVN Redesign Problem

A. Limitations of Conventional VN Redesign Techniques

One of the most commonly adopted redesign techniques for recovery against single node failure are formally known as the 1-redundant and k-redundant schemes. In the case of the 1-redundant scheme, the VN request is augmented with a single backup node that needs to be connected to the neighbors of each critical node via backup virtual links. Next, the resultant SVN is embedded onto the substrate network while forcing the primary and backup nodes in a given SVN to occupy



(a) Backup Resource Sharing (b) 1-Redundant SVN (c) 2-Redundant SVN

(d) 1-Redundant SVN Embedding Solution (1) (e) 2-Redundant SVN Embedding (f) 1-Redundant SVN Embedding Solution (2)

Fig. 2. Designing and Embedding Reliable VNs

distinct substrate nodes. This ensures that a single substrate node failure will not affect more than one virtual node in the same VN request. Figure 1(c) illustrates the case where the VN request presented in Figure 1(a) is augmented with a single backup node b_1, as per the 1-redundant scheme. The backup node must be provisioned with the maximum CPU demand of all the critical nodes, so it can assume any single facility node failure. Hence 8 units of CPU is reserved on backup node b_1. Moreover, for each backup virtual link connecting b_1 to any critical node v, it is sufficient to reserve the maximum bandwidth demand on v's adjacent links, since backup link (b_1,v) will only be activated upon the failure of one of v's neighbors. For example, the backup link (b_1,v_1) will only be activated in the case where virtual node v_2 or v_3 fails. In the case where v_2 fails, 1 unit of bandwidth is required to resume the communication on backup link (b_1,v_1). Similarly, in the case where v_3 fails, it will also migrate to b_1 and communicate with v_1 with 2 units of bandwidth. Given that at any point in time either v_2 or v_3 would fail, it is sufficient to reserve 2 units of bandwidth on the link connecting b_1 to v_1. The set of backup links that are activated simultaneously upon the failure of a virtual node v are denoted as the Backup-Group of v ($BG(v)$) [15]. For instance, the $BG(v_2)$ contains backup links (b_1,v_1) and (b_1,v_3). Similarly, the backup group of $BG(v_3)$ are also (b_1,v_1) and (b_1,v_2).

Now, for the k-redundant scheme, the VN is augmented with k backup nodes, where k represents the number of primary critical nodes. In this case, each backup virtual node protects a single primary node, and hence it only connects to its neighbors via backup virtual links. Each backup node along with its associated backup links will be provisioned with the same amount of resources as the primary node it protects and its adjacent links, respectively.

When a facility node fails, only the affected node will be disconnected from the substrate network. However, its adjacent network node and substrate links will remain active and capable of routing traffic. Thus, upon the failure of a facility node that hosts a virtual node v, the bandwidth on the original working paths that connect v to its neighbors in the substrate network will be released, and hence becomes available. This released bandwidth can thus be reused by the corresponding backup paths of v's backup node. Such type of sharing is known as $cross-sharing$ [15] between working and backup paths. Each virtual node v is associated with a working-group ($WG(v)$) that contains the set of v's working paths. For instance, the $WG(v_1)$ contains (v_1,v_2) and (v_1,v_3). Hence, the $BG(v_1)$ can reuse the bandwidth of the $WG(v_1)$ upon v_1's failure through cross-sharing. Moreover, given that a single node might fail at any point in time, the backup paths belonging to different backup groups can share their bandwidth in the substrate network. Such type of sharing is referred to as $backup-sharing$ [15]. Figure 2(a) shows a mapping solution for the 1-redundant SVN presented in Figure 1(c) over the substrate network in Figure 1(b). We observe that for backup link (b_1,v_3), 4 units of bandwidth needs to be reserved, since the substrate links that route this backup path do not overlap with any other appropriate backup or working paths. However, backup paths (b_1,v_1) and (b_2,v_2) overlap over substrate link $\{s_2,s_4\}$; and given that these backup paths belong to distinct backup groups, only 2 units of bandwidth need to be reserved on substrate link $\{s_2,s_4\}$, rather than 3 due to backup-sharing. Moreover, backup path (b_1,v_1) further overlaps with working path (v_1,v_2) on substrate link $\{s_1,s_2\}$; hence 0 units of bandwidth needs to be reserved on this substrate link via cross-sharing.

The problem with the 1-redundant and k-redundant schemes is that by forcing the number of backup nodes to be either 1 of k, we may end-up with infeasible or costly mapping solutions. This is due to the fact that the substrate might not have enough bandwidth capacity to route the traffic between

1 backup node to the neighbors of all critical nodes, in the case of the 1-redundant scheme. Whereas, in the case of the k-redundant scheme, a substantial amount of CPU resources remain idle until a failure occurs, since k-redundant requires as many backup nodes as primary critical nodes, not to mention the large number of backup virtual links needed to associate each backup node with its appropriate primary critical node. This motivates the need for a cost-efficient redesign technique that is capable of exploring the space between 1 and k, and finding the balance between the amount of provisioned CPU and bandwidth to yield feasible and cost-efficient embedding solutions.

B. Illustrative Example

To further illustrate the inconvenience of the conventional redesign techniques, consider the case of a 4 nodes VN, where each virtual node is considered to be critical. Using the 1-redundant scheme, we augment this VN with a single backup node, connected to the neighbors of all critical nodes via backup virtual links, as illustrated in Figure 2(b). Now, consider a substrate network with 6 facility nodes interconnected via substrate links, each with a bandwidth capacity of 1 unit, as shown in Figure 2(d). Given the 1-redundant SVN, there exist no feasible mapping solutions on the aforementioned substrate network. For instance, consider embedding the SVN using the mapping solution illustrated in Figure 2(d). When the substrate node s_1 fails, the virtual node v_1 migrates to b_1 which needs to communicate with virtual nodes v_2 and v_4. b_1 is capable of reaching virtual node v_2 through path $\{s_3 \rightarrow s_2\}$. However, the substrate network's capacity, with the current embedding solution inhibits b_1 from reaching node v_4, since the working path of $\{v_3$-$v_4\}$ remains operational. This renders the embedding solution illustrated in Figure 2(d) infeasible. By examining all possible mapping solutions of the 1-redundant SVN on the given substrate network, we find that they are all infeasible. This is because the 1-redundant scheme connects a single backup node to the neighbors of all critical nodes. Hence b_1's bandwidth demand along with the given substrate network capacity, inhibits b_1 from protecting this VN against any single node failures.

On the other hand, consider the case where the aforementioned VN is augmented with 2 backup nodes b_1 and b_2, as shown in Figure 2(c). b_1 assumes the failure of critical nodes v_1 and v_2, and b_2 replaces v_3 and v_4 in case any of them failed. Upon embedding the resultant SVN, we notice that this reliable design does indeed yield a feasible solution and requires 0 units of reserved bandwidth due to cross-sharing, as illustrated in Figure 2(e). For example, consider the case where the facility node s_1 fails; subsequently, v_1 will migrate to b_1, and that latter needs to resume v_1's communication with v_2 and v_4. The failure of virtual node v_1 leads to the release of the active bandwidth on working paths $\{s_1$-$s_2\}$ and $\{s_1$-$s_4\}$ connecting virtual node v_1 to v_2 and v_4, respecitvely. The released bandwidth will be reused by b_1 to reach v_2 and v_4 through cross-sharing. By employing cross-sharing for all other virtual node failures in the given VN, we can conclude that indeed the 2-redundant SVN requires 0 unit of reserved bandwidth.

Further, consider the same substrate network, where link $\{s_2 \rightarrow s_5\}$ has a capacity of 2 units, as illustrated in Figure 2(f). In this case, we can indeed find a feasible embedding solution for the 1-redundant SVN with a provisioned bandwidth cost of 2 units, whereas the 2-redundant scheme still requires 0 units of provisioned bandwidth.

These motivational examples prove our proclamation that by forcing the number of backup nodes to be either 1 or k, we might end up with infeasible or costly mapping solutions. Whereas when we augment the VN with i ($1 \leq i \leq k$) backup nodes (i = 2 in the above example), we achieve a balance between the amount of backup bandwidth and CPU that needs to be reserved. In fact, this balance yields a feasible solution, when the 1-redundant and k-redundant fail to find one.

This motivates the need for a redesign approach that is capable of finding that balance, rather than being fixed to either 1 or k backup nodes. By exploring the space in the range between 1 and k, we can obtain lower-cost mapping solutions, and increase the network's admissibility. This is one of Pro-Red's unique capabilities. Another advantage of Pro-Red is that it redesigns the VN in a way to promote the backup bandwidth sharing at the substrate network. In the next section we present Pro-Red's theoretical foundation that enables it to fulfil these two promises.

IV. Prognostic Redesign Approach (Pro-Red) :

A. Theoretical Foundation

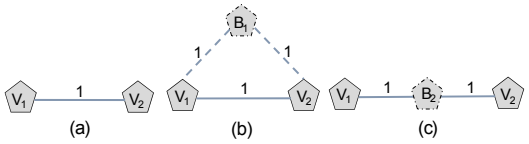


Fig. 3. Theoretical Foundation

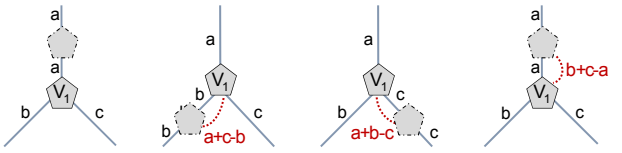
In this section, we present the theoretical foundation on which Pro-Red's redesign technique is established. We begin our explanation with a motivational example: Consider a 2 nodes VN illustrated in Figure 3(a). Augmenting the VN with a single backup node, using the 1-redundant scheme, requires 2 units of reserved bandwidth (as shown in Figure 3(b)). By employing an effective embedding approach, this estimated bandwidth cost could be minimized at the substrate network level via cross-sharing and backup-sharing. Observe, however, that by placing this backup node along the path connecting v_1 and v_2, the resulting SVN will require 0 additional units of bandwidth once embedded into the substrate network. This is due to the fact that by placing the backup node in between its associated primary nodes, we force the primary path that routes the traffic between v_1 and v_2 in the substrate network to pass through b_1 [1]. Subsequently, if either one of these

[1]Note that once a backup node is placed between v_1 and v_2, the associated working path connecting v_1 and v_2 in the substrate network will be routed differently.

primary nodes fail, the backup node will cross-share (reuse) the released primary bandwidth. It should be noted here that this redesign approach is indeed prognostic to backup resource sharing, as it is able to predict (promote) the cross-sharing (bandwidth reuse) at the VN level. Indeed, throughout our numerical results, we show that Pro-Red achieves considerable gains in terms of reducing the total bandwidth cost against the conventional redesign techniques, and greatly decreasing the network's blocking ratio.

We build on this motivational example to formulate a novel redesign technique that determines the location of backup nodes in the VN, such that cross-sharing and backup-sharing can be fully exploited in the substrate network. Placing the backup node between every two virtual nodes is definitely costly in terms of idle CPU resources. Hence, we resort to clustering a subset of virtual nodes into distinct sets, where nodes in a particular set are covered by a single backup node. In each set, the backup node is positioned such that the maximum amount of backup resource sharing is guaranteed upon the embedding. This clustering technique can thus create a balance between the amount of provisioned backup nodes and links.

To create a set, we begin by selecting the virtual node with the highest degree. This allows a larger number of primary virtual nodes to be clustered within a single set, which can substantially decrease the amount of reserved CPU resources. Once the starting node is identified, we place the backup node on the adjacent link with the highest bandwidth demand, which guarantees the most backup resource sharing. To support this



(a) Total Cost = 2a + b + c (b) Total Cost = 2a + b + 2c (c) Total Cost = 2a + 2b + c (d) Total Cost = a + 2b + c

Fig. 4. Designing Reliable VNs

analysis, consider the following example illustrated in Figure 4. Let v_1 be the node with the highest nodal degree 3. Its adjacent links have a bandwidth demand of a, b and c respectively. We assume (without loss of generality) :

$$a > b + c > b > c \tag{1}$$

In order to protect V_1, we need to place a backup node on one of its adjacent links. In this case, we have 3 different scenarios, we can either place the backup node on the link with bandwidth demand a, b, or c. These different scenarios are illustrated in Figure 4(a), 4(b), and 4(c), respectively. Notice that in the case where the backup node is placed on the link with the highest bandwidth demand a (shown in Figure 4(a)), 0 units of reserved bandwidth is needed. In fact, by placing the backup node on this former link we can always achieve the lowest total cost, since upon failure the backup node is able to reach all of v_1's neighbors by fully reusing the released bandwidth through cross-sharing. Whereas, by placing the backup node on the link with bandwidth demand b (shown in Figure 4(b)) additional bandwidth needs to be reserved in order to reach v_1's neighbors. In fact, since $b < a$, the backup node can never reach v_1's neighbor at link a without reserving an additional (a - b) units of bandwidth. The same applies to reach v_1's neighbor at link c, hence an overall (a + c - b) units of bandwidth needs to be reserved. This renders a total cost of ($2a$ + b + $2c$), which is obviously more expensive that the redesign solution presented in Figure 4(a). Note that in the case where $a \leq (b + c)$, and a is the link with the highest bandwidth demand; to place the backup node on link a, a total of (b + c - a) must be reserved, as illustrated in Figure 4(d). However, this solution still renders the lowest total bandwidth cost.

B. Pro-Red Algorithm :

Algorithm 1 Pro-Red: Prognostic Redesign Heuristic

1: **Given** $V(U, E)$ */*Virtual Network Topology*/*
2: */*Set cover flag for nodes and links to false*/*
3: **for** ($u \in U$) **do**
4: $u.covered$ = false;
5: **end for**
6: C = { }; */*Initialize the list of covered nodes*/*
7: **while** (|C| < U) **do**
8: $\hat{C}$ = {U} - C;
9: **Step 1: Find Starting Node**
10: n = $GetHighestNodeDegree(\hat{C})$;
11: L = $GetAdjacentLinks(n)$;
12: **Step 2: Find Starting Link**
13: l = $GetHighestBW(L)$;
14: **Step 3: Create a new Set**
15: s = $CreateSet(n,l)$;
16: C = $C \cup \{s\}$;
17: **end while**

In this section, we present the SVN redesign heuristic that is founded on the theories and observations presented in Section IV-A. The objective of this algorithm is to assign a backup node for each critical node in the given VN topology; we refer to a critical node that is assigned to a backup node as $covered$ (or protected). Initially, all the virtual nodes in the VN topology are considered as $uncovered$; hence, we initialize the virtual nodes with a cover flag set to false. Next, we define two new sets C and $\hat{C}$ that are updated at the end of every iteration with the list of covered and uncovered nodes, respectively. The process terminates when C contains all the critical nodes in the VN request. At each iteration, the algorithm creates a single set. We define a set as an ensemble of critical nodes protected by a single backup node. To create a set, we first need to identify a starting point, from which a set will begin and grow. Based on the previous observations presented in Section IV-A, the starting point is defined by node v_1 with the highest nodal degree in the list of uncovered nodes $\hat{C}$, and its adjacent link e with the highest bandwidth demand. Next,

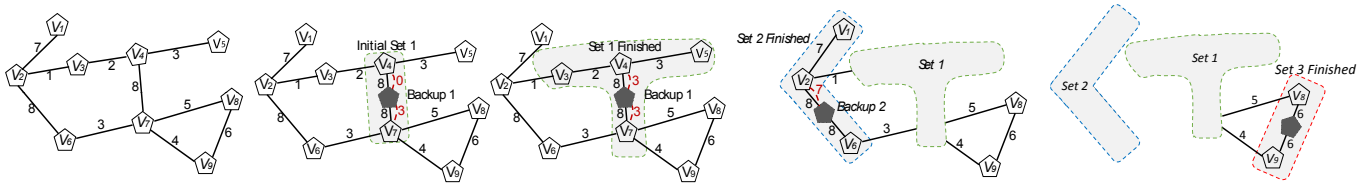
the algorithm invokes the $CreateSet$ function that returns a set s which contains the critical nodes covered by the newly discovered set.

Algorithm 2 CreateSet(virtual_node v_1,virtual_link e)

1: s = {};
2: v_2 = $GetOtherNode(v_1,e)$;
3: **Step 4: Create a new backup node** b
4: $\hat{e_1}$ = new virtual_link(v_1,b);
5: $\hat{e_2}$ = new virtual_link(v_2,b);
6: $setCPU(b,\max(v_1,v_2))$;
7: $d_{\hat{e_1}} = d_e$ - Sum($GetAdjacentLinksBW(v_1)$);
8: $d_{\hat{e_2}} = d_e$ - Sum($GetAdjacentLinksBW(v_2)$);
9: $v_1.covered = v_2.covered$ = true;
10: $s = s \cup \{v_1,v_2\}$;
11: **Step 5: Protect Adjacent Leaf Nodes**
12: **while** (T: $v_1.hasAdjacentLeafNodes()$) **do**
13: t = T.next();
14: $t.covered$ = true;
15: $s = s \cup$ {t};
16: $setBW(\hat{e_1},\max(d_{\hat{e_1}},d_{(v_1,t)}))$;
17: $setCPU(b,\max(b,t))$;
18: **end while**
19: Repeat the same while loop for Adjacent leaf nodes of v_2
20: **Step 6: Protect Adjacent non-leaf Nodes**
21: **while** (R: $v_1.hasAdjacentNonLeafNodes()$) **do**
22: r = R.next();
23: **if** ($(2d_{(v_1,r)} \geq$ Sum($GetAdjacentLinksBW(r)$))&&
24: ($d_{\hat{e_1}} \geq d_{(v_1,r)}$ && ($r.hasAdjacentLeafNodes()$ = null)) **then**
25: $r.covered$ = true;
26: $s = s \cup r$;
27: $setCPU(b,\max(b,t))$;
28: **end if**
29: **end while**
30: Repeat at line 21 for v_2
31: **return** s;

In Algorithm 2, we highlight the procedural details of the $CreateSet$ function. It begins by creating a new backup node b to be placed between the edge nodes (v_1, v_2) of link e. To exploit cross-sharing, link e will be replaced by two backup virtual links $\hat{e_1}$ and $\hat{e_2}$ that position backup node b in between nodes v_1 and v_2. This would force the primary virtual link connecting nodes v_1 and v_2 to be routed through b. Hence, if any one of them failed, the released bandwidth on links $\hat{e_1}$ and $\hat{e_2}$ can be reused by backup node b. Initially, the CPU demand of b is set to the maximum CPU demand of critical nodes v_1 and v_2. Also, the bandwidth demand on link $\hat{e_1}$ is set to the sum of the bandwidth demands of v_1's adjacent links, subtracted by the bandwidth demand of link e, since that latter will be released and cross-shared (reused) upon failure of node v_1. The same applies when assigning the bandwidth demand on link $\hat{e_2}$. Subsequently, nodes v_1 and v_2 are now protected (covered) by backup node b. Once this set is established, we need to grow it in order to cover the highest number of adjacent nodes possible without incurring too much additional backup bandwidth. First, we need to include all the adjacent leaf nodes in the set, otherwise leaf nodes will be left uncovered, or would require a dedicated backup node, which is seemingly not cost efficient. To cover leaf nodes, we need to adjust the bandwidth demand on links $\hat{e_1}$ and $\hat{e_2}$, appropriately, with enough bandwidth to assume the failure of any leaf node, as well as the CPU demand of backup node b. Finally, the algorithm will also attempt to cover non-leaf neighbors nodes of v_1 and v_2 using backup-sharing. Meaning, without reserving any additional bandwidth on backup virtual links $\hat{e_1}$ and $\hat{e_2}$. Given a non-leaf neighbor node v' of v_1, if the sum of the bandwidth demand on n''s adjacent links, including link (v',v_1) is smaller than the reserved bandwidth on link $\hat{e_1}$, and excluding link (v',v_1) is smaller than the bandwidth demand on link (v',v_1); further, if (v',v_1) is the link with the highest bandwidth demand among v''s adjacent links, then v' could be included in v_1's set and subsequently protected by backup node b without incurring any additional backup bandwidth via backup-sharing. Finally, the algorithm returns the set of nodes that are covered by the newly created set s. The $CreateSet$ function has a complexity of O(n), which renders the complexity of Pro-Red's redesign heuristic to be O(n^2), since we call the $CreateSet$ function for each uncovered node in the VN request.

To further illustrate the enactment of Pro-Red's redesign algorithm, consider the VN topology presented in Figure (5-a). The algorithm begins by identifying a starting node and link, which in this case are node v_7 with link {v_4,v_7}, since they correspond to the node with the highest degree, and its adjacent link with the highest bandwidth demand. Next, a set is created by placing a backup node b_1 on link {v_4,v_7}, as shown in Figure 5(a). This implies that nodes v_4 and v_7 are now protected by backup node b_1. Since the sum of the adjacent links to v_4 (excluding link {v_4,b_1}) is smaller than the bandwidth on link {v_4,b_1}, 0 units of bandwidth is required to protect node v_4. However, the sum of the adjacent links of v_7 is 12, which implies that 3 units of bandwidth must be reserved on link {v_7,b_1} in order to protect node v_7. Next, the set is grown by adding the adjacent leaf nodes of v_4 and v_7. The only leaf node found is v_5 which will be added to the set, and subsequently incurs 3 additional units of bandwidth to be reserved on link {v_4,b_1}. Finally, the potential of adding non-leaf nodes is explored. Indeed, we find that only node v_3 can be added to the set with no additional bandwidth, as shown in Figure 5(c). When no additional nodes can be further added to the set, the set becomes saturated. Subsequently, the $CreateSet$ function returns set s_1 with backup node b_1 protecting virtual nodes v_3, v_4, v_5, and v_7, which leaves 5 critical nodes in the given VN uncovered. Hence, a new set is initiated starting with node v_2, since it represents the next uncovered node with the highest nodal degree. The same process repeats, and returns set s_2 with backup node b_2 protecting virtual nodes v_1, v_2, and v_6, as shown in Figure 6(d). Finally, set s_3 is created with backup



(a) (b) (c) (d) (e)

Fig. 5. Step-by-Step SVN Redesign Algorithm.

node b_3 covering nodes v_8 and v_9, illustrated in Figure 5(e). Once all critical nodes are protected, the algorithm terminates. At the end, we obtain 3 sets with 3 backup nodes protecting 9 critical nodes with only 12 units of reserved bandwidth.

V. The SVN Embedding

Upon obtaining the redesigned VN, the next step is to embed this latter onto the substrate network. Since the SVNE problem is NP-Hard, we adopt a disjoint mapping approach, where we preform the node mapping first and then the link mapping. For the node mapping, we use the VMP algorithm in [11] to find a set $\mathcal{M}$ of feasible node mapping solutions. Note both the primary and backup node placement is performed jointly. For the link mapping, we formulate an ILP model that will select the lowest cost mapping solution $m \in \mathcal{M}$, and determine its corresponding link mapping solution. Given our prognostic redesign, our ILP model assumes as input the backup resource sharing identified during the redesign phase. Our link mapping model is thus formulated as follows:

- Parameters:

$G^s(N, L)$: substrate network with N nodes and L links.
$G^v(V, E)$: virtual network with V virtual nodes and E virtual links.
$\hat{E}$: the set of backup virtual links $\hat{e} \in \hat{E}$.
$\mathcal{M}$: the set of all node mapping solutions $m \in \mathcal{M}$.
S : the list of constructed sets $s \in S$.

$$\delta^m_{n,n} = \begin{cases} 1, & \text{if } v \text{ is mapped onto substrate node } n \text{ in } m, \\ 0, & \text{otherwise.} \end{cases}$$

$$\sigma^s_{\hat{e}} = \begin{cases} 1, & \text{if backup link } \hat{e} \text{ belongs to set } s, \\ 0, & \text{otherwise.} \end{cases}$$

$\gamma_{\hat{e}}$: the amount of bandwidth to be reserved on $\hat{e}$.

- Decision Variables:

$$x_m = \begin{cases} 1, & \text{if node mapping solution } m \text{ is chosen,} \\ 0, & \text{otherwise.} \end{cases}$$

$$y^{e,m}_{i,j} = \begin{cases} 1, & \text{if } e \text{ is mapped on substrate link } (i,j) \text{ in } m, \\ 0, & \text{otherwise.} \end{cases}$$

$$y^{\hat{e},m}_{i,j} = \begin{cases} 1, & \text{if } \hat{e} \text{ is mapped on substrate link } (i,j) \text{ in } m, \\ 0, & \text{otherwise.} \end{cases}$$

$t_{i,j}$: the primary traffic mapped on substrate link (i,j).
$\hat{t}_{i,j}$: the backup traffic reserved on substrate link (i,j).

$$Min \sum_{(i,j)\in L} (t_{i,j} + \hat{t}_{i,j})$$

Subject to

$$\sum_{m\in\mathcal{M}} x_m = 1 \tag{2}$$

$$y^{e,m}_{i,j} \le x_m \quad \forall e \in E, m \in \mathcal{M}, (i,j) \in L. \tag{3}$$

$$t_{i,j} = \sum_{m\in\mathcal{M}} \sum_{e\in E} y^{e,m}_{i,j} d_e \quad \forall (i,j) \in L. \tag{4}$$

$$y^{\hat{e},m}_{i,j} \le x_m \quad \forall \hat{e} \in \hat{E}, m \in \mathcal{M}, (i,j) \in L. \tag{5}$$

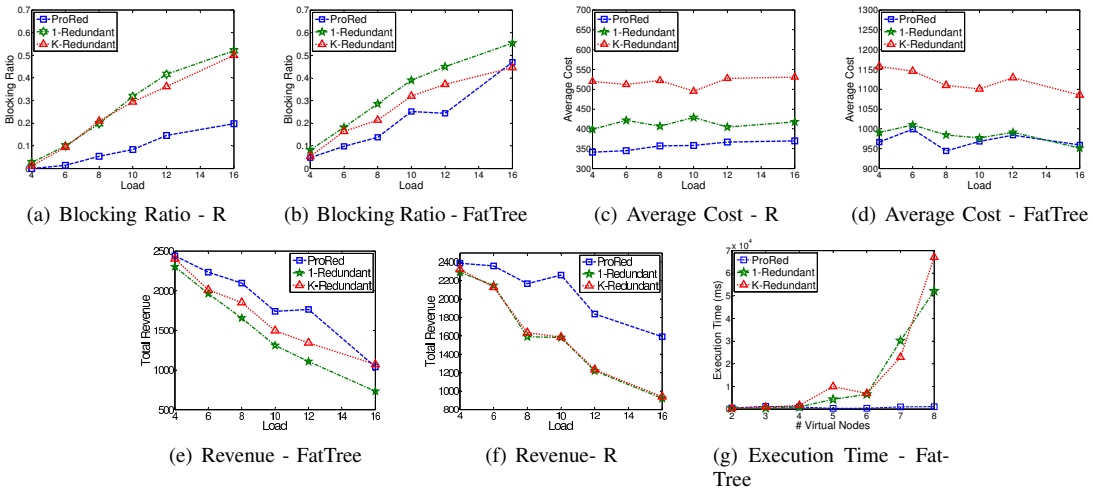
$$\hat{t}_{i,j} \ge [\sum_{m\in\mathcal{M}} \sum_{\hat{e}:\sigma^s_{\hat{e}}=1} y^{\hat{e},m}_{i,j} \gamma_{\hat{e}} \quad \forall s \in S] \quad \forall (i,j) \in L. \tag{6}$$

$$t_{i,j} + \hat{t}_{i,j} \le d_l \quad \forall l : (i,j) \in L. \tag{7}$$

We aim at minimizing the overall bandwidth cost for the given SVN mapping solution. This encourages the model to select a node mapping solution where the nodes are not too widely spread. Hence, we set the model's objective function to minimize the sum of primary and backup traffic on the substrate links. Two flow conservation constraints are needed to route the primary and backup virtual links. The details of these constraints have been omitted due to space limitation. Constraint (2) forces the model to select a single node mapping solution. Constraint (3) indicates that a primary link mapping solution will only be constructed for the chosen node mapping solution. Constraint (4) measures the primary traffic routed on every physical link in the substrate network. Constraint (5) indicates that a backup link mapping solution will only be constructed for the chosen node mapping solution. Constraint (6) measures the backup link traffic routed on every physical link in the substrate network. Notice that for every given link, the amount of backup bandwidth reserved is the max of the sum of backup bandwidth provisioned within each given set. This is because our approach enables backup sharing among the backup virtual links belonging to distinct sets, since these latter will not be activated at the same time. Constraint (7) ensures that the sum of the primary and backup bandwidth routed on each substrate link does not violate its capacity.

VI. Numerical Results

We evaluate the performance of Pro-Red against the 1-redundant and k-redundant schemes for various metrics: Blocking Ratio, Average Cost, Revenue and Execution Time. We adopt two different substrate network topologies to conduct this evaluation. The substrate networks used for our



(a) Blocking Ratio - R (b) Blocking Ratio - FatTree (c) Average Cost - R (d) Average Cost - FatTree

(e) Revenue - FatTree (f) Revenue- R (g) Execution Time - Fat-Tree

Fig. 6. Comparison between Pro-Red, 1-Redundant and K-Redundant Scheme.

simulation are FatTree (K=4) [1], in addition to a randomly generated network [18] R, with 36 nodes and 48 links. In both these substrate networks, we set the CPU capacity of each host node to 48 units, and the bandwidth capacity on the substrate links is set to 750 units. We perform the redesign and mapping of VNs in an online fashion, upon the arrival of each request. Hence, we assume that the VNs arrival and departure follow a Poisson distribution. The VN requests are randomly generated, where the size of each VN can range between 2 to 8 virtual nodes. Each virtual node can be connected to any other virtual node in the VN request with a probability of 50%. The CPU demand of the virtual nodes is set to be in the range [1:5], and the bandwidth demand on the virtual links is in the range [10:50]. To conduct the comparison with 1-redundant and k-redundant, we employ the same embedding approach presented in section V; however, we replace Constraint (6) with Constraint (10) that performs cross-sharing and backup-sharing [15]:

$$\sum_{m\in\mathcal{M}} \sum_{\hat{e}:\in BG(v)} y_{i,j}^{\hat{e},m}\ d_{e:\{v,d(\hat{e})\}} - \sum_{m\in\mathcal{M}} \sum_{e:\in WG(u)} y_{i,j}^{e,m}\ d_{\hat{e}} \leq \hat{t_{i,j}} \quad \forall\ (i,j) \in E,\ v \in V \quad (10)$$

In all test cases, the results are averaged over 5 runs.

1. **Blocking Ratio :** The first metric we evaluate is the blocking ratio. We vary the load of the poisson process between 4 to 16 and run Pro-Red, 1-redundant and k-redundant over the same distribution and generated VN list. The results are shown in Figure 6(a) and 6(b). We observe that for FatTree (K = 4), Pro-Red achieves a lower blocking ratio over 1 and k-redundant. This gain is mainly attributed to Pro-Red's ability to explore the space between 1 and k. Since FatTree connects each host node to the substrate network with a single substrate link, this architecture puts 1-redundant at a great disadvantage, as the backup node is forced to go through a single substrate link in order to reach the neighbors of all the critical nodes in a given VN. Indeed, we observe that Pro-Red achieves 51% lower blocking ratio over 1-redundant when the load on the substrate network is equal to 8. Similarly, we observe that Pro-Red achieves 40% lower blocking ratio over k-redundant when the load is equal to 6. Though k-redundant does not concentrate the backup bandwidth load on a single substrate link, its redesign technique requires as many backup nodes as the number of critical nodes in a VN request, which renders a substantial amount of CPU and bandwidth demand to associate each backup node with its corresponding primary virtual node. Whereas Pro-Red maintains a balance between the number of allocated backup nodes and links, thus its blocking ratio prevails over its peers. Given that the FatTree topology does not allow Pro-Red to employ its prognostic redesign technique, we further compare these 3 redesign techniques over a randomly generated topology to evaluate the advantage of this property. We observe that Pro-Red achieves encouraging gain in terms of decreasing the blocking ratio. We find that as we increase the load to 16, 1-redundant and k-redundant's blocking ratio becomes at least 50%, while Pro-Red's blocking ratio remains below 20%. In this case, Pro-Red achieves a 60% gain. The rich interconnection of the random network topology enables Pro-Red from exercising its prognostic redesign technique, i.e. finding node mapping solutions that are capable of placing the backup node in between its associated primary virtual nodes. Hence, Pro-Red is capable of greatly decreasing the incurred bandwidth cost for each VN, and subsequently increasing the network's admissibility.
2. **Average Cost :** For a given VN, the cost is measured using the objective function of the SVN embedding model presented in Section V, which represents the sum of the primary and backup bandwidth cost incurred by this VN in the substrate network. For each of the aforementioned techniques, we average the cost of the admitted VNs as we vary the load. First, we compare the average cost obtained by Pro-Red against 1 and k-redundant using FatTree.

Again, we observe that in addition to Pro-Red's lower blocking ratio, it can also achieve a lower average cost. This greatly motivates the inconvenience of fixing the number of backup nodes to either 1 or k. Further, as we compare the average cost over the randomly generated topology, we observe that Pro-Red's unique redesign technique enables it to greatly decrease the average cost over the substrate network. While the gain over 1-redundant is between 8 and 17%; however compared to k-redundant, Pro-Red achieves a constant gain of 30%. Pro-Red's prognostic redesign technique for backup resource sharing enables it to achieve this gain, while 1-redundant and k-redundant falls short due to their agnostic approach.

3. **Revenue :** Revenue is an important metric that highly complements the blocking ratio metric. A low blocking ratio does not necessarily implicate a high revenue. This is because the concerned model may only be capable of admitting small-size VNs with low CPU demands. When in fact, large size VNs with substantial CPU requirements are more profitable to the cloud provider. In this regard, we measure the revenue obtained using Pro-Red, versus the 1-redundant and k-redundant schemes. Given that the aim of the metric is to evaluate each of the aforementioned techniques's ability to admit large and profitable VNs, we measure the revenue of each admitted VN in function of its overall CPU demands and size using the following equation: Revenue = $\sum_{v \in V} c_v + \pi_v$ |V|. We observe that for the FatTree network presented in Figure 6(e), Pro-Red achieves encouraging results, with a 59% gain over 1-redundant and 31% gain over k-redundant for a load of 12. Similarly, Figure 6(f) presents the obtained results over the random network. We observe that as we increase the load, Pro-Red's revenue gain increases. In fact, for a load of 16, Pro-Red achieves 40% revenue gain over both 1 and k-redundant. This gain is mainly attributed to ProRed's unique redesign properties, which significantly reduce the average cost, and hence leverage the efficient utilization of the substrate network. Subsequently, ProRed is capable of admitting larger and more profitable VNs in comparison with the 1 and k-redundant schemes.
4. **Execution Time :** Finally, we measure the runtime of embedding a single SVN by varying its size between 2 to 8 virtual nodes, and we compare the execution time of the 3 redesign techniques over the FatTree network (illustrated in Figure 6(g)). We observe that the runtime of 1 and k redundant follows a step increase as we vary the size of the SVN. This is due to its cross-sharing and backup-sharing (Constraint 10) that measures the allocated bandwidth on each substrate link for each primary virtual node during the embedding phase. The size of this constraint grows more complex as the number of virtual nodes in the SVN increases. However, our prognostic redesign technique alleviates this load from the embedding algorithm, which is reflected in the linear execution time achieved by Pro-Red. Pro-Red returns the sets of backup nodes and their associated backup links, as well as the amount of required backup resources to be reserved, while promoting backup resource sharing. Hence, all of these information will serve as an input to the model, rather than being explored at the embedding phase. This alleviated load is reflected in the runtime gain that Pro-Red achieves.

VII. Conclusion

In this paper, we presented Pro-Red a novel prognostic redesign technique for survivable virtual networks against single facility node failures. Pro-Red swerves from the dogmatic redesign techniques that fix the number of backup nodes to either 1 or k. Further it is equipped with a unique property that enables it to design SVNs that can highly promote backup resource sharing once embedded in the substrate network. This property lays in positioning of the backup nodes in the SVN such that backup-sharing and cross-sharing can be fully exploited. We compared Pro-Red against 1-redundant and k-redundant schemes, and we show that it achieves significant gains in terms of decreasing the blocking ratio, achieving lower average cost and substantially higher revenue, in considerably lower execution times.

References

[1] NM. Chowdhury et al. A survey of network virtualization. *Computer Networks*, 54(5):862–876, 2010.

[2] D. G Andersen. Theoretical approaches to node assignment. *Computer Science Department*, page 86, 2002.

[3] NM. Chowdhury et al. Virtual network embedding with coordinated node and link mapping. In *INFOCOM*, pages 783–791. IEEE, 2009.

[4] M. Chowdhury et al. Vineyard: Virtual network embedding algorithms with coordinated node and link mapping. *TON*, 20(1):206–219, 2012.

[5] Y. Zhu et al. Algorithms for assigning substrate network resources to virtual network components. In *INFOCOM*, pages 1–12, 2006.

[6] M. Yu et al. Rethinking virtual network embedding: substrate support for path splitting and migration. *ACM SIGCOMM*, 38(2):17–29, 2008.

[7] J. Lischka et al. A virtual network mapping algorithm based on subgraph isomorphism detection. In *ACM workshop on Virtualized infrastructure systems and architectures*, pages 81–88. ACM, 2009.

[8] U. Budnik. Lessons learned from recent cloud outages. 2013.

[9] JR. Raphael. The worst cloud outages of 2013. 2013.

[10] M. Rahman et al. Survivable virtual network embedding. In *NETWORKING 2010*, pages 40–52. Springer, 2010.

[11] J. Xu et al. Survivable virtual infrastructure mapping in virtualized data centers. In *IEEECLOUD 2012*, pages 196–203. IEEE, 2012.

[12] H. Yu et al. Migration based protection for virtual infrastructure survivability for link failure. In *OFC 2011*, 2011.

[13] H. Yu et al. Survivable virtual infrastructure mapping in a federated computing and networking system under single regional failures. In *GLOBECOM 2010*, pages 1–6. IEEE, 2010.

[14] W. Yeow et al. Designing and embedding reliable virtual infrastructures. *ACM SIGCOMM Computer Communication Review*, 41(2):57–64, 2011.

[15] H. Yu et al. Cost efficient design of survivable virtual infrastructure to recover from facility node failures. In *ICC*, pages 1–6. IEEE, 2011.

[16] B. Guo et al. Survivable virtual network design and embedding to survive a facility node failure. *Journal of Lightwave Technology*, 32(3):483–493, 2013.

[17] Q. Hu et al. Survivable network virtualization for single facility node failure: A network flow perspective. *Optical Switching and Networking*, 10(4):406–415, 2013.

[18] A. Medina et al. Brite: An approach to universal topology generation. In *Modeling, Analysis and Simulation of Computer and Telecommunication Systems, 2001. Proceedings. Ninth International Symposium on*, pages 346–353. IEEE, 2001.