

# Constructing Minimal Overlay to Support Policy-Based Access Control Groups (Policy-compatible overlay networks)

Bong Jun Ko, Starsky H. Y. Wong, Kang-Won Lee  
IBM T. J. Watson Research Center  
Hawthorne, NY, USA

Email: {bongjun\_ko, hwong, kangwon}@us.ibm.com

Chi-Kin Chau  
Computer Laboratory, University of Cambridge  
Cambridge, UK

Email: ckc25@cam.ac.uk

**Abstract**—Overlay networks have been studied extensively in recent years to achieve more reliable and failure-resistant communications, to ensure certain performance characteristics among a group of nodes, or to support efficient search in a peer-to-peer networks. However, one of the important considerations for building an overlay, namely policies, has been neglected in the past research. In this paper, we formally study the problem of constructing an overlay that satisfies a given set of policy-based access control groups, and prove that it is NP-complete; (b) We design centralized algorithms that can achieve results very close to the optimal case (within 3% in terms of the total edge cost); (c) We also design distributed algorithms that does not require a global knowledge of the network and show that it performs well. In particular, it can generate overlays whose cost is up to 30% smaller than a naive construction based on a minimum spanning tree.

## I. INTRODUCTION

Overlay networks have been studied extensively in recent years for their practical importance in many applications. Various types of overlay networks and mechanisms to construct them have been proposed to achieve more reliable and failure-resistant communications among nodes [6], [13], to ensure certain performance characteristics among a group of nodes, including bandwidth [14], data transfer delay [5], [11], efficient P2P search [10], [15], reliable content delivery [1], and multicast/group communication [7]. However, one of the important considerations for building an overlay has been missing in the past research, namely *policies*. In the networking community, cyber security and information assurance issues (i.e., making sure that information access is granted to the users that have credential) are becoming increasingly more critical. Thus it is important to design a mechanism to build an overlay that can support access control policies in a flexible and scalable manner.

Broadly speaking, policies are the rules that specify high level intent of a person or an organization, and are often described in the form of constraints [3]. In our context, we consider policies that define constraints as to which nodes can

*touch* which types of sensitive data. Here by *touch* we mean common network operations such as receive, forward, and process (e.g., encode, decode, downsample, retransmit, etc.). Needless to say such policies will be useful in distributing and controlling sensitive information, such as classified and secret information in the military and intelligence community, sensitive business information (e.g., HR data such as salary information), and private information of an organization (e.g., tentative budget numbers) or of an individual (e.g., health record of a patient).

Of particular interest to this research is sharing information across multiple collaborating groups in the emerging smart network applications where certain types of information can be shared across groups but other types of information should be kept within the group responsible for it. Let us consider two examples. First, in a smart grid system, multiple energy companies and municipalities may share information at a coarse granularity so that they can monitor the global trend of energy consumption in a wide geographic area. But detailed information (e.g., real-time power consumption of each household) is retained within the entities that are in charge (i.e., the energy company and the city that the household belongs to) for privacy. This also helps mitigate information overload to the other parties. Second, in a smart health care system, multiple hospitals, households, and government entities may share medical information but different entities may have access to different pieces of information at different granularity. For example, CDC may query hospitals for aggregate pathological data to analyze the trend of a suspected pandemic. On the other hand, a specialist may pull detailed patient record from a primary care physician under the consent of the patient. These policies could be stated as follows.<sup>1</sup>

- If (organization is (energycompany(household) or municipality(household))) then allow access(MeterReading)
- If (queryOrigin is CDC) then allow access(aggregatePathologyData)

<sup>1</sup>There are several policy languages that can encode the abstract policy rules shown below, including [4], [12], including a system that can translate policy rules in natural language (with constrained vocabularies) into machine processible policies [2].

- If (specialist isIn ConsentList(patient)) then allow access(fullMedicalRecord)

From these examples, we can see that policy-based control of information flow will be highly flexible and scalable in the sense that the policy rules themselves are general but still could be used to evaluate numerous instances. For instance, we do not need to specify whether the inquiring municipality is West Chester, Rockland, or Orange county in the first example. We just need to specify which attributes (in this case municipality) of a household should be checked.

Although the flexibility and scalability of policy rules are powerful, they can create problems when we try to construct overlays for access controlled groups based on them. More specifically, since each attribute of a data object (e.g., household, patient) can be used to specify policies and typically there will be multiple attributes per object (e.g., gender, race, occupation, etc. for a patient object), the number of groups can potentially be huge. In the current IT systems that use policies the number of policies could be in the order of  $O(10)$  -  $O(1000)$ . If we assume the similar scale for the emerging smart network applications, the number of groups that need to be maintained could be also large because a single policy may result in multiple groups (as illustrated in the example below), and maintaining a large number groups will be costly to ensure that they are consistent with the policies whenever the policies or the group membership change. Thus the main focus of this paper is to create a minimum cost overlay that satisfies a set of policy rules.

For concrete discussion, consider the example in Figure 1. In this figure, there are two policies: *Financial data* should only be accessed by *Directors*; and *Project data* should only be accessed by the nodes in the same *Department*. These two policies effectively generate three groupings ( $G_1 - G_3$ ) of nodes as shown in the figure. We assume that group members can trust each other in the sense that node  $v$  only forwards group  $i$ 's data to node  $u$  iff node  $u$  also belongs to group  $i$ . With this assumption, our goal is to construct a single overlay to distribute data on which a group's communication does not have to traverse through any member outside the group.<sup>2</sup> In the figure, overlay (c) violates the policy because  $N_1$  cannot talk with  $N_3$  without involving a non-group member  $N_2$ , hence violating the grouping rule  $G_1$ . Similarly overlay (d) violates the grouping rule  $G_2$ .

In this paper, we study the problem of constructing an overlay that satisfies a given set of policy-based grouping. Based on graph theoretical analysis and rigorous algorithm design we present the following major contributions:

- We formulate an optimization problem in a graph-theoretic setting for constructing an overlay network with minimum number of links that satisfies the group-level connectivity requirements derived from a set of

<sup>2</sup>While one can define many different versions of interesting overlay construction problems (e.g., for isolation policy, which prohibits having a link between isolated groups), this simple formulation turned out to be rich and complex enough; Hence we just focus on this simple policy in this paper.

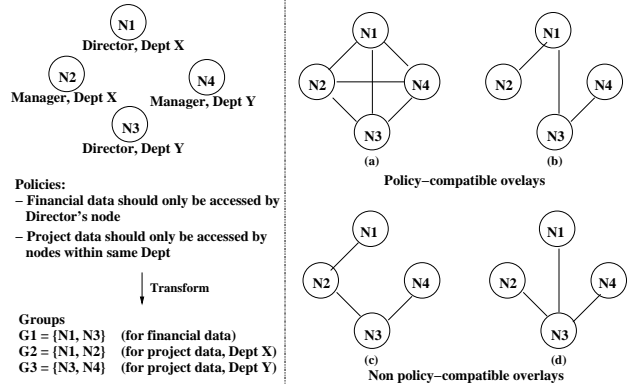


Fig. 1. Examples of policy-compatible overlays

information access policies. We formally prove its NP-completeness.

- We design centralized heuristic algorithms for the policy-compatible overlay construction problem. We prove that the proposed algorithms will always construct policy-compatible graphs while trying to keep the number of links added in the network as minimum as possible. Simulation result shows that the proposed algorithm can achieve results that are very close (within 3%) to optimal for a small set of nodes. We also design distributed algorithms and show that they can significantly reduce the cost (by up to 30%) compared to a naive construction based on minimum spanning trees. The proposed distributed algorithms are shown to achieve performance close to the centralized algorithms.
- We extend our algorithms to support a secondary optimization goal of minimizing the total link cost along with the primary objective of minimizing the number of links. We provide a single, tunable parameter that can control the behavior of the algorithms to gracefully transition between the two objectives.

The remainder of this paper is organized as follows: Section II provides a short introduction on policy-based access control. Section III formulates the policy-compatible overlay construction problem and proves that this problem is NP-complete. Sections IV and V propose centralized and distributed overlay construction algorithms, respectively. Section VI evaluates the proposed algorithms using simulation, and finally Section VII concludes the paper.

## II. POLICY-BASED ACCESS CONTROL

We consider a policy model that follow a general format of *event-condition-action* (ECA) [3], which is described as “when *event* occurs, if *condition* is true then execute *action*”. Suppose we have a set of *nodes*  $N$  that want to access some data, and a set of *data contents*  $D$ . In a policy statement, nodes are described by a well-defined set of *attributes*,  $A = \{a_1, \dots, a_{|A|}\}$ , where each attribute is a variable that supports the operations  $\{=, \neq, <, >, \leq, \geq, \in, \notin, \dots\}$ . Each node  $u$  is then described by the set of attributed-value pairs

$\{(a_i(u), v_i(u)) : i = 1, \dots, |A|\}$  The contents are similarly described by a set of attributes  $B = \{b_1, \dots, b_{|B|}\}$ . For example, the attributes of the nodes can be `ipSubnet`, `subscriptionLevel`, `organizationId`, etc. and the content attributes can be `contentType`, `securityLevel`, `authorName` in a particular context. In policy statements, these attributes can be used to check the condition to determine whether a certain node can access certain data, and subsequently can decide how the nodes should be grouped into access control groups.

Suppose we are given a set of  $K$  policy statements,  $X = \{x_1, \dots, x_K\}$  where all  $x_k$ 's are the approval policies (i.e., the action is always *Approve* when the condition is met).<sup>3</sup> Let  $h_k^{(n)}$  and  $h_k^{(c)}$  respectively denote the portion of the boolean expression for the node attributes and the data attribute, i.e., the  $condition(x_k) = h_k^{(n)} \wedge h_k^{(c)}$ . Then for each  $x_k$ , we define a set of nodes  $N_k \in N$  such that  $N_k = \{u \in N \mid h_k^{(n)}(u) = true\}$ , which can access a subset of contents  $D_k = \{d \in D \mid h_k^{(c)}(d) = true\}$ . In other words, for each policy  $x_k$ , we form a ‘‘distribution group’’ of nodes  $N_k$  for a set of data  $D_k$ , within which a secure group communication should be enabled.

In Section III, we discuss how to realize the secure communication groups imposed by the policies based on an overlay network, and formulate it in a graph-theoretic optimization problem.

### III. COMPLEXITY OF PO CO PROBLEM

We prove the NP-completeness of the PoCO optimization problem. The decision version of the problem is as follows.

*Definition 1: (Policy-compatible Overlay (PoCO) Decision Problem)* Given a set of nodes  $N$ , and a collection of subsets,  $\Omega = \{N_1, \dots, N_K \subseteq N\}$ . Decide if there exists a compatible  $G = (N, E)$ , such that  $|E| \leq \Delta$ .

*Theorem 1:* Policy-compatible overlay decision problem PoCO  $(N, \Omega, \Delta)$  is NP-complete.

*Proof:* It is easy to show that PoCO is in NP. We examine each induced subgraph  $G_k$  of  $G$  to see if it is connected or not. Checking the connectivity in a graph is polynomial in time.

To show PoCO is NP-hard, we rely on a polynomial time reduction from 3SAT problem.

*Definition 2: (3SAT Problem)* Consider a 3-CNF formula  $F$  consisting  $m$  clauses and  $h$  variables, i.e.  $F = c_1 \wedge c_2 \wedge \dots \wedge c_m$ , where each  $c_i = y_{j_1} \vee y_{j_2} \vee y_{j_3}$  and  $y_{j_1}, y_{j_2}, y_{j_3} \in \{x_1, \bar{x}_1, \dots, x_h, \bar{x}_h\}$ .  $F$  is said to be satisfiable, if there exists a truth assignment to  $F$ , such that every clause has at least one true literal. 3SAT is well-known to be NP-complete.

Given a 3-CNF formula  $F$ , we assume each clause does not contain a literal and its complement (as this is trivially satisfiable). We construct a corresponding PoCO  $(N, \Omega, \Delta)$ , such that  $F$  is satisfiable, if and only if  $(N, \Omega, \Delta)$  is satisfiable.

<sup>3</sup>We assume the policy set does not include conflicting or ineffective policy, e.g., using policy analysis techniques described in [3].

First, set  $N = \emptyset$ . For each  $x_j, \bar{x}_j$ , we add three nodes  $a_j, b_j, c_j \in N$ , and create two subsets in  $\Omega$ :  $N_{x_j} = \{a_j, b_j, c_j\}$  and  $N_{x_j}^{a_j, b_j} = \{a_j, b_j\}$ .

Then, for each  $c_i = y_{j_1} \vee y_{j_2} \vee y_{j_3}$ , we add three nodes  $o_i, p_i, q_i \in N$ , and create eight subsets in  $\Omega$ :

$$N_{c_i}, N_{c_i}^{p_i, x_{j_1}}, N_{c_i}^{p_i, x_{j_3}}, N_{c_i}^{q_i, x_{j_2}}, N_{c_i}^{q_i, x_{j_3}}, N_{c_i}^{o_i, x_{j_1}}, N_{c_i}^{o_i, x_{j_2}}, N_{c_i}^{o_i, x_{j_3}}$$

We next describe the construction of these subsets:

- 1) First, we set  $N_{c_i} = \emptyset$  and add  $o_i, p_i, q_i \in N_{c_i}$ .
- 2) Next, if  $y_{j_1}^{j_1} = x_{j_1}$ , then
  - i) Add  $a_{j_1}, c_{j_1} \in N_{c_i}$ ,
  - ii) Create  $N_{c_i}^{p_i, x_{j_1}} = \{a_{j_1}, p_i\}$ ,  $N_{c_i}^{o_i, x_{j_1}} = \{c_{j_1}, o_i\}$ .
 elseif  $y_{j_1}^{j_1} = \bar{x}_{j_1}$ , then
  - i) Add  $b_{j_1}, c_{j_1} \in N_{c_i}$ ,
  - ii) Create  $N_{c_i}^{p_i, x_{j_1}} = \{b_{j_1}, p_i\}$ ,  $N_{c_i}^{o_i, x_{j_1}} = \{c_{j_1}, o_i\}$ .
- 3) Repeat for  $y_{j_2}$ , but we use  $q_i$  instead of  $p_i$ .
- 4) Repeat for  $y_{j_3}$ , but we use both  $q_i$  and  $p_i$ .

It is easy to see that the construction of  $(N, \Omega)$  is polynomial in time. See Fig. 2 for an illustration of the construction of PoCO  $(N, \Omega, \Delta)$  for a given  $F$ .

There are some remarks. If a subset  $N_k \in \Omega$  has two nodes only (i.e.  $N_k = \{a, b\}$ ), then  $(a, b) \in E$  for any compatible  $G = (N, E)$ . Thus,  $N_{x_j}^{a_j, b_j}$ ,  $N_{c_i}^{p_i, x_{j_1}}$ ,  $N_{c_i}^{p_i, x_{j_3}}$ ,  $N_{c_i}^{q_i, x_{j_2}}$ ,  $N_{c_i}^{q_i, x_{j_3}}$ ,  $N_{c_i}^{o_i, x_{j_1}}$ ,  $N_{c_i}^{o_i, x_{j_2}}$ ,  $N_{c_i}^{o_i, x_{j_3}}$  are two-node subsets, which must be included as edges in any compatible  $G$ .

Finally, we set  $\Delta = 2h + 7m$ .

(If Part): We show that if  $F$  is satisfiable, then  $(N, \Omega, \Delta)$  is satisfiable by constructing compatible  $G = (N, E)$ . First, we set  $E = \emptyset$ , and we include the two-node subsets  $N_{x_j}^{a_j, b_j}$ ,  $N_{c_i}^{p_i, x_{j_1}}$ ,  $N_{c_i}^{p_i, x_{j_3}}$ ,  $N_{c_i}^{q_i, x_{j_2}}$ ,  $N_{c_i}^{q_i, x_{j_3}}$ ,  $N_{c_i}^{o_i, x_{j_1}}$ ,  $N_{c_i}^{o_i, x_{j_2}}$ ,  $N_{c_i}^{o_i, x_{j_3}}$  as edges in  $E$ , for all variable  $j$  and clause  $i$ . Thus, the induced subgraphs of these subsets are connected.

Then, for each variable  $x_j$ , either one of  $x_j$  or  $\bar{x}_j$  is true. If  $x_j$  is true, then we add  $(a_j, c_j) \in E$ . Otherwise, if  $\bar{x}_j$  is true, then we add  $(b_j, c_j) \in E$ . Thus, the induced subgraph of  $N_{x_j}$  is connected.

Next, for each clause  $c_i$ , one literal must be true. Hence, the corresponding  $(x_j, c_j)$ , where  $x_j \in \{a_j, b_j\}$ , has been already included in  $E$ . Thus, the induced subgraph of  $N_{c_i}$  is connected.

Therefore,  $G = (V, E)$  is compatible. Moreover,  $|E| = 2h + 7m = \Delta$ , and  $(N, \Omega, \Delta)$  is satisfiable.

(Only-if Part): We show that if  $(N, \Omega, \Delta)$  is satisfiable, then  $F$  is satisfiable. Suppose  $G = (V, E)$  is compatible and  $|E| = \Delta = 2h + 7m$ . Since the induced subgraphs of two-node subsets  $N_{x_j}^{a_j, b_j}$ ,  $N_{c_i}^{p_i, x_{j_1}}$ ,  $N_{c_i}^{p_i, x_{j_3}}$ ,  $N_{c_i}^{q_i, x_{j_2}}$ ,  $N_{c_i}^{q_i, x_{j_3}}$ ,  $N_{c_i}^{o_i, x_{j_1}}$ ,  $N_{c_i}^{o_i, x_{j_2}}$ ,  $N_{c_i}^{o_i, x_{j_3}}$  must be connected in  $G$  and their edges are distinct, they take up  $h + 7m$  edges from  $E$ . Hence, there remain  $h$  edges.

Next, we know that the induced subgraphs of  $h$  number of  $N_{x_j}$  subsets are also connected in  $G$  using distinct edges. That totally take up all remaining  $h$  edges. Hence, each  $N_{x_j}$  takes

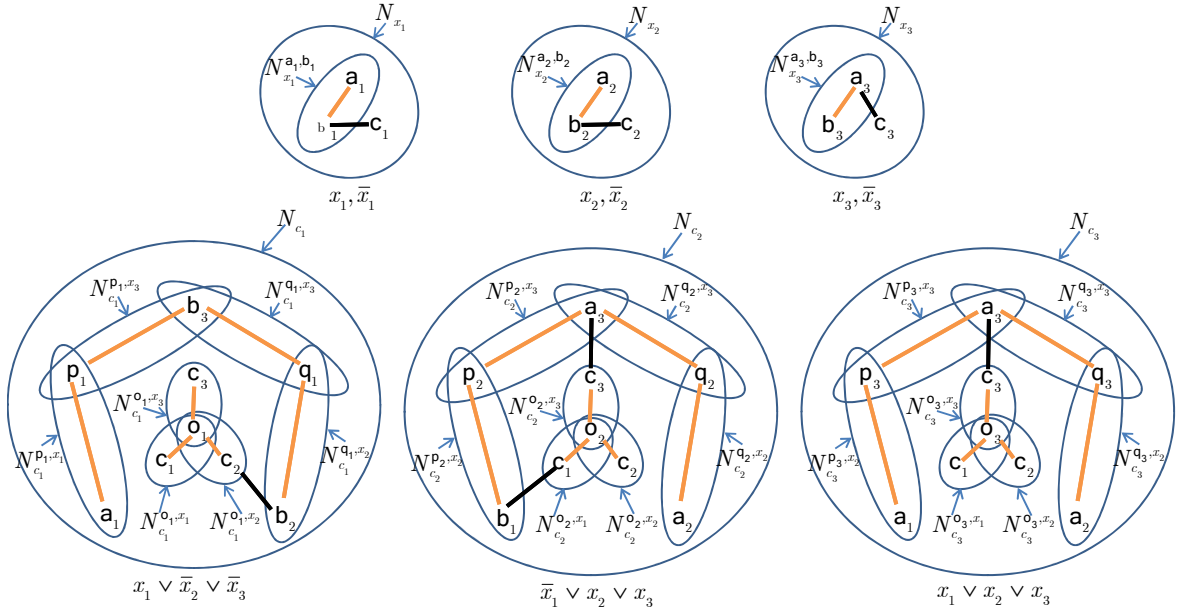


Fig. 2. An illustration of construction of PoCO  $(N, \Omega, \Delta)$  for a given  $F = (x_1 \vee \bar{x}_2 \vee \bar{x}_3) \wedge (\bar{x}_1 \vee x_2 \vee x_3) \vee (x_1 \vee x_2 \vee x_3)$ . The truth assignment is  $x_1 = 0, x_2 = 0, x_3 = 1$ , which is depicted as the selected black edges in the figures.

up exactly one remaining edge. Therefore, we set  $x_j$  as true, if  $(a_j, c_j) \in E$ . Otherwise, set  $\bar{x}_j$  as true, if  $(b_j, c_j) \in E$ . This is a consistent assignment for each variable.

Finally, we also know that the induced subgraphs of all  $N_{c_i}$  are connected in  $G$ . This implies that at least one  $(x_j, c_j)$  in  $N_{x_j}$ , where  $x_j \in \{a_j, b_j\}$ , is also present in  $N_{c_i}$ . Otherwise, other edge in  $N_{c_i}$  will require to take up an extra edge in  $E$ . Hence, every clause is satisfiable.

Therefore, we show that PoCO is NP-hard, because 3SAT problem is NP-complete. ■

Since PoCO is NP-complete, the optimization problem is unlikely to be solvable in polynomial time.

#### IV. CENTRALIZED OVERLAY CONSTRUCTION ALGORITHMS

We begin by defining a few terms and notations that will be used in the description of the algorithms throughout this section.

Suppose we are given a set of nodes  $N$  and a collection of its subsets  $\Omega = \{N_1, \dots, N_K\}$ . We denote by  $\Omega_v \subset \Omega$  the groups that a node  $v \in N$  belongs to, i.e.,  $\Omega_v = \{N_k \in \Omega | v \in N_k\}$ . Similarly,  $\Omega_{u,v}$  denotes the groups that nodes  $u$  and  $v$  both belong to, i.e.,  $\Omega_{u,v} = \Omega_u \cap \Omega_v$ .

A pair of nodes  $u$  and  $v$  both in some group  $N_k$  are *group-connected* in  $N_k$  ( $N_k$ -*group-connected* in short) in a graph  $G = (N, E)$  if there exists a connected path between them in the subgraph  $G_k = (N_k, E_k)$ , where  $E_k \subset E$  is the set of edges in  $E$  incident only on the nodes in  $N_k$ . We denote by  $C_{E_k}(N_k) \subset N_k \times N_k$  the set of  $N_k$ -group connected node pairs in  $G_k = (N_k, E_k)$ .

Given a subgraph  $G_k = (N_k, E_k)$ , the following function returns the number of *new* group-connected pairs in  $N_k$  when

an edge  $(u, v)$  is added to  $E_k$ :

$$\text{NEW-CONNS}(u, v, N_k, E_k) = C_{E_k \cup \{(u,v)\}}(N_k) - C_{E_k}(N_k).$$

The value returned by NEW-CONNS function is used in our algorithms hereafter as the “utility” of an edge for a graph  $G_k$ , in terms of how many node pairs to be connected due to the addition of an edge to a graph. It is easy to see NEW-CONNS returns in a polynomial time since there exist efficient algorithms to verify the connectivity of all node pairs in a graph.

##### A. Centralized algorithms

Our first centralized algorithm shown in Algorithm IV-A is based on a greedy decision such that, at each step, an edge is inserted whose addition maximizes the number of new group-connected node pairs in the graph constructed thus far. Let  $E^U$  be the set of all pairs of nodes in  $N$ , i.e.,  $E^U = \{(u, v) | u \in N, v \in N, u \neq v\}$ .

At each step of the WHILE loop (line 3-13), the GREEDY-CONN keeps selecting an edge whose insertion to the graph can get the most pair of nodes group-connected (lines 5, 10), and adds it one by one to the graph (lines 11, 12), until no edge can add any new group-connected node pairs (line 7). When more than one edges can be selected, one of them will be selected arbitrarily for breaking the tie.

It is easy to see GREEDY-CONN completes in a polynomial time w.r.t.  $|N|$  and  $|\Omega|$  since, first, the NEW-CONNS routine returns in a polynomial time which is repeated for each of  $K$  groups (line 5), and, at each step of the main loop, one edge is taken from all potential edge sets of size  $|N|(|N| - 1)/2$  (line 11) and added to the graph  $G$  (line 12).

Fig. 3 illustrates an example of the execution of GREEDY-CONN algorithm for two groups  $N_1 = \{1, 2, 3\}$  and  $N_2 =$

---

**Algorithm 1** *GREEDY-CONN*: Input  $(N, \Omega)$ , Output  $G = (N, E)$

---

```

1:  $E \leftarrow \emptyset$ 
2:  $F \leftarrow E^U$ 
3: while  $F \neq \emptyset$  do
4:   for each  $(u, v) \in F$  do
5:      $m_{(u,v)} \leftarrow \sum_{k=1}^K \text{NEW-CONNS}(u, v, N_k, E_k)$ 
6:   end for
7:   if  $\max m_{(u,v)} = 0$  then
8:     break
9:   end if
10:   $(u^*, v^*) \leftarrow \arg \max \{m_{(u,v)} : (u, v) \in F\}$ 
11:   $F \leftarrow F - \{(u^*, v^*)\}$ 
12:   $E \leftarrow E \cup \{(u^*, v^*)\}$ 
13: end while
14: return  $G = (N, E)$ 

```

---

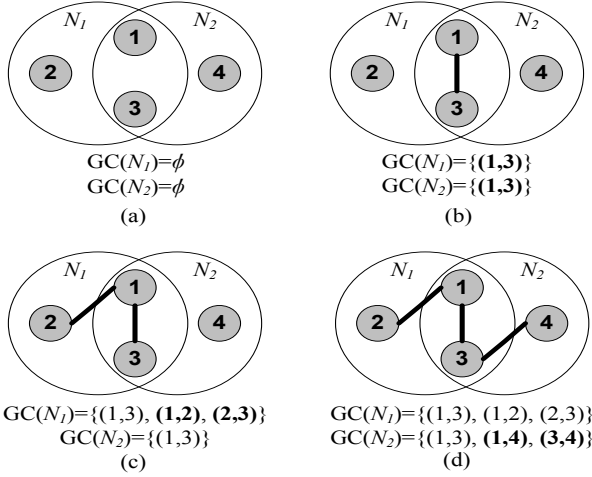


Fig. 3. Example edge assignment by GREEDY-CONN algorithm for two groups  $N_1 = \{1, 2, 3\}$  and  $N_2 = \{1, 3, 4\}$ . The new node pairs that become group-connected in each group is indicated in bold letters.

$\{1, 3, 4\}$ , where, at each stage of the execution, the set of group-connected node pairs for each group is shown in  $GC(N_k)$ . For instance, starting from the empty set of edges in Fig. 3(a), the first edge added is  $(1, 3)$  since it will make two group-connections (one in each group), as shown in Figure 3(b)—all others make only one group-connection except  $(2, 4)$  which is useless in this grouping. Then one of the edges  $(1, 2)$ ,  $(2, 3)$ ,  $(1, 4)$ , and  $(3, 4)$  can be added at the second step as any of them would make two pairs of nodes get group-connected in its respective set;  $(1, 2)$  is added in this example with the tie broken arbitrarily (Fig. 3(c)). Note that the addition of  $(1, 2)$  not only make not only the node pair 1 and 2 get connected in  $N_1$  but also make 2 and 3 get connected via node 1, which makes the addition of edge  $(2, 3)$  unnecessary thereafter. Finally, edge  $(3, 4)$  is added from group  $N_2$  (Figure 3(d)), resulting in a graph compatible with the grouping. Note that, our greedy algorithm achieves the optimal assignment in this simple example.

The following lemma shows GREEDY-CONN returns a graph compatible w.r.t.  $\Omega$ .

*Lemma 1:* Graph  $G = (N, E)$  returned by GREEDY-CONN( $N, \Omega$ ) is compatible w.r.t.  $\Omega$ .

*Proof:* The proof is by contradiction. Suppose there is a subgraph  $G_k = (N_k, E_k)$  that is induced by a set  $N_k \in \Omega$  and is not connected. By definition,  $G_k$  has at least one pair of nodes,  $u$  and  $v$ , between which there is no connected path in  $G_k$ . The edge  $(u, v)$  is not  $E_k$  (hence not in  $E$  either) because it would make a connected path between  $u$  and  $v$ . Therefore, since  $(u, v) \notin E$ ,  $F \cup E = E^U$ , and  $F \cap E = \emptyset$ , upon the completion of the algorithm,  $F$  is not empty and contains  $(u, v)$ , for which  $m_{(u,v)} > 1$  (the edge  $(u, v)$  can make the node pair  $u$  and  $v$   $N_k$ -group connected). This means none of the two terminating conditions of the algorithm (line 4 and 10) is satisfied, contradicting to the assumption that the algorithm has returned. ■

While GREEDY-CONN makes the greedy choice of the edges toward the our primary objective of minimizing  $|E|$  in the graph compatible w.r.t.  $\Omega$ , it is oblivious of the total cost of edges added to the graph. To address our secondary objective of minimizing  $c(E)$ , we modify GREEDY-CONN to what is called *GREEDY-MARGIN-CONN*, for which a single parameter  $\alpha$  can be used to systematically balance the algorithm's tendency toward either objectives.

More specifically, we say a non-negative integer  $m$  is in the  $\alpha$ -margin of another integer  $n$  for  $m \leq n$  if  $\frac{n}{m} \geq \alpha$  for real-valued constant  $\alpha \in [0, 1]$ . Then, given a link cost function  $c : N \times N \rightarrow \mathfrak{R}^+$  and a constant  $\alpha$ , GREEDY-MARGIN-CONN selects an edge at each step in the following manner.

Like in GREEDY-CONN algorithm. GREEDY-MARGIN-CONN keeps adding edges until all node pairs in all groups are group-connected. The main difference is that, instead of selects an edge only among the ones that maximize the number of new group-connected node pairs, GREEDY-MARGIN-CONN chooses the one with the smallest cost among those whose utilities ( $\sum_{k=1}^K \text{NEW-CONNS}(u, v, G, N_k)$ ) are within  $\alpha$ -margin of the maximum utility, hence the name GREEDY-MARGIN-CONN.

By taking into account the edges with smaller utilities in addition to the ones with maximum utility, GREEDY-MARGIN-CONN can search and select at each step an edge of smaller cost out of the ones in the larger candidate set ( $H$ ). The parameter  $\alpha$  determines how smaller the utility of an edge can be to be in the search space—the smaller (close to 0)  $\alpha$  is, the larger the candidate set is.  $\alpha = 1$  is a special case that reverts back to GREEDY-CONN algorithm with the tie broken by the link cost (smallest cost edge is added among the ones with the same, maximum utility), while  $\alpha = 0$  is the case that only the link cost is considered in constructing the graph. Thus we expect the algorithm to perform well for our primary objective (the number of links) when  $\alpha$  is large, and to gracefully degrade as  $\alpha$  becomes smaller but improve its performance for reducing link cost.

### B. Further reducing the number of links

The greedy algorithms carefully add an edge whose utility is always positive, thus avoid adding unnecessary edge to a graph

built thus far. However, since the choice of the edges is only based on the graph topology selected in the previous steps, but not on the edges added in the later stage, the final output graph may contain *redundant* edges (to be formally defined shortly), which can be removed from the graph without making any pair of nodes in the same group disconnected from each other. Thus it is advantageous to our objective of minimizing the number of overlay links to look back at the output graph from the greedy algorithm and perform some “clean-up” procedure.

Specifically, the *redundancy* of an edge for a graph  $G$  is defined in the opposite way of its utility. More specifically, for a given graph  $G = (N, E)$ , we say an edge  $(u, v) \in E$  is *redundant* in  $G$  if, for each  $N_k \in \Omega_{u,v}$  (recall  $\Omega_{u,v}$  are the set of groups that both  $u$  and  $v$  belong to), there exists a connected path between  $u$  and  $v$  other than the edge  $(u, v)$  in the induced subgraph  $(N_k, E_k)$ . Note that an edge  $(u, v)$ 's redundancy in a given graph  $G$  can be easily verified by comparing the number of connected node pairs in  $(N_k, E_k)$  and that in  $(N_k, E_k - \{(u, v)\})$  for all  $k$ .

The following lemma shows a redundant edge can be safely removed without hurting the group-connectivity of the nodes.

*Lemma 2:* Suppose there is a redundant edge  $(u, v)$  in a graph  $G = (N, E)$ . If a pair of nodes  $u'$  and  $v'$  are  $N_k$ -group-connected in  $G$  for any group  $N_k$ ,  $u'$  and  $v'$  are also  $N_k$ -group-connected in  $G^- = (N, E - \{(u, v)\})$ .

*Proof:* Consider a redundant edge  $(u, v)$  in  $G = (N, E)$ . Let  $G_k = (N_k, E_k)$  be the subgraph induced by any arbitrary  $N_k$  in  $G = (N, E)$ . Suppose an arbitrary pair of nodes  $u'$  and  $v'$  are  $N_k$ -group-connected for some  $N_k$  in  $G$ , and let a sequence of nodes  $(u' = u_0, u_1, \dots, u_l = v')$  denote a path in between  $u'$  and  $v'$  that go through nodes  $u_1, \dots, u_{l-1}$  ( $u_i \in N_k$ ). If the path does not include a subsequence  $(u, v)$ , then the removal of edge  $(u, v)$  does not affect this path, hence  $u'$  and  $v'$  are still connected in  $(N_k, E_k - \{(u, v)\})$ . If the path does include the subsequence  $(u, v)$ , because there is a path between  $u$  and  $v$  in  $N_k$  other than the edge  $(u, v)$  by definition of the redundant edges, after the removal of  $(u, v)$ , the pair  $u'$  and  $v'$  must be connected by another path which is represented by replacing the subsequence  $(u, v)$  in  $(u' = u_0, u_1, \dots, u_l = v')$  with the node sequence representing the alternate path between  $u$  and  $v$ . This completes the proof. ■

Since removing a redundant edge does not hurt the group-connectivity of a graph, once we have a graph  $G$  compatible with  $\Omega$ , we can check the edges in  $G$  one by one to see if there are redundant edges to remove from  $G$ . As in the process of adding the edges to the graph, where the order of edge addition affect the final  $|E|$ , the order of visiting the edges in the removing process can also affect  $|E|$  after the removal process ends. In the interest of reducing the total cost of the link, we take a heuristic approach to remove the redundant edges by checking the edges in  $E$  in the decreasing order of their cost.

## V. DISTRIBUTED OVERLAY CONSTRUCTION ALGORITHM

### A. Distributed algorithm

In the distributed algorithm, each node continually makes a local decision as to whether to add and delete edges incident only to itself based on its local connectivity information for its own groups.

Given a subgraph  $G_k = (N_k, E_k)$  for some  $N_k \in \Omega_u$ , let us denote by  $\mathcal{N}_u(G_k)$  the set of neighbor nodes of  $u$  in  $G_k$ , i.e.,  $\mathcal{N}_u(G_k) = \{v \mid (u, v) \in E_k\}$ . Similarly,  $\mathcal{E}_u(G_k)$  represents the set of local edges incident to  $u$  in  $G_k$ . Also  $\mathcal{E}_u$  denotes all edges incident to  $u$  in the entire graph  $G = (N, E)$ , where  $E = \bigcup_{k=1}^K E_k$ .

The following two conditions must hold before the distributed algorithm is executed for the correctness:

- (C1) Before executing the algorithm,  $u$  has the correct, up-to-date topology information of  $G_k = (N_k, E_k)$  for all  $N_k \in \Omega_u$ .
- (C2) When  $u$  adds or deletes an edge  $(u, v)$  for some  $v \in N_k$  after executing the algorithm, no other node in  $N_k$  adds or deletes its own edge at the same time.

The above two conditions ensure the correctness of the decision made by each node. In Section V-C, we describe how a distributed protocol of message exchange between the nodes can ensure these conditions.

Given the topology  $G_k = (N_k, E_k)$  for all  $N_k \in \Omega_u$ , DISTRIBUTED-GREEDY algorithm (Algorithm 2) outputs the updated set of a node  $u$ 's local edges  $\tilde{\mathcal{E}}_u$  to be included in the graph at its completion.

---

**Algorithm 2**     *DISTRIBUTED-GREEDY:*     Input  $(u, \mathcal{N}_u, N_k, E_k) \forall N_k \in \Omega_u$ , Output  $\tilde{\mathcal{E}}_u$

---

```

1:  $\tilde{\mathcal{E}}_u \leftarrow \mathcal{E}_u$ 
2:  $V_u \leftarrow \bigcup_{N_k \in \Omega_u} N_k$ 
3:  $F_u \leftarrow V_u - \mathcal{N}_u$ 
4: for all  $v \in F_u$  do
5:    $m_v \leftarrow \sum_{N_k \in \Omega_u} \text{NEW-CONNS}(u, v, N_k, E_k)$ 
6: end for
7: if  $\max_v m_v > 0$  then
8:    $v^* \leftarrow \arg \max_{v \in F_u} m_v$ 
9:    $\tilde{\mathcal{E}}_u \leftarrow \tilde{\mathcal{E}}_u \cup \{(u, v^*)\}$ 
10:  for all  $N_k \in \Omega_u$  do
11:     $E_k \leftarrow E_k \cup \{(u, v^*)\}$ 
12:  end for
13: end if
14:  $A_u[\cdot] \leftarrow$  Sort edges in  $\tilde{\mathcal{E}}_u$  in decreasing order of edge cost
15: for  $i \leftarrow 1$  to  $|\tilde{\mathcal{E}}_u|$  do
16:    $(u, v) \leftarrow A_u[i]$ 
17:   if  $(u, v)$  is redundant in any  $G_k = (N_k, E_k)$  for  $N_k \in \Omega_u$  then
18:      $\tilde{\mathcal{E}}_u \leftarrow \tilde{\mathcal{E}}_u - \{(u, v)\}$ 
19:   end if
20: end for
21: return  $\tilde{\mathcal{E}}_u$ 

```

---

Like the centralized GREEDY-CONN algorithm, DISTRIBUTED-GREEDY chooses an edge such that it can maximize the number of new group-connected node pairs added to the current graph (lines 4-13). Besides that,

however, DISTRIBUTED-GREEDY is quite distinguished from its centralized counterpart in many aspects despite their seeming resemblance:

- Each node only adds its own edges (the algorithm returns new set of edges for node  $u$  only: line 21), by selecting one from those that are in at least one of its groups (line 2) but are not yet in its own edges (line line 3). The decision and action is thus fully localized.<sup>4</sup>
- Each node only considers the graph topology and the utility of its edges only within the groups it belongs to as the input topology includes only the groups the node belongs to.
- At each invocation of the algorithm, a node not only adds an edge, but also it removes redundant edges among its own existing edges (lines 14-20), by checking the redundancy of the existing edges in in the decreasing order of their link cost (line 14).

Note that, after each execution of the algorithm, at most one edge is added, but multiple redundant edges can be removed, and sometimes no edge is added or deleted when there is no edge with positive utility and no redundant edge. Through the repeated execution of DISTRIBUTED-GREEDY algorithm by all nodes, the collective edge addition and deletion results in the policy-compatible graph as a whole. We formally show the convergence of this process in Section V-B.

Note also that, while each node adds and deletes only its own edges, its decision is based on the utility and redundancy of the edges not only for the node itself but for other pairs of nodes in the groups that the node belongs to. This is possible because each node keeps getting updated with the graph topology of its groups through the distributed protocol described in the Section V-C.

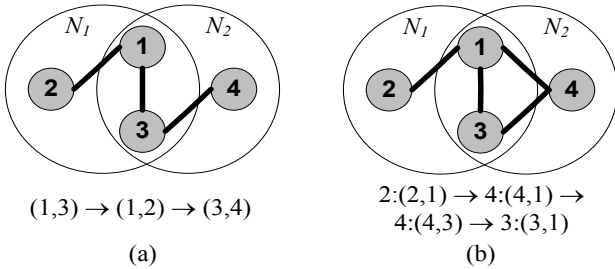


Fig. 4. Distributed greedy assignment can assign more edges than its centralized counterpart. (a) The same assignment by the centralized algorithm as in Figure 3 (b) The edges added by a run of distributed assignments by nodes in the order of 2, 4, 4, and 3. The number before each edge in the sequence indicates the node that adds that edge.

While the benefit of the distributed algorithm (reduced overhead for each node, localized decision based on local information), it comes with some expense. Due to each node's myopic view of the local network topology, the number of

<sup>4</sup>We assume that other nodes are fully cooperative to a node's addition of edges to them as long as they are in the same group. This is a reasonable assumption because a node would add edges only to the other node in the same group, and in our trust model (Section III), nodes in the same group are mutually trusted.

links collectively added by all nodes can be greater than the one assigned by centralized algorithm. Figure 4 illustrates such an example, where the edges are added by each individual nodes in a particular order of nodes 2, 4, 4 (again), and 3. This order of edge addition, especially two successive ones by node 4, is possible due to the fully distributed execution of the algorithm by each individual node, which only controls its own edges and does not force other nodes to add or delete their edges in a specific order. However, our simulation study in Section VI that compares the performance of the distributed algorithm against the centralized one shows that the performance of DISTRIBUTED-GREEDY is generally within small margin of what the centralized counterpart achieves.

Finally, the extension of DISTRIBUTED-GREEDY to the distributed counterpart of GREEDY-MARGIN-CONN is also possible and straightforward. In what is called DISTRIBUTED-GREEDY-MARGIN, the selection of an edge to be added is made not only from the set of the edges with the highest utility, but also from the edges that have the utility within  $\alpha$ -margin of the maximum utility, with the edge with the smallest cost selected from the extended set. The purpose of DISTRIBUTED-GREEDY-MARGIN is the same as that of its centralized counterpart: we would like the edges with as small cost as possible while sacrificing the primary objective of reducing the number of links in to some extent controlled by a single parameter  $\alpha$ .

### B. Convergence of distributed algorithm

We show in this section that the distributed process of adding and deleting edges by each node converges to a global graph compatible w.r.t.  $\Omega$ . As we shall see below, the convergence is guaranteed under the two conditions C1 and C2 in Section V-A.

For a given graph  $G = (V, E)$  and given groupings of nodes  $\Omega = \{N_1, \dots, N_K\}$ , let us denote  $Q_\Omega(G)$  be the total number of group-connected node pairs under  $G$  in all  $N_k \in \Omega$ , i.e.,

$$Q_\Omega(G) = \sum_{k=1}^K \sum_{u,v \in N_k} \mathbb{1}(u \text{ and } v \text{ are } N_k\text{-group-connected}).$$

Everytime a node changes its edge, the global graph  $G = (N, E)$  changes: when an edge  $(u, v)$  is added (deleted) by either  $u$  or  $v$ ,  $E$  will change to  $E \cup \{(u, v)\}$  ( $E - \{(u, v)\}$ , respectively). Since only one edge is added or deleted at a time (due to the condition C2), WOLG, we denote the evolving sequence of graphs by  $(G_0, G_1, G_2, \dots)$ .

*Theorem 2:* Starting from an arbitrary graph  $G_0$ , the evolving sequence of the graphs  $(G_0, G_1, G_2, \dots)$  generated by the distributed process DISTRIBUTED-GREEDY converges to a stable graph  $\tilde{G} = (N, \tilde{E})$  which is compatible w.r.t.  $\Omega$  in a final number of steps.

*Proof:* We first prove the convergence of the process. For any  $i = 0, 1, \dots$ , the transition from  $G_i$  to  $G_{i+1}$  occurs when some edge  $(u, v)$  is either added or deleted. Since an edge  $(u, v)$  is added only when it can create some new group-connected node pair in some  $N_k \in \Omega_{u,v}$  (line 7),

$Q_\Omega(G_i) < Q_\Omega(G_{i+1})$  if the transition is caused by an edge addition. If the transition occurs due to an edge deletion,  $Q_\Omega(G_i) = Q_\Omega(G_{i+1})$  because only redundant edges can be deleted, and, by Lemma 2, the number of connected pairs does not decrease for any group due to the removal of redundant edges ( $Q_\Omega(G_i)$  does not increase either).

Now since  $Q_\Omega(G_i)$  is bounded from above by some constant  $\Delta = \sum_{k=1}^K \frac{|N_k|(|N_k|-1)}{2}$ , the number of times that  $Q_\Omega(G_i)$  increases (hence the number of edge additions) is also bounded by  $\Delta$ . Also, for each interval between two consecutive edge additions, there can be only up to  $\frac{|N|(|N|+1)}{2}$  edge deletions. Therefore, the distributed process should terminate within a number of steps less than  $\frac{\Delta|N|(|N|+1)}{2}$ .

Next we show the converged graph is a compatible graph by contradiction. Suppose the distributed process results in a graph  $G = (V, E)$  not compatible w.r.t.  $\Omega$ . Then there must be some pair of nodes  $(u, v)$  in some group  $N_k$  that are not group-connected in  $G_k = (N_k, E_k)$ . For each such pairs, however, this condition must be recognized by all nodes in  $N_k$  as all nodes in  $N_k$  have the correct topology information of  $G_k = (N_k, E_k)$ . If no other node in  $N_k$  adds an edges to improve the connectivity of  $u$  and  $v$  in  $G_k$ , either  $u$  or  $v$  will add an edge between them since adding such an edge can certainly increase the number of group-connected pairs in  $N_k$ . This contradicts to the assumption that the process has terminated, and hence completes the proof. ■

The proof of Theorem 2 reveals the following property.

*Corollary 3:* The distributed process of DISTRIBUTED-GREEDY terminates in a polynomial number of steps w.r.t  $|N|$  and  $|\Omega|$ .

*Proof:* In the proof of Theorem 2, we have shown the total number of times DISTRIBUTED-GREEDY is executed by all nodes is bounded by  $\sum_{k=1}^K \frac{|N_k|(|N_k|-1)}{2} \times \frac{|N|(|N|+1)}{2}$  which is in  $O(|\Omega||N|^4)$ . Also since each execution of DISTRIBUTED-GREEDY is also polynomial in  $|N|$  and  $|\Omega|$ , the whole process finishes in a polynomial time steps. ■

### C. Obtaining group topology

Nodes use a distributed protocol to ensure the two conditions C1 and C2 presented in Section V-A. Our mechanism is similar in spirit to the distributed protocol proposed in [9], which is used for a distributed resource replication algorithm and its convergence. Here we provide only a high-level sketch of the protocol, with particular emphasis on how it is used in our context of distributed overlay construction.

Before a node changes its edges in a group, it initiates a three-way handshake of messages with all other nodes in the same group to get consent of other nodes in the group for the change. The handshake is composed of three stages: *REQUEST-ACCEPT/REJECT-UPDATE/ABORT*: (i) First, when node  $u$  decides to change an edge  $(u, v)$ ,  $u$  sends REQUEST messages to other nodes in the group  $u$  and  $v$  belong to, indicating  $u$ 's intention to change its edge. Other nodes respond to this message either by accepting the change or rejecting it. (ii) Any node  $w$  that receives the request from

$u$  REJECTs it if the responding node has not itself initiated a handshake process for its own edge's change, to prevent a simultaneous edge change by  $u$  in the same group. Otherwise,  $w$  ACCEPTs  $u$ 's request, after which it refrains from initiating its own change process until it receives the further outcome (UPDATE or ABORT) of the change process from  $u$ . (iii) Finally, if node  $u$  receives ACCEPT messages from all nodes in the group, it changes the edge  $(u, v)$ , and sends UPDATE messages to group nodes. If it receives at least one REJECT, it does not change the edge, and sends ABORT messages.

The handshake process serves two purposes: (i) A node  $u$  changes its edge in group  $N_k$  only when all other nodes in  $N_k$  have accepted the change. Since, if all of them have accepted the changes, they are prevented from changing their own edges until the requesting node completes its change, the edge change by  $u$  must be the only one at the time of its change. This ensures condition C2. (ii) If a node  $v \in N_k$  receives an UPDATE message from some other node  $u \in N_k$ , the edge change contained in the message is used to update  $v$ 's view of the topology of  $N_k$ . Since every node will send the UPDATE message within  $N_k$  for every edge change, and since a node  $v$  would not attempt to change its edge in  $N_k$  if some other node in  $N_k$  is in the changing process, the group topology  $G_k = (N_k, E_k)$  is always accurate by the time any node  $v$  in  $N_k$  changes its edge. This ensures condition C1.

In our simulation study in Section VI, however, we forgo the full implementation of this distributed protocol as our primary focus in this paper is to present and evaluate the performance of the distributed algorithm. Instead, we emulate the behavior of this process by having each node take the turn of changing the edges in random order, from which we are able to obtain results that indicate the dynamic property of our distributed algorithm, such as the number of edge changes. We leave the implementation and comprehensive treatment of the distributed protocol as future research item.

## VI. PERFORMANCE EVALUATION

In this section we evaluate the performance of our proposed algorithms by simulation study. In our simulation, we create random grouping of nodes  $\Omega$ , for which each node  $u$  is assigned to each group  $N_k$  with some probability  $p$ . We conduct simulations by varying the size of inputs  $|N|$  and  $|\Omega|$ , and also  $p$ . Due to space limitation, most results in this section represent the case with  $|\Omega|=10$  and  $p=0.5$ , unless otherwise specified. Other results showed similar trends. We also assign the cost of link between each pair of nodes uniformly at random in  $[1, |N|]$ . We evaluate both the number of links  $|E|$  and the total link cost  $c(E)$  assigned by the algorithms. For each different setting of  $|N|$  and  $|\Omega|$ , we obtain these performance values by averaging the ones obtained from 10 simulations.

We first see the  $|E|$ -performance of GREEDY-CONN algorithm against the optimal value. Figure 5 shows the ratio of  $\frac{|E_{greedy}|}{|E_{optimal}|}$ , where  $|E_{greedy}|$  and  $|E_{optimal}|$  are the numbers of edges assigned respectively by GREEDY-CONN and an exhaustive search. Due to the limitation in the computing

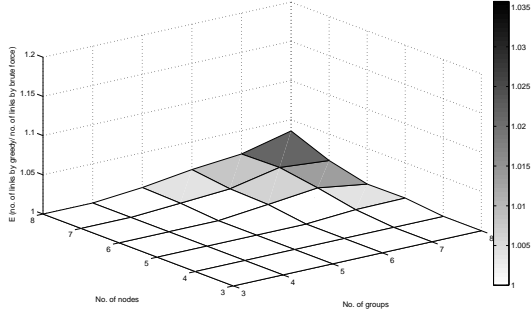


Fig. 5. Performance ratio of GREEDY-CONN versus optimal solution

power, we are only able to show the results for small ranges of  $|\Omega|$  and  $|N|$ , where the greedy algorithm achieves near-optimal  $|E|$  (less than 3% of the optimal).

Figures 6 to 9 compares  $|E|$  and  $c(E)$  performance of the centralized algorithms and distributed algorithms. For the comparison with another approach, we include the results of an algorithm that, starting from complete graph for  $G$ , removes redundant edges one by one in the decreasing order of edge cost. We call this algorithm G-MST (Group-Minimum Spanning Tree) because it is similar to Reverse-delete algorithm [8] that solves optimal minimum spanning tree problem for single tree, except that it removes only the redundant edge w.r.t. the grouping  $\Omega$ . Note that G-MST is designed only toward the goal of minimizing  $c(E)$  but not  $|E|$ , and thus the comparison against it is made to evaluate our GREEDY-MARGIN-CONN and DISTRIBUTED-GREEDY-MARGE algorithms.

Figures 6 and 7 show the average number of edges,  $|E|$ , selected by our centralized algorithms and distributed algorithm, respectively, with  $|\Omega| = 10$ . Our first observation (and rather obvious one) is that  $|E|$  increases as  $|N|$  increases. It is because with fixed  $p$ , average group size  $|N_k|$  increases with  $|N|$ , for which more links are needed for group-connectivity.

Another thing we notice in these figures is that, as intended by the design, the  $|E|$  performance of MARGIN-version of the algorithms degrades gracefully as we decrease the marginal parameter  $\alpha$  from 1 to 0; With  $\alpha = 1$ , the design of (DISTRIBUTED-)GREEDY-MARGIN algorithms are in fact equivalent to simple greedy algorithms in terms of  $|E|$  performance.

The benefit of increased  $\alpha$  in terms of  $c|E|$  performance is clear in Figures 8 and 9, where the total link cost gradually decreases as we decrease  $\alpha$ . At  $\alpha = 0$  for instance, GREEDY-MARGIN algorithm essentially results in the same performance by G-MST, which is solely designed for  $c(E)$  performance. In summary, the results in Figures 6-9 validates that our MARGIN-CONN algorithms effectively achieve our goal of balancing the weights given to  $|E|$  and  $c(E)$  performances easily by controlling a single parameter  $\alpha$ .

We can also see the distributed algorithms' performance is within small margin of their centralized counterparts. This can be verified more clearly in Figure 10 ( $|\Omega|=20$ ), where

$|E|$  resulted by the distributed greedy algorithm is within 5% to 10% range of the centralized one. Figure 11 shows the impact of varying group size with  $p$  varied to the  $|E|$  performance. Interestingly, there is a trend of increasing  $|E|$  initially when  $p$  increases in small values (0.2 to 0.4) but it decreases as  $p$  further increases. This is because when  $p$  is small, increasing  $p$  means bigger sizes of individual groups, thus requiring more links for each group. However if  $p$  keeps increasing and approaching to 1.0, the overlapping of nodes across the groups also intensifies, and links assigned to a group can be shared and reused by other groups.

Finally, in Figure 12, we plot the total number of times that the links are added or deleted by the collective process of DISTRIBUTED-GREEDY algorithm. The curves suggest that the total number of link changes increases at most linearly with the number of the groups and nodes. This in turn means the edge changes per node is kept within a constant level (less than 2 times on average in the figure) regardless of the size of input  $|N|$  and  $|\Omega|$ , suggesting our algorithm is highly scalable in large-scale networks.

## VII. CONCLUSION

In this paper, we formally studied the problem of constructing an overlay that satisfies a given set of policy-based grouping of nodes to support access control and cybersecurity. Based on graph theoretical analysis and rigorous algorithm design we presented the following major contributions: (a) We formulated an optimization problem to construct a minimum overlay that satisfies the policy-based access control groups, and formally prove that it is NP-complete; (b) We designed centralized algorithms that can achieve results very close to the optimal case (within 3%); (c) We also designed distributed algorithms that are comparable to the centralized scheme and can improve the performance by up to 30% compared to a naive construction; (d) We extended our algorithm to support a secondary optimization goal to optimize the sum of edge cost with a tunable parameter so that one can continuously change the result from edge optimization to cost optimization.

## REFERENCES

- [1] Akamai. <http://www.akamai.com>.
- [2] Sparcle policy management workbench. Online document. [http://domino.research.ibm.com/comm/research\\_projects.nsf/pages/sparcle.index.html](http://domino.research.ibm.com/comm/research_projects.nsf/pages/sparcle.index.html).
- [3] D. Agrawal, S. Calo, K.-W. Lee, J. Lobo, and D. Verma. Policy technologies for self-managing systems. 2008.
- [4] D. Agrawal, S. Calo, K.-W. Lee, J. Lobo, and A. Westerinen. Cim simplified policy language (cim-spl). In *DMTF standard*, 2009.
- [5] A. Bharambe, J. R. Douceur, J. R. Lorch, T. Moscibroda, J. Pang, S. Seshan, and X. Zhuang. Donnybrook: Enabling large-scale, high-speed, peer-to-peer games. In *Proceedings of ACM SIGCOMM*, 2008.
- [6] D. A. Hari, D. Andersen, H. Balakrishnan, F. Kaashoek, and R. Morris. Resilient overlay networks. pages 131–145, 2001.
- [7] Y. hua Chu, S. G. Rao, S. Seshan, and H. Zhang. A case for end system multicast. In *Proceedings of ACM Sigmetrics*, 2000.
- [8] J. Kleinberg and E. Tardos. *Algorithm Design*, 2006.
- [9] B.-J. Ko and D. Rubenstein. Distributed self-stabilizing placement of replicated resources in emerging networks. *IEEE/ACM Transactions on Networking*, 13(3):476–487, 2005.
- [10] I. Stoica, R. Morris, D. Karger, M. F. Kaashoek, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. pages 149–160, 2001.

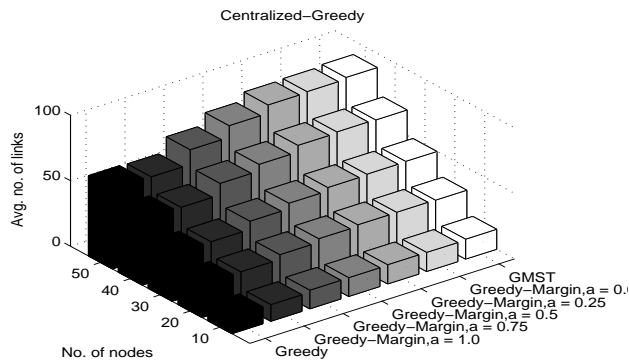


Fig. 6. Avg. no. of links with 10 groups (Centralized-Greedy)

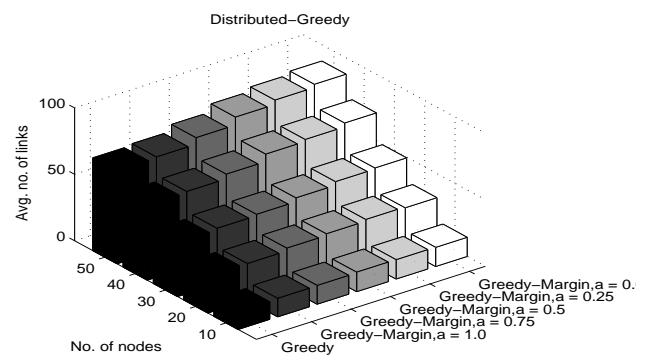


Fig. 7. Avg. no. of links with 10 groups (Distributed-Greedy)

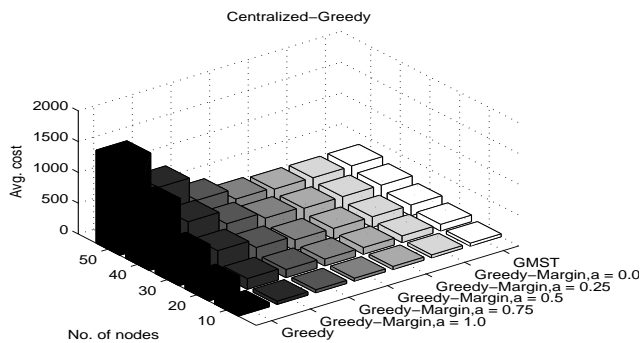


Fig. 8. Avg. cost with 10 groups (Centralized-Greedy)

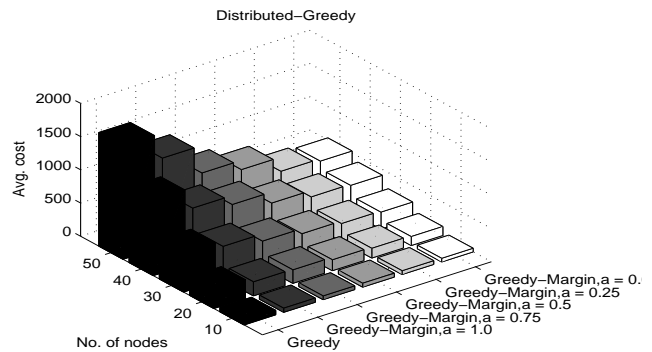


Fig. 9. Avg. cost with 10 groups (Distributed-Greedy)

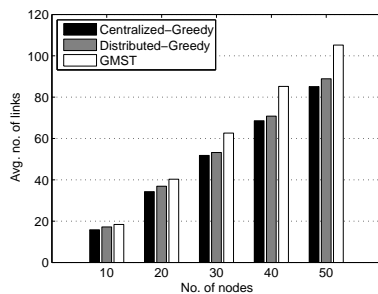


Fig. 10. Avg. no. of links with 20 groups

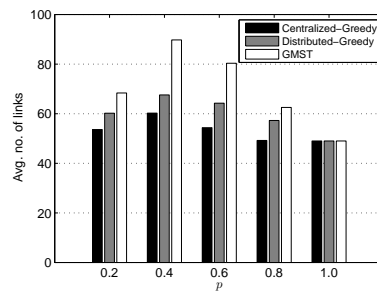
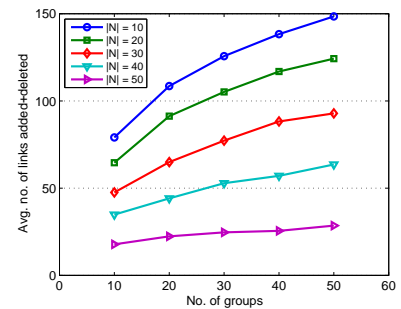
Fig. 11. Avg. no. of links with 10 groups and 50 nodes with different  $p$ 

Fig. 12. Avg. no. of links added and deleted by DISTRIBUTED-GREEDY

- [11] D. A. Tran, K. Hua, and T. Do. Zigzag: An efficient peer-to-peer scheme for media streaming. In *In Proc. of IEEE Infocom*, 2003.
- [12] K. Twidle, E. Lupu, N. Dulay, and M. Sloman. Ponder2 - a policy environment for autonomous pervasive systems. In *in Proc. Of IEEE Workshop on Policies for Distributed Systems and Networks (Policy)*, 2008.
- [13] A. K. Vishal, V. Misra, and D. Rubenstein. Sos: Secure overlay services. In *In Proceedings of ACM SIGCOMM*, pages 61–72, 2002.
- [14] D.-N. Yang and W. Liao. On bandwidth-efficient overlay multicast. *IEEE Trans. Parallel Distrib. Syst.*, 18(11):1503–1515, 2007.
- [15] B. Y. Zhao, L. Huang, J. Stribling, S. C. Rhea, A. D. Joseph, and J. D. Kubiatowicz. Tapestry: A resilient global-scale overlay for service deployment. *IEEE Journal on Selected Areas in Communications*, 22:41–53, 2004.